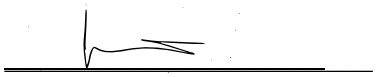
APPROVAL SHEET

Title of Dissertation: Analysis of Irregular Event Sequences in Healthcare using Deep Learning, Reinforcement Learning, and Visualization

Name of Candidate: Filip Dabek
Doctor of Philosophy, 2020

Dissertation and Abstract Approved: 

Dr. Tim Oates
Professor
Department of Computer Science and Electrical Engineering

Date Approved: November 23, 2020

NOTE: *The Approval Sheet with the original signature must accompany the thesis or dissertation. No terminal punctuation is to be used.

ABSTRACT

Title of dissertation: Analysis of Irregular Event Sequences in Healthcare using Deep Learning, Reinforcement Learning, and Visualization

Filip Dabek, Doctor of Philosophy, 2020

Dissertation directed by: Professor Tim Oates
CSEE Department

Each year over 880 million doctor visits occur in the United States. Current estimates place the yearly healthcare data records at 2,314 exabytes with projections of reaching zettabytes and yottabytes. For a country with 329 million people that spend $4 trillion on healthcare each year, understanding and uncovering the hidden patterns and trends within this big data is paramount in improving healthcare outcomes through preventative care and early diagnosis, creating a more efficient healthcare system with optimizations, and making clinicians' and patients' lives easier. With the ever increasing availability of big data, these problems are more important now than ever before.

While many event analysis and time series tools have been developed for the purpose of analyzing such datasets, most approaches tend to target clean and evenly spaced data (i.e., with a fixed time interval between observations). When faced with noisy or irregular data, it is typical to use a preprocessing step of transforming the data into being regular. This transformation technique arguably interferes on a

fundamental level as to how the data is represented, and may irrevocably bias the way in which results are obtained. Therefore, operating on raw data, in its noisy natural form, is necessary to ensure that the insights gathered through analysis are accurate and valid.

In this dissertation novel approaches are presented for analyzing irregular event sequences using a variety of techniques ranging from deep learning, reinforcement learning, and visualization. We show how common tasks in event analysis can be performed directly on an irregular event dataset without requiring a transformation that alters the natural representation of the process that the data was captured from. We focus our efforts on healthcare specifically, but also evaluate our approaches against other domains to test for generalizability. The three tasks that we showcase include: (i) predicting the probability of a future event occurring, (ii) summarizing large event datasets, and (iii) modeling the processes that create events.

Analysis of Irregular Event Sequences in Healthcare using Deep Learning, Reinforcement Learning, and Visualization

by

Filip J. Dabek

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:
Professor Tim Oates, Chair/Advisor
Professor Nilanjan Banerjee
Professor David Chapman
Professor Tim Finin
Dr. Jesus J. Caban

For my parents who

taught me the value of hard work

and allowed me to dream big.

Dziękuję bardzo!

Acknowledgments

This dissertation would not exist but for the support and guidance of many close friends and family. I would like to take a moment to acknowledge their role in assisting this endeavor.

Firstly, I would like to thank my advisor, Dr. Tim Oates, for taking me on as a pupil for my graduate career, being a helpful resource, and cultivating my growth as a student and a scientist. He helped walk me through difficult machine learning challenges and focused on ensuring that I learned the correct process and workflow. I reflect fondly upon the countless times that Dr. Oates and I discussed research as well as topics beyond my PhD: entrepreneurship, running, soccer, and events taking place around the world. He has always provided positive guidance and has helped me achieve my goal of completing this PhD thesis. Whatever future accomplishments I may yet achieve, I thank him for his role in building my career.

I would also like to thank my mentor, Dr. Jesus J. Caban, for his guidance and unwavering support for over 9.5 years. I met Dr. Caban in my freshman year during my undergraduate studies as a shy, nerdy, and unsure of himself student and he helped me grow into an outgoing, confident, and eager to tackle the world man. He has provided me countless opportunities and always guided me along the right path, seemingly acting like a father figure to me. In addition, he has taught me many invaluable lessons ranging from public speaking, PowerPoint story telling, managing teams of individuals, research & development, and always working hard each and every single day to produce results and make an impact. I owe a large

part of this thesis and the tremendous growth that I have made to him.

I would also like to thank my committee members for their support throughout my PhD and always pushing me with tough questions to ensure that I continue to do great work.

I would like to thank Dr. Daniel Mollura for providing me with freelance work opportunities to apply my talents and mentoring me on career and entrepreneurship opportunities. He has always placed my school work as a priority from the moment that I met him, all the while pushing me to become a better data scientist, product builder, and business man. He has helped lay the groundwork for a successful career ahead.

I give my thanks to the Computer Science and Electrical Engineering (CSEE) department at UMBC for providing me with a solid educational foundation for years to come. I have now been at UMBC and this department for 10 years (2010-2020) and I greatly appreciate the flexibility afforded to me as I did not take a traditional approach to my education: I worked on freelance projects throughout my undergraduate career and then a full-time research job throughout my graduate career, as well as taking advantage of the BS/MS program. The collective staff of UMBC and the CSEE Department have always been there to provide guidance and served as inspiration to me as pioneers in their fields.

I would like to thank my "work family", the NICoE Informatics team, for all of their support throughout this PhD thesis. They saw my ups and downs throughout this thesis and were always there to provide support, whether that was statistical advice, listening to me map out my problem, a simple laugh, or discussing sports

with me. The team that I was a part of afforded me an outlet to express my research interests and build products, while always pushing me to grow. I would like to especially thank Dr. Jesus Caban, Peter Hoover, Niki Noprapa, and Elizabeth Jimenez for their unwavering support and guidance.

I would like to thank my parents, Stanley and Ivona Dabek, for the selfless support they have provided me throughout the years. They have made extraordinary sacrifices to ensure that I had access to every opportunity possible to succeed in my endeavors. My parents have been shining examples of hard work and perseverance that has guided me my entire life. I would especially like to thank my Dad for being my mentor in business strategies and the pursuit of my education; and I would especially like to thank my Mom for always providing an ear to listen and for prioritizing my education over everything else. During the final completion of this PhD thesis I purchased a house and my parents took on the tremendous work of renovating it, all the while ensuring that I was not disturbed from my research. They have taught me many valuable life lessons and continue to guide me towards the right path. Thank you, Mom and Dad.

I would like to thank my sister, Gosia, and brother-in-law, Bryan Curtin, for setting an example of what it takes to succeed. Their dedication to excellence in the medical field has shown me the value of sheer determination. They work long and hard hours to be the best in their field, through nights, weekends, and holidays and are truly an inspiration to push myself harder. In addition to a strong work ethic, they have also taught me the importance of relaxation by hosting board game nights for the three of us. I would like to also thank my niece, Claire, and nephew,

Max, for reminding me of the beautiful parts of life through play, laughter, and love.

I would also like to thank my family overseas in Poland for their words of encouragement. I would like to thank my grandmother and grandfather (Babcia and Dziadzio) that repeatedly inquired, through our countless Skype calls, about the progress of my PhD and provided me words of support and guidance. I would also like to thank the rest of my family in Poland, both living and deceased, for always reminding me of my extraordinary talents and teaching me that I can do anything that I set my mind to.

I would also like to thank my friends for supporting me throughout my PhD by listening to me discuss my research, attempting to brainstorm new ideas with me, and for being a caring support system and outlet that I could always rely on. Of note I would like to thank Justin Pallett and Travis Barry for always being there for me. Additionally, I would like to thank my soccer friends for providing me with an escape from my research and for welcoming me with open arms into the various teams that I found myself on.

Finally, I would like to thank those that I met along the way: each and every person helped to shape me into who I am today, and I am forever grateful for that.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

α	alpha
β	beta
ϵ	epsilon
γ	gamma
AI	Artificial Intelligence
AHRQ	Agency for Healthcare Research and Quality
AUC-PR	Area Under the Precision Recall Curve
AUC-ROC	Area Under the Receiver Operating Characteristic Curve
AWP-Span	Adaptive-Window-PrefixSpan
BoW	Bag-of-Words
CCS	Clinical Classifications Software [2, 3]
CNN	Convolutional Neural Network
CPT	Current Procedural Terminology
DC/PC	Direct Care/Purchased Care
DoD	Department of Defense
EHR	Electronic Healthcare Records
FSM	Frequent Sequence Mining
HCI	Human-Computer Interaction
HCPCS	Healthcare Common Procedure Coding System
ICD-9/ICD-10	International Classification of Diseases
LIME	Local Interpretable Model-agnostic Explanations [4]
LSTM	Long Short-term Memory
MHS	Military Health System
ML	Machine Learning
mTBI	Mild Traumatic Brain Injury
NLP	Natural Language Processing
NN	Neural Network
PCA	Principal Component Analysis
RL	Reinforcement Learning
RNN	Recurrent Neural Network
TBI	Traumatic Brain Injury
VHA	Veterans Health Administration
Vis/Viz	Visualization

Chapter 1: Introduction

1.1 Overview

Each year over 880 million doctor visits occur in the United States [5–7]. Current estimates place the yearly healthcare data records at 2,314 exabytes with projections of reaching zettabytes and yottabytes [8]. For a country with 331 million people that spend $4 trillion on healthcare each year [9], understanding and uncovering the hidden patterns and trends within this big data is paramount in improving healthcare outcomes through preventative care and early diagnosis, creating a more efficient healthcare system with optimizations, and making clinicians' and patients' lives easier. With the ever increasing availability of big data, these problems are more important now than ever before.

While many event analysis and time series tools have been developed for the purpose of analyzing such datasets, most approaches tend to target clean and evenly spaced data (i.e., with a fixed time interval between observations). When faced with noisy or irregular data, it is typical to use a preprocessing step of transforming the data into being regular through the removal of timestamps. This transformation technique arguably interferes on a fundamental level as to how the data is represented, and may irrevocably bias the way in which results are obtained. Therefore,

operating on raw data, in its noisy natural form, is necessary to ensure that the insights gathered through analysis are accurate and valid.

In this dissertation novel approaches are presented for analyzing irregular event sequences using a variety of techniques ranging from deep learning, reinforcement learning, and visualization. We show how common tasks in event analysis can be performed directly on an irregular event dataset without requiring a transformation that alters the natural representation of the process that the data was captured from. We focus our efforts on healthcare specifically, but also evaluate our approaches against other domains to test for generalizability. The three tasks that we showcase include: (i) predicting the probability of a future event occurring, (ii) summarizing large event datasets, and (iii) modeling the processes that create events.

First, effectively utilizing historical data to make predictions for the future is important to ensure that we can properly optimize systems and tasks. Many approaches in recent years have showcased improved results in prediction tasks on sequential datasets, but there exists the potential to both expand on the existing literature by embedding the sequential nature of the dataset into a prediction model and to provide a method for prediction on irregular event sequences without biasing the dataset through a transformation.

Second, as event datasets grow in both the number of sequences that they contain and the number of events per sequence, understanding the common trajectories and the various paths that are taken by sequences is important in developing a thorough understanding of these large datasets. This issue is evermore present in irregular datasets in which produced visualizations become very messy and compli-

cated due to the inherent noise caused by the irregularity. Therefore, an approach that is able to generate an effective summary is important to ensure that a comprehensive overview can be gathered, compared to the existing technique of requiring each individual sequence to be analyzed.

Third, for each event dataset there is a process that created the events. While analyzing a dataset alone can provide information into the patterns contained, being able to accurately model the process that created the events could allow for a multitude of analyses to be performed, such as applying automatic learning to identify the path to a certain event, comparing event sequences based on the differences learned by the model, and generating a textual summary of the events that have taken place. Through this model and its applications, a more thorough understanding of the events could be obtained.

1.2 Contributions

In this dissertation, we aim to provide contributions in the analysis of irregular events through the three different tasks described in Section 1.1. While our contributions are aimed at irregular events within healthcare, because we generalized our approaches, they also apply to regular event datasets and/or other domains.

First, we propose a predictive model that utilizes BERT for predicting suicide risk in mild traumatic brain injury (mTBI) patients within a year post-injury. Our model takes as input the entirety of the patient's EHR prior to their mTBI. To preserve the diagnosis codes, as were input by a clinician, we convert them into

their corresponding textual descriptions, and to preserve the irregular nature of the EHR we embed time tokens into the constructed patient text. Through the combination of a patient NLP-based representation and the embedded NLP knowledge present within BERT, our model outperforms state of the art approaches for diagnosis prediction, standard clinical models used for suicide prediction, and other common approaches utilized for binary classification tasks.

Second, we present our adaptive window-based algorithm that builds a summary of an event dataset by introducing the notion of temporal context into frequent sequence mining algorithms. We show the mapping of the results of the algorithm to our novel event summary diagram that reduces the visual complexity and analytical capacity required. Through the summary that is produced, we show the utility of our approach to visualizing events on three datasets from different domains.

Third, we present our automatic and adaptive approach, based on reinforcement learning techniques, for modeling a process that creates events. For this task, we explore the domain of user interaction events that are captured within a visualization system; and show how our approach is able to find the optimal path to solve a specific task and to gain an understanding of the behavior exhibited by users. We perform a user study in which we show the wide applicability of our approach for understanding user interactions, and thus how models can be built to learn from the underlying structure embedded within events.

1.3 Outline

The rest of this document is broken up into five chapters.

Chapter 2 provides a formal definition for temporal events and background on the existing work for events in both machine learning and visualization.

Afterwards, the document contains three approach chapters where each chapter corresponds to a different task that we have addressed: Chapter 3 presents our BERT model that utilizes an NLP-based patient representation to predict suicide risk, Chapter 4 presents a mining algorithm and shows its results on our novel event summary diagram, and Chapter 5 introduces the concept of automatically and adaptively learning from the interaction event data generated from visual analytic systems.

Following the three chapters of our approach, we conclude the dissertation in Chapter 6.

Chapter 2: Temporal Events

2.1 Definition

It will be useful in what follows to formally define an event dataset. Event datasets can be defined as a set of sequences $D = \{S_0, S_1, ..., S_n\}$ where each sequence $S_i =< I_0, I_1, ...I_m >$ contains a timestamped series of items I_i arranged in a specific order with respect to the timestamps. The arrangement of the items can be used to derive temporal relationships within the data such as $I_i \prec I_{i+1}$ describing that I_i happened before I_{i+1}. Each itemset, I_i, contains a set of event types such that $I_i =< T_0, T_1, ..., T_p >$ and $T_i \in E$ where $E = \{e_1, e_2, ..., e_k\}$ is a dictionary of event types. An example of a dataset of sequences is shown in Figure 2.1, where each row represents a sequence S_i that contains itemsets I_i with one or more event types.

Event datasets also come in the form of time series datasets. A time series dataset is identical to the definition of an event dataset with the exception of each itemset I_i. In time series data each itemset contains a set of measurements such that $I_i =< M_0, M_1, ..., M_p >$ where M_i is an n sized vector corresponding to the n values collected at each timestep.

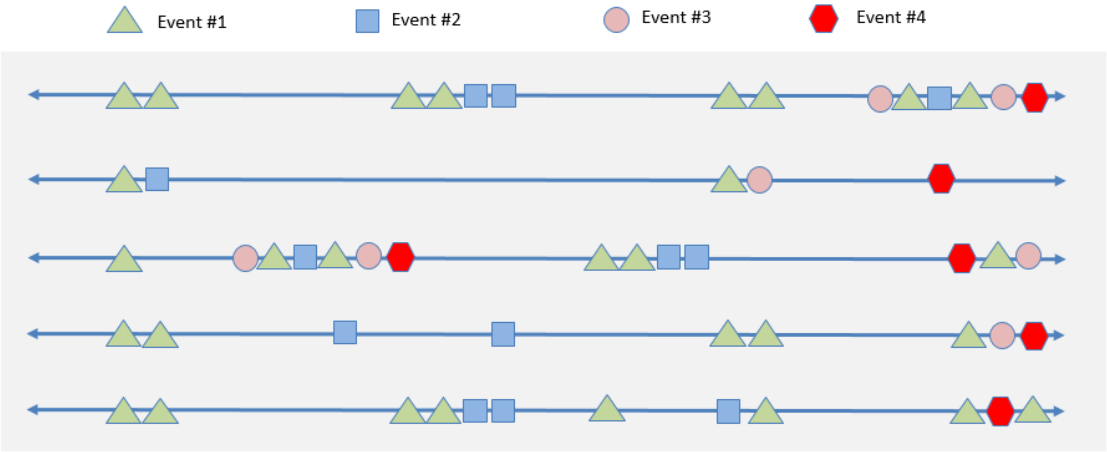


Figure 2.1: Illustration showing a set of temporal sequences with the trajectory between event #1 (triangle) and event #4 (hexagon). As the size and complexity of the data increases, there is a need to generate a visual summarization of the sequences to better understand the relationships between events.

2.1.1 Regular and Irregular

Both event and time series datasets are structured in two different ways: regular and irregular; or also referred to as evenly spaced and unevenly spaced respectively.

A regular or evenly spaced time series is one that always occurs at fixed intervals. A weather sensor that takes measurements every minute would produce a dataset of evenly spaced measurements that are always exactly 60 seconds apart. However, evenly spaced time series do not commonly occur in real world scenarios due to either missing data or the data occurring at different frequencies. In the case of a weather sensor, a single missing time point would alter the dataset from having

a perfect composition of each data point being exactly 60 seconds apart to being one that has multiple different types of intervals.

Therefore, to describe this alternate type of dataset, irregular or unevenly spaced time series are defined as a dataset where all of the intervals between timesteps are not identical. These types of datasets are common and representative of a vast array of processes that datasets are created from. For example, a dataset consisting of customer orders would not be able to exist in the realm of being evenly spaced due to a customer's shopping patterns being sporadic and not always occurring at the exact same moment in time. However, while irregular datasets commonly occur, they also pose great challenges in analyzing them.

2.2 Machine Learning

As machine learning has roots in statistical methods, the common statistical models are the initial approaches that are taken for the modelling and predicting of events and time series. These common models include: autoregressive integrated moving average (ARIMA), autoregressive–moving-average (ARMA), autoregressive–moving-average model with exogenous inputs (ARMAX), and Poisson distributions. However, to move from the standard statistical models with the hope of achieving better results, John Gamboa outlined the various deep learning approaches that have been utilized for analyzing event and time series data, and concluded that deep learning could still contribute a lot to these fields of event and time series data [10]. Thus, there has been a growing interest in developing neural network based approaches for these tasks.

Restricted Boltzmann Machines (RBM's), are a special type of neural network in which nodes are separated into two groups, only each pair of nodes from the two groups have a connection between them, and there exist no connections between nodes within a group [11]. In 2009 RBM's became popular when Taylor and Hinton developed a conditional RBM that was able to perform learning faster and more efficiently. They showed its application in modelling time series data consisting of captured motion [12]. Later in 2013, Ilya Sutskever developed a temporal RBM model that had an RBM at each timestep and further improved the performance gathered on time series tasks [13].

While RBM's are one type of neural network architecture that has been ex-

plored, one of the most popular variants utilized for time series and event-based analysis are recurrent neural networks (RNN's). Jaeger and Haas adapted RNNs based on echo state networks (ESNs) by only modifying the weights between the hidden and output nodes during training. They showed the application of this new architecture in predicting wireless communication activity, allowing for engineering processes to reduce a signal's error rate [14]. Sharat et al. implemented a better method for back-propagation across many layers and then utilized deep RNN's to predict the future values of a time series [15]. Alex Graves showed how Long Short-term Memory (LSTM) networks, a type of RNN, could be used to generate longitudinal sequences, similar to those of event sequences, through an application of predicting the next character or word in a document as well as predicting the correct corresponding characters in a handwriting task [16]. Kaiser et al. at Google Brain identified the difficulty in remembering events that rarely occur and thus they built a memory module for use in deep learning applications that is capable of performing life-long one-shot learning. They showed how their approach works on a variety of tasks, and most specifically on predicting the next event in a sequence [17].

While both these statistical and neural network approaches have been able to model time series and event datasets, they rely solely on datasets that occur at a fixed interval and are evenly spaced. In practice events do not only occur at a set frequency: network intrusion events occur at random moments in time, customers place online orders with different gaps of time between orders, etc. Common approaches to addressing these types of datasets have consisted of repeating samples or applying exponential smoothing in order to convert the irregular time series into

a regular one. This conversion, however, does not result in a true representation of the original dataset and assumes that the data exists at a frequent interval. Therefore, to avoid these standard techniques, steps have been taken to address this gap in the literature and utilize irregular data in its purest form.

From a statistical perspective, Andreas Eckner has recently identified this need and developed a framework for analyzing unevenly spaced time series data in which he defined the current limitations and formulated the problem in a way similar to that of evenly spaced data [18]. He has since presented algorithms that are able to solve some of the basic tasks, such as computing the moving averages of a dataset [19]. Zumbach and Müller also recognized the focus on regular time series and presented a toolbox of operators that could be applied specifically to unevenly spaced time series [20]. However, there still remains a gap in solving some of the more complex tasks, such as prediction and classification, which has been explored using RNN's.

Using RNN's, Lipton et al. utilized clinical measurements that were unevenly spaced in order to classify 128 different diagnoses. To overcome the issue of the irregular data, the authors repeated the previous test results for the points at which data was not available [21]. Tresp and Briegel presented a combination model consisting of both a nonlinear RNN and a linear error model that are trained together to predict the glucose/insulin metabolism of patients based on their at-home blood glucose measurements that can occur at irregular intrervals [22]. Neil et al. recognized that data comes in different frequencies and thus they modified a standard LSTM by adding a time gate that determined whether or not the node should acti-

vate at a particular timestep. This not only improved the performance on standard benchmarks, but also allowed for two connected networks to be built that accepted separate data streams at different frequencies [23]. The concept of accepting different frequences is similar to that of irregular or unevenly spaced data that does not all occur at a fixed interval. In a recent masters thesis, it was identified that predicting the time until a next event occurs in irregular datasets is a difficult and unsolved task. Therefore, the author proposed a model: the Weibull Time To Event RNN (WTTE-RNN), in which an RNN is trained to output the parameters of a Weibull distribution at each timestep and then the distribution is utilized to determine whether or not an event will occur at the corresponding timestep [24].

2.3 Visualization

In the visualization field, a vast array of approaches have addressed both time series and event analysis. Work has been done into evaluating existing approaches and providing the best interface for user exploration and analysis of datasets. Javed et al. evaluated multiple visual techniques and found that for visualizing and comparing multiple time series both small multiples and horizon graphs were most effective over longer time spans while standard line graphs were ideal for shorter spans [25]. Aigner et al. evaluated the different visual and interactive techniques for exploring time series data and stressed the importance of building visualizations and systems that are user-centered and interaction-based [26]. Hao et al. presented an approach for aggregating and visualizing time series data to fit different screen resolutions [27]. Wang, Deshpande, and Schneiderman presented a more efficient algorithm for searching for temporal patterns in an event dataset [28].

Moving beyond the standard visualization techniques, novel visualizations for understanding the composition of datasets have been constructed. CloudLines visualized multiple time series using a standard technique of representing each moment in time as a circle. However, to help spot episodic moments they mapped an importance function to the opacity and size of circles, allowing for multiple time series to easily be compared in a compact display [29]. Weber et al. mapped time series to a spiral chart and showed how using spirals revealed patterns in the data better than a bar graph. For example, plotting the intensity of the sun's rays in a spiral allows for easily identifying the points throughout the year where the rays are the

strongest [30]. TimeNotes created a hierarchy of time series data allowing for a user to select sections to zoom in onto a new panel and found that exploration of data was enhanced using their technique [31]. ActiviTree provided an interface for easily searching through event sequences through the notion of a graph. By entering a query, the possible subsequent events were drawn as branching nodes in a tree where the opacity of each edge corresponded to the frequency of that event occurring next [32]. Lin and Wei developed a symbolic approach to time series that allowed for time series to be converted into a set of symbols without suffering from the high dimensionality and lack of accurate distance measures that typical approaches such as Fourier transforms, wavelets, eigenwaves, etc. have experienced. This better symbolic representation was then utilized for various tasks, such as classification through a decision tree, and shown to achieve superior results [33]. Building on this symbolic approach, VizTree utilized the ability to convert a time series into a set of symbols and then visualized the sequence of symbols into a suffix/subsequence tree in order to arrive at a compact visual representation and make recurring subsequences easy to notice [34].

However, as the size of datasets continue to grow, summarization approaches have been increasingly more important. With this, Kumar et al. addressed the issue of summarizing large time series databases through the use of bitmaps. Their approach converted the features of a time series into a set of symbols and then created a bitmap color representation from the symbols. Through the small bitmaps created for each large dataset, they showed how a user scrolling through a list of datasets could easily distinguish the similarities and differences between them

[35]. Another way to summarize time series was accomplished through linguistic summaries, where the features and patterns of a time series are extracted into a natural language summary that is easy to understand. This was initially introduced by Yager [36], then further developed by Kacprzyk and Yager [37] and Kacprzyk et al. [38,39]; and shown to help the interpretability of a dataset through experiments on investment funds performance over eight years. LifeLines was built to aggregate and understand a series of event records by stacking multiple records on top of one another while providing the user with the ability to simplify and manipulate the display [40,41]. Later LifeLines2 extended LifeLines by building temporal summaries using bar charts that indicated the distribution of event types at each timestep using stacked bars [42]. And most recently, EventFlow built on this system by introducing the notion of merging and simplifying temporal event sequences [43]. Sips et al. built a visualization to summarize a time series at multiple different time scales, allowing for overall trends at a large time scale as well as smaller trends at small time scales to be evident; compared to that of a typical line graph [44]. In VAET, the authors recognized that previous research on E-transaction time series has focused primarily on identifying the temporal trends of transactions, but that understanding each transaction individually is also important. Therefore, they built a system that summarized the entire time series of transactions into a visualization to allow an analyst to look at both the overall trends as well as interesting individual transactions [45]. RankExplorer presented a visualization method to understand the change in search query rankings over a long time series data [46]. They built on the work of ThemeRiver [47], in which the authors visualized the change in themes for

a collection of documents over time, by utilizing the concept of flow and grouping time series together to present an overall summary of the ranking changes.

While datasets continue to grow, the types of datasets that are analyzed also continues to expand. The visualization community has also identified the need to address unevenly spaced or irregular datasets. Aris et al. presented four different representation methods to produce a normal, evenly-spaced time series from an irregular one. Their methods consisted of repeating events, sampling a dataset at lower frequencies, and representing the dataset without timestamps, thus solely relying on the event indices [48]. Stroscope was developed for irregular time series with the focus on being able to analyze both the frequency interval and actual measurements simultaneously, something that a regular time series does not have to deal with. The authors combined a line graph with a bar graph into a "ripple graph" which made these two different aspects of an irregular time series easily interpretable [49]. Schulz and Stattegger presented an approach for analyzing unevenly spaced time series in the context of paleoclimatic data. They showed how their approach did not require interpolating the dataset into a regular time series, but rather the analysis and visualizations highlighted the irregular aspect of the dataset [50].

Chapter 3: Suicide Prediction

3.1 Introduction

Every 11 minutes a person in the US dies from suicide [51]. In 2018 suicide claimed the lives of 48,000 people [51]. Beyond these numbers, 10.7 million American adults seriously thought about suicide, 3.3 million made a plan, and 1.4 million attempted suicide [52]. Suicide is clearly a large and growing public health problem and is currently the 10^{th} leading cause of death in the US [51].

Suicide rates can vary by race/ethnicity, age, and other characteristics. However, veterans and military members face significantly higher incidence of suicide than other groups. It has been estimated that veterans and active duty personnel commit suicide at a rate of 18 to 22 per day [53]. In 2014 this count was 7,400 which accounted for 18% of all suicides in the US, while the military population makes up less than 9% of the entire US population [54]. Veterans are 1.5 times more likely to die by suicide than Americans who never served in the military [54].

The World Health Organization (WHO) has called on nations to make suicide prevention a "global imperative" [55]. About 60-70% of at risk individuals that are seen by primary care practitioners prior to their suicide attempt are not properly identified [56, 57]. Many of the people who die by suicide are known to the mental

health care system well before death. In the US, two-thirds of suicide decedents are in contact with the mental health care system in the year before their death, 30% are either hospitalized for a psychiatric disorder or make an emergency department visit for a psychiatric problem during that year, and one-third of suicide decedents are in outpatient treatment for a mental disorder in the month before their death [56, 58–60].

To address this ever growing problem, a significant amount of research has been done from both medical and predictive modeling perspectives. Within the medical literature a series of risk factors have been identified: mental and addictive disorders, substance abuse, neurochemical risk factors, family matters, etc. [61]. These risk factors support healthcare workers at the point of care to be aware and be able to take action when necessary.

While risk factors help to identify patients that are at risk to have suicidal thoughts or attempt to commit suicide, predictive models have been built to summarize the risk, flag patients, and help to avoid oversights [62]. Most notably, within the military and veteran population, the US Veterans Health Administration (VHA) developed a model, REACH-VET, that identifies the patients that are at a high risk of committing suicide using their Electronic Health Record (EHR) history [63, 64]. A majority of these have used hand-tuned features, such as the presence of a depression diagnosis, which can be inaccurate and do not capture the entirety of data available [65, 66]. To build more accurate models it is necessary to utilize machine learning techniques that can analyze a large number of variables and identify the complex associations [67].

In this chapter we present our work on building a model for predicting suicide risk, where suicide risk is defined as both attempts and ideation, based off of the popular BERT model. We introduce our NLP-based technique in which we convert the diagnosis codes into their corresponding textual description before inputting them into a model. We believe that this is novel and allows for the model to discern the data in its natural form, as clinicians search for diagnosis codes via text rather than memorizing codes [68].

This chapter is organized as such: first, we compare our model against the state of the art techniques for diagnosis prediction and a famous military population model for suicide (REACH-VET). Then, we provide an in depth overview of how we built our model and design decisions made. Next, we present our work on creating a visualization to inspect the BERT model to evaluate the feature importances towards the binary classification task. Finally, we address the approaches that we attempted and did not work before concluding the chapter.

	Inpatient	Outpatient	Totals
Direct Care	98,872	29,509,032	29,607,904
Purchased Care	79,280	15,253,048	15,332,328
Totals	178,152	44,762,080	44,940,232

Table 3.1: Encounters grouped into different categories.

3.2 Dataset

3.2.1 Overview

The dataset used was collected from the Military Health System (MHS). The parameters for the collection included patients over the age of 18 who underwent an mTBI while active duty. This yielded 182,596 patients and a total of 45 million encounters.

The encounters can be separated into two different types: Direct Care (DC) and Purchased Care (PC), where Direct Care indicates that the patient went to a DoD medical facility and Purchased Care indicates that the patient sought care at a non-DoD medical facility and then the facility was reimbursed. The encounters can also be split into inpatient and outpatient, indicating whether the patient stayed for an extensive period of time or not. The four different groupings that result from these two categories, along with their respective counts, can be seen in Table 3.1. Additionally, Figure 3.1 shows the distribution of the encounter types to give a visual overview of how the dataset is weighted towards outpatient encounters.

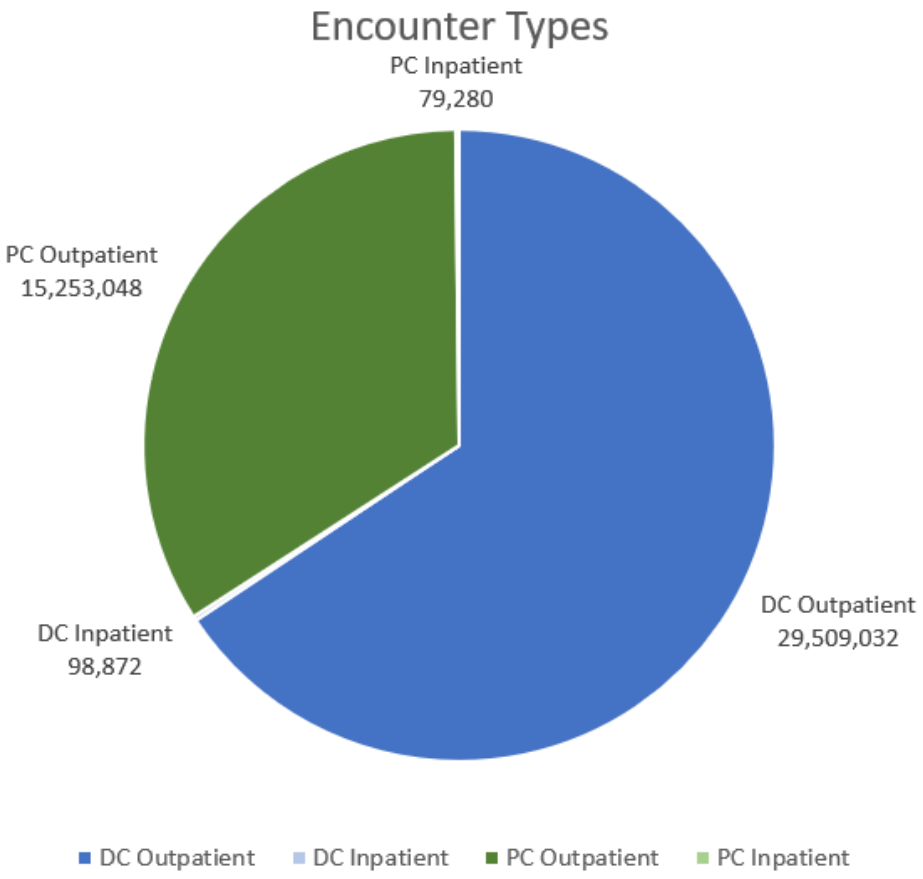


Figure 3.1: Distribution of the encounter types.

Comprehensive metadata is attached to each encounter. First, each encounter includes the date on which it occurred. For the outpatient encounters this is a single date, whereas for the inpatient encounters this consists of admittance and discharge dates. However, to create a common structure between these two types of encounters, the admittance date is assumed to be the single date for the inpatient encounters.

Basic patient demographics at the time of the encounter are also attached, these include: patient age, gender, state of residence, military branch, pay grade, and whether the patient is active at the time of the encounter.

Each encounter also includes the diagnosis and procedure codes that were assigned to it. These codes are assigned to the encounter in the order of relevance and severity. For this data pull, only the top three diagnosis codes and top three procedure codes were used. Some encounters have less than three codes or have zero diagnosis/procedure codes. The diagnosis codes are in either ICD-9, ICD-10, or DoD specific formats. As a pre-processing step, all of the ICD-9 codes were converted to their equivalent ICD-10 code. A total of 44,170 unique diagnosis codes exist within the dataset where 25 are DoD specific codes (prefixed with "DOD") and the rest are ICD-10 diagnosis codes. Compared to the over 69,000 possible ICD-10 diagnosis codes [69], this is a significant portion.

The procedure codes are in ICD-9, ICD-10, CPT, HCPCS, and DoD specific formats. No pre-processing steps were performed for these codes as these five formats vary widely and creating a mapping between them is not guaranteed to have exact matches as was possible with diagnosis codes. The total number of unique procedure

codes within the dataset is 19,751 which, compared to the 71,000 ICD-10 procedure codes [69] alone, shows that the diagnoses vary widely and that the procedures tend to be a pre-defined set.

3.2.2 Cohort Construction

Because the relation of suicide to a patient's mTBI is of interest, the cohorts were constructed with this in mind. First, the set of TBI and suicide diagnosis codes was obtained. The TBI codes were pulled from the Health.mil Surveillance Case Definitions [70] and were merged with the TBI codes supplied within the MHS data repository during the data pull to create a comprehensive list. These TBI codes provide a classification of mild, moderate, penetrating, severe, and unclassified. Because we want to make sure that our patient count remains high so that the models have a lot of examples to learn from, we keep the noise within the dataset by keeping the codes corresponding to personal history of TBI within the dataset.

To obtain the suicide codes, the Clinical Classifications Software (CCS) mapping created by the Agency for Healthcare Research and Quality (AHRQ) [3] was consulted and category *5.13* was utilized. However, the ICD-10 diagnosis code Z915, indicating personal history of self-harm, was removed due to it signifying a previous history of suicide and was assumed to not be a new occurrence.

Using these suicide codes along with the encounters allows us to identify suicide ideation, failed suicide attempts, and successful suicides. However, it must be noted that there may be other occurrences of each of these conditions within the dataset as

it may not have been reported, did not warrant a hospitalization, or was recorded within a different system. This is a natural limitation of this dataset and many health records that must be taken into account when interpreting the results.

With the TBI and suicide codes defined and because of particular interest in mTBI's, each patient's first occurrence of an mTBI was identified. This date will henceforth be referred to as "Day 0" and will be used as the prediction point, the moment in time where the model will attempt to make a prediction. The models will be given the entirety of data that is available prior to and including Day 0, and this time period is considered the "Observation Period". Days within this period will be referred to as negative days. For example, "Day -10" corresponds to the 10th day prior to Day 0.

To build a clinically applicable scenario, in which a clinician is interested in understanding the risk of the patient attempting or having thoughts of suicide within the next year, the 365 days after Day 0 are considered to be the "Prediction Period". Contrary to the observation period, these days will be referred to as positive days; such that "Day 30" corresponds to 30 days post the mTBI date/Day 0. If a patient has a suicide diagnosis code within the prediction period then they are placed into the Suicide cohort. All other patients, those that do not have a suicide code within the prediction period, are then placed into the Control cohort. A visual timeline of this cohort construction is shown in Figure 3.2.

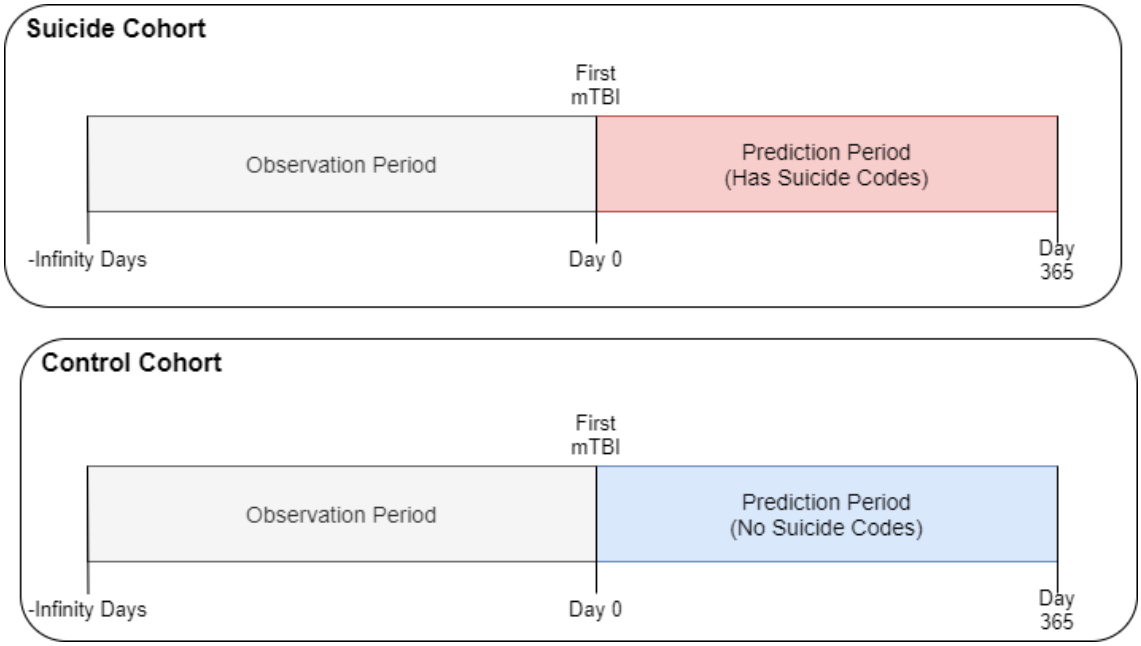


Figure 3.2: Patient timeline for constructing the cohorts. Patients that had a suicide code in the prediction period were placed into the Suicide cohort, while all others were placed into the Control cohort.

3.2.3 Statistics

To understand the dataset and the cohorts, a statistical breakdown is provided.

Of the 182,596 patients there were 180,390 that had a TBI code. While the data pull was done with the parameter of all patients possessing a TBI encounter, the removal of the personal history of TBI code removed these patients. All 180,390 patients had a mild TBI.

Looking at the other severities of TBIs: 51,154 patients had an additional TBI other than mild. 43,219 of these patients had the non-mTBI occur prior to their first mTBI (Day 0). Within these 43,219 patients: 1,071 were severe, 42,998 were moderate, and 215 were penetrating. While it may be unusual to have one of these more severe TBI's prior to the first mTBI, this could be attributed to a misclassified TBI or the patient simply having an additional more severe TBI prior to their first mTBI. This once again shows the uncertainty of health record data that must be taken into account when building a model.

Looking at the entirety of the data and not limiting within the first 365 days after the mTBI (known as the prediction period), a total of 14,831 patients had a suicide code somewhere in their history. Prior to Day Zero 10,806 patients had a suicide code. On the other hand, post Day Zero 5,003 patients had a suicide code.

Analyzing the occurrence of the first suicide code after the mTBI (day zero): the mean was 472 days, standard deviation was 658.9, twenty five percent fell within 73 days, and 202 fell within 202 days. Breaking down the first twelve months post mTBI: 2,635 patients were within the first six months, 3,122 were within the first

nine months, and 3,417 were within the first twelve months. Note that these are total patient counts for each timeframe.

Comparing the two time periods, observation period and prediction period, for the patients that had a suicide code within the first twelve months: 858 had a suicide code in both time periods, 2,561 had a suicide code in the prediction period but not in the observation period, and 10,031 had a suicide code in the observation period but not in the prediction period. Through this, it is possible to see that this dataset of patients does not have a bias towards being able to automatically predict patients with a history of suicide as high risk for a future suicide. Rather the model will have to discern which patients it should discard that have a history and which it should newly identify.

3.3 Models

This section outlines the set of models that were built and presents a comparison of the results to evaluate the optimal model. These models are split into standard models and NLP-based models. The standard models perform feature engineering that attempts to summarize the dataset through transformations that may cause a loss of information. Whereas in the NLP-based models we perform a novel transformation of the dataset into text that attempts to preserve the raw nature of the dataset and use it with pre-trained NLP models that have a rich understanding of language and words built into them.

3.3.1 Regular Models

3.3.1.1 REACH-VET

The US Veterans Health Administration (VHA) has identified the need to predict suicide risk among their retired veteran population. Because of this, they have created a statistical model, named REACH-VET, that attempts to predict the risk of suicide within a patient by utilizing the available data from the VHA.

To create this model, McCarthy et al. aggregated 381 measures that were found to be risk factors for suicide based on previous studies [63]. Some examples of these measures include: age, gender, race/ethnicity, marital status, urban or rural residence, geographic region, homelessness, any presence of mental health or substance abuse diagnosis, utilization of VHA inpatient service, and prescription for certain medications. Most of these were calculated for the previous 12 or 24 months from the prediction time point. But in other cases some measures, such as number of emergency department visits, were calculated for the previous 1, 2, 3, 6, 12, 18, and 24 previous months.

Utilizing these measures, a multivariate logistic regression model was fit and the predicted probability of suicide for each person was extracted. Due to the computational complexity of running a logistic regression model with the full features, as described by the REACH-VET authors, an extension of the 381 measures was performed by passing them through an elastic net (or penalized logistic regression) in order to arrive at the most significant and contributing predictors [64]. This

resulted in a reduction to 61 predictors which had comparable sensitivity to the original logistic regression with all 381 predictors.

To evaluate this statistical model technique against this new dataset, the 61 measures were attempted to be recreated. However, as this dataset currently does not contain the same breadth of data sources as the REACH-VET system, about half (37) of the 61 predictors were able to be re-created. These 37 included (note that only variables that begin with "Days of" are raw numbers, otherwise they are boolean flags):

Demographics

- Age ≥ 80
- Male
- Region (West)

Prior Suicide Attempts

- Any suicide attempt in prior 1 month
- Any suicide attempt in prior 6 months
- Any suicide attempt in prior 18 months

Diagnoses

- Arthritis (prior 12 months)
- Arthritis (prior 24 months)

- Bipolar I (prior 24 months)
- Head and neck cancer (prior 12 months)
- Head and neck cancer (prior 24 months)
- Chronic pain (prior 24 months)
- Depression (prior 12 months)
- Depression (prior 24 months)
- Diabetes mellitus (prior 12 months)
- Systemic lupus erythematosus (prior 24 months)
- Substance Use Disorder (prior 24 months)
- Homelessness or services (prior 24 months)

VHA utilization

- Emergency Dept visit (prior 1 month)
- Emergency Dept visit (prior 2 months)
- Psychiatric Discharge (prior 1 month)
- Psychiatric Discharge (prior 6 months)
- Psychiatric Discharge (prior 12 months)
- Psychiatric Discharge (prior 24 months)

- Any mental health (MH) tx (prior 12 months)
- Any mental health (MH) tx (prior 24 months)
- Days of Use (0-30) in the 13th month prior
- Days of Use (0-30) in the 7th month prior
- Emergency Dept visits (prior 1 month)
- Emergency Dept visits (prior 24 months)
- First Use in Prior 5 Years was in the Prior Year
- Days of Inpatient MH (0-30) in 7th month prior, squared
- Days of Outpatient (0-30) in 7th month prior
- Days of Outpatient (0-30) in 8th month prior
- Days of Outpatient (0-30) in 15th month prior
- Days of Outpatient (0-30) in 23rd month prior
- Days with outpt MH use in prior month, squared

This was then run against both a base logistic regression model and a penalized logistic regression model (elastic net).

3.3.1.2 RETAIN

Recently, the popularization of recurrent neural networks (RNNs) has resulted in state of the art results across a wide range of domains, including healthcare.

However, the adoption of these models has lowered the interpretability of these clinical models due to the "black box" nature of neural networks. Because of this Choi et al. utilized RNNs with an attention mechanism that is able to provide insight into the encounters and diagnoses that contribute towards a positive prediction [71]. They apply their technique to a heart failure prediction task, which like suicide prediction has a big class imbalance of 10 controls to 1 heart failure.

Due to the high number of ICD-9 diagnosis codes, the authors grouped the codes based off of the Clinical Classifications Software for ICD-9 [2] which resulted in a reduction from 14,000 to 283 codes. A similar process was performed for medication and procedure codes. While this reduction helps the network to have fewer codes to discern from and understand the groupings of the codes, this also results in a loss of information.

A visualization, RetainVis, was built on top of this model [72]. This dashboard-like interface displays all of the patients in a scatterplot, an overview of the different patient attributes, and a detailed patient view that shows the encounters for each patient and the extracted contribution score to the overall prediction from the RETAIN model. This tool allows users to be able to explore the RNN's predictions and attempt to make it less of a black box.

The RETAIN model was taken from a Keras implementation built from the original paper [73].

3.3.1.3 Deepr

Statistical models and neural networks typically require significant feature engineering: referencing previous medical studies, performing correlation analysis between variables, and attempting to input different combinations or alterations of variables. All of this creates a massive bottleneck for creating predictive EHR models and does not help to utilize the power of neural networks to learn from extensive, raw data. Because of this Nguyen et al. found an approach, named Deepr, where they input the diagnoses for patient encounters into a convolutional neural network (CNN) without any extensive data manipulation [74].

Borrowing from the NLP domain where CNNs have found a boost in performance over RNNs, Deepr treats each patient as a document and transforms each encounter into a sentence. The sentence transformation is performed by thinking of each diagnosis as a word. To limit the number of possible diagnoses, and thus vocabulary size, the ICD diagnosis codes were shortened to level 3. For example, the code "S06.2X9D" was shortened to "S06". Furthermore, borrowing from traditional NLP tasks, any words/diagnoses that occur infrequently were converted to a unique "RAREWORD" token. Infrequent was defined as occurring less than 100 times in the dataset.

To express a context of time within the patient document, each sentence was separated by a time token indicating the amount of time until the next encounter/sentence. The time tokens were created based on months and included five intervals: "(0-1], (1-3], (3-6], (6-12], 12+". These intervals were then represented as

```
1910 Z83 911 1008 D12 K31 1-3m R94 RAREWORD H53
Y83 M62 Y92 E87 T81 RAREWORD RAREWORD 1893 D12
S14 738 1910 1916 Z83 0-1m T91 RAREWORD Y83 Y92
K91 M10 E86 6-12m K31 1008 1910 Z13 Z83.
```

Figure 3.3: Example representation of a patient within Deepr.

a word where "(0-1]" was converted to "0-1m".

An example of a patient encoded using the Deepr technique can be seen in Figure 3.3. Using this representation, the authors input patients into a CNN and attempted to predict the risk of unplanned readmission within 6 months post discharge. The results of their experiments show that their Deepr model outperforms a standard bag of words (BoW) approach using a logistic regression model [74]. Furthermore, comparing the classification boundaries of the Deepr model against logistic regression shows that the Deepr model had a clearer boundary with minimal overfitting, while the BoW approach suffered from severe overfitting.

3.3.1.4 Baselines

For a baseline evaluation against these regular models a standard RNN was built. Each diagnosis code was transformed into a unique integer to be input to an embedding layer. We explored using all of the diagnoses or only the primary diagnosis at each encounter. Additionally, we attempted different truncations of the diagnosis codes to simplify them and create a smaller embedding space for when converting the diagnoses into integers. The embedding layer is ultimately connected

to a recurrent layer with GRU nodes [75], followed by a hidden layer, and finally an output layer with a single sigmoid output.

Additionally, borrowing from our past work on constructing windows of time and aggregating the number of diagnoses within each window [76], a second baseline was evaluated. For this we converted all of the diagnoses into a corresponding high level group as defined in Clinical Classifications Software for ICD-10 [3]. Then we divided the observation period into distinct windows, using the optimal windows identified in the past, and counted the number of diagnoses for each group. The optimal windows that are utilized are: "[-365, -30], [-30, 0]" in terms of days within the observation period. These count vectors were then input into a standard feed-forward neural network as we have found that it outperforms both logistic regression and SVM models.

3.3.2 NLP-based Models

3.3.2.1 NLP Representation

When faced with the task of entering their final diagnosis/diagnoses into an EHR system, clinicians often search for ICD codes through textual search rather than memorizing codes or searching for them directly [68]. Because of this, models that rely on the ICD code hierarchy are not able to capture the exact thought process followed by clinicians.

Additionally, there exist key terms within the textual descriptions of ICD codes that can differentiate the meaning of the diagnoses, which cannot be appreciated by a simple movement in the hierarchy. For example, the general ICD-10 code for a single episode of a major depressive disorder is **F32**. The corresponding textual descriptions for this general code and its more detailed codes include:

- **F32** Major depressive disorder, single episode
- **F32.0** Major depressive disorder, single episode, mild
- **F32.1** Major depressive disorder, single episode, moderate
- **F32.2** Major depressive disorder, single episode, severe without psychotic features
- **F32.3** Major depressive disorder, single episode, severe with psychotic features

When entering a diagnosis code, clinicians select between the more detailed versions, and looking at these descriptions it is clear that a small change in the code

can have a big impact on the severity and nature of the disease that the patient experienced. For a model to learn this big difference using only the hierarchy is a tremendous task; and when typical models, such as RETAIN and Deepr, reduce all codes to a general hierarchy removing the detailed information, this makes it even harder for models to fully grasp the nature of the patient.

Therefore, to overcome the limitation of encoding the ICD hierarchy into a model, to provide the full context of a diagnosis, and to build off of the great progress shown recently with NLP models, a transformation of the dataset into an NLP representation, similar to that of Deepr in Section 3.3.1.3 is performed.

To perform this transformation, each patient is treated as a document and each encounter is treated as a sentence. To form a textual sentence from each encounter, the diagnosis codes for each encounter are converted into the ICD-10's corresponding full text description. Using this representation, a set of NLP-based models is explored.

3.3.2.2 Word2Vec

Text cannot be directly input to a model, but requires a conversion to numbers. A popular technique to convert each word within a document into a vector is word2vec [77]. The word2vec model is a shallow neural network that is trained using one of two methods: continuous bag of words (CBOW) or skip-gram. Within the CBOW method, the network predicts a word given the words surrounding it. Whereas in the skip-gram method, the network takes a word as input and must

predict the words surrounding it.

Upon training completion, for each word the corresponding vector in the network's hidden layer is extracted and utilized as the word's representation. These vectors have been found to be effective at a wide range of NLP tasks [78] due to the information that is embedded within them. For example, with a pre-trained word2vec model released by Google [79] the vectors representing the words "king", "man", and "woman" can be extracted. Then by performing the mathematical operation $\vec{king} - \vec{man} + \vec{woman}$, the result is the vector corresponding to the word "queen".

As the number of words within each patient document will be extremely large for this dataset, inputting these word vectors into an RNN where each timestep corresponds to a word would result in a very deep network. Therefore, it is necessary to attempt to reduce the number of timesteps. We accomplished this by transforming each patient from words into sentences, as previously explained. A common technique to construct sentence vectors from word2vec is to average all of the word vectors that make up the sentence. While this potentially removes some information, it has shown success across a variety of tasks [80–83].

Given that each patient will have n sentence vectors, an RNN is built where the n sentence vectors are input for each patient and then the network attempts to predict suicide risk using a sigmoid activation output node. To compute the word vectors, Gensim's implementation of the word2vec model was utilized [84].

3.3.2.3 Sent2Vec

While the averaging of word vectors to create a sentence vector is a common technique, this results in some loss of information as the word2vec model was not built for this use case. To overcome this limitation a recent approach, sent2vec, was introduced to build the sentence vectors directly [85]. This algorithm is based off of the word2vec CBOW training method and extends it to include n-grams as an input, rather than just words; and the entire sentence is used as a context instead of the dynamic context windows in word2vec. The sent2vec model also includes an averaging operation across the words of a sentence within the network's architecture to allow for it to learn the single sentence vector.

The final sentence vectors obtained by sent2vec are input into a similary built RNN as the one described for the averaged word2vec vectors in Section 3.3.2.2. To compute the sentence vectors the publicly available C++/Python implementation of the original paper was utilized [85].

3.3.2.4 Doc2Vec

Due to the difficulty of training RNN's, especially when a large number of sentences (encounters) are present, creating a vector for the entire patient "document" is another potential avenue. With this, the algorithm Doc2vec can be utilized. Doc2vec is an approach that was created to embed an entire paragraph of sentences into a singular vector [86]. The network behind Doc2vec is trained using the same methods as word2vec, but extended to include a document/paragraph id: the

CBOW method still predicts the surrounding word but also includes the id for the input and averages or concatenates the words together prior to the hidden layer, and the skip-gram method attempts to predict context words given an id.

Using the doc2vec approach, all of a patient's encounters are concatenated together to create a document and then a vector is learned for each patient. This vector is then fed into a variety of classifiers: standard feedforward neural network, logistic regression, support vector machine (SVM), and random forest. To compute the document vectors, Gensim's implementation of the doc2vec model was utilized [84].

3.3.2.5 BERT

A popular architecture within NLP that has recently shown state of the art (SoTA) performance across many domains and tasks is BERT [87]. BERT is a model, based on transformers, that was developed by Google AI [1] and is currently used for suggestions within Google Search, GMail compose, and across other Google products [88]. To understand how the BERT model is structured and processes text, a detailed explanation is provided below.

First, recurrent neural networks (RNNs) were utilized to accept input of sequences and to allow information to persist throughout the sequence. However, the issue of exploding gradients (long-term dependencies) arises when the RNN is unable to remember a critical keyword over a long sequence [89]. To overcome this the Long-Short Term Memory (LSTM) neuron was created, where it has additional logic

gates that learn when to remember or discard each part of a sequence [90]. However, LSTMs are still faced with multiple issues: (i) over really long sequences they tend to suffer from similar long-term dependencies problems as RNNs, (ii) both RNNs and LSTMs cannot be parallelized resulting in significant computational concerns, and (iii) the distance between parts of a sequence is linear such that the probability of keeping a word decreases exponentially.

To again overcome the long-term dependencies, an attention mechanism was found in which the RNN chooses which parts of the sequence to pay attention to (attend to) when making predictions [91]. Additionally, to overcome the issue of parallelization, convolutional neural networks (CNNs) were adapted to NLP as they can easily be parallelized and the distance between any part of a sequence to another is $log(N)$ where N is the height/number of layers in the network [92–95]. Using a CNN allows for the entire sequence of words to be read in at once, rather than a word at a time, thus strengthening the prediction ability.

Using all of the above breakthroughs, a transformer is a CNN that uses multiple attention models and more specifically self-attention as seen in Figure 3.4 [1]. One of the additional innovations included within transformers is a positional encoding: the position of each word is also encoded as the positions are key to being able to perform many NLP tasks.

With this, researchers at Google built the Bidirectional Encoder Representations (BERT) model based on a transformer [87]. One of the other innovations included in the BERT model are WordPiece tokens [96]. Rather than mapping each individual word to an integer, where different prefixes and suffixes create unique

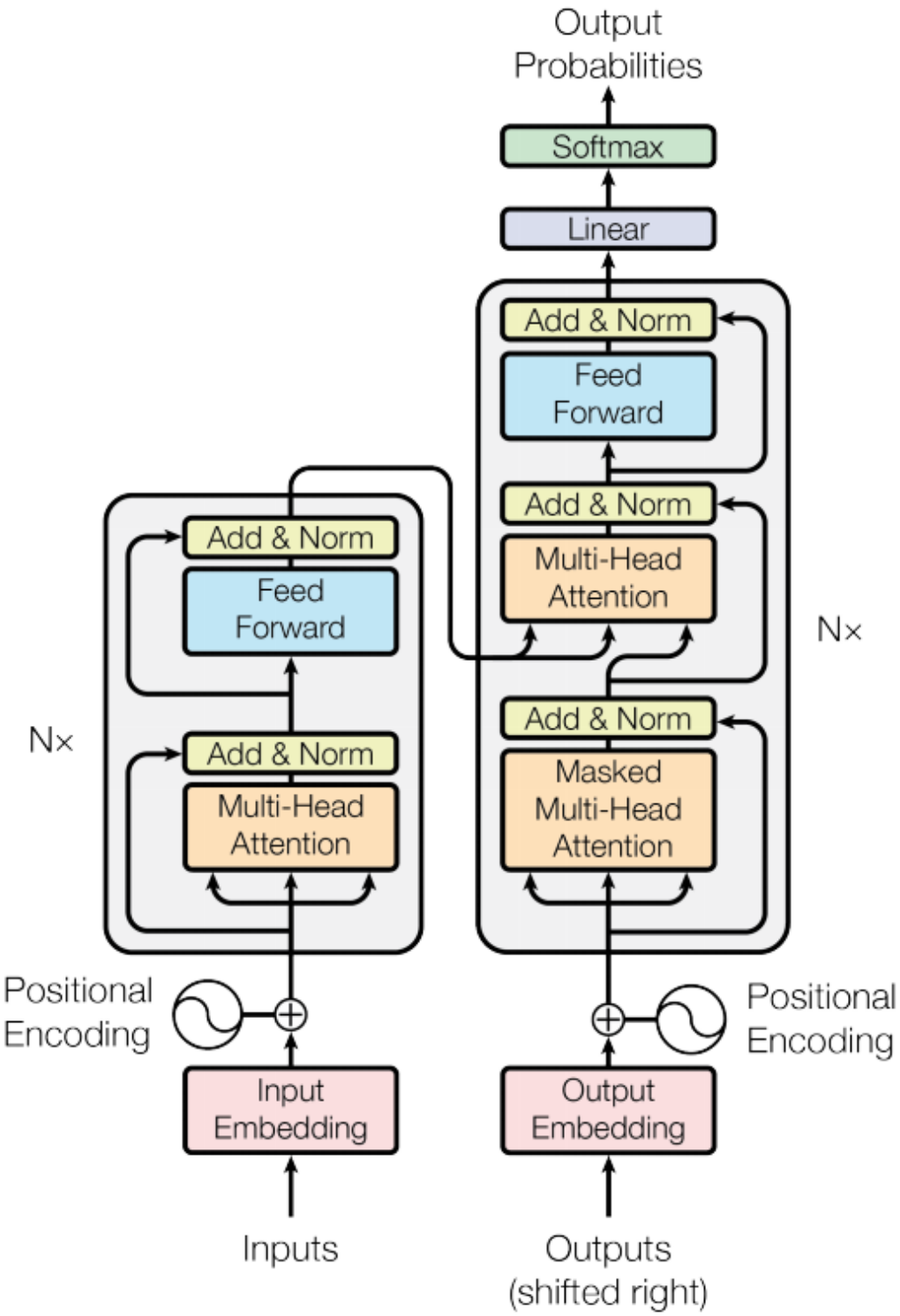


Figure 3.4: Model architecture of a transformer. Taken from original paper [1].

integers, the vocabulary of the model is represented at a subword level. For example, the word "walk" is broken up into "wa" and "##lk" or "embeddings" is broke up into: "em", "##bed", "##ding", "##s". This allows for the model to handle a larger variation of words with a smaller set of tokens.

The BERT model training process is broken up into two stages: pre-training and fine-tuning. During pre-training a language model that understands the associations between words is built. During fine-tuning, the model is applied to a specific NLP task. This shows how the knowledge embedded within BERT can be applied to a wide variety of tasks and domains.

The pre-training process attempts to maximize its performance on two tasks simultaneously: next sentence prediction (NSP) and word masking. For the first task the model is given two sentences, A and B, and it must predict the probability that B follows and is connected to A. For the second task, 15% of the words in a sequence are replaced with a universal "[MASK]" token and the model must predict the original word that occupied the space. This pre-training process was run on the Wikipedia (2,500M words) and BookCorpus (800M words) [97] datasets for 1 million update steps.

For the fine-tuning process, an untrained layer of neurons is added to the end of the model and then it is trained for the specific task of interest.

Because the BERT models trained by Google on the Wikipedia and BookCorpus datasets have a maximum sequence length of 512 tokens, the patient document representation as described in Section 3.3.2.1 was truncated to only include the first 500 tokens. Further work on understanding the impact of this truncation is explored

in Section 3.4.

3.3.2.6 NLP Baseline

To establish a baseline performance on our dataset, a bag of words (BoW) approach was used. Term frequency-inverse document frequency (TF-IDF) [98] was the preferred BoW method of choice due to its popularity and ability to encode the statistical importance of a word to a document. The TF-IDF vectors were fed into logistic regression, elastic net, and support vector machine models.

3.3.3 Results

3.3.3.1 Metrics

Given a set of patients, a model predicts the probability that each patient will be at risk for suicide in the future. These probabilities can be used to identify which group to place each patient into: suicide or control. Due to the significant class imbalance, the model performance was evaluated using the precision-recall curve. To understand this curve, understanding precision, recall, and their associations is important.

Precision is the measure of how well the model can predict suicide patients out of all of the patients that it placed into the group: given 10 patients predicted as suicide but only 4 are correct then the precision is $4/10 = 40\%$. Recall is the measure of how well the model is able to identify all of the suicide patients: given 25 total suicide patients in the test set and 4 of them were correctly predicted as suicide then the recall is $4/25 = 16\%$.

It is difficult to perform well at both precision and recall at the same time. If precision is high, then recall tends to be low. If recall is high, then precision tends to be low. For example, if a model can identify a single patient as suicide then the precision will be 1.0 or 100%, but the recall will be awful. Similarly, if the model attempts to identify all suicide patients to achieve high recall, it is bound to mistakenly add non-suicide patients which will lower its precision.

With this in mind, the precision-recall curve shows the trade off between the

two. An example curve is shown in Figure 3.5. With this the trade-off between precision and recall can be seen. An end-user could choose to have a recall of about 10% and thus obtain a high precision of 80%. This would result in a small number of patients to be predicted, but very accurately. Alternatively to attempt to predict a lot of patients with a low accuracy, the end-user could select a recall of 80% and obtain a precision of about 40%.

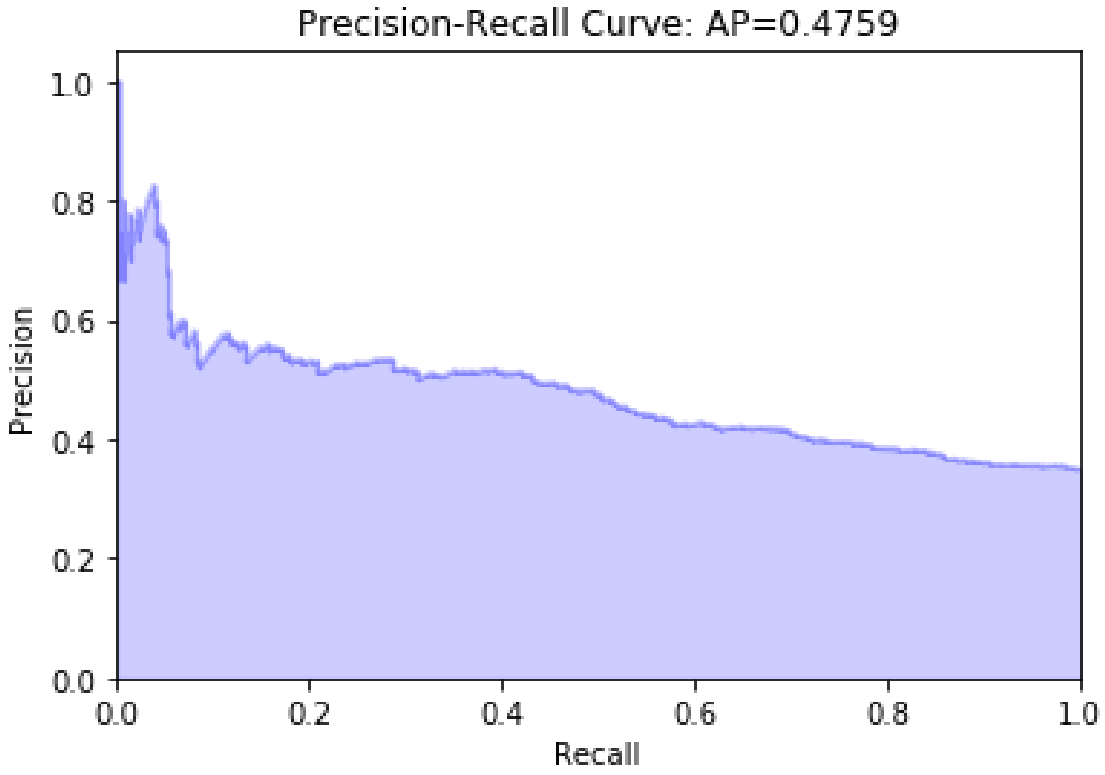


Figure 3.5: Example PR Curve.

To summarize the curve in a singular value, for easy comparison of model performances, the area under the curve can be calculated. This is referred to as the Area Under the Precision Recall Curve (AUC-PR). Of note the AUC-PR metric is different than the Area Under the Receiver Operating Characteristic Curve

(AUC-ROC) and has been found to be more informative in the context of binary classification tasks on class imbalanced datasets [99, 100].

The AUC-PR will typically have smaller values. To give context into expected values, two recent papers measure their performance in terms of AUC-PR. Edward Choi et al. build a multilevel medical embedding, named MiME, for heart failure prediction [101] where the class imbalance was about 10 to 1 (3,414 positive out of 30,764 total cases). MiME and baseline models were tested against several experiments and the comparisons between the baseline and MiME respectively are as follows: (0.2462 and 0.2831), (0.2771 and 0.2750), (0.2505 and 0.2589), and (0.4415 and 0.4787).

In another paper, Kemp et al. from Google Health build a hierarchical recurrent neural network model for discharge diagnosis classification [102]. While the class breakdowns are not cited, two tasks were evaluated using the AUC-PR metric. For the first task, mortality prediction, the baseline bag-of-words (BoW) model achieved 0.479($\pm$0.008) while the author's model achieved 0.479($\pm$0.007). For the second task, all ICD-9 diagnosis codes at discharge were attempted to be predicted: BoW was at 0.331($\pm$0.001) while the author's model was at 0.352($\pm$0.001).

Within both of these papers, the difficulty to obtain a high AUC-PR score compared to the AUC-ROC has been brought into context. To provide more information, we will report the loss and AUC-ROC in addition to the AUC-PR.

3.3.3.2 Optimization

For each model that was presented in Sections 3.3.1 and 3.3.2, different hyperparameters were explored and the configuration with the best performance is reported. A train/val/test split methodology was utilized where the dataset was split into training and testing using an 80%/20% split and then the training set was split again using 80%/20% to arrive at the final training and validation sets. To account for the variation that exists within the patient dataset, this train/val/test split was performed three separate times to construct three datasets. When reporting the results the metrics across these three datasets are aggregated and the mean and standard deviation are reported.

Below the attempted hyperparameters and final best configuration of each model is provided:

- **REACH-VET**
  - *Final* A linear elastic net with alpha of 0.01, l1 ratio of 0.15, maximum iterations of 1000, and tolerance of 0.0001.
  - *Explored* A comparison between logistic regression, elastic net utilizing stochastic gradient descent (SGD), and a native linear elastic net were tested. For the elastic net alpha's of $[0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0]$ were explored. For the logistic regression C's of $[0.25, 0.5, 0.75, 1, 2, 4]$ and both "l1" and "l2" penalties were explored.
- **RETAIN**

- *Final* All of the diagnoses were utilized for input to the network. Embedding and recurrent dimensions were 128, l2 of 0.0001, maximum timestep length before truncation of 300, 10 epochs, batch size of 32, and dropouts (both embedding and context vector) of 0.6.
- *Explored* The utilization of all diagnosis codes compared to only the primary diagnosis for each encounter was tested. Epochs $[5, 10]$ and dropouts of $[0, 0.6]$. Bayesian Optimization [103] was attempted as it has been found to be useful [104], but no improvement and rather a decrease in performance was observed.

- **Deepr**
  - *Final* 250 filters, kernel size of 3, embedding layer of 50, hidden layer of 250, learning rate of 0.001, 5 epochs, 32 batch size, and dropouts (both embedding and output) of 0.4.
  - *Explored* Epochs $[2, 5, 10]$, learning rates $[0.1, 0.01, 0.001, 0.0001]$, dropouts $[0.0, 0.2, 0.4, 0.6, 0.8]$, embedding dimension $[25, 50, 100]$, hidden dimension $[125, 250, 500]$, filters $[125, 250, 500]$. Bayesian Optimization [103] was attempted as it has been found to be useful [104], but no improvement and rather a decrease in performance was observed.

- **RNN with Embedding Baseline**
  - *Final* All diagnosis codes truncated to first two characters, embedding layer of 25, recurrent layer of 50 using GRU nodes, hidden layer of 50, 5

epochs, 32 batch size, no dropout, and learning rate of 0.001.

  - *Explored* Using all diagnosis codes versus only the primary at each encounter, truncating diagnosis codes to the first $[1, 2, 3]$ characters, epochs $[2, 5, 10]$, learning rates $[0.01, 0.001]$, embedding size $[25, 50, 100, 200]$, hidden dimension $[25, 50, 100]$, recurrent layer dimensions $[25, 50, 100]$.

- **Window Configuration Baseline**

  - *Final* Epochs of 20, learning rate of 0.0001, hidden dropout of 0.4, hidden layer activation function of tanh, hidden layer of 100 nodes, batch size of 32.

  - *Explored* Window configurations tested: "$[-365, 0]$", "$[-730, 0]$", "$[-365, -30], [-30, 0]$", "$[-730, -365], [-365, -30], [-30, 0]$", epochs $[5, 10, 20, 30]$, learning rates $[0.01, 0.001, 0.0001]$, hidden layer node counts $[50, 100, 200]$, hidden layer activation functions $[relu, tanh]$, hidden layer dropout rate $[0.0, 0.2, 0.4, 0.6, 0.8]$.

- **Word2Vec RNN**

  - *Final* Word2Vec: size of 100, window of 5, min count of 1, 5 iterations. RNN: learning rate of 0.0001, 10 epochs, hidden layer of 50, embedding layer of 400, recurrent layer of 200, batch size of 32, zero dropout.

  - *Explored* Word2Vec: Default gensim implementation. RNN: Epochs $[10, 20]$, learning rates $[0.1, 0.01, 0.001, 0.0001]$, embedding dimensions $[200, 400, 600]$, hidden dimensions $[50, 100, 200]$,

recurrent dimensions $[100, 200, 400]$,

recurrent layer dropout $[0.0, 0.2, 0.4, 0.6, 0.8]$.

- **Sent2Vec RNN**

  - *Final* Sent2Vec: Default FastText implementation [85]. RNN: Learning rate 0.0001, recurrent layer dropout rate 0.4, embedding size of 200, recurrent layer number of nodes 200, hidden layer of 100 nodes, 20 epochs, batch size of 32.

  - *Explored* Sent2Vec: Default FastText implementation [85]. RNN: Learning rates $[0.1, 0.01, 0.001, 0.0001]$, epochs $[5, 10, 20]$, recurrent layer dropout rates $[0.0, 0.2, 0.4, 0.6, 0.8]$, embedding dimensions $[200, 400, 600]$, hidden layer dimension $[50, 100, 200]$, recurrent layer dimension $[100, 200, 400]$.

- **Doc2Vec NN**

  - *Final* Doc2Vec: Default Doc2Vec implementation in scikit-learn [105]. NN: hidden dropout of 0.5, suicide class weight of 100, 20 epochs, batch size of 32, learning rate of 0.01, hidden layer activation relu.

  - *Explored* Doc2Vec: Default Doc2Vec implementation in scikit-learn [105]. NN: hidden droput rates $[0.0, 0.3, 0.5, 0.7, 0.9]$, hidden layer dimensions $[25, 50, 100, 200, 400]$, suicide class weights $[1, 50, 100]$.

- **Doc2Vec LogReg**

  - *Final* Doc2Vec: Default Doc2Vec implementation in scikit-learn [105].

LogReg: scikit-learn default implementation [105] using liblinear solver. L2 penalty, C of 1.0, 100 maximum iterations, suicide class weight of 100.

- *Explored* Doc2Vec: Default Doc2Vec implementation in scikit-learn [105]. LogReg: suicide class weights $[1, 100]$, C values $[0.2, 0.5, 1, 2, 4]$.

- **Doc2Vec SVM**
  - *Final* Doc2Vec: Default Doc2Vec implementation in scikit-learn [105]. SVM: scikit-learn default SVC implementation [105].
  - *Explored* Doc2Vec: Default Doc2Vec implementation in scikit-learn [105]. SVM: scikit-learn default SVC implementation [105]. **Long time to converge.*

- **Bag of Words (BoW) SVM**
  - *Final* Default TF-IDF implementation in scikit-learn. SVM: default SVC implementation in scikit-learn [105].
  - *Explored* Default TF-IDF implementation in scikit-learn. SVM: default SVC implementation in scikit-learn [105]. **Long time to converge.*

- **BERT**
  - *Final* "bert-base-uncased" model base, diagnoses as input, time words for tokens, 2 epochs, truncation at 500 tokens using the oldest encounters first, no pre-training, and no demographics.

– *Explored* A comprehensive exploration into optimizing BERT is provided in Section 3.4.

3.3.3.3 Evaluation

	Train		
Model	**Loss**	**AUC-ROC**	**AUC-PR**
REACH-VET	—	0.531(±0.03)	0.021(±0.00)
RETAIN	0.074(±0.00)	0.871(±0.00)	0.177(±0.01)
Deepr	0.074(±0.00)	0.902(±0.01)	0.209(±0.02)
Basic RNN with embed*	0.075(±0.00)	0.860(±0.00)	0.179(±0.01)
Window Configuration*	0.070(±0.00)	0.894(±0.00)	0.286(±0.01)
Word2Vec RNN	0.085(±0.00)	0.764(±0.00)	0.067(±0.00)
Sent2Vec RNN	0.086(±0.00)	0.748(±0.03)	0.081(±0.02)
Doc2Vec NN	1.731(±0.04)	0.779(±0.01)	0.090(±0.00)
Doc2Vec LogReg	—	0.629(±0.01)	0.025(±0.00)
Doc2Vec SVM	—	0.413(±0.27)	0.110(±0.17)
BoW*	—	0.640(±0.05)	0.041(±0.01)
BERT	0.008(±0.00)	0.889(±0.00)	0.298(±0.01)

Table 3.2: Performance of various models on the suicide prediction task (* denotes baseline) on the train datasets.

	Test	
Model	**AUC-ROC**	**AUC-PR**
REACH-VET	0.530(±0.03)	0.021(±0.00)
RETAIN	0.723(±0.02)	0.073(±0.01)
Deepr	0.831(±0.01)	0.146(±0.00)
Basic RNN with embed*	0.813(±0.00)	0.136(±0.01)
Window Configuration*	0.827(±0.00)	0.173(±0.01)
Word2Vec RNN	0.760(±0.01)	0.064(±0.00)
Sent2Vec RNN	0.625(±0.03)	0.034(±0.00)
Doc2Vec NN	0.741(±0.03)	0.072(±0.01)
Doc2Vec LogReg	0.624(±0.01)	0.026(±0.00)
Doc2Vec SVM	0.497(±0.07)	0.023(±0.01)
BoW*	0.620(±0.04)	0.029(±0.01)
BERT	0.868(±0.01)	0.227(±0.02)

Table 3.3: Performance of various models on the suicide prediction task (* denotes baseline) on the test datasets.

Tables 3.2 and 3.3 contain the performance of the regular and NLP-based models that were evaluated on the train and test datasets respectively. Analyzing this table it is evident that the BERT model outperforms the other techniques in terms of both AUC-ROC and AUC-PR across both the train and test data, with the sole exception of the train AUC-ROC being higher for two other models.

Our decision to utilize both the AUC-ROC and AUC-PR as metrics, with a primary focus on the AUC-PR, is validated by the results. Most models that perform poorly in terms of the AUC-PR tend to have a lower AUC-ROC. A generalization that can be realized by the table is that an AUC-PR of less than 0.100 coincides with an AUC-ROC of less than 0.800. However, an increase in one metric does not directly translate to an increase in the other metric. For example, the Deepr model has an AUC-ROC/AUC-PR of 0.831/0.146 whereas the Window Configuration baseline model has 0.827/0.173. An increase in the AUC-PR was accompanied with a decrease in the AUC-ROC. Ultimately, we can infer that we can utilize both metrics for an overall decision, but for the best performing models the AUC-PR should be the deciding factor as was also previously outlined in Section 3.3.3.1.

The two state of the art (SoTA) approaches, RETAIN and Deepr, had drastically different results. RETAIN could be classified as a poor performer with its AUC-ROC under 0.800 and AUC-PR under 0.100. Whereas Deepr was the third best model in terms of AUC-PR, which was double that of RETAIN (0.073 versus 0.146). We initially hypothesized that the reason for RETAIN's poor performance was due to its simplification of diagnosis codes using the CCS mapping [2]. However, a similar simplification was performed for the Window Configuration model

which did not see the same performance decrease. Thus, we believe that this poor performance can be attributed to the difficulty of training an RNN and the required truncation of sequences. In contrast, the Deepr model chose to simplify the diagnosis codes in their approach to a lesser degree by truncating each code to its first three characters. This, combined with the CNN utilized, seems to have benefited the Deepr model greatly. It should be noted that the Deepr model also removed the rare diagnosis codes, with good reason. However, our BERT model does not have this limitation and allows us to keep the dataset intact through the conversion of codes to their textual description.

The models that performed the worst across all metrics, REACH-VET, Doc2Vec LogReg, Doc2Vec SVM, and BoW, all shared a commonality of utilizing either a logistic regression or SVM model as its classifier. These models also had two different types of inputs: (REACH-VET) one was hand-selected count features and (Doc2Vec & BoW) the other was our NLP-based patient documents. The structure of these features are drastically different, which allows us to conclude that logistic regression and SVM based models do not seem to be a good fit for these tasks compared to neural networks. However, we cannot conclude that both of these feature types are not suitable for this task as they are reused across other models.

The baseline models, which we define as models that are commonly used across domains and not specifically tailored towards a specific task, varied widely in performance. The BoW model performed the worst, while the other two were within the top performing models. The "Basic RNN with embed" model outperformed the RETAIN model, when it utilizes a similar technique of simplifying diagnosis codes

and inputting them into an RNN. This shows that the additional modifications to RETAIN provided no benefit for our suicide prediction task. Much to our surprise, the Window Configuration model was the second best model out of all models that we tried. This model shows the benefit of counting the number of diagnoses per time interval/window as a baseline, but at the same time its high training AUC-PR, that rivals that of the BERT model, did not translate to the test AUC-PR shows its lack of generalization and the relative ease of overfitting it.

Our NLP-based models, excluding the BERT model, did not perform as well as expected and overall performed worse than the "regular" models. The Doc2Vec NN model matched the performance of the SoTA model, RETAIN, but otherwise ended up being the best performer within this group. We ultimately think that these poor results are due to the difficulty of summarizing the large text documents that we created as well as the difficulty of training RNN's over long sequences.

With that, the BERT model was the best performing model by a drastic margin. The BERT model had a higher testing AUC-PR than that of the second best model by a factor of 0.054. This is equivalent to the difference between the RETAIN (SoTA) model and the worst performing, VHA created, REACH-VET model. This also rivals the drop off from the worst of the higher performing models (AUC-PR ≥ 0.100) to the best of the lesser performing models (AUC-PR < 0.100). Clearly our NLP approach did not fare well across the majority of models, but the BERT model was the exception. We hypothesize that our NLP-based patient representation pares strongly with the recent growth of NLP models, most notably the BERT model, and that these two make for a great pair. Additionally, while the

baseline models perform reasonably well, they suffer from a lack of generalizability seen in the transition from training to testing data and they also lack the ability to provide that extra boost in performance needed for this task. This boost can be the difference needed to identify a patient and save a life.

3.3.3.4 BERT Evaluation

With the BERT model outperforming all of the approaches, an in depth evaluation of its predictions and results was performed. This evaluation was done on the test data to understand its application on a new dataset. For this evaluation a cut point of 0.2 was used on the predictions, such that patients that were predicted to have a probability of 0.2 or higher were classified as suicide and anything below as control.

The test set contained 36,078 patients and of these 692 were positive for suicide. The AUC-ROC is 0.8748 and the AUC-PR is 0.2444. The confusion matrix for these predictions is shown in Figure 3.6 and the Precision-Recall (PR) curve is shown in Figure 3.7. From the confusion matrix it can be seen that the model predicts the control patients very well, as to be expected. Only a minimal 319 control patients were incorrectly predicted. This results in precision and recall scores of 99% for the control patients. The suicide patients are much more difficult to predict and thus 160 of the 692 were correctly predicted resulting in a precision score of 33% and recall score of 23%.

While the model is not able to predict all of the suicide patients, it is important for the model to at least be able to predict the high risk patients well. Looking at the PR curve, it can be seen that at about a recall of 10% the precision is relatively high at about 70%. This is promising for the deployment of such a model in a clinical setting, where the end-user is able to choose the configuration of the model that they would like to utilize. With this, we break the patients down into risk

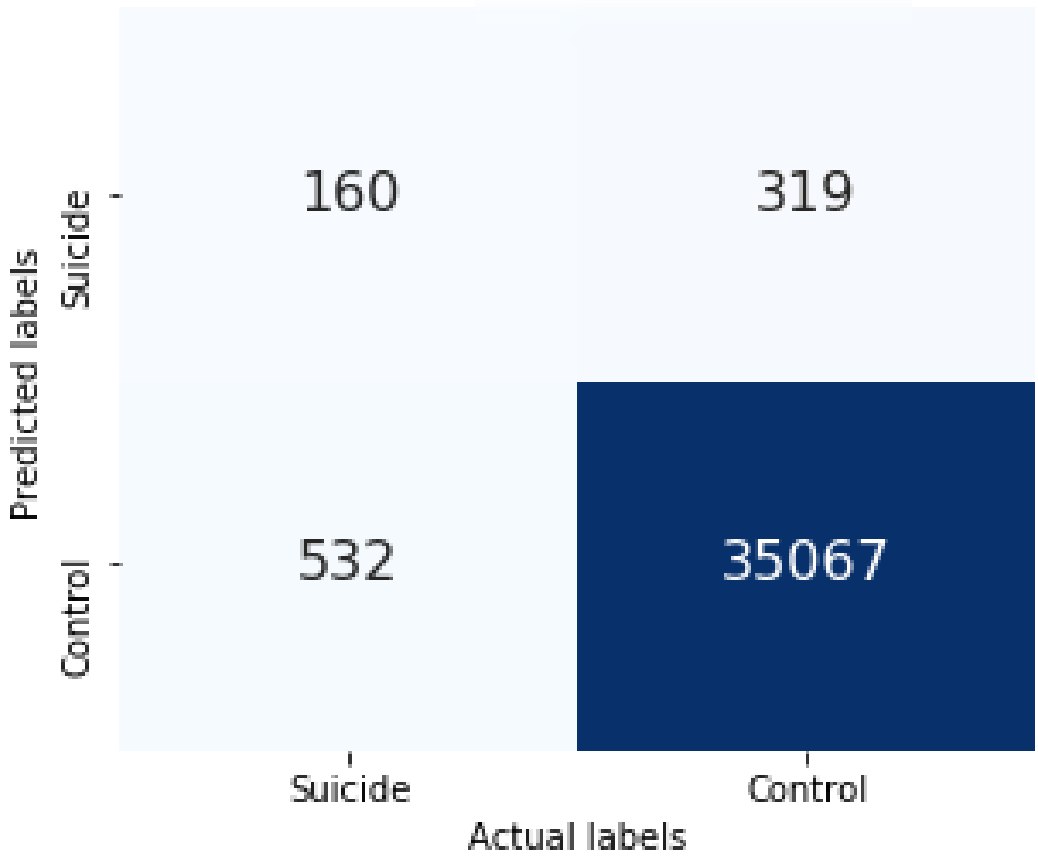


Figure 3.6: Confusion matrix for the test set.

categories based on their predicted probabilities.

Figure 3.8 presents a histogram of the predicted probabilities showing a significant skewed right distribution. Table 3.4 provides the raw bin cut points and the respective patient count for each bin. Using these probabilities we construct a hierarchy of risk for patients:

- **High**: $0.48 \leq P(Suicide) \leq 1$
- **Medium**: $0.2 \leq P(Suicide) < 0.48$
- **Low**: $0 \leq P(Suicide) < 0.2$

Bin	Count
0.0749	34,207
0.1553	1092
0.2357	463
0.3161	163
0.3965	59
0.4769	38
0.5572	28
0.6376	11
0.7180	12
0.7984	5

Table 3.4: Probability bins and counts for the histogram shown in Figure 3.8.

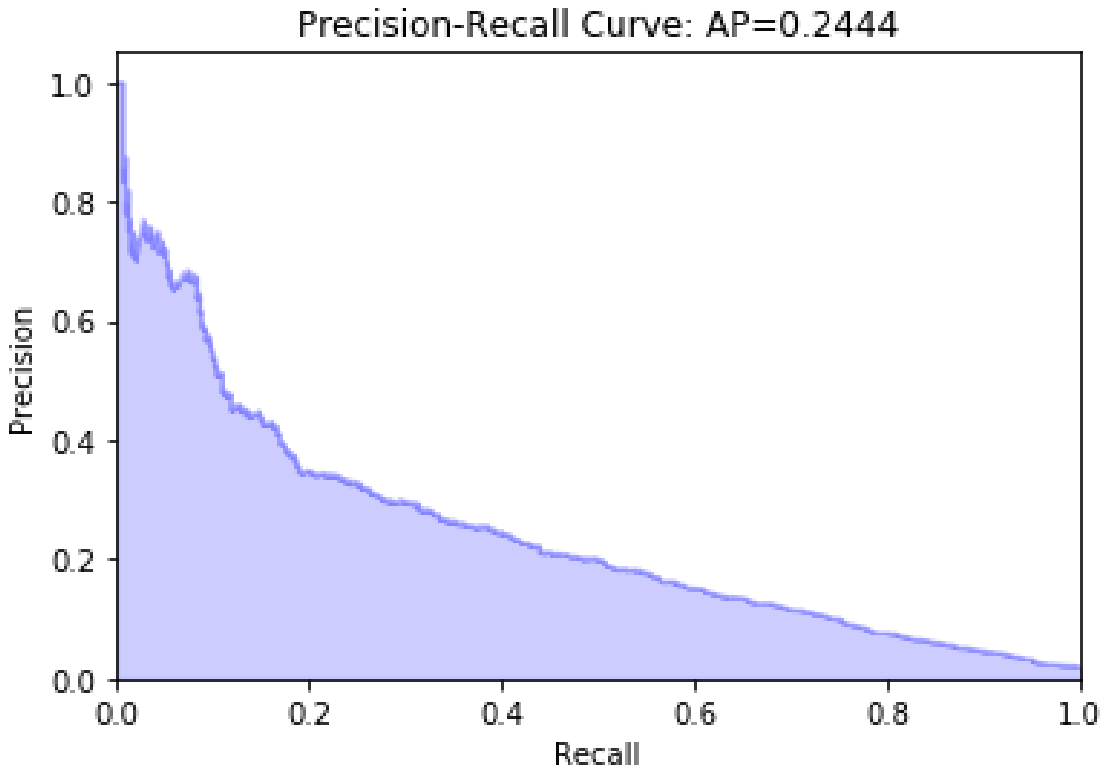


Figure 3.7: Precision-Recall Curve for the test set.

With these risk levels, the test set is broken up such that 53 (0.15%) are high risk, 426 (1.18%) are medium risk, and 35,599 (98.67%) are low risk. Breaking down these risk levels further: the PR Curve and confusion matrix for the high risk patients are shown in Figures 3.9 and 3.10, medium risk are shown in Figures 3.11 and 3.12, and low risk are shown in Figures 3.13 and 3.14. The AUC-PR for each of these levels is 0.7843, 0.3895, and 0.1167 respectively.

The high AUC-PR for the high risk patients is very promising and shows the model's ability to predict these patients well. Furthermore, the medium risk patients also maintain a high AUC-PR of 0.3895 which outperforms the model's overall AUC-PR of 0.2444.

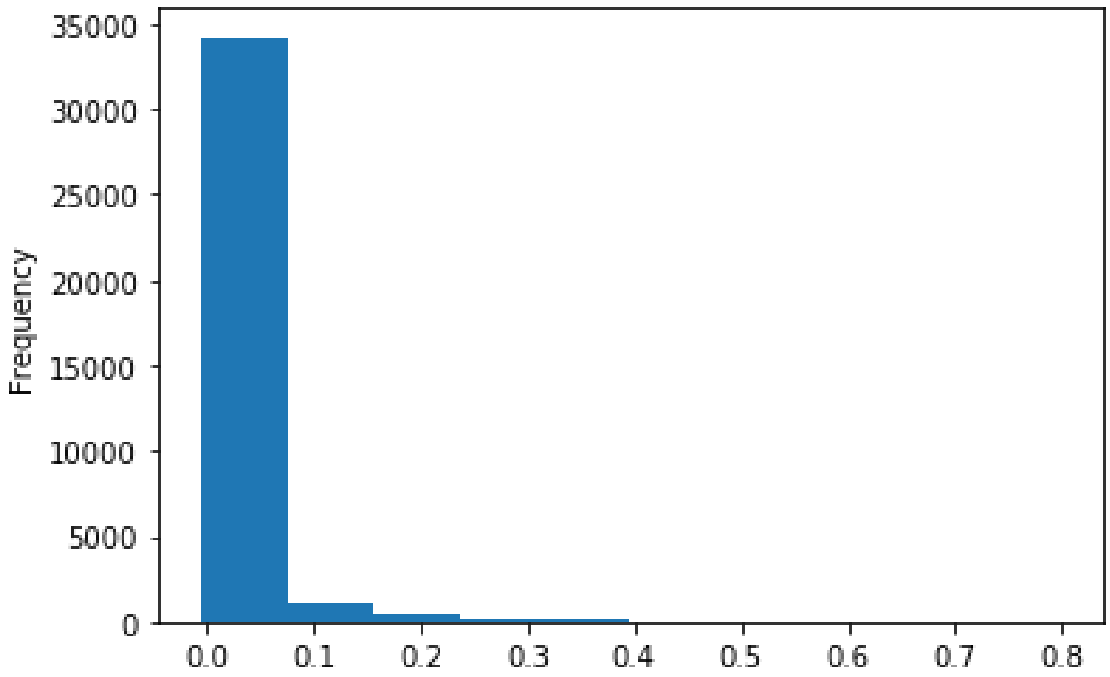


Figure 3.8: Histogram of the predicted probabilities for the test set.

In Section 3.2.3 an analysis of patients with a history of suicide versus those without was performed and it was found that an even distribution of those patients exists, such that a model cannot simply only choose to predict patients with a prior history as suicide and expect to perform well. With this, we will check this breakdown to ensure that the model did not attempt to make these types of predictions and that the model can generalize. First, for patients that did not have a suicide code before or after their mTBI (control cohort) the model was 99.27% accurate. Next, for the patients that had a suicide code before and after their mTBI (suicide cohort) the model was 33.71% accurate. For the patients that had suicide before, but did not have it after (control cohort) the model was 96.09% accurate. And finally for the patients that did not have a suicide code before, but had it after the

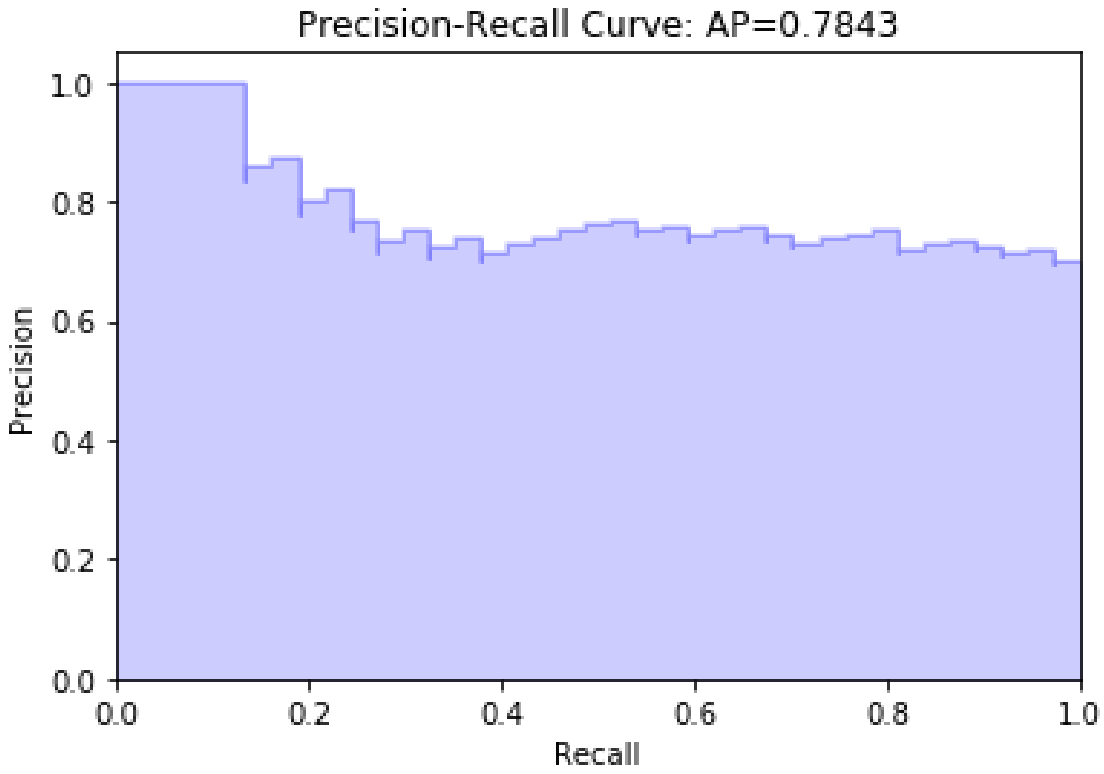


Figure 3.9: Precision-Recall Curve for the high risk patients of the test set.

model was 19.46% accurate. These values are visualized in Table 3.5.

It is promising that the model was able to correctly identify that the 1,968 patients with a history of suicide should be predicted as control. Clearly the model struggled the most with the patients without a history of suicide before their mTBI, but the fact that the model did not completely fail on these patients is a good indicator that it did not simply utilize a history of suicide to make a determination.

Overall, predicting suicide at the point of an mTBI within patients is a difficult task. Compared to state-of-the-art approaches on diagnosis prediction, the conversion of diagnoses into an NLP representation and the use of BERT outperformed and shows promise.

Figure 3.10: Confusion matrix for high risk patients of the test set.

Suicide Before?	**Suicide After?**	**# Patients**	**Accuracy**
Yes	Yes	178	33.71%
Yes	No	1,968	96.09%
No	Yes	514	19.46%
No	No	33,418	99.28%

Table 3.5: Breakdown of accuracy for patients with suicide before/after their mTBI on the test set.

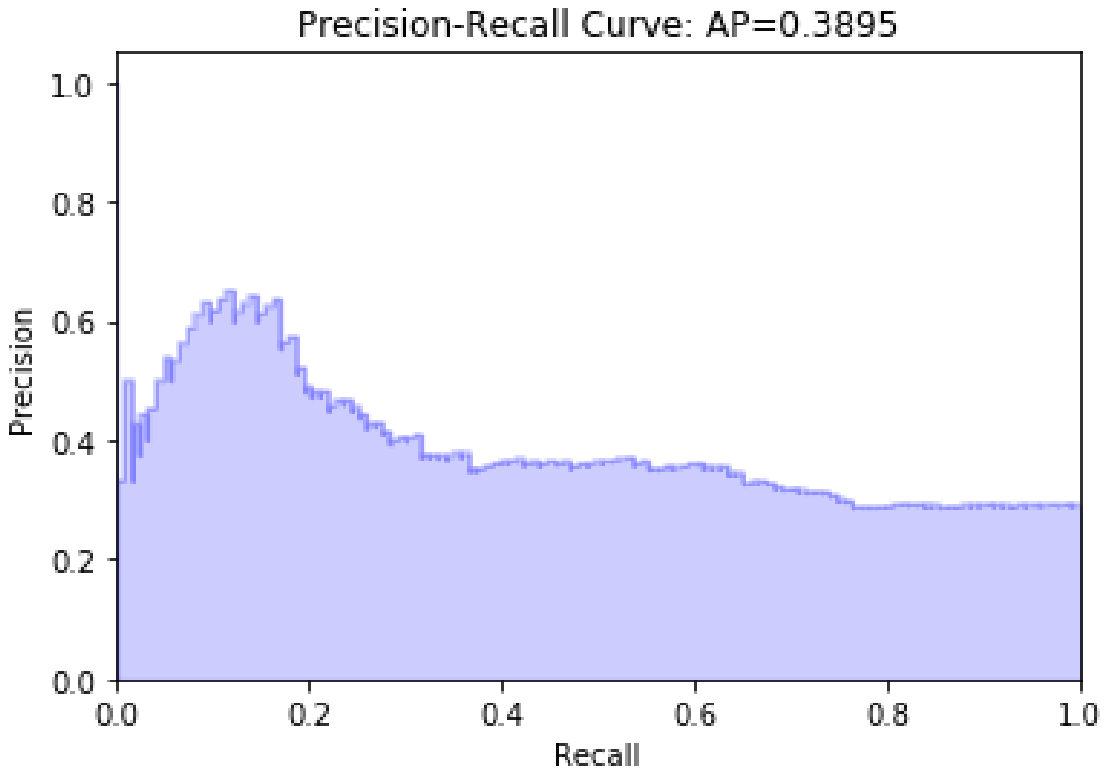


Figure 3.11: Precision-Recall Curve for the medium risk patients of the test set.

Figure 3.12: Confusion matrix for medium risk patients of the test set.

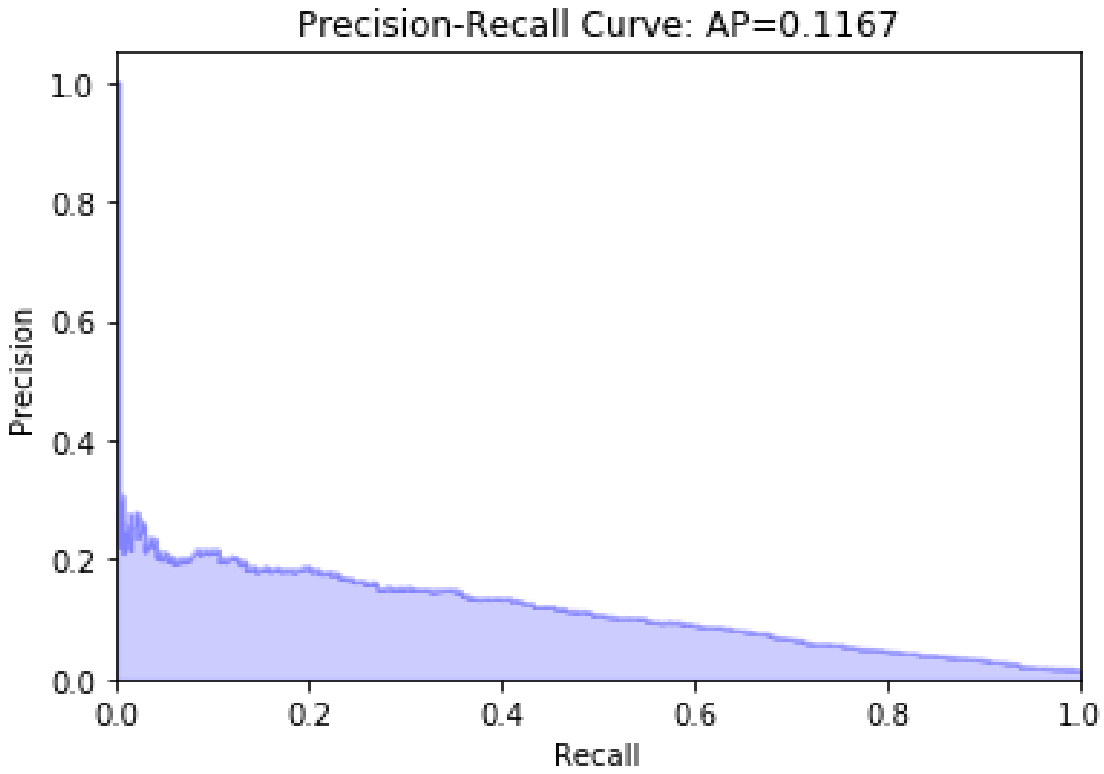


Figure 3.13: Precision-Recall Curve for the low risk patients of the test set.

Figure 3.14: Confusion matrix for low risk patients of the test set.

3.4 BERT

To understand the tuning of the suicide BERT model that was performed to arrive at the most optimal model in Section 3.3.3, an in depth breakdown of each parameter will be provided. For each parameter tuned, the rest of the model's parameters will be kept constant, and the constant values will be listed as well as the performance comparison of the various parameter values.

3.4.1 Model Base

As explained in Section 3.3.2.5, the BERT model undergoes two stages: pre-training and fine-tuning. During the pre-training phase, the model is trained on masked token prediction and next sentence prediction. Then, for the fine-tuning phase an untrained layer of neurons is added to the end of the model and then it is trained for the specific task of interest. For the original BERT model, the Google researchers pre-trained on a Wikipedia and BookCorpus dataset for 1 million update steps and released the associated models. These pre-trained models thus contain a wealth of NLP knowledge embedded within the datasets and can be used for different domains and tasks. With this, different pre-trained versions from the original paper, as well as subsequent work, was compared. Table 3.6 contains the standard configuration used across the different bases.

The version released with the original BERT paper is the "bert-base-cased" model. This model contains word tokens with capital letters which allow the model to understand pronouns, beginnings of sentences, etc. However, these capitalizations

Configuration	Value
Model Base	—
Extra Pre-Training?	No
Num Epochs	2
Max Seq Length	500
Reverse Truncate?	No
Demographics?	No
Time Tokens	None
Data Input	Diagnoses

Table 3.6: Model configuration used for comparing different BERT model bases.

of letters may not always be necessary. We hypothesize that for our suicide task this would be the case because the patient documents made up of ICD-10 diagnosis descriptions do not contain capital letters, compared to formal writing.

With this, "bert-base-uncased" is an alternate version where all of the word tokens are lower cased. The number of word tokens (vocabulary size) for the model changes from 28,996 for the cased version to 30,522 for the uncased version. The reason for the increase in size is that the uncased version came subsequently and thus contains updated word tokens as well as significantly more "unused" tokens that can be replaced with custom words.

As these official BERT model bases were trained on a corpus that contains a wide array of domains, researchers have attempted to further pre-train them on clin-

ical notes to embed clinical knowledge and make them more specialized. Alsentzer et al. utilize the MIMIC-III dataset [106] to pre-train a BERT model on (i) all note types and (ii) only discharge summaries [107]. These clinically trained models were found to outperform the standard BERT on various NLP tasks. As the version trained on all notes, named Clinical BERT, is most applicable to suicide prediction, a comparison will be performed.

As the notes contained within the MIMIC-III dataset do not follow a similar structure as the patient documents that have been constructed for this approach, pre-training a BERT model on the constructed patient documents may lead to an increase in performance. Thus, the patient documents are aggregated and pre-trained on the "bert-base-uncased" base. With this, the BERT model should theoretically learn the structure of the code descriptions and the characteristics of this unique dataset. For this pre-training only, masked token prediction is used as next sentence prediction was found to not be important to the training success [108].

Finally, to put the benefit of using any pre-trained models into context, a BERT model that was never pre-trained was evaluated. Note that this model was only fine-tuned and never had any pre-training, thus it is expected to perform worse, which would confirm that the pre-trained versions do have knowledge built into them that contributes towards the suicide prediction task.

Tables 3.7 and 3.8 contain a summary of the comparison of the various pre-trained model bases/versions that were evaluated. Analyzing the results, it can be seen that the model that was never pre-trained ("None") performed significantly worse in terms of all metrics. Notably the AUC-PR for test was 0.141 compared

	Train		
Base	**Loss**	**AUC-ROC**	**AUC-PR**
bert-base-cased	0.008(±0.00)	0.885(±0.00)	0.281(±0.02)
bert-base-uncased	0.008(±0.00)	0.885(±0.00)	0.281(±0.01)
Clinical	0.008(±0.00)	0.883(±0.00)	0.262(±0.03)
None (cased)	0.009(±0.00)	0.846(±0.01)	0.141(±0.02)
Pre-train	0.009(±0.00)	0.748(±0.22)	0.164(±0.13)

Table 3.7: Results obtained by training BERT on different model bases (train datasets evaluation).

	Test	
Base	**AUC-ROC**	**AUC-PR**
bert-base-cased	0.864(±0.01)	0.207(±0.01)
bert-base-uncased	0.863(±0.01)	0.209(±0.02)
Clinical	0.861(±0.01)	0.203(±0.01)
None (cased)	0.835(±0.01)	0.140(±0.02)
Pre-train	0.738(±0.20)	0.140(±0.11)

Table 3.8: Results obtained by training BERT on different model bases (test datasets evaluation).

to the 0.20 for the rest of the models. On the other hand, the pre-trained model performed about the same or worse than the model that was never trained. But more importantly, this pre-trained model had a significantly higher standard deviation than all other models.

Amongst the rest of the models there was a small deviation between each. Surprisingly the Clinical version performed the worst out of the three, showing that the clinical knowledge embedded from the MIMIC-III dataset did not help to improve the model. It was expected for the "bert-base-uncased" version to perform better than the "bert-base-cased", but the difference is not significant. Ultimately, due to the tiny differences in performances either the "bert-base-cased" or "bert-base-uncased" would be a reasonable choice. However, due to the more expressive vocabulary for the uncased model, the "bert-base-uncased" is the base that we settled on for the final optimal model.

3.4.2 Truncating Sequences

When creating a BERT model, a maximum sequence length must be defined. Input to the model must then be at most the sequence length and may be any value less than the maximum. Because the BERT model utilizes WordPiece tokens, as previously described, rather than word indices, a larger count of inputs will be utilized for each sequence. The pre-trained models released in the original BERT paper have a maximum sequence length of 512 tokens, which for larger documents is easy to surpass. To modify this value to be larger the knowledge embedded from

the massive pre-training would be lost and thus for larger sequences an approach must be made that satisfies this constraint.

An example of the truncation is shown in Figure 3.15 where a patient has 6 diagnoses, but the fictional model has a maximum sequence length of 4. With this 2 of the diagnoses must be removed. Two different approaches for truncating the sequence are visualized within the figure: keeping the 4 oldest diagnoses or keeping the 4 newest diagnoses. It would be expected for the newest truncation method to outperform the oldest truncation due to it containing the most recent patient history and current state of the patient, but both of these approaches will be attempted and compared.

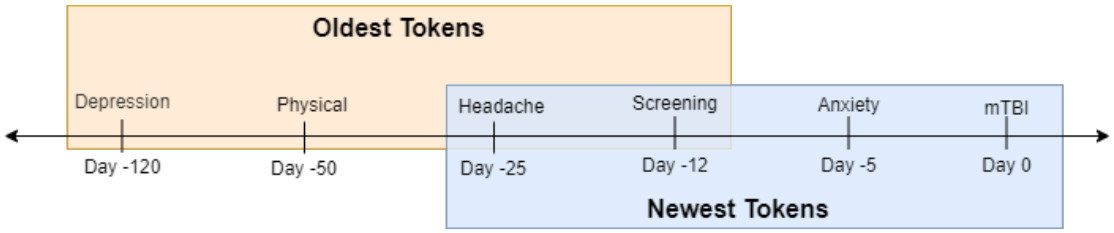


Figure 3.15: An example of truncating a sequence of 6 to fit a fictional model with a maximum sequence length of 4 using an oldest and newest truncation method.

To understand the necessity of truncation to fit within the 512 maximum sequence length set by the BERT models, the number of word tokens required for representing each patient's diagnosis history was calculated. For all 180,390 patients, the mean number of tokens was 2,720 with a standard deviation of 3,539.18. The minimum number of tokens for a patient was 12. 25% of the patients fell within 551 tokens, 50% within 1,508, and 75% within 3,521. Only 42,530 patients could fit within the 512 maximum sequence length set by the pre-trained BERT models. For

the purposes of our analysis we utilized 250 and 500 for truncation and only 22,374 and 41,750 patients could fit into each without truncation.

Additionally, the suicide cohort tends to have longer sequences than that of the control cohort: 3,250 vs 2,709 for the mean with similar standard deviations. Of the 3,434 total suicide patients only 558 could fit into BERT's maximum sequence length, whereas of the 176,956 control patients 41,972 could fit into the same limit. This 16% versus 23% difference illustrates the tendency for the suicide cohort to have longer sequences and the bias that could exist when choosing a truncation strategy and limit.

Table 3.9 contains the standard configuration used across the models and Table 3.10 includes the results of the attempted truncation methods. Clearly keeping the newest diagnoses performed significantly worse than the oldest diagnoses, and in terms of AUC-PR it had half of the performance. While this was not expected and is surprising, analyzing the patients and method of cohort construction further can reveal insights into this significant gap. When truncating to keep the newest data first, all of the patients will have an mTBI and other associated diagnoses from Day 0 within the data. Therefore the similarity between the patients will be high and the BERT model will have less potential patterns to distinguish between. Whereas the older data contains much more variability that a model is able to learn from. Additionally, as the suicide cohort tend to have longer sequences compared to the control cohort, the suicide cohort's diagnoses are more likely to be truncated and for their past to not be included within the model.

The importance of the older data can further be seen in the results of the

oldest 250 and 500 truncation methods. For both of these, the test AUC-PR's are identical at 0.207. This shows the significance of the oldest diagnoses and how only a small number of the oldest are needed before the performance levels off. Compared to the newest truncation method where doubling from 250 to 500 resulted in a boost in performance. Therefore, we can conclude that the oldest diagnoses outperform and are the method of choice when truncation is required.

Configuration	**Value**
Model Base	bert-base-cased
Extra Pre-Training?	No
Num Epochs	2
Max Seq Length	—
Reverse Truncate?	—
Demographics?	No
Time Tokens	None
Data Input	Diagnoses (Cased Tokens)

Table 3.9: Model configuration used for comparing sequence truncation methods.

In future work alternate forms of trimming the sequences down to the maximum sequence length could be attempted. For example, each encounter contains up to three diagnosis codes that are ranked by order of importance: the diagnoses in the third position across all encounters could first be removed to attempt to fit the patient document within the maximum sequence length. This would allow for

	Train			Test	
Truncation	**Loss**	**AUC-ROC**	**AUC-PR**	**AUC-ROC**	**AUC-PR**
Oldest 250	0.008(±0.00)	0.880(±0.00)	0.273(±0.01)	0.857(±0.00)	0.207(±0.01)
Oldest 500	0.008(±0.00)	0.885(±0.00)	0.281(±0.02)	0.863(±0.01)	0.207(±0.01)
Newest 250	0.009(±0.00)	0.816(±0.00)	0.121(±0.01)	0.787(±0.01)	0.091(±0.01)
Newest 500	0.009(±0.00)	0.836(±0.01)	0.147(±0.01)	0.809(±0.01)	0.105(±0.01)

Table 3.10: Results of different sequence truncation methods for BERT.

the entirety of patient data to still be utilized without missing a potentially crucial moment in time.

Additionally, extensions of the BERT model, such as XLNet [109] and BigBird [110], have developed techniques to overcome the maximum sequence limit and could be attempted to allow for the entire patient history to be input.

3.4.3 Demographics

Thus far the BERT models have only received encounters as input in the form of diagnosis descriptions. However, most models tend to utilize the patient's demographics as a lot of patterns and risks can be identified or ruled out from them. For example, the REACH-VET model has multiple boolean features based off of demographics: age ≥ 80, patient lives on the west coast, and patient is a male to name a few.

To similarly utilize the demographics into our BERT model we input them

raw to allow the model to identify the patterns that should be expected. The demographics are appended to the beginning of the patient document in the following manner: age represented as an integer, gender as text ("male"/"female"), and the branch of service for the patient as text (i.e. "army", "air force", "navy"). Once again, it should be noted that the inclusion of demographics reduces the number of diagnoses that can fit within the maximum sequence length.

Configuration	**Value**
Model Base	bert-base-uncased
Extra Pre-Training?	No
Num Epochs	2
Max Seq Length	500
Reverse Truncate?	No
Demographics?	—
Time Tokens	None
Data Input	Diagnoses (Uncased Tokens)

Table 3.11: Model configuration used for comparing the effect of including demographics.

We compare a standard model that strictly has diagnoses against one that has demographics appended at the front of the diagnoses. Table 3.11 contains the configuration for these two models and Table 3.12 contains the results. The difference between these two models is minimal, but the inclusion of demographics

	Train			Test	
Demo?	**Loss**	**AUC-ROC**	**AUC-PR**	**AUC-ROC**	**AUC-PR**
No	0.008(±0.00)	0.885(±0.00)	0.281(±0.01)	0.863(±0.01)	0.209(±0.02)
Yes	0.008(±0.00)	0.884(±0.00)	0.276(±0.01)	0.861(±0.01)	0.208(±0.02)

Table 3.12: Results comparing input with and without demographics in the BERT model.

seems to slightly decrease the performance of the model. The AUC-PR for the train dataset shows the biggest decrease from 0.281 to 0.276, whereas for the test dataset it is minimal from 0.209 to 0.208. While the expectation is for the demographics to increase the performance, the patients within this dataset exhibit little variability: the mean age is 28 with a standard deviation of 8.11. Additionally, there exists the potential that the demographics can be inferred from the diagnoses and thus the network does not see an advantage in duplicated information.

In summary the demographics seem to provide no benefit to the network, but rather they cause a slight decrease in performance.

3.4.4 Number of Epochs

When fine-tuning, the number of epochs to run for must be optimized to ensure that the model is properly trained and does not overfit the validation/test data. Most applications of the BERT model tend to fine-tune it for 2 to 3 epochs [87, 111, 112]. However, to identify the optimal number of epochs for this task, a

comparison of epochs 1 through 8 was performed. The standard configuration used across runs is shown in Figure 3.13.

Configuration	Value
Model Base	bert-base-cased
Extra Pre-Training?	No
Num Epochs	—
Max Seq Length	500
Reverse Truncate?	No
Demographics?	No
Time Tokens	None
Data Input	Diagnoses (Cased Tokens)

Table 3.13: Model configuration used in comparing number of epochs for training.

Table 3.14 displays the results of the runs and Figure 3.16 visualizes the AUC-PR over the epochs. While the AUC-PR on the train data increases along with the epochs, the test AUC-PR begins to show a higher standard deviation at epoch 3 and eventually begins to decrease. Therefore, 2 epochs appears to be the most optimal for this application.

3.4.5 Time Tokens

While the BERT models have been able to perform exceptionally well on the encounters, and more specifically the diagnosis descriptions, embedding the time

	Train			Test	
Epochs	**Loss**	**AUC-ROC**	**AUC-PR**	**AUC-ROC**	**AUC-PR**
1	0.299(±0.50)	0.878(±0.00)	0.234(±0.01)	0.861(±0.01)	0.199(±0.02)
2	0.008(±0.00)	0.885(±0.00)	0.281(±0.02)	0.864(±0.01)	0.207(±0.01)
3	0.008(±0.00)	0.887(±0.00)	0.310(±0.05)	0.862(±0.01)	0.205(±0.02)
4	0.008(±0.00)	0.891(±0.00)	0.391(±0.02)	0.862(±0.01)	0.205(±0.02)
5	0.007(±0.00)	0.894(±0.00)	0.433(±0.02)	0.859(±0.00)	0.191(±0.01)
6	0.007(±0.00)	0.893(±0.00)	0.418(±0.05)	0.859(±0.01)	0.189(±0.01)
7	0.006(±0.00)	0.902(±0.00)	0.542(±0.01)	0.853(±0.01)	0.170(±0.01)
8	0.006(±0.00)	0.901(±0.00)	0.548(±0.02)	0.850(±0.00)	0.161(±0.01)

Table 3.14: Results for training BERT models over different number of epochs: 1 through 8.

between encounters is crucial in building a contextual understanding of what the patient underwent. For example, a depression code many years before the patient's mTBI may impact their suicide risk differently than one that was present shortly before the mTBI. Or a patient that routinely sees a doctor, where less than seven days passes between their encounters, may be predicted differently than that of a patient that waits months between their visits. Because of this, we hypothesize that it is important to embed the context of time into the patient documents that are input.

Existing work tends to remove time or apply simple aggregations. In the

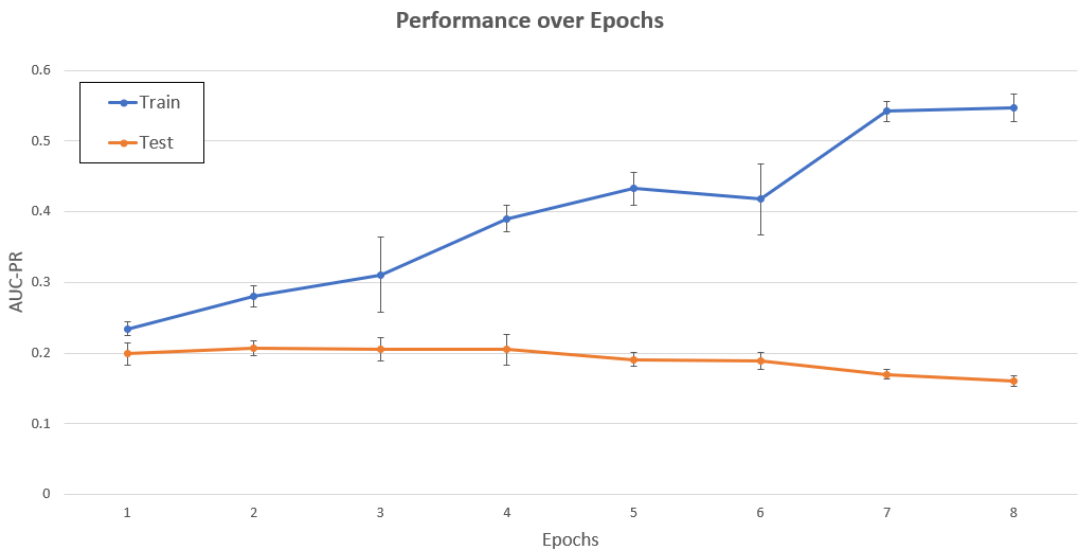


Figure 3.16: The AUC-PR performance over the number of epochs that the model was trained for. Error bars are the standard deviation across the three patient split at each epoch.

RETAIN model, the time between encounters was removed. The REACH-VET model calculated the total number of diagnoses within arbitrarily picked time periods such as 1 month, 6 months, 12 months, 24 months, etc. In our own past work, we employed a window technique similar to that of REACH-VET when inputting into a neural network [76].

Rather than removing time, the Deepr model embedded time directly into their CNN by creating time tokens represented as words. However, to avoid exploding the number of word tokens within the model and to allow the model to understand the similarity between times, they had to aggregate chunks of time into a single word. The intervals used were measured in months: $(0-1], (1-3], (3-6], (6-12], 12+$, and each interval was translated into a word such that $(0-1]$ was represented as

"0-1m". While the Deepr model is able to embed a more true representation of the EHR compared to that of the window aggregation technique, it still lacks the ability to provide the network with the detailed time information and requires expertly crafted intervals.

We extend the Deepr time token implementation by utilizing the ability of the WordPiece tokens to represent any word, and thus any time input. With this the number of days between each patient's encounter, d, was utilized to create several different embeddings. Each embedding was placed in between encounters within the patient documents.

- **Integer Days**: d was represented as an integer.
- **Word Summary**: d was converted to its logical word representation.

  $d < 7$ day

  $7 \leq d \leq 30$ week

  $d > 30$ month
- **Plural Time Words**: d was converted to a plural integer and word represen-

tation.

$d = 1$	1 day
$7 \leq d$	$d$days
$7 \leq d \leq 14$	1week
$7 \leq d \leq 30$	$\lfloor d/7 \rfloor$weeks
$30 \leq d \leq 60$	1month
$60 \leq d \leq 365$	$\lfloor d/30 \rfloor$months
$365 \leq d \leq (365 * 2)$	1year
$d >= (365 * 2)$	$\lfloor d/365 \rfloor$years

The intuition behind the *plural time words* approach is that BERT has been found to understand the importance of plural words and the context associated with them [113].

It should also be noted that the inclusion of time tokens influences the maximum sequence length of 512 and results in less diagnoses being able to be included. With this, a large number of time tokens would reduce the amount of diagnosis information that a model is able to learn from and could ultimately hurt performance. To potentially overcome this issue, both the *word summary* and *plural time words* techniques were extended to create variations of each that did not contain the "SEP" token between encounters. The rationale behind this is that a visualization of the feature importances, as will be introduced in Section 3.5, it was found that the BERT model pays a lot of attention to these "SEP" tokens. Plus, the removal of these tokens would minimize the amount of diagnoses that likewise would need to be removed.

Configuration	Value
Model Base	bert-base-cased
Extra Pre-Training?	No
Num Epochs	2
Max Seq Length	500
Reverse Truncate?	No
Demographics?	No
Time Tokens	—
Data Input	Diagnoses (Cased Tokens)

Table 3.15: Model configuration used in comparing different time token embeddings.

Table 3.15 outlines the model configuration used for the evaluation of the time tokens and Table 3.17 contains the results for these models. Based on the results obtained, the embedding of time tokens does in fact improve the performance of the models. The AUC-PR of 0.207 when no time tokens are used instantly improves to 0.226 with the "Integer Days" embedding technique. By including contextual words of "day", "week", and "month" within the "Time Words" embedding, a further improvement to 0.231 is seen.

Introducing plurality to the time words decreases the performance to 0.224, despite BERT's ability to understand singular versus plural. Typically BERT distinguishes between singular and plural when trying to predict the next word to place within a sentence and the correct grammar is important. In this diagnosis scenario,

	Train		
Time Tokens	**Loss**	**AUC-ROC**	**AUC-PR**
None	0.008(±0.00)	0.885(±0.00)	0.281(±0.02)
Integer Days	0.008(±0.00)	0.891(±0.00)	0.298(±0.01)
Time Words	0.008(±0.00)	0.891(±0.00)	0.296(±0.02)
Time Words, No SEP	0.008(±0.00)	0.888(±0.00)	0.273(±0.02)
Plural Words	0.008(±0.00)	0.887(±0.00)	0.282(±0.01)
Plural Words, No SEP	0.008(±0.00)	0.886(±0.00)	0.274(±0.01)

Table 3.16: Results on different time token embeddings within patient sequences in BERT (evaluated on the train datasets).

it could be reasoned that the plurality is an unnecessary addition as the meaning of the word exists regardless and BERT does not need to predict the next word or token. However, because the "Plural Words" embedding takes utilizes more tokens and thus removes diagnoses due to the maximum sequence length limit, the reduced performance could also be attributed to this.

The removal of the separator tokens provided an overall decrease in performance. For the "Time Words" embedding it significantly dropped from 0.231 to 0.219. However, for the "Plural Words" embedding the performance increased to 0.225 along with an increase in standard deviation.

Ultimately, the "Time Words" technique with the separator tokens fared the best within this application. However, the minimal decrease in performance seen

	Test	
Time Tokens	**AUC-ROC**	**AUC-PR**
None	0.864(±0.01)	0.207(±0.01)
Integer Days	0.871(±0.01)	0.226(±0.01)
Time Words	0.874(±0.01)	0.231(±0.01)
Time Words, No SEP	0.871(±0.00)	0.219(±0.00)
Plural Words	0.869(±0.01)	0.224(±0.01)
Plural Words, No SEP	0.868(±0.01)	0.225(±0.02)

Table 3.17: Results on different time token embeddings within patient sequences in BERT (evaluated on the test datasets).

by the removal of separator tokens for the "Plural Words" technique could be an insight into the additional context providing benefit that could only be seen with a model that does not have a maximum sequence length limit. Therefore, for this work "Time Words" will be utilized, but "Plural Words" should be re-explored in future extensions.

3.4.6 Procedures

Thus far the diagnoses, and occasionally demographics, have been the only source of input into the BERT models. While diagnoses provide a wealth of information, the procedures that patients undergo is an extra source of data that both complements diagnoses and contains unique insights that the diagnoses cannot

describe. Therefore, we construct patient sentences and documents using the procedure codes in a similar fashion as that of the diagnoses. Because we do not know how long it will take the model to be fine-tuned on procedures, we run the model for 2 and 3 epochs as a comparison. Table 3.18 shows the standard configuration utilized and Table 3.19 shows the results of the procedure runs against a diagnosis model.

Configuration	**Value**
Model Base	bert-base-uncased
Extra Pre-Training?	No
Num Epochs	2
Max Seq Length	500
Reverse Truncate?	No
Demographics?	No
Time Tokens	None
Data Input	— (Uncased Tokens)

Table 3.18: Model configuration used in comparing procedures vs diagnoses for input.

Clearly, the procedures are unable to match the performance of the diagnoses. Part of this can be explained by the dominance of diagnoses within the dataset and the sparseness of procedures. Of the 45 million encounters within the dataset, 28.7 million spanned the observation period for constructing features. For these

	Train			Test	
Input	**Loss**	**AUC-ROC**	**AUC-PR**	**AUC-ROC**	**AUC-PR**
Diagnoses	0.008(±0.00)	0.885(±0.00)	0.281(±0.01)	0.863(±0.01)	0.209(±0.02)
2 Epochs	0.009(±0.00)	0.848(±0.01)	0.176(±0.02)	0.831(±0.01)	0.131(±0.01)
3 Epochs	0.009(±0.00)	0.854(±0.01)	0.218(±0.01)	0.829(±0.01)	0.136(±0.01)
4 Epochs	0.008(±0.00)	0.0857(±0.01)	0.261(±0.02)	0.831(±0.01)	0.132(±0.02)

Table 3.19: Results on comparing diagnoses vs procedures as input for BERT.

encounters, all but 56,983 contained at least one diagnosis code and these empty diagnosis encounters did not have any procedure codes either. More importantly, procedure codes only appeared along with diagnosis codes and never on their own. Of the almost 28.7 million encounters with diagnosis codes, 11 million of them had procedure codes.

This 28.7 versus 11 shows the disparity of procedure codes and how the BERT model received less information when using only procedure codes resulting in lower performance. However, because the procedures can serve as a complement to the diagnoses a method to utilize both needs to be explored. In Section 3.6.1 we describe an ensemble approach that we attempted, but were initially unsuccessful with.

3.4.7 Optimal Configuration

Using the learned information about the different configurations of BERT we run a final model to obtain the optimal performance. Table 3.20 contains the con-

figuration for this optimal model and Table 3.21 contains the performance achieved. An in depth evaluation of this optimal model was presented in Section 3.3.3.3.

Configuration	Value
Model Base	bert-base-uncased
Extra Pre-Training?	No
Num Epochs	2
Max Seq Length	500
Reverse Truncate?	No
Demographics?	No
Time Tokens	Time Words
Data Input	Diagnoses (Uncased Tokens)

Table 3.20: Optimal BERT configuration based on evaluating each parameter individually.

Train			Test	
Loss	**AUC-ROC**	**AUC-PR**	**AUC-ROC**	**AUC-PR**
0.008(±0.00)	0.890(±0.00)	0.298(±0.01)	0.868(±0.01)	0.227(±0.02)

Table 3.21: Result of the BERT model with the optimal configuration.

3.5 BERT Classifier Introspection

While the BERT model has been shown to improve performance over existing approaches, understanding the inner workings of the model and how it arrives at the decisions that it provides is imperative. Clinicians seek AI tools that are easily understandable and that can complement their workflow [114]. Building trust and ensuring proper care is essential and can only be achieved through an explainable model.

3.5.1 Related Work

BERT, because it is based on neural networks, is inherently considered a black box and difficult to discern the reasoning behind the decisions that it makes. However, a tremendous amount of effort has recently been placed into explaining neural networks and the decisions that they make.

First, attention mechanisms have been introduced into neural networks to improve performance by allowing the network to learn to only focus on specific components of the input. With this, the attention vectors can be analyzed to identify the components that the network pays attention to and thus understand how it arrived at the corresponding decision. For example, when predicting the next word in the sentence "In France they speak _____" the network would ideally focus on the previously seen word "France" and thus the vector of possible words would be weighted the most at the index corresponding to this word. This method of utilizing attention was seen within the RETAIN approach previously described in

Section 3.3.1.2 where the authors utilized the attention mechanism to identify which past diagnoses contributed the most to a future prediction of heart failure.

An alternative to the purely computational approach is to create visualizations that attempt to provide introspection. To understand networks learned on images, CNNVis was created in which the visualization shows what is learned at each layer of a CNN [115]. For example, in the early layers the network learns colors, patterns, and textures before slowly evolving into objects and concepts that are finally utilized to make a prediction. Krause et al. developed a workflow and visualization based on "instance-level explanations" that attempt to identify which features influence a correct/incorrect decision [116]. In our own previous work, we have attempted to build a visualization, similar to that of CNNVis, that attempts to summarize an entire neural network on raw-valued/vector-valued datasets rather than images [117]. Within this work, we found that it is especially difficult to understand all of the connections within a network when faced with raw-valued datasets as the data cannot be visualized as simply as an image can.

As the BERT model is complex such that a series of attention layers are utilized in the last layer, this makes it even more difficult to visualize the inner workings compared to that of a simple network that we attempted in our previous work. Most work on visualizing BERT has focused on understanding the structure of the model and the correctness of individual neurons. Clark et al. visualized the various heads of BERT and evaluated the accuracy of each head for part of speech tagging [118]. Whereas Reif et al. visualized the word embeddings formed and attempted to understand whether or not the model's attention vectors contain

a representation of the syntactic features [119]. Jesse Vig created a tool called BertViz that creates a visual representation of each attention layer for the next word prediction task. The end-user is able to inspect each individual layer to find model bias or to link a certain neuron to a model behavior [120].

While these existing visualizations have begun the difficult step of understanding BERT, they have focused on the next word prediction task or on the inner structure of the model. Visualizing the influence of features for a binary decision, such as suicide risk, presents a different challenge of summarizing the attention heads and their influence into the final output neuron. This ultimately requires a new visualization and technique to be created.

As these existing BERT visualizations focus on next word prediction and our model is based on a binary decision of suicide risk, a new visualization was necessary to be created. To achieve this, we will attempt to identify which features contributed the most towards the prediction and then present a visualization.

3.5.2 Feature Attribution Methods

Two feature attribution methods were utilized: Local Interpretable Model-agnostic Explanations (LIME) [4] and Integrated Gradients [121]. A brief overview of each of these is given next.

3.5.2.1 LIME

As an extensive amount of effort is required to dissect the inner workings of a model in order to understand the predictions that it makes, LIME was created to be able to explain the predictions of any classifier, regardless of the underlying model. For text classification tasks, the end result is a list of the words and their contribution score for each class. To achieve this LIME takes a text sample and inputs it into the model to obtain the prediction probability. Then it perturbs the words within the text to create additional samples. For example, given the sentence "I hate this movie", the following samples will be created: "I hate movie", "I this movie", "I movie", "I hate", etc.

The probabilities for each of these samples is obtained and then LIME analyzes how similar each sample is to the original by a metric of distance and applies a linear model to learn how well the model works around the original text. With the linear model LIME is able to extract an approximate feature importance towards a class for each word.

We run LIME on our patient documents to compute the feature importance scores. The hyperparameters utilized for the LIME explainer were 20 for the number of samples (size of the neighborhood) and 999999 for the maximum number of features to include in the explanation. Higher values for the sample size was not able to run, due to the size of our documents, and the maximum number of features was set at a large number to ensure that all possible words could be explored.

3.5.2.2 Integrated Gradients

Another common approach to computing feature importance is to multiply the gradient of a neural network by its input to identify which components, often in the form of pixels, contributed the most to the prediction made by the network. However, due to a flattening phenomenon within gradients this method is unable to account for a large part of the prediction and mischaracterizes the features that contributed the most. Therefore, a new method called Integrated Gradients was created where a baseline input is utilized and the input is changed one feature at a time towards the baseline and the gradients are analyzed. With this, small changes within the gradient are able to be more clearly seen and better contribution scores are computed more correctly.

For a text prediction task, the input and baseline are tokens to input to a network and the baseline usually consists of filler tokens. With this, we ran our patient documents against this Integrated Gradients method using the Captum implementation [122]. The method was run for a total of 20 steps for each input.

3.5.3 Visualization

We integrate the aforementioned feature attribution methods and our best performing BERT model into an interactive visualization, which can be seen in Figure 3.17. The split point for this visualization was chosen to be 0.2, where $p \geq 0.2$ was classified as a suicide patient while $p < 0.2$ was classified as a control patient.

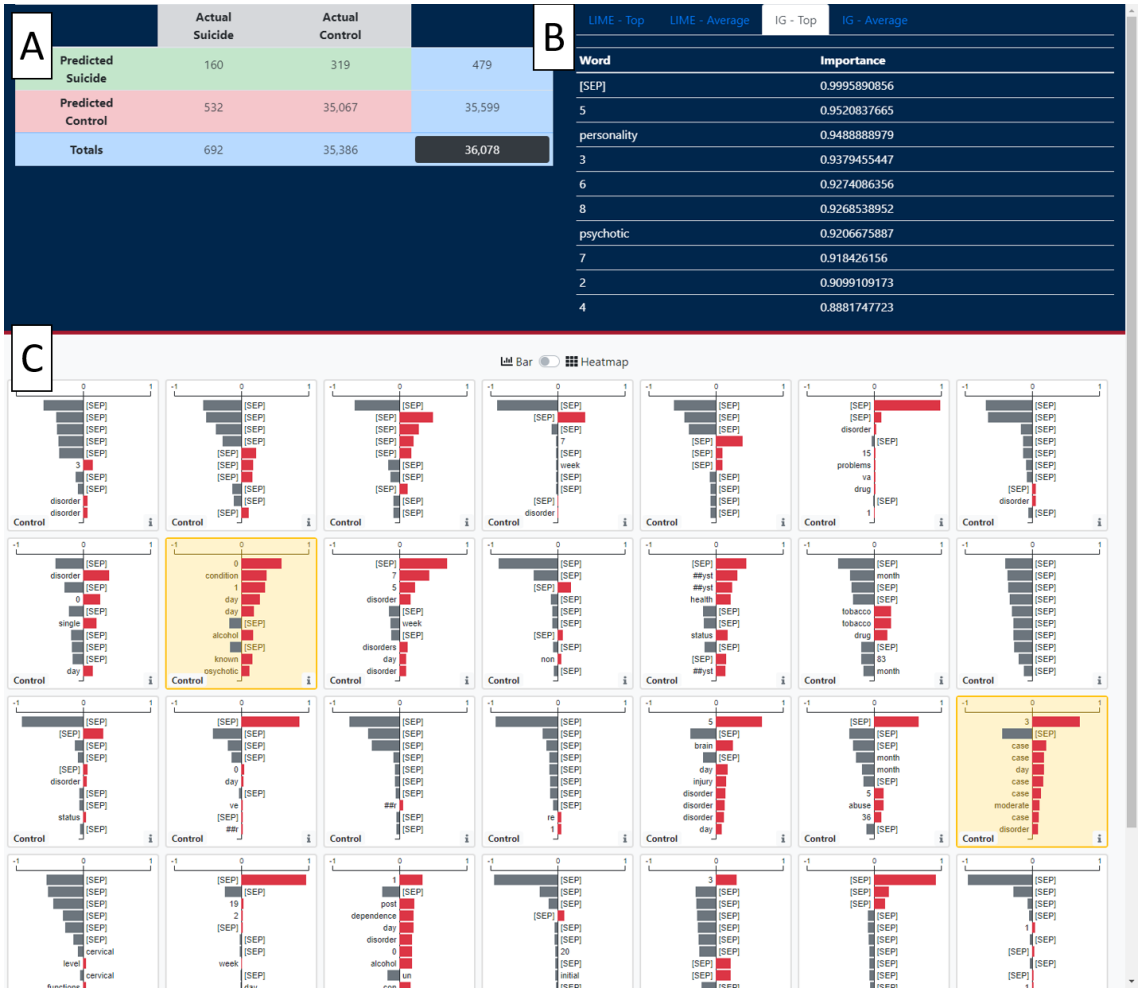


Figure 3.17: An overview of the interface for exploring BERT predictions.

Breaking this interface down: (a) a confusion matrix of the model's predictions, (b) the top words or tokens for each respective feature attribution method over all patients, and (c) the top feature importances for each individual patient.

3.5.3.1 Confusion Matrix

The confusion matrix displayed in section A of Figure 3.17 breaks down the classification accuracy for each group of patients. Along the vertical columns, the true suicide and control groups can be seen. Whereas along the horizontal rows the

predicted groups are seen. With this it can be seen that 160 patients had suicide and were predicted correctly, while 532 suicide patients were incorrectly classified as being control patients.

To encourage exploration of the data and to drill down on the other visualizations, each cell of the confusion matrix can be selected to restrict the other views of the interface to only utilize the data within the selection.

3.5.3.2 Overall Feature Importances

Because each feature attribution method computes different importances, Figure 3.17b contains the top and average words/tokens for each attribution method.

3.5.3.3 Patient Matrix

To encourage exploration of the dataset of patients and to create an understanding of the makeup of patients, we build a patient matrix display into Figure 3.17c where each box corresponds to a single patient. A zoomed in version of this matrix is shown in Figure 3.18. We break down the components of each patient box: (i) the cohort that the patient belongs to is shown in the bottom left hand corner, (ii) a two-sided bar chart showing the top tokens that influenced either a suicide or control prediction based on positive and negative values respectively, and (iii) a yellow border and background for patients that were incorrectly classified.

Alternatively the user can choose to toggle the matrix display to heatmaps as shown in Figure 3.19. With this approach the patient document, that is fed into

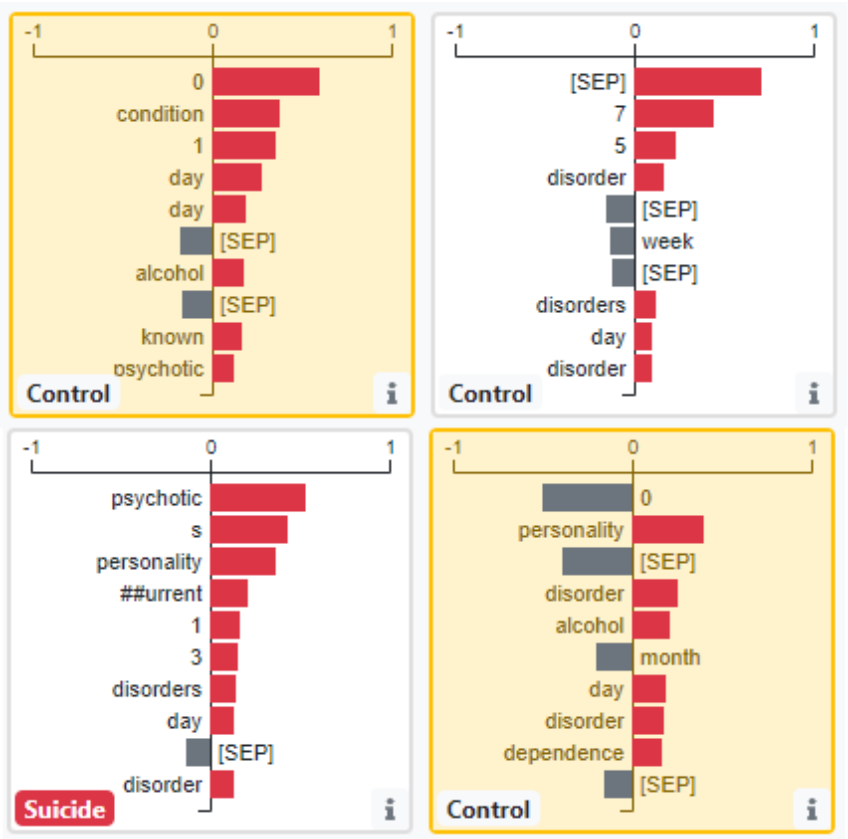


Figure 3.18: An extracted patient matrix from the visualization.

the BERT model, is displayed token by token within the heatmap filling each row left to right and then top to bottom. Each token is then colored with its respective positive or negative influence color, where red corresponds to suicide/positive and grey corresponds to control/negative. Note that only a subset of the patient document is able to be displayed within each box as the entire documents are too large, but this still gives the user a brief overview of the patient.

The displayed influence scores for both the two-sided bar chart and the heatmaps are obtained from the selected feature attribution method selected within Figure 3.17b. This patient matrix view allows the end-user to easily identify patients that were incorrectly classified and to seek out interesting patterns within the patient

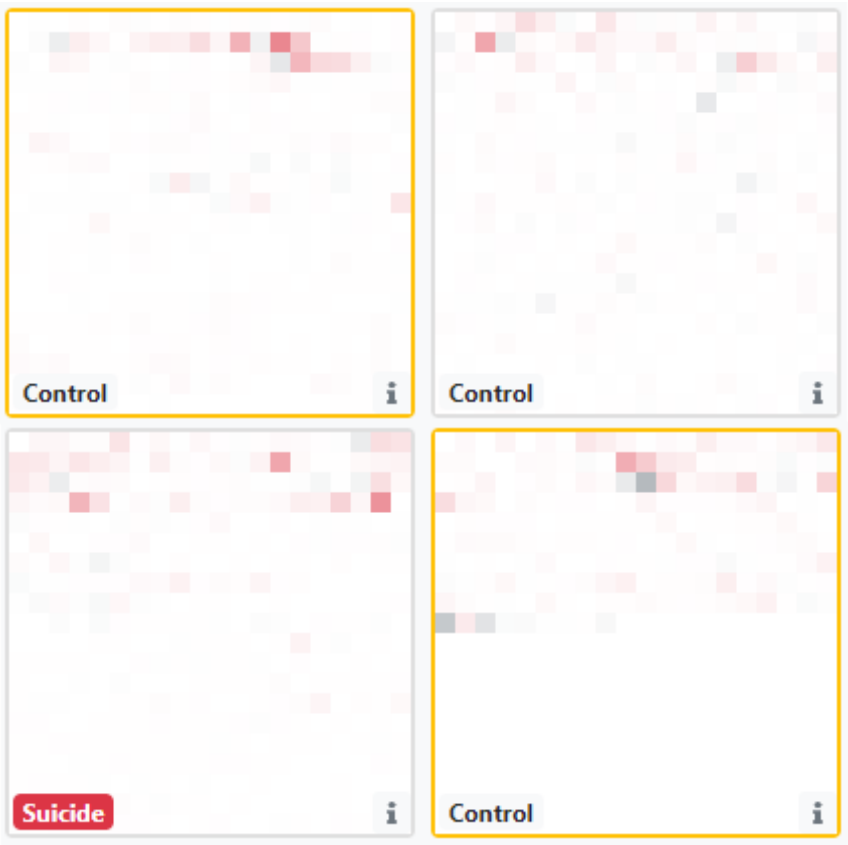


Figure 3.19: An extracted patient matrix from the visualization with the heatmap setting turned on.

documents.

3.5.3.4 Individual Patient View

To provide a more in-depth look at a patient the user is able to click on the "More Information" button located at the bottom right corner of each box to be presented with a popup of the patient. An example of this is shown in Figure 3.20. Within this view, the available cohort, accuracy of the prediction, and demographics are shown on the left hand side. In the center of the screen the patient document

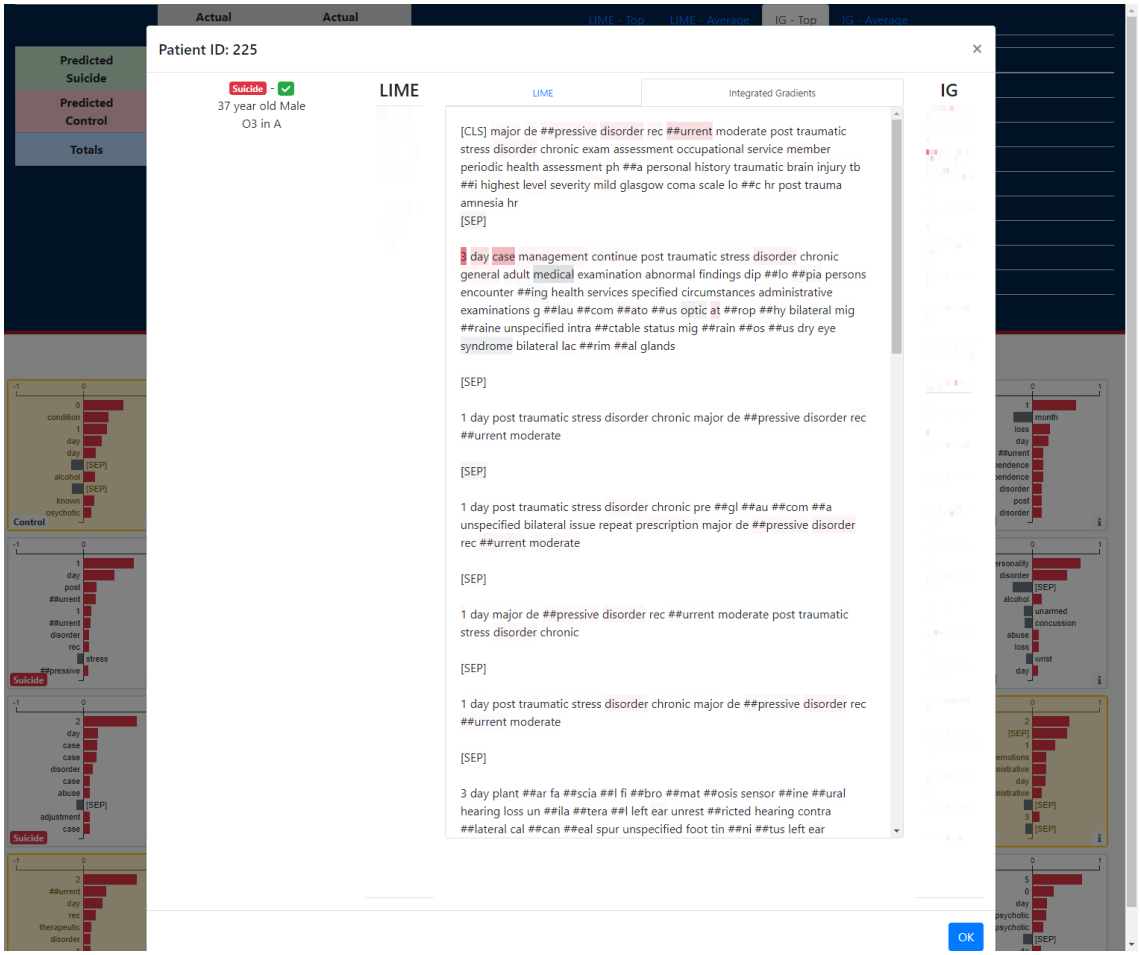


Figure 3.20: Viewing a single patient's document overlaid with feature importances.

is displayed, in the format that the feature attribution method received it. Each word of the document is highlighted based on its classification influence. In the example shown it can be seen that the words "3 day case" strongly influenced the ultimate suicide prediction. Whereas the words "medical" and "syndrome" were the most significant at attempting to sway the prediction towards control. Users can hover over words to identify their feature importance value via a tooltip as shown in Figure 3.21.

Additionally, to provide an overview of the entire document, on the left and

right hand side there exists a column for LIME and IG respectively. Each of these columns contain an enlarged heatmap of the patient document, as was described for the patient box in the matrix. By hovering over the sidebar the user can see a preview of the text that is contained within that section of the document and can jump directly to it by clicking on it. An example of this is shown in Figure 3.22.

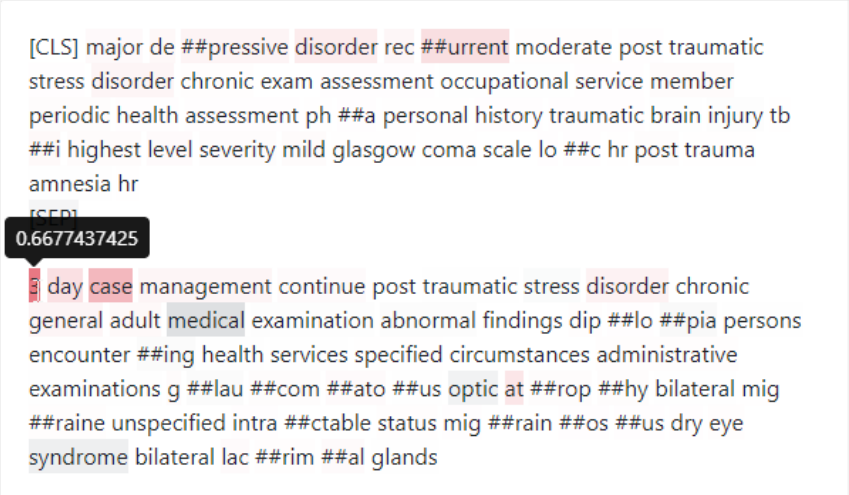


Figure 3.21: Example of hovering over a word in the patient document to identify its feature importance value.

Altogether this visualization takes a step towards encouraging exploration into the feature importances for a binary classification task within a BERT model that previously was not possible. This visualization tool can be used to both identify weaknesses of the model from a machine learning perspective or to provide clinicians and public health decision makers the ability to evaluate this model prior to deployment. While this is a first step towards interpreting the binary classifier component of BERT models, more advanced techniques for identifying the feature importances could greatly improve this visualization.

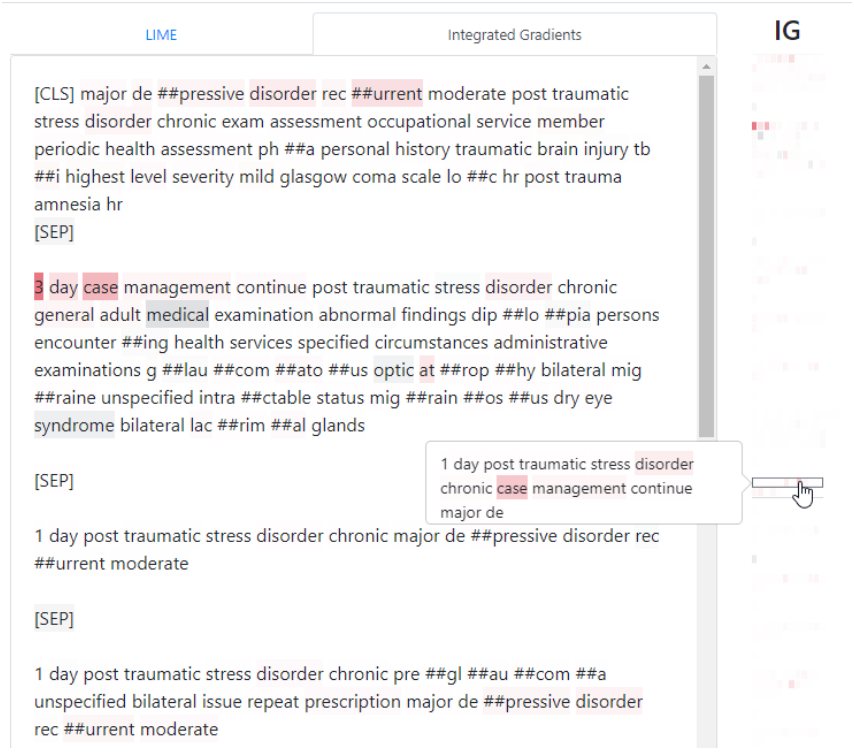


Figure 3.22: Example of hovering over the heatmap sidebar to view a distant location in the patient document.

3.5.4 Doc2Vec Visualizations

To give additional context into the difficulty of the problem at hand, we utilize the doc2vec vectors created in Section 3.3.2.4 and visualize them. First, we generate word clouds for both the suicide patients and the control patients. Two random suicide patients are shown in Figure 3.23, while two random control patients are shown in Figure 3.24. Inspecting these word clouds, and those for many other patients, it can quickly be seen the difficulty in predicting suicide for a given patient. For the suicide patient labeled "a" there exists "alcohol dependence" and depressive words which lead to an easier prediction. However, for the suicide patient labeled "b"

there exists a lot of words, but nothing that jumps out as being a typical suicide risk factor. Ultimately, this suicide "b" patient is very similar to the control patients that also do not have anything that can easily be highlighted as a reason for predicting suicide.

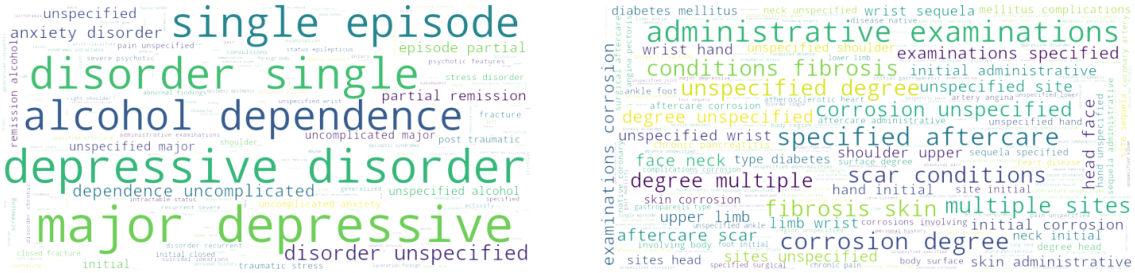


Figure 3.23: Word clouds for two random patients in the suicide cohort.

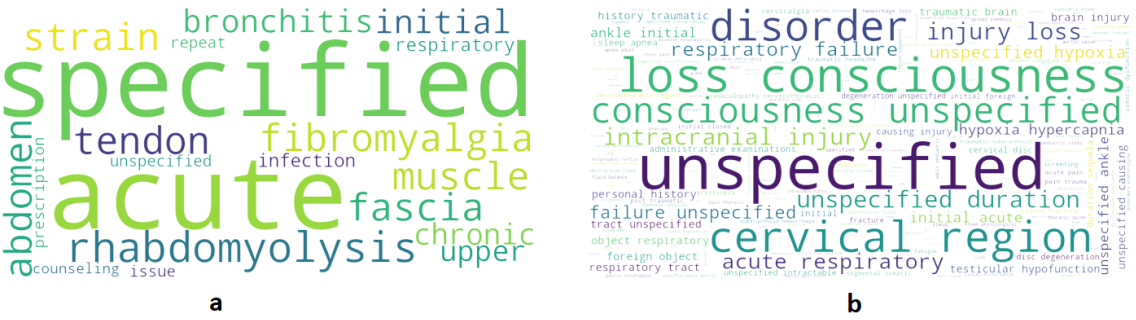


Figure 3.24: Word clouds for two random patients in the control cohort.

To better visualize and understand the similarities between the suicide and control cohorts, we extract the doc2vec vectors for each document and run principal component analysis (PCA) to reduce the vectors from a high dimensional space to a low dimensional space that can be visualized. For this the vectors were of size 100 and we utilized the first two or three PCA components to plot the vectors in 2D and 3D space respectively.

Figure 3.25 shows the first two PCA components where red data points correspond to the suicide cohort and blue data points correspond to the control cohort. It is easy to see that the suicide cohort generally is contained to a specific area, but these patients are directly overlapped on top of control patients. Figure 3.26 further shows this overlap in the 3D space where (a) contains the combined suicide and control patients, (b) contains only the control patients, and (c) contains only the suicide patients. In (a) it is impossible to see the red data points for suicide patients and exploring this 3D space interactively the red data points are not visible from any angle. Analyzing (b) and (c) to understand the distribution of these two cohorts, we identify that the red suicide patients are nested within the blue control patients.

While this is a massive simplification of the problem, through the use of doc2vec and PCA, these examples underscore the difficulty of discerning between the suicide and control cohorts. The computed patient documents are relatively similar in nature and identifying the correct characteristics for classification is a challenging task for any model to perform.

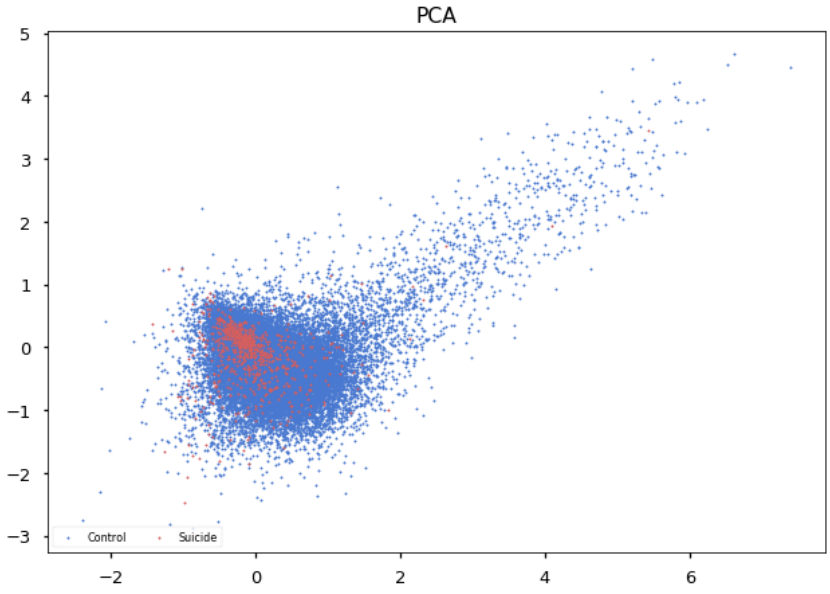


Figure 3.25: A 2D visualization for the first two components of PCA that was run on the doc2vec vectors.

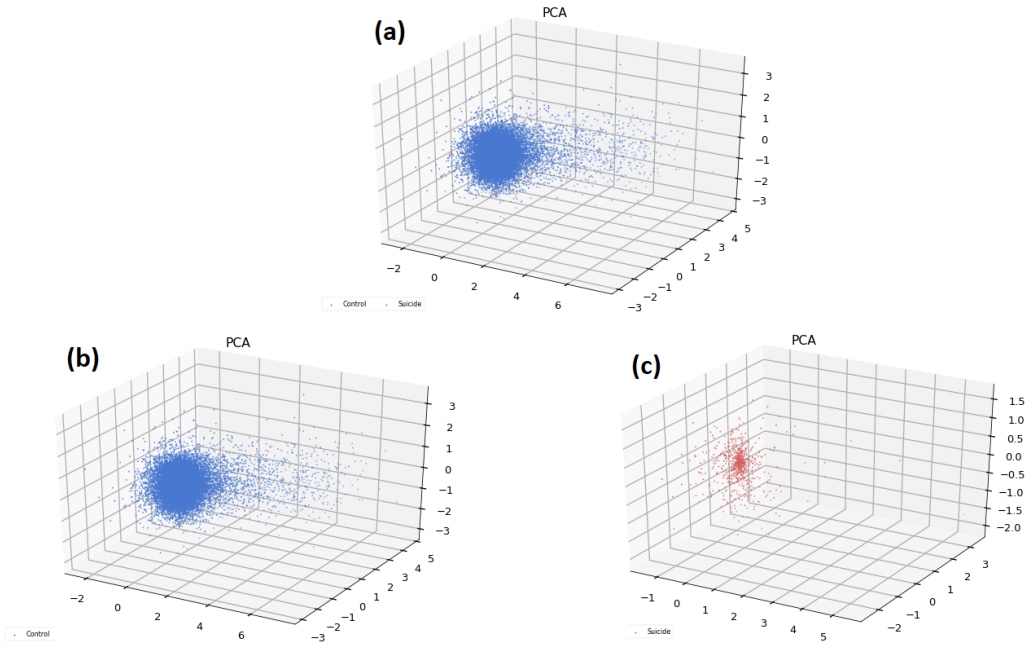


Figure 3.26: A 3D visualization for the first three components of PCA that was run on the doc2vec vectors. (a) Both suicide and control patients, (b) only control patients, and (c) only suicide patients.

3.6 What Didn't Work?

There were a couple of approaches that were attempted throughout this chapter, but were unsuccessful at achieving better performance. For completeness and for future work to build off of, these approaches are described below.

3.6.1 BERT Ensemble

As the base BERT model is limited by the 512 token maximum sequence length that was defined by Google when building and training their BERT model bases, we took steps to work within this limit by truncating the patient sequences. We found that keeping the oldest diagnoses first when trimming produced the best results. However, there could be additional information in the newer diagnoses that could help steer the model towards better predictions. Additionally, as we focused only on diagnoses there may be indicators within the procedures that could further improve our models.

With this in mind we attempted to create a BERT ensemble where we trained two separate BERT models: one on truncated diagnoses and one on truncated procedures. Then we extracted the final hidden state of the BERT model, prior to the binary output layer, and created an ensemble-like neural network. This neural network took as input the extracted diagnosis hidden states and fed them into an RNN to create a final vector, and similarly took the extracted procedure hidden states and fed them into another RNN to create a final vector, and then concatenated these two final vectors along with a vector of patient demographics to

then feed through a standard feedforward layer. Both of the RNN layers utilized LSTM nodes and the hidden layer used the RELU activation.

There were two challenges faced within this ensemble model: (1) which hidden state to extract and (2) poor performance when utilizing multiple factors.

First, the architecture of the BERT model is such that prior to the single output node for the binary classification task, there exist a series of attention layers. In a typical RNN layer there is a single hidden state at each timestep and thus the single vector can be extracted. But within the series of attention layers in BERT this results in a two-dimensional vector. This vector can either be flattened, resulting in an extremely large and difficult to train vector, or a single layer can be utilized. Commonly the first layer's hidden state, corresponding to the beginning of the sentence token, is extracted. When only using this hidden state in an RNN without any other factors we found that the performance was similar to or worse than that of the BERT model, which is to be expected as information was removed. Therefore, identifying a way to utilize the entirety of the BERT model within an ensemble would be a logical next step.

Second, we found introducing a demographic vector to the diagnoses when training the ensemble caused the performance to significantly decrease. We attempted to encode the age into the demographic vector by normalizing it between 0 and 1, $age/120$, or logarithmically scaling the age, $log(age)$ or $log(120 - age)$. The gender was encoded as 0 for males and 1 for females. And the patient's branch of service was converted to a one-hot vector. Throughout each of these encodings, the model's performance would significantly decrease in an unknown manner. There are

two intuitions behind this drastic decrease in performance: (i) the diagnoses may already have some encoding or representation of the patient embedded within them and (ii) the age and gender distribution of the patients possesses little variation and the branch of service may not have as big of an impact on suicide as was expected.

For the future we seek to attempt to use one of the iterations of BERT that have been developed, such as XLNet [109] or Big Bird [110], that allow for infinitely long sequences to be input. With these, our model would not require the truncation of sequences and we could attempt multiple variations of models: (i) we could input the diagnoses, procedures, and demographics all at once into a single BERT model and allow it to discern between the features or (ii) we could build a separate model for each component (diagnoses, procedures, and demographics) and build an ensemble that uses what each of these models has learned to arrive at a final prediction.

3.6.2 Attempts without Ideation

Initially when building our model, we defined the patient populations such that the suicide cohort consisted only of patients with a suicide attempt diagnosis code. This meant that the patients with only a suicide ideation code were placed in the control cohort. The intuition behind this was that a suicide ideation would be more noisy and less informative than that of an attempt. However, the performance of our models significantly suffered such that the BERT model was only achieving an AUC-PR of about 0.07 (which was still the best performing model). Upon consultation with a clinician (MD and board certified), we identified that both

attempts and ideation are important for this task and that suicide ideation codes are well documented and clearly defined.

3.6.3 Downsampling & Weighted Loss

We attempted to utilize downsampling such that our roughly 1 to 100 ratio of suicide to control was converted to 1 to 10 by removing control patients. Additionally, we attempted to weight the loss of each classifier by placing a higher importance on the suicide patients compared to that of the control patients. For both of these variations we found that the performance of our models slightly decreased or stayed about the same, thus nullifying the need for either of these alterations.

3.7 Limitations

While our NLP patient representation and optimal BERT model have shown to outperform the existing state of the art, suicide-based, and baseline models, there is additional work that is necessary to improve the efficacy of this model. Given that the REACH-VET model [64] is currently deployed by the Veterans Health Administration (VHA), there is evidence that these models are currently being utilized within healthcare settings. Thus, when making a selection between a model such as the REACH-VET and our own BERT model, a decision between performance and interpretability must be made. While we have attempted and hope to lessen the gap to inspecting the BERT model through the feature visualization that we outlined in Section 3.5, additional work is still necessary. Furthermore, a recent critical review of suicide prediction models found that most of the developed tools have no clinical value [62]. They suggest that new approaches should be linked directly to concrete clinical questions where the models and tools can be evaluated based on an overall net-benefit rather than sensitivity or positive predictive value (PPV). While we believe that our approach is different from that of the traditional logistic regression and time removal techniques, we seek to learn from this review and further evaluate our model in this new manner.

Additionally, while our BERT model did not see a boost in performance, and occasionally a decrease in performance, when appending demographics to the input this can be seen as a cause for concern. The hypothesis behind this result is that the ages within the dataset are relatively similar, that the diagnoses inherently em-

bed the demographics within them, and that the addition of demographic tokens removed potentially vital diagnosis tokens due to the maximum sequence length limitation of the base BERT model. However, both demographics [123] and service branch [124] have been shown to be vital suicide risk factors. Because of this, we should expect to see an increase in performance through the addition of demographics rather than the result that we observed. We believe that an ensemble model should be utilized in the future to overcome this as was discussed in Section 3.6.1.

3.8 Conclusion

Within this chapter we introduced our model for predicting suicide risk in mTBI patients a year post injury. For this we explored state of the art models for diagnosis prediction and baseline models for clinical data as well as NLP data. Ultimately our BERT model significantly outperformed the existing models in terms of AUC-PR. For the future, we look to extend this model onto different variations of the BERT model that do not possess the same maximum sequence length as the base BERT.

Through our introduction of BERT into this suicide prediction task, we presented our novel NLP-based patient representation. For this we identified that clinicians search for diagnosis codes within the EHR using text rather than directly searching for codes. Because of this and the difficulty of expressing differences between two codes, we chose to operate on the diagnoses using their textual descriptions rather than the raw ICD-10 code structure to preserve the original form

of the code as intended by the clinician. This results in a complex "document" for each patient that can be fed into a model. While past NLP models and approaches did not seem to fare well with this patient representation, the recent explosion of NLP model performances and the introduction of BERT seem to make for a great pairing as can be evidenced through our results.

Finally, to begin to explore the interpretability of a BERT model fine-tuned on the binary classification task we constructed a visualization using multiple feature attribution methods. To our knowledge this is the first work of its kind, as other visualizations and inspections have been based on the next sentence prediction task. Within this visualization it is possible to identify the tokens/words that our model attends to the most and finds the most important in making its final prediction. While the increase in performance of our model, compared to that of existing suicide prediction models, already encourages deployment into a clinical scenario, by providing clinical interpretability we aim to strengthen the use case.

As most of these datasets contain large amounts of sequences and are ever-increasingly growing, being able to visualize and understand the entirety of the dataset is a challenging task. For example, a dataset that has a million sequences does not have enough screen space, in terms of pixels, to display all of the records. Additionally, even if there was enough screen space there would be a large amount of patterns for one to have to inspect and discern between to understand the dataset as a whole. Therefore, while we are now able to predict irregular events through the work shown within this chapter, it is necessary to support the exploration of a dataset through a summarization technique that allows for large scale data to be

simplified for easy interpretation.

Chapter 4: Visual Summarization of Temporal Sequences

4.1 Introduction

Events exist everywhere in life: online consumers perform a series of comparisons and clicks before purchasing an item [125], patients go to different doctors and try different therapies during the recovery from a specific condition [43, 126, 127], sports teams make plays and score back-and-forth during a match [43]. Throughout all domains, a temporal event can be described as *a data point denoted by a timestamp and event type*. During the last few years, the increased availability of large-scale temporal datasets that contain complex and long sequences has underscored the importance for designing scalable techniques to easily identify trends and to answer important questions from irregular, noisy, and large longitudinal datasets.

However, while events can exist in a vast array of domains, summarizing a large temporal event dataset remains a difficult task. Many techniques have been developed in recent years that provide analysts with the ability to filter, explore, and simplify their data. However, while these approaches have been shown to work on regular event datasets, they face significant challenges when time is introduced. Rather than taking into account that a series of events during a certain temporal window can be vastly different than the exact same events within another temporal

window, these approaches treat them as identical and thus lose crucial information.

Because time can be vital to the exploration and understanding of event datasets, an approach that is able to handle large scale temporal data, generate a summary, and provide the ability to explore at multiple levels of detail could greatly assist the analytical requirements of temporal event researchers and ensure that vital temporal information is not lost.

In this chapter we present *TrajectoryFlow*, a scalable algorithm and visualization, that has been built to effectively analyze and summarize temporal event datasets. The primary contributions of this work include: (1) An adaptive window-based algorithm that summarizes large datasets of temporal sequences by taking into account the temporal context of patterns and trajectories and creating an overall view, and (2) a novel event summary diagram that reduces the analytical complexity used to visualize the results of event mining approaches.

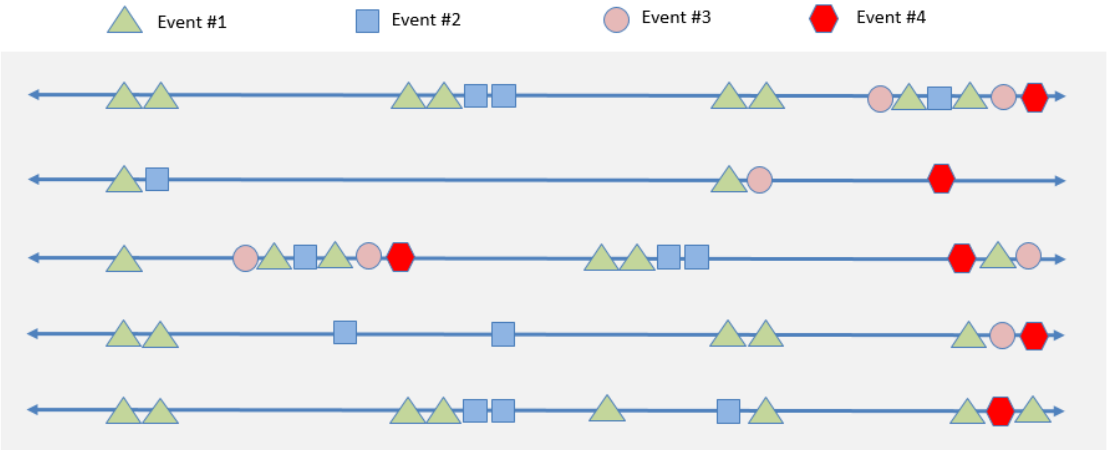


Figure 4.1: Illustration showing an example of a temporal event dataset.

4.2 Temporal Events

A temporal event dataset can be defined as a set of sequences $D = \{S_0, S_1, ..., S_n\}$ where each sequence $S_i =< I_0, I_1, ...I_m >$ contains a timestamped series of items I_i arranged in a specific order with respect to the timestamps. The arrangement of the items can be used to derive temporal relationships within the data such as $I_i \prec I_{i+1}$ describing that I_i happened before I_{i+1}, also referred to as "temporal context". Each itemset, I_i, contains a set of event types such that $I_i =< T_0, T_1, ..., T_p >$ and $T_i \in E$ where $E = \{e_0, e_1, ..., e_n\}$ is a dictionary of event types. An example of a dataset of sequences is shown in Figure 4.1, where each row represents a sequence S_i that contains itemsets I_i with one or more event types. Additionally, each row represents time in the horizontal direction such that time increases from left to right. Note that the term sequence and record are and can be used interchangeably.

At first glance some of the patterns in Figure 4.1 are easy to notice due to

Figure 4.2: An example summarization of Figure 4.1, showing the usefulness of summarizing a dataset of sequences.

the relatively small nature of the dataset, but more hidden and embedded patterns are difficult to identify and would require analyzing each sequence individually to be able to pick them out. As the size of this dataset grows from only 5 rows to hundreds or even millions of rows and from a maximum of 15 data points to millions of data points, these patterns become increasingly more difficult to identify and spot.

Additionally, a common task associated with these datasets is to extract the trajectory between two events. Within the healthcare domain this allows for the diagnoses that a patient will experience to be highlighted. For example, understanding the path/trajectory from the moment that a patient has a concussion to when they commit suicide could help to identify beneficial moments for intervention.

With this in mind, Figure 4.2 shows a summary of the dataset where the summary contains the most common path of events that occur between event #1 (triangle) and event #4 (hexagon), taking temporal positioning (horizontal) into account. This event summary provides a concise description of the dataset and shows how an analyst could easily understand the patterns contained within their dataset easier.

4.3 Related Work

In the visualization field, a common method of understanding event sequences has been to visualize a single record/sequence to understand its path over time [41, 128–131]. More recently, a dataset of sequences has been visualized by stacking sequences vertically on top of one another and attempting to merge identical paths to allow for interactive clustering [132, 133], filtering [134], exploration [135], and searching [32, 134, 136–139].

There are several notable examples of systems that employ these visualization techniques on event datasets. EventFlow [43] presents an aggregated display of event sequences and allows users to simplify large, complex datasets into a concise display through filters and merging of events. MatrixWave [140] adapted the concept of a Sankey diagram into a matrix visualization for comparing event sequences. CareFlow [127] used a similarity distance metric to identify a population of similar records and display them in a flow visualization allowing for users to be able to identify the most effective outcomes based on sequence paths. Outflow [141] aligned sequences on a specific event and aggregated them into a directed acyclic graph (DAG) which was used to visualize results in a Sankey-type interactive visualization.

While these approaches are powerful and novel in their applications, they lack the ability to properly evaluate temporal sequences. Each of these approaches rely on merging events together based off of their index within a sequence rather than the time at which the event occurred. For example, we obtained a soccer match dataset of 9,084 games and over 900,000 data points and loaded it into the popular

EventFlow system [43] to explore the longitudinal patterns that occur over the span of the game. The result can be seen in Figure 4.3. At first glance we can see that the tool is effective at making it easier for users to identify that the green and blue events are the most common events that occur at the beginning of the sequences. However, beyond that initial starting point the visualization becomes complex and patterns become very difficult to identify. What are the most common types of events to occur near the end of the game? What about events that occur around halftime? Knowledge of the game of soccer would cause a user to expect to see substitutions to be very common towards the end of the game, but in this visualization this cannot be found.

The reason for this limitation in existing systems is that sequences are attempted to be merged together at each event and the moment that a mismatch occurs, the sequences are diverged. However, knowledge of temporal sequences tells us that there could be noise in between common patterns and thus those common patterns should ideally be identified to be merged together.

To help identify common patterns throughout a dataset, recent approaches have utilized Frequent Sequence Mining (FSM) algorithms to identify the frequent patterns contained within a dataset and present a concise view to users. FSM is a popular technique for finding the most frequently occurring subsequences of events in datasets of sequences [142]. These algorithms are able to identify the most common patterns of events within a dataset and extract them, regardless of where they occur. Using FSM, Liu et al. presented both the most frequent patterns and an icicle plot of all sequences within a dataset [143]. Kwon et al. also utilized an FSM algorithm

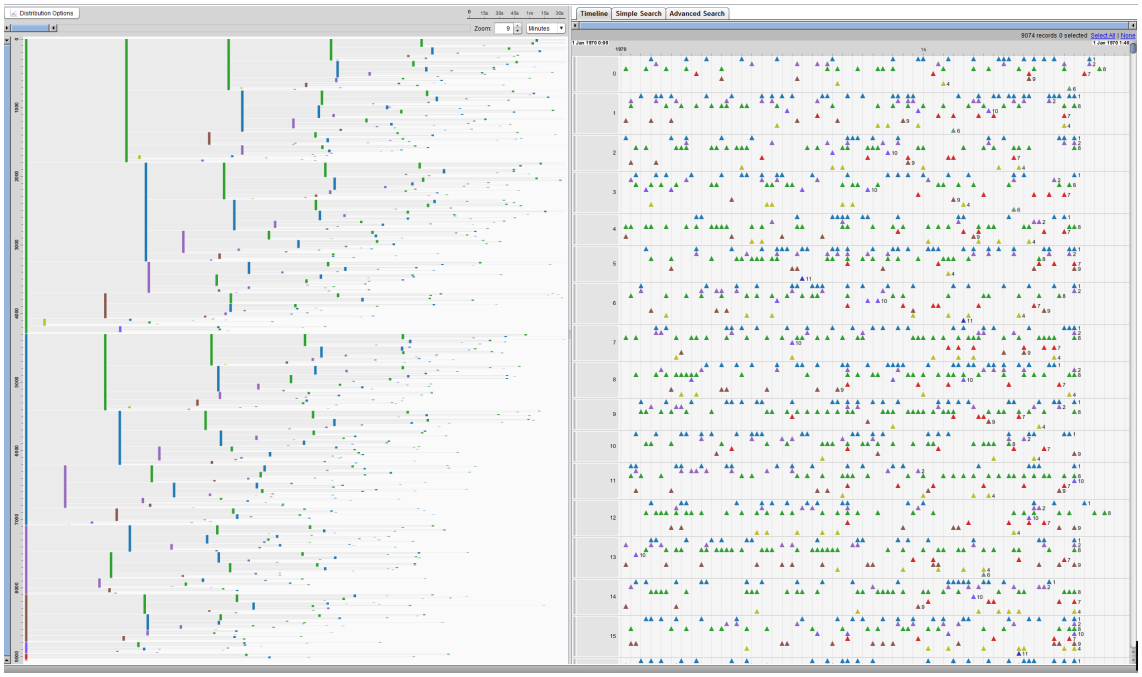


Figure 4.3: A screenshot of a popular even mining tool used to visualize and explore the longitudinal patterns within a soccer match dataset of 9,084 games and over 900,000 data points.

to display similar information in addition to a histogram and network of the overall dataset [144]. DecisionFlow [126] combined FSM and statistical methods to identify frequent sequences and their correlations to specific outcomes which was presented in a timeline and pattern diagram to understand the statistical results. Frequence [145] modified an FSM algorithm to visualize the most frequent subsequences and their outcome measures in a Sankey diagram.

While these approaches using FSM assist users in identifying patterns that frequently occur they still lack the ability to take into account the context of time due to the limitation that FSM algorithms do not factor in the temporal position of each frequent pattern. Thus, a pattern that exists at the beginning of a sequence is considered the same as one that occurs at the end of a sequence. Continuing to use the game of soccer as an example from Figure 4.3, two substitutions in a row at the beginning of a game would be considered the same as ones that occurred at the very end of the game. However, in the domain of soccer, these are not the same as the ones at the beginning of a game are considered more of an anomaly. Therefore, while FSM algorithms have provided progress in analyzing event datasets, there still exists room for integrating temporal context into these analyses.

4.4 Mining Temporal Sequences

Our first contribution is an adaptive window-based algorithm that introduces a temporal context to a popular mining algorithm, PrefixSpan [146, 147]. We title our algorithm: Adaptive-Window-PrefixSpan (AWP-Span) and the pseudocode is shown in Algorithm 1. In order to understand our algorithm, we first provide an explanation and example of FSM algorithms, and most specifically the PrefixSpan algorithm.

4.4.1 Frequent Sequence Mining

Frequent Sequence Mining (FSM) is a popular technique for finding the most frequently occurring subsequences of events in datasets of sequences [142]. These algorithms define a subsequence as being "frequently occurring" when the number of sequences that it is contained in is more than a certain support threshold. FSM algorithms stem from either Apriori-based or pattern-growth-based approaches [142, 148–150], but the general structure of an FSM algorithm is to build a frequent set of events that are each of length one and then to continue growing these events by concatenating more events until the defined threshold is met. This results in the algorithm producing a list of frequent subsequences and the number of sequences that each of them appear in.

When computing the frequent subsequences of a dataset, a sequence, $S_i =< I_{i1}, I_{i2}, ...I_{im} >$, is considered to be a subsequence of another sequence, $S_j =< I_{j1}, I_{j2}, ...I_{jn} >$ if there are m itemsets in S_j that satisfy $I_{i1} \subseteq I_{j1}, I_{i2} \subseteq I_{j2}, I_{i3} \subseteq$

$I_{j3}, ..., I_{im} \subseteq I_{jm}$. It can also be said in reverse that S_j contains S_i.

The support, τ, of a subsequence, S_k, is defined as the fraction of all sequences in a dataset that contain the subsequence S_k. In FSM algorithms a support threshold is predefined such that when the "frequently occurring" subsequences are returned, they have a support at least as large as the threshold.

To better explain FSM algorithms, we will provide an example of how to determine if a subsequence is frequently occurring. In this example we will utilize the dataset D shown in Table 4.1 and a support threshold, τ, of 50% to determine if the subsequence $S_f =< A, B, D >$ is frequently occurring. The support threshold of 50% means that at least 50% of all sequences in the dataset, D, must contain the subsequence S_f for it to be considered "frequently occurring".

Iterating through the sequences we first can see that S_1 does contain S_f as it has A in the first itemset, followed by B in the second itemset, and D in the third itemset. Looking at the next sequence, S_2, we can see that it contains A in the first itemset, but does not contain B in its second itemset which would initially seem that it does not contain S_f. However, starting from the second itemset we can see that it contains A, followed by a B in the third itemset, and finally a D in the fourth itemset. This means that S_2 does in fact contain S_f and shows how the subsequence can be contained anywhere in the location of the sequence. Moving along to the final sequence, S_3, we can see that it contains A followed by B in its first two itemsets, but in its third itemset it lacks the D event resulting in it not containing S_f.

Through iteratively looking at the sequences we could see that S_1 and S_2

Table 4.1: Example Dataset

	Items #1	Items #2	Items #3	Items #4
S_1	{A, C}	{B, D}	{C, D}	
S_2	{A}	{A, C}	{B, D}	{C, D}
S_3	{A}	{B}	{C}	{D}

contained S_f, while S_3 did not, resulting in S_f being present in 66% of the dataset, thus we can conclude that S_f is frequently occurring due to 66% being at least as high as the threshold of 50%. In this example we have shown how FSM algorithms are able to identify all of the frequent subsequences, but instead of taking a brute-force approach they operate on efficient data structures for an optimal search strategy.

While FSM algorithms can identify the most frequent subsequences in a dataset, they do not possess the ability to take into account temporal context and build a trajectory between two events. One of the issues with lacking temporal context was highlighted in the example shown in Section 4.4.1 and Table 4.1 where the frequent subsequence S_f occurred in a different location within S_2 than it did in S_1. While the example contained sequences of relatively short length, in real world data the sequences may be very long and frequent patterns at the beginning of a sequence may not necessarily contain the same meaning as a pattern at the end of a sequence. This limitation poses a problem for tasks where an event dataset researcher seeks to identify the most common patterns within a certain stage of all sequences because the frequent subsequences obtained from FSM algorithms will potentially be present

in a different section of each sequence.

In addition, as event researchers also attempt to understand the various paths that sequences may take, mapping the output of an FSM algorithm to a visualization would result in little information on the trajectories embedded within the dataset. For example, representing the frequent subsequences within a Sankey diagram would simply show the most common patterns within the dataset, but not provide an analyst with information on the most common trajectories and the varying paths followed by sequences.

With these limitations in mind we extend the work of PrefixSpan to include a window technique that is able to take into account the temporal context of a dataset in each window and build an overall pattern followed by the majority of sequences.

4.4.2 AWP-Span

Our algorithm, AWP-Span takes as input a dataset of sequences D, a support threshold τ, and a list of windows W of size ω denoting the number of segments the dataset should be split into. The execution of AWP-Span begins with splitting the dataset D into the windows defined by W and then identifying the most frequently occurring subsequences in each window using PrefixSpan. For example, Figure 4.4 provides an example of this beginning execution step. In this example the dataset in Figure 4.4 (left) is split into two windows of size ω such that $W = W_1, W_2$. Then Figure 4.4 (right) shows a table of the subsequences that occur in W_1 and the subsequences that occur in W_2. The red lines through certain subsequences indicate

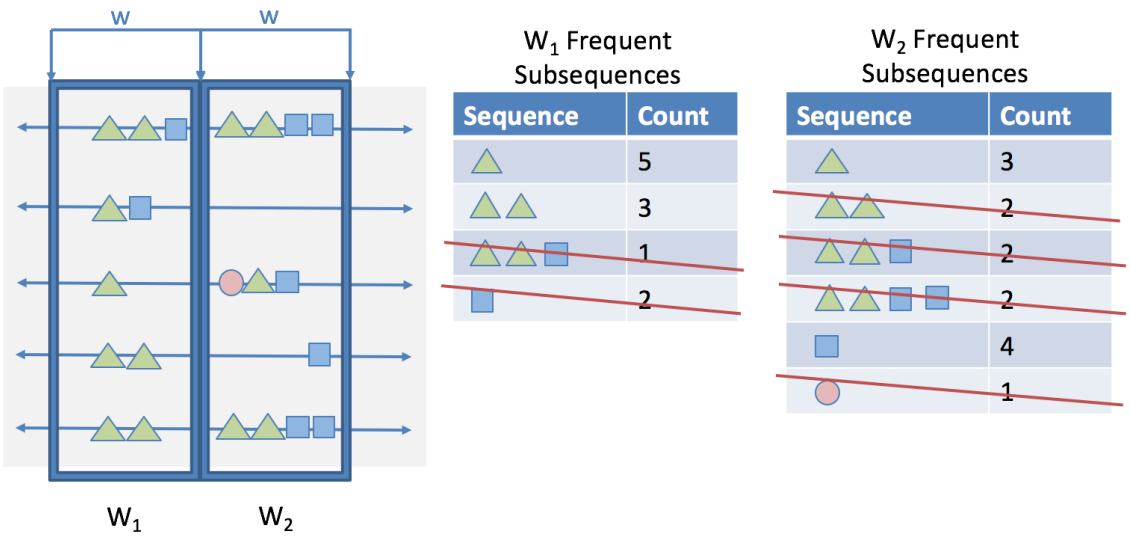


Figure 4.4: (Left) An example of a dataset being split into two windows each of size ω and a support threshold of $\tau = 3$. (Right) The frequent subsequences that result from running PrefixSpan on W_1 and W_2. The red lines indicate the subsequences that were ignored by the algorithm as they occurred less than the support threshold.

that they are not "frequently occuring" as the number of times that they occur is less than the support threshold of three. By splitting a dataset into windows, like we have shown, we are able to incorporate a temporal context into PrefixSpan allowing for it to identify the common patterns at different stages of a sequence rather than ignoring the premise that sequences change over time and patterns in different temporal locations may not be indicative of the same pattern. Being able to evaluate the various stages of a sequence is an important task that event researchers seek to accomplish and this modification of the PrefixSpan algorithm builds this ability directly into the algorithm.

Once we have identified the most common subsequences in each window, con-

necting the windows together such that events flow from one window to the next is the next step in building a common flow and understanding the long term patterns in a dataset.

4.4.2.1 Window Merge

To connect two adjoining windows, a simple approach would be to concatenate two windows' frequent subsequences together based on the beginning and ending events. For instance given the frequent subsequences for two example windows $W_1 = [< A, B, C >, < A, D >]$ and $W_2 = [< C, D >, < C, F >]$, the subsequence "ABC" in the first window would be connected to "CD" in the second window such that it forms "ABCD". "ABC" would also be concatenated with "CF" to form "ABCF" and this process would continue for all possible combinations between the two windows. However, connecting windows in this manner is a challenging task as it is not guaranteed that there will be connecting events, which can be seen in the subsequences "AD" in W_1 and "CF" in W_2 where the ending event of 'D' in "AD" is not equal to 'C' in "CF".

Another issue with this simple method of concatenation is that there may exist extra information between the frequent subsequences that could be valuable. For example, many patients could have the frequent subsequences of <"Cancer", "Chemotherapy">followed by <"MentalHealth", "Chemotherapy">, but some patients may have gone to a "GroupTherapy" class in between the frequent subsequences that would reveal some extra information. Rather than discarding this

additional information it is important to keep it.

In the example of "ABC" and "CD" forming "ABCD", there may be information in the dataset records that exist between these two subsequences. For example, there may be a record that consists of "ABCEFCD" which would indicate that the concatenation "ABCD" is not representative of all records in the dataset. While analyzing an event dataset, researchers are interested in all of the varying paths that sequences can take and this simple concatenation would not represent the smaller, hidden patterns.

Therefore, we recognize that a single event sequence may contain one of two types of information in between its two windows: either noise or no information at all; and to resolve the issues present in the simple concatenation, we develop a method WindowMerge that conjoins windows by using both a direct connection and a wildcard connection, where a direct connection is indicative of no information in between subsequences and a wildcard connection is indicative of extraneous information. Using this methodology, we can iteratively combine the frequent patterns found in adjoining windows to create an overall flow of events throughout a dataset. We include our WindowMerge in Algorithm 1.

Breaking this method down WindowMerge takes two windows to connect, W_i and W_j, where $W_i < W_j$, and returns the number of direct connections, C_d, and the number of wildcard connections, C_w between the two windows. The method's execution begins by filtering the dataset of sequences D to only look at the section of the sequences that span the windows such that: $S_{ij} = D \in \{W_i, W_j\}$. Then for each frequent subsequence in each window, $F_{im} \in W_i$ and $F_{jn} \in W_j$ where m is the m^{th}

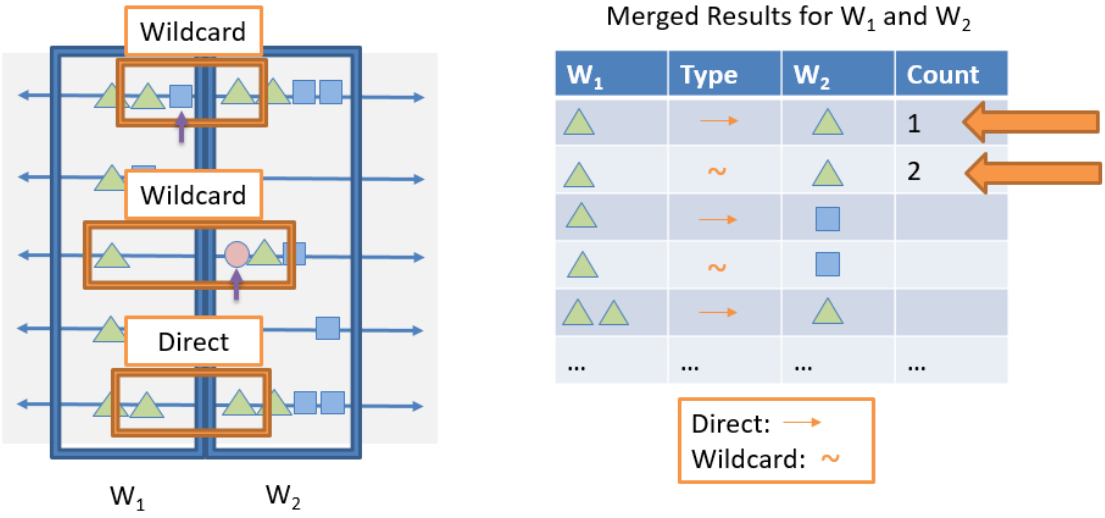


Figure 4.5: An example of running WindowMerge on windows W_1 and W_2. On the right the table is filled in both direct (orange arrow) and wildcard (orange ~) connections for each combination of frequent subsequences found in Figure 4.4. On the left, the connections for the first two rows and how the counts are determined is shown.

frequent subsequence in W_i and n is the n^{th} frequent subsequence in W_j, it identifies the index locations of these frequent subsequences in the sequences of the windows such that the index locations are at the closest possible indices to the connection between the windows: $L_{im} = max(index(S_{ij}, F_{im}))$ and $L_{jn} = min(index(S_{ij}, F_{jn}))$. By choosing the indices closest to the edge where the two windows meet, this ensures that each sequence can only be counted once in the case of repeated events.

Then for each pair of index locations that exist, the method identifies if there exist events between the locations or not. If the locations contain no events in the middle, $L_{jn} - L_{im} = 0$, then the direct connection count for that pair of frequent subsequences is incremented $C_d \leftarrow C_d + 1$, or otherwise if $L_{jn} - L_{im} > 0$ then the wildcard connection is incremented $C_w \leftarrow C_w + 1$.

An example of the WindowMerge method is shown in Figure 4.5 where the windows W_1 and W_2 are merged together. The table on the right is populated with the combination of frequent subsequences that were obtained previously for both windows. Each respective combination is connected via a direct connection and a wildcard connection which can be seen by the arrows and tilde in the table. To compute the number of occurrences for the first combination of frequent subsequences, which corresponds to the first two rows of the table, Figure 4.5 highlights where the subsequences from the combination occurs in the dataset between the windows. In this example it can be seen that the first sequence contains an extra blue square between the two green triangles indicating that the triangles are joined together by a wildcard connection, then in the third sequence there exists a red circle in between the triangles indicating that it is also a wildcard connection, and finally in the last

sequence there exists no extra events between the two green triangles indicating that they have a direct connection between each other. Therefore, the table is filled in with a value of one for the direct connection and a value of two for the wildcard connection. This process would continue for the rest of the combinations of frequent subsequences to obtain the final result between windows.

In this example that we have presented, we can see the issue of repeated events that was mentioned in the definition of the window connections: the first and last sequences in Figure 4.5 contain multiple triangle events which would alter the counts that we obtain. Had each possible index location been considered for these two sequences then each possible pair of triangles would have been counted resulting in the first sequence containing four wildcard connections and the last sequence containing three wildcard and one direct connection. However, counting multiple connections for a single sequence would present the assumption that the sequence takes multiple varying paths which would not be accurate against the premise that each sequence is a series of events. For example, in the first sequence if we evaluated both $triangle, triangle, square, triangle$ and $triangle, square, triangle$ then we would be making the assumption that the sequence went through both of these subsequences at different times when in reality the second is just a subset of the first. By misrepresenting the number of sequences in the dataset, we would not be able to correctly evaluate the task of identifying the most common trajectories that sequences followed as a minority trajectory consisting of repeated events would become heavily weighted and lead to inaccurate insights of the dataset.

4.4.2.2 Initial Evaluation

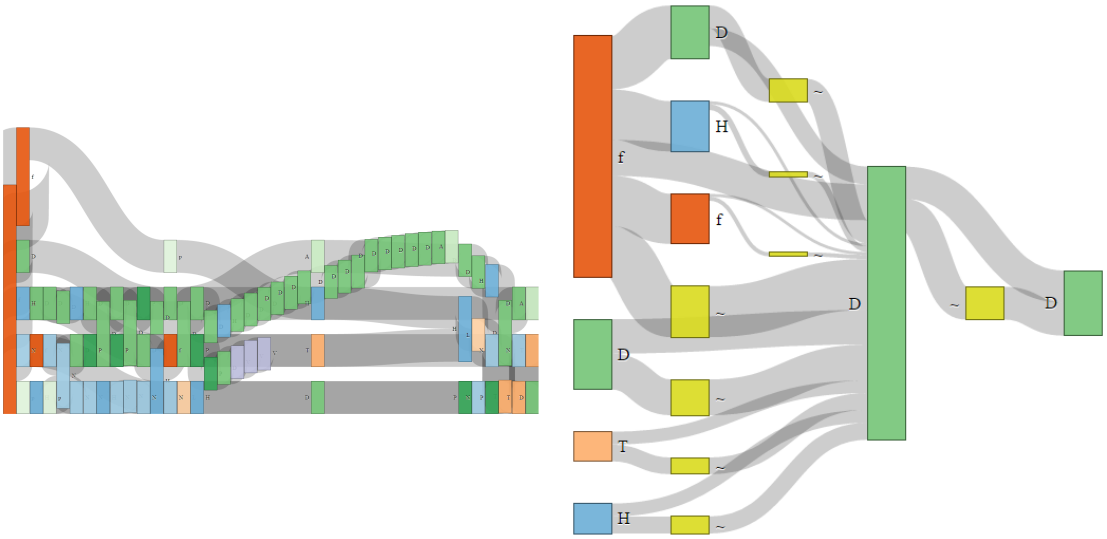


Figure 4.6: An example showcasing the reduction that our AWP-Span algorithm performs. (Left) Shows a Sankey diagram of 10 sequences in the dataset, without any reduction or algorithm run. Only 10 sequences are able to be shown due to the space limitations. (Right) Shows the results produced by our AWP-Span algorithm with window size of 30 for the large dataset of sequences. The yellow nodes correspond to wildcard connections.

To show an example of how our AWP-Span algorithm is able to provide a reduction and a summary of the overall sequences in a dataset, we extract seven sequences from an EHR dataset that will further be explored in the EHR case study in Section 4.6.3. This dataset contains a patient's diagnosis at each temporal event. Figure 4.6 Left shows the seven sequences prior to our algorithm being run. We limit the Sankey diagram to only seven sequences due to space limitations, but we can already see the complexity that these sequences contain and the difficulty in

performing any analysis. The complexity increases exponentially as the size of the dataset increases.

With this, we show the results of our algorithm in Figure 4.6 Right where the mined patterns are displayed on the Sankey diagram and the yellow nodes correspond to wildcard connections. We will describe the approach that we take in generating this Sankey diagram in more detail in Section 4.5, but at first glance we can already see the reduction in complexity and the ease of identifying patterns in our algorithm's results.

4.4.2.3 Trajectory Analysis

A common challenge faced by researchers when analyzing temporal sequences is exploring the trajectory from one event to another. For example, a researcher commonly asks the question: "What path did sequences follow to get from A to C?". The path that is followed can vary from being short in length, "A ->B ->C", or very long, "A ->B ->E ->F ->G ->C". We embed the ability to build such a trajectory into our algorithm by providing the option to specify beginning, E_b, and/or ending, E_e, events. When either of these events is set, then the algorithm will only consider frequent subsequences that occur between the two events and prune any incoming connections to the beginning event or any outgoing connections from the ending event. We will now define the pruning that our algorithm employs.

Given a sequence, S_1, our algorithm will identify the index of the first occurrence of E_b such that $I_b = index(S_1, E_b)$ and the index of the last occurrence

of E_e such that $I_e = index(S_1, E_e)$. Then for each frequent subsequence that is encountered in the algorithm, S_f, it will identify the index of the subsequence in the encompassing sequence such that $I_f = index(S_1, S_f)$. If the index of the subsequence falls between the first and last occurrence of the beginning events and ending events then it will be considered ($I_b \leq I_f \leq I_e$). However, if it does not fall within the bounds then the algorithm will skip that frequent subsequence.

To remove the connections leading into the beginning event and stemming from the ending event, anytime that the algorithm encounters a beginning event E_b in a frequent subsequence, no incoming connections are made to the event. For example, a subsequence of $< A, B, C >$ in window W_i with a beginning event $E_b =$ "B'' will cause the subsequence to be pruned to $< B, C >$ as the "A" event occurred before the beginning event. In addition, any frequent subsequences in the prior window, W_{i-1}, will not make a direct or wildcard connection to this subsequence.

Conversely, anytime that the algorithm encounters an ending event E_e in a frequent subsequence, no outgoing connections are made from the event, such that a subsequence of $< D, E, F >$ in window W_j with an ending event $E_e =$ "E'' will be pruned to $< D, E >$ and no connections to the window W_{j+1} will be made.

By allowing for a trajectory to be built from the first beginning event to the last ending event, there exists the possibility that there may be multiple beginning or ending events contained within these bounds. We do not attempt to remove these repeated events as keeping them allows for researchers to better understand varying pathways. For example, if the beginning event exists in multiple windows then this indicates that the relevant trajectories may have begun at different time

points which can assist in providing researchers the ability to accomplish their task of understanding all of the various paths that the sequences within their dataset can follow.

4.4.2.4 Adaptive Windows

While the AWP-Span algorithm that we have developed is effective at identifying common subsequences or building a trajectory within a temporal context, we recognized through test runs of our algorithm that identifying the optimal windows to split a dataset on is challenging and can be the deciding factor in obtaining a meaningful result for a specific domain. For example, consider a dataset that contains events on an interval unit of a day and events that are sparsely distributed over the sequences, where a lot of events occur at the beginning of sequences but very few events occur at the end of sequences. Setting a fixed window size of $\omega = 30$ days would cause the number of events in later windows to be significantly less than those in earlier windows. With this difference important patterns may become "chunked" together in earlier windows or not enough events may exist in later windows for patterns to be identified. In addition, identifying the exact number of windows that should exist is another difficult challenge similar to that of identifying the optimal number of clusters in clustering algorithms [151–153]. Because of these challenges, it would be beneficial to provide an optional functionality to automatically identify the optimal number of windows and for these windows to be varying in size such that areas where data is sparse would be combined and areas of density would be

split.

To provide a solution to this challenging problem, we devised a complementary algorithm that returns the optimal window configuration, W, to be used as input to the AWP-Span algorithm. We define optimal as a window configuration where the events are about evenly spaced out across the windows, such that no window has too sparse or too dense of temporal data. We refer to this algorithm as "AdaptiveWindows" as it adapts the windows such that they fit the dataset and domain that are being analyzed. This adaption of windows is accomplished by ensuring that the windows are expanded or collapsed such that the number of sequences embedded within each window is similar. We present the algorithm in Algorithm 3 and will provide a brief explanation.

The AdaptiveWindow algorithm takes a dataset of event sequences, D, an application specific window measurement, n, and returns a list of windows with their respective upper and lower bounds. We introduce the idea of an application specific window measurement as it allows for domain knowledge to be embedded into the configuration of the windows. The window measurement, n, is a window size that represents how data in the domain is usually segmented or it can be a value that the researcher believes would be the best fit for segmentation through windows. For example, in a financial domain a researcher may input a quarter of a year for the window measurement as sales data is typically measured per quarter. In another domain, such as sports, a researcher may choose to split the data based on possession of a ball as changes in possessions may lead to different types of patterns. By providing an avenue to embed domain knowledge and application

specific requirements into our AdaptiveWindow algorithm, the windows returned will fit both the dataset and the domain.

Breaking down the algorithm it first creates a temporary list of windows: $W_t =< T_0, T_1, ..., T_k >$ where each temporary window is equally spaced such that they are all of size n: $(T_1 - T_0) = n$. As each sequence in the dataset, D, contains a list of itemsets, I, and each itemset is denoted by a timestamp such that I_t is the timestamp of when the events in I occurred; the first window in the temporary windows begins at the earliest timestamp, t_0, that exists in the dataset. Building from there the window boundaries occur at $t_0 + (n * i)$ where i is the window index. Using this domain specific window measurement the algorithm places the itemsets of the data into the respective temporary window in which they occur. Once the temporary windows have been built the number of itemsets in each window is calculated, $|T_k|$, and then averaged together to identify the average across the dataset, $A = \langle < |T_0|, |T_1|, ..., |T_k| > \rangle$. This average itemsets, A, represents the average number of events contained in each window when spaced by the domain measurement n.

Once the average has been calculated it is then used to construct the optimal list of windows, W_o, where each window contains approximately A itemsets. By constructing windows that contain approximately A itemsets each, this means that the final set of windows will be reflective of the domain that was input into the algorithm. To construct the optimal windows, W_o, a famous method of discretization, also known as equal-frequency binning is utilized [154].

With the optimal windows identified, both sparse and dense areas of events

in the dataset will be eliminated allowing for windows of varying size that contain approximately an equal amount of data.

Algorithm 1 AWP-Span Part 1

Input: D, dataset of event sequences

E, event dictionary

W, windows of size ω

τ, support threshold

E_b, beginning event(optional)

E_e, ending event (optional)

Output: F, frequent subsequences for each window

C, connection values between each window

{Filter for beginning and ending events}

if $E_b \in E$ **then**

for $S_i \in D$ **do**

$S_i \leftarrow S_i \in index(S_i, E_b),$

end for

end if

if $E_e \in E$ **then**

for $S_i \in D$ **do**

$S_i \leftarrow S_i \in index(, S_i, E_e)$

end for

end if

{Separate D into windows, W, of size ω}

$n = |W|$

$W_D \leftarrow < T_0, T_1, ..., T_n >$ where $T_i = W_i$

for $T_i \in W_D$ **do**

$T_i \leftarrow |D \in T_i, T_{i+1}|$

end for

{Compute PrefixSpan results, $R =< R_1, R_2, ...R_i >$, for each window, W_i}

$F \leftarrow < F_0, F_1, ..., F_n >$

$F_i \leftarrow$ PrefixSpan(sequences=F_i, support=τ)

{Sort each result F_i by support value}

$F_i = \text{sort}(F_i, \text{dir}=ASC)$ {Sorting by support value}

{Keep only the top k subsequences in each result F_i}

$F_i = F_i[0 : k]$

{Merge adjoining windows using WindowMerge}

$C \leftarrow <>$

for $W_i \in W_D$ **do**

  for $W_j \in W_D$ **do**

    $C \leftarrow C \cup$ WindowMerge(W_i, W_j)

  end for

end for

return F, C

Algorithm 2 WindowMerge(i, j)

$S_{ij} \leftarrow D \in W_i, W_j$

$C_d \leftarrow C_w \leftarrow 0$

for $f_i \in F_i$ **do**

  for $f_j \in F_j$ **do**

    if $E_b \notin f_j$ **and** $E_e \notin f_i$ **then**

      if $f_i \in D_{ij}$ **and** $f_j \in D_{ij}$ **then**

        if $index(D_{ij}, f_j) - index(D_{ij}, f_i) = 0$ **then**

          $C_d \leftarrow C_d + 1$

        end if

        if $index(D_{ij}, f_j) - index(D_{ij}, f_i) > 0$ **then**

          $C_w \leftarrow C_w + 1$

        end if

      end if

    end if

  end for

end for

return $< C_d, C_w >$

Algorithm 3 AdaptiveWindows

Input: a dataset of event sequences D

domain window size n

Output: adapted windows W

$t_0 \leftarrow$ earliest timestamp in D

$W_t \leftarrow < T_0, T_1, ..., T_k >$ where $T_i = t_0 + (n * i)$

for $T_i \in W_t$ **do**

$T_i \leftarrow |D \in < T_i, T_{i+1} > |$

end for

$A \leftarrow \langle W_t \rangle$ {Average itemsets per windows of n}

$W_o \leftarrow$ equal_freq_bin(dataset=D, freq=A)

return W_o

4.5 Visualization

Once we obtained the results from our AWP-Span algorithm, we sought to visualize the overall flow of events and understand the various connections between windows. We show the results of our algorithm on two different visualizations: the Sankey diagram and our own event summary design.

4.5.1 Mapping Common Sequences

Due to the atypical nature of our algorithm results which contain various types of connections (direct and wildcard), we needed to devise a mapping from the results obtained to the Sankey diagram. Next we will describe the mapping that we have defined.

First, we represent each event as a node in the Sankey diagram and place the restriction that each window in our results must contain its own separate set of nodes, such that the windows are initially not connected. By requiring each window to have its own set of nodes, this ensures that the Sankey diagram contains a temporal context as events that happen in different locations of a dataset (in different windows) are not equivalent to one another. To build each window's set of nodes we convert the frequent subsequences contained in the window to a series of nodes such that similar events that occur in the same index of a subsequence are merged. For example, given a window that contains the frequent subsequences $S_1 =< A, C, B >$ and $S_2 =< C, D, A >$, C would be represented as a single node due to it occurring in the same itemset index 0, but A would not be merged as it

occurs in different indices 0 and 1.

Second, for each pair of nodes in a window that have different index positions, an edge is drawn in between them. Using the previous example of S_1 and S_2, two edges would be drawn from the node C: an edge from C to B and an edge from C to the A node in the second index. As each edge represents the flow from one event to the next, we encode the number of times that the respective transition occurred between the events into the edges.

Third, we link adjoining windows, W_i and W_j, through two methods: a direction connection and a wildcard node. Given the frequent subsequences of each window, $F_{im} \in W_i$ and $F_{jn} \in W_j$, these two methods are utilized on the last event in each F_{im} such that $E_{im} = F_{im}(-1)$ and the first event in each F_{jn} such that $E_{jn} = F_{jn}(0)$. For the direct connection the two events, E_{im} and E_{jn}, are connected by an edge that is encoded with the direct count between F_{im} and F_{jn} that was identified by our algorithm.

For the wildcard node an extra node is drawn between the two events, E_{im} and E_{jn}, which is labeled as a "wildcard node". Two edges are then drawn: (1) from the event E_{im} to the wildcard node and (2) from the wildcard node to the event E_{jn}. Both of these edges are encoded with the wildcard count that was returned by the algorithm.

While the wildcard node is powerful at representing extra information that is contained between two events, for some visualization techniques this creates extra nodes and thus noise. Therefore, to reduce this noise and clutter caused by the wildcard nodes we alter the mapping so that instead of creating a wildcard node,

a colored edge between E_{im} and E_{jn} is drawn. This change reduces the amount of nodes in the graph and still allows for the difference between direct and wildcard connections to be understood through visual inspection of the graph.

4.5.2 Sankey Diagram

A popular visualization, the Sankey diagram, has been used to understand the flow of resources in data [155]. We utilize the Sankey diagram by mapping our algorithm's results to the diagram which allows for us to visualize the flow of events in the various trajectories.

An example of a Sankey diagram produced by mapping our algorithm is shown in Figure 4.7 Left and Middle where wildcard nodes and edges are used respectively. Through visual inspection we can see that Figure 4.7 Middle contains less clutter and is easier to understand due to the encoding of the wildcard into the edges.

Analyzing Figure 4.7 Middle we can describe the overall flow of events: first an event of f is the most frequent while D, T, and H also occur at much lower frequencies. Next, the events D, T, and H proceed to another D with the sequences being split about 50% for going to the second D on a direct connection or with extra, infrequent events in between. At the same time, the f event proceeds to a different D, f, and H before ultimately ending up at the same D as the previous events. Once all of the events have reached the large D on the right side of the diagram, only a small portion move further into another repeated event of D. The reason that there exists a small path out of a large event node is that the final D

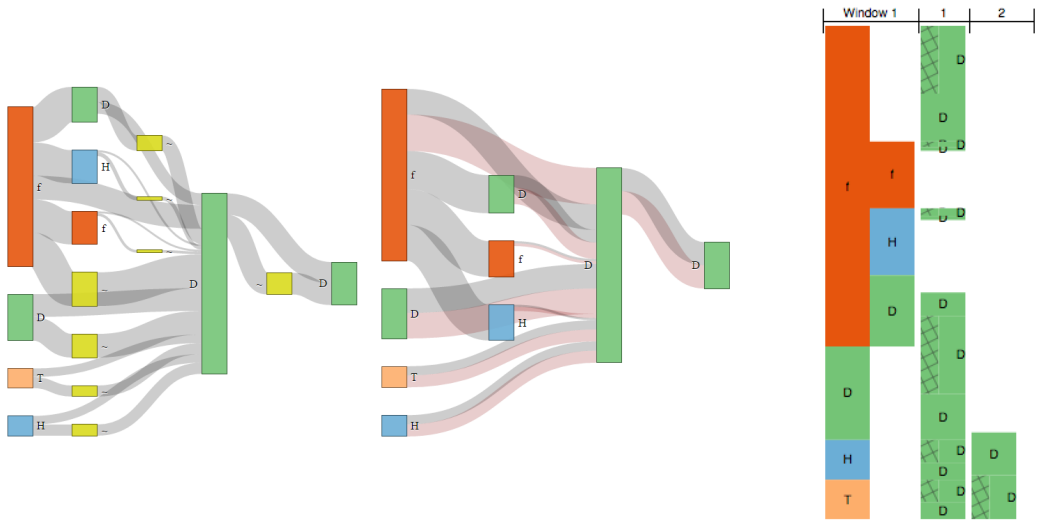


Figure 4.7: An example output of our AWP-Span algorithm mapped to a Sankey and our event summary diagrams. (Left) Shows a Sankey diagram containing wildcard nodes causing for extra clutter and noise to exist. (Middle) Shows a Sankey diagram with the wildcard nodes converted into red-colored edges that can be expanded with a click to show the underlying wildcard data which provides multiple levels of detail. (Right) Our event summary diagram where the edges have been removed to decrease the analytical capacity needed to understand the overall flow of events. Wildcard edges are now mapped to a texture rather than a red-colored edge and windows are separated by a vertical gap.

is infrequent in its window relative to the other windows. This means that the D event is a frequent subsequence in its own window, but that the other windows have frequent subsequences of a much higher frequency.

Through this analysis of the Sankey diagram it can be seen how a business analyst could identify the most popular marketing channels, in this case the D channel would be indicative of being very popular, as well as understand the varying

paths that users take: most users will end up at the largest D node but in the first column the D, T, and H are all just as likely to end up at D as the f channel. Furthermore, in the healthcare domain a clinician would be interested to see that most patients end up with a diagnosis of D, but that the paths that the patients took differed.

While the Sankey diagram is able to show the flow of sequences between events for all of the windows, it suffers in its visual complexity and ease of use. When analyzing the Sankey diagram, a user is required to follow the edges out from a node to be able to understand the destination events because of the fact that edges can be curved and are overlapped in the diagram. In addition, as the size of the data being represented increases, and in our case the number of windows, the size of the diagram grows significantly causing for it to be difficult to obtain a general overview of the data. These are limitations of the Sankey diagram which directly apply to the data that is attempted to be visualized from our AWP-Span algorithm, and thus we sought to identify a better method of visualizing the data.

4.5.3 Event Summary Diagram

As the Sankey diagram suffers from difficulties in identifying the flow of events with the spaced-out edges, as was identified by MatrixWave [140], we developed an edge-free visualization that reduces the complexity of analyzing and understanding the different flow of trajectories. We will outline the steps taken to construct our event summary diagram to overcome these limitations.

First, in the Sankey diagram there is no requirement for event nodes to be aligned as the edge between a set of nodes is flexible to ensure that the connection is fully made. While this ensures flexibility, it also requires immense visual thinking to be able to thoroughly understand the connections. To overcome this issue we removed all edges in the graph and aligned each node next to the node that it originated from. This alignment causes for a clear path to be understood through a quick glance at the graph.

Second, as we removed edges between nodes, and thus removed the encoded wildcard edges, we textured a portion of the destination event node to indicate that the sequence passed through a wildcard prior to reaching this event. Adding a texture to the node adds minimal noise to the diagram while adding key information. We chose to only texture a small portion of the destination node over applying the texture to the entire node, because through a user evaluation we found that texturing the entire node created confusion as to which connection contained the wildcard: whether it was the connection into or out of the node.

Third, as our AWP-Span algorithm is built on the premise of dividing event sequences into windows, we added a vertical spacing between each set of windows so that through visual inspection it is possible to understand the frequent subsequences contained within the window. By ensuring that each window can be individually analyzed, users across many domains can understand a particular section of a dataset. For instance, a clinician could isolate a specific window to evaluate the most common diagnoses and paths followed within a certain time span.

Finally, as each window in our AWP-Span algorithm contains its own frequent

subsequences, it is possible that certain windows will contain more sequences than another window. As an example situation: an event A could go to event B with a value of 5, followed by B going to C with a value of 10, where A, B, and C are in separate windows. In this example, the event B would be of size 5 while the event C would be of size 10 and following our first requirement of aligning nodes next to the node that they originated from, this difference in size would not be possible as there is not enough B to flow into C. Therefore, to overcome this limitation we utilize proportions on each window to ensure that the events can be drawn and that the relative frequency within a window is sustained.

We utilize the example data shown in Figure 4.7 (Left and Middle) to present our new design in Figure 4.7 Right. Through visual inspection, we can see that our new design contains the same information present in the Sankey diagram, but does so in a simpler and more easily understood manner. For instance, we can see that the majority of the events in the first window were red, but in the event summary diagram we can better see the proportion difference as there are no gaps between the events. Then, we can see that the first window also has an additional series of events that come from the red nodes, but the rest of the events/nodes do not have any additional events. This is very difficult to identify in the sankey diagram due to its algorithm for attempting to move the nodes around and thus they do not line up with the first large red node.

Looking at the additional windows, it is much easier to see that all of the events have wildcard edges and the proportion of them is also easier to identify. Finally, we can see that the blue and orange nodes in the first window are the only

nodes that have any events in the final window. Compared to the sankey diagram, this is impossible to identify in it. Ultimately, the simplification of edges decreases the analytical capacity required by the user and simultaneously reduces the virtual screen space occupied by the diagram.

4.5.4 Levels of Detail

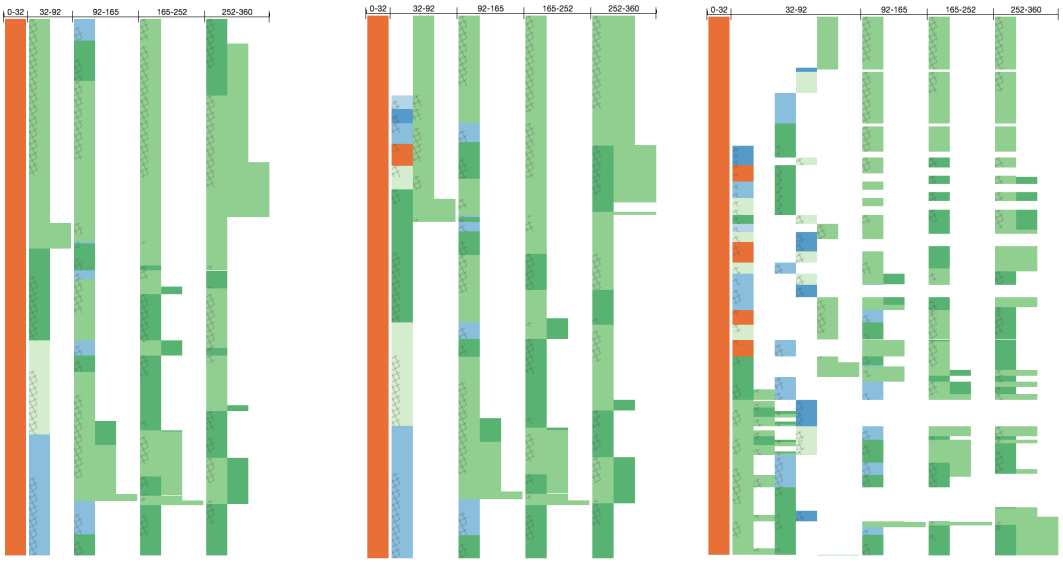


Figure 4.8: An example showing the multiple levels of detail available upon clicking on a wildcard edge. In this example, the user clicked on subsequent wildcard edges to show the underlying data.

As an extension to the wildcard edges produced, we recognize that the information contained in these connections may be interesting and contain useful information, thus we extend both of our visualizations to allow for expanding and collapsing of wildcard connections. Through an interactive action of clicking on a red-colored or textured wildcard edge, the wildcard is exploded such that the events

contained within it are drawn as nodes in place of the edge. An example of this is shown in Figure 4.8, where a wildcard on the blue "H" node was clicked to display the underlying wildcard data. This provides both of our visualizations with multiple levels of detail by helping to identify certain patterns or trends embedded within the wildcards and allowing for further exploration of a dataset of sequences than what the overall summary is able to present.

4.6 Case Studies

We demonstrate the capabilities of our algorithm and visualization on three datasets from vastly different domains that contain a large amount of records and have been used in real world analysis scenarios. For the purposes of evaluating the insights that can be discovered within these datasets, we interviewed three analysts that each have experience in their own respective domains.

4.6.1 Soccer Matches

Sports interests analysts and researchers because of the high volume of data that is produced throughout the course of a game: events are produced on and off the field and each player contains an assortment of attributes that can be learned from. For this case study, we utilized a Kaggle dataset that contains 9,084 soccer matches captured from the 2011/2012 season to the 2016/2017 season for the 5 biggest European leagues[1]. These matches, in total, produced over 900,000 events which were categorized into 11 different event types. The generated event summary diagrams for two different window counts, n, are shown in Figure 4.9 where the images correspond to 6 and 12 windows respectively.

We first presented the sports analyst this dataset loaded into an existing event analysis system as we had previously shown in Figure 4.3. Then we presented our tool with the initial number of windows set to 6, as shown in Figure 4.9 Left. Immediately upon seeing this visualization, before even attempting to change the

[1] https://www.kaggle.com/secareanualin/football-events

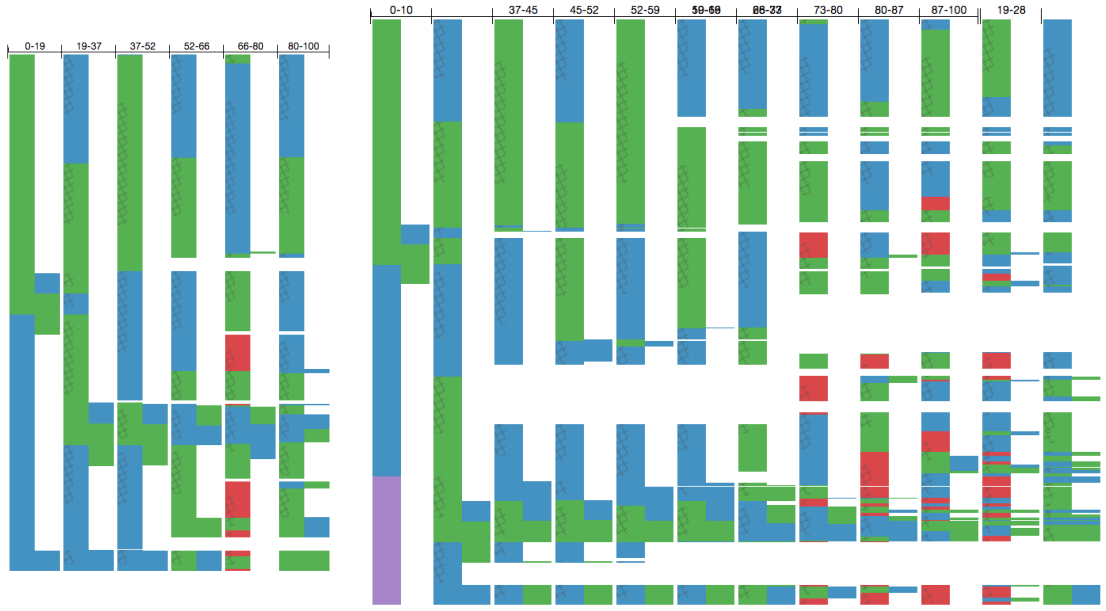


Figure 4.9: A dataset of over 9,000 European soccer matches were summarized into event summary diagrams containing (Left) 6 windows and (Right) 12 windows.

number of windows parameter, n, the analyst was intrigued as to the existence of the red events in the 5th window. Upon analyzing the legend, the analyst identified that the red event corresponds to a substitution event and confirmed that this is a typical behavior present in all soccer matches.

Next, the analyst began to change the parameter, n, and saw minimal changes in the visualization until they arrived at a value of 12 windows, shown in Figure 4.9 Right. At this window configuration, the purple event became present for the first time and through these series of parameter changes the analyst highlighted to us that they noticed two different patterns. The first pattern that they said they were able to uncover is that the purple event, which corresponds to a corner kick, did not appear until they had reached 12 windows. The analyst quickly began to think

out loud and realize that as the number of windows increased, the size of each window decreased. Thus they concluded that corner kicks were not frequent enough to appear in the larger size windows, but when confined to only the first 10 minutes of a game corner kicks were frequent enough. This was a very interesting trend to the analyst that they wished to investigate further.

The second pattern that they highlighted and were puzzled by was the large presence of green and blue events, corresponding to free kicks and attempts on goal respectively. The analyst eagerly opened the dataset in a statistical software and produced a histogram of the event types over the dataset where they noted that the green and blue events dominate the entire dataset. Through this they were able to identify a flaw with the dataset: that the majority of the dataset contained green and blue events which results in a dataset that contains very little useful information. This insight ultimately saved the analyst from performing any potential future analysis or at least identifying the flaw before they choose to analyze..

The analyst concluded by stating that compared to the existing event analysis system, our visualization was key in performing a top-down analysis of the dataset rather than a bottom-up approach. Most notably, they did not find it possible to arrive at the same insight of substitutions occurring in the later parts of a game in the existing system as its purpose is not catered to capturing a summary of the contained events.

4.6.2 Web Traffic Logs

We performed the next case study on web traffic logs with an analyst that manages front-end developers and is in charge of ensuring that a system is optimized for the end users. Traffic and access log files are an interesting data source to researchers from many different perspectives ranging from cybersecurity [156,157], to advertising and marketing purposes [158,159], to website and usability optimization [160,161]. However, these datasets commonly exist in large data sizes to the extent of gigabytes and even terabytes of data. At these large scales, it is paramount that analysts are able to fully understand the dataset that they are presented with, and thus a summary is greatly beneficial for these types of analyses.

For this case study, we utilized a dataset released by msnbc.com containing the server logs for the entire day of September 28, 1999 [162]. The dataset contains 989,818 tracked users which make an average of 5.7 visits across 17 different categories of pages on the website. Due to extreme outliers present in the data we removed the 10,000 users that made over 50 clicks, leaving us with a still sizable dataset of 979,133 users.

We presented the web traffic analyst with the summary mined from our AWP-Span algorithm mapped to both a Sankey diagram and our event summary diagram as displayed in Figure 4.10. The analyst immediately was offput by the Sankey diagram, and stated that the only insights that they could safely discover in this "mess" were that users typically visit the blue page first, corresponding to the frontpage of the website, and that over time less users clicked due to the smaller nodes in the

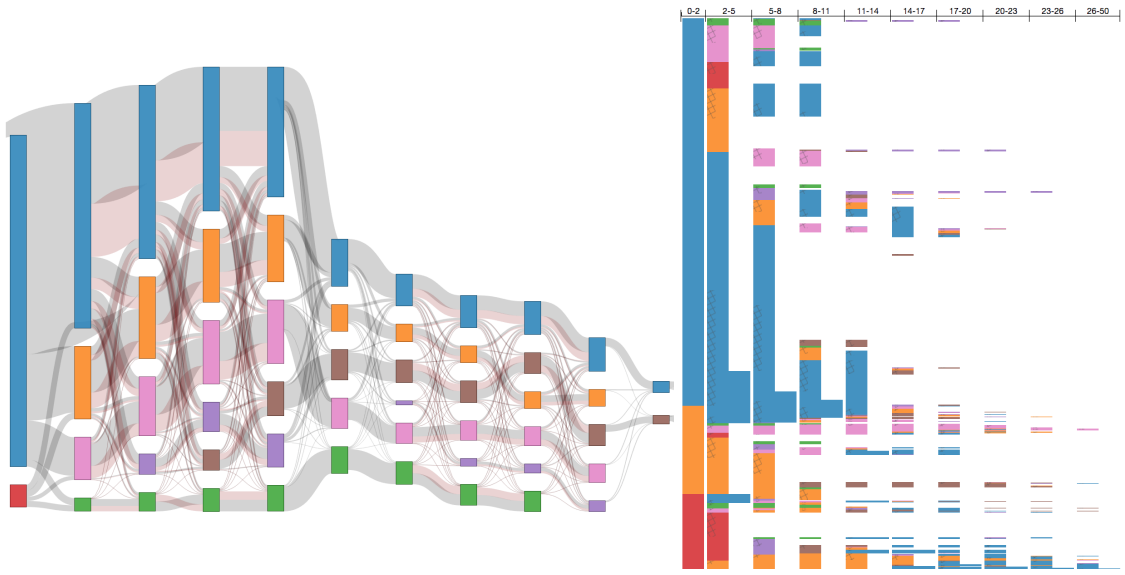


Figure 4.10: A web traffic dataset containing 989,818 users was summarized using our AWP-Span to produce a (Left) Sankey diagram and (Right) event summary diagram.

Sankey diagram.

However, upon switching to our event summary diagram, the analyst began to quickly pick out interesting patterns and trends. They identified that most users stay on the frontpage between clicks, because of the large series of blue in the middle of the diagram, but that some of them move to the orange (news), red (as seen tv), and pink (sports) before coming back to the frontpage, as seen in the upper left section. Next they eagerly pointed to the bottom section of the diagram where the initial red events lead into repeated orange events that continue through all of the windows, which indicates that users that visit the website through the "as seen on tv" (red) section will stay on the website for a long time, clicking through different news (orange) pages. They noted this as being a helpful insight into user retention

and pointed out a comparison to the middle section of the diagram where users that entered the website through the frontpage leave after a significantly lower amount of clicks than those that enter through the "as seen on tv" section. Based on this insight, they stated that the frontpage could potentially be optimized to attempt to guide users towards these pages with higher retention rates or that more advertising of the website could be done on TV.

4.6.3 EHR Data

With the challenge of EHR data in mind, as discussed in Section 4.2, we performed a final case study with a large, complex, clinical dataset. This dataset consisted of 89,840 patients that have been diagnosed with a mild Traumatic Brain Injury (a.k.a. concussion). The longitudinal data included 5.3 million TBI-related clinical encounters over 10 years and over 8.7 million clinical diagnoses (i.e., events), where a TBI-related encounter was defined as a visit to the doctor where the patient was treated with one or more of the conditions that are commonly known to be related to concussions such as behavioral disorder, sleep problems, cognitive deficiencies, and audiology complaints. The patients under consideration had an average of 59.06 encounters.

As a common task in evaluating and understanding TBIs involves analyzing the first year after an injury, we filtered the dataset to contain the first 365 days after a patient's concussion before running our AWP-Span algorithm. A mined event summary is shown in Figure 4.11 Left and a subset of the dataset (only 1,000 patients

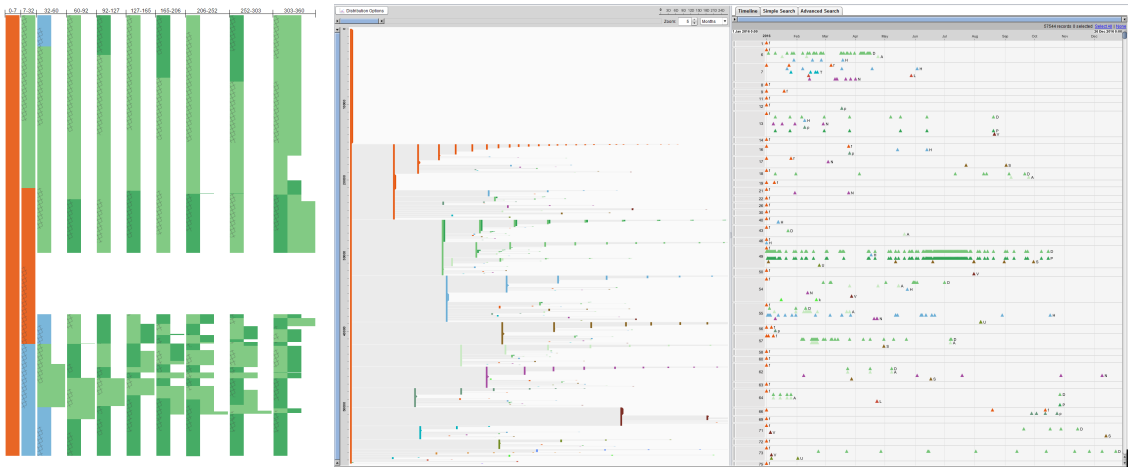


Figure 4.11: (Left) An event summary diagram for EHR data. (Right) The same EHR dataset loaded into an existing event analysis system.

due to performance limitations) loaded into an existing event analysis system is shown in Figure 4.11 Right.

The healthcare analyst looked at both types of visualizations displayed in Figure 4.11 and first noted three similarities: (1) they both show that patients that suffered a concussion (orange) and then had no more events available afterwards, (2) they both indicate that patients have repeated diagnoses, and (3) that Depression (light green) and PTSD (dark green) are very common to occur amongst concussion patients. Beyond these similarities, the analyst liked that the Figure 4.11 Right showed all of the different types of events, but they stated that they struggle to identify any patterns due to little change in diagnoses/colors for each record. Thus, they moved to the diagram shown in Figure 4.11 Left and expressed their satisfaction for time periods being present at the top of the diagram to aid in the understanding of the trajectory. With these time periods (window configurations), they found

it interesting to see that the size of the windows increased in the later windows, indicating that patients utilized less healthcare and doctor visits as time went on. Additionally, the analyst noticed that the light blue (headaches) diagnoses only occurred in the first couple of windows after a concussion which they confirmed follows the medical literature.

The analyst was further interested in understanding what diagnoses the patients had between diagnoses that our system did not find to be common and thus they began to click wildcard edges and arrived at a diagram similar to the one found in Figure 4.8 Right. Through the expansion of these wildcard edges, the analyst discovered that patients seemed to only have diagnoses of sleep (dark blue) near their concussion, but not in the later windows. They found this interesting and noted that there could be a relation between the diagnoses of sleep and headaches only occurring in similar windows.

4.7 Limitations

While our approach is able to produce an effective summary visualization, there are limitations that must be taken into account when utilizing it. Many approaches take a bottom-up approach, such as EventFlow [43], where the user is presented the entire dataset and must perform operations to simplify it and create a summary. However, our approach takes a top-down approach by presenting the user with a simplified visualization that they can interact with to gradually view more detail. Because of this different method there may be information that is crucial for the user to identify within the dataset, but was removed during the process of summarization. Thus, extra care should be taken to understand that our summary visualization is strictly used for providing a high level overview and does not reveal all patterns within the dataset.

Also, because we build a representation of the dataset using the mined frequent subsequences and their connections between windows the summary visualization generated is not a true view of the dataset but rather a simulated view. Thus, there is a lot of information that is hidden or removed that causes for the summary visualization to be prone to misinterpretation. To overcome this limitation, in the future we seek to build a summary that attempts to follow the bottom-up approach and create an algorithm that performs the typical simplification techniques followed by users in an automatic manner. With this alternate approach the entire dataset would be visualized and the frequent patterns could be highlighted while the rest of the dataset is hidden (or greyed out into transparency). Additionally, the algorithm

could identify sequences to remove from the visualization or condense into another in order to arrive at a visualization that is able to support large scale datasets. Ultimately, this alternate technique would ensure that the summary visualization is a true representation of the dataset while still providing the benefits of a top-down approach.

4.8 Conclusion

Previously frequent sequence mining (FSM) algorithms had been used to understand the patterns contained within a dataset and did not take into account the location of each pattern. Through the development of our algorithm we have shown how the temporal context of each mined pattern can be utilized by breaking a dataset into temporal windows and constructing a trajectory of the various common paths that sequences may follow. Compared to displaying all possible event paths which suffers from scaling issues, our unique method of utilizing an FSM algorithm provides a condensed overview of an event dataset while still allowing for exploration at multiple levels of detail. In addition, due to the window nature of our algorithm the computation for each window can be run in parallel allowing for results to be obtained quickly.

Additionally, through our new novel event summary design we have reduced the visual complexity existent in a typical Sankey diagram. We have demonstrated the effectiveness and usefulness of our algorithm and event summary diagram through three case studies, where analysts from different domains were able

to quickly find insights and identify the aid that our approach provides for solving tasks faced each and everyday by event researchers in large scale temporal datasets.

Because we are now able to both predict and summarize irregular events, analysts and end-users can be provided with the models and visualizations to further understand their datasets and utilize them correctly. As users interact with these different components, there is a lot of knowledge that can be learned from the way in which they interact. For instance, "can the visualization be improved?", "is the user lost?", "did the user identify a pattern in the dataset?", and many more questions can be asked. With this, the interactions that the users perform are also irregular in nature and we can model these interactions as irregular events to also learn from them and further improve our models, visualizations, and users' analytic capabilities.

Chapter 5: Reinforcement Learning on User Interactions

5.1 Introduction

Imagine a typical scenario where a user has a specific task or question to answer: they open a visualization tool, load their dataset, map attributes of the data to visual representations, employ filters to manipulate the data or visual representations, and continue to go through an analytical process of exploring the information until they reach a state where the task under consideration is complete. Throughout this process, no matter the knowledge level of the user, each interaction that they take leads them closer or further away from their goal as well as provides some value, whether that be large or small. There is a wide range of questions that can be asked of these interactions, such as: Does the user seem to be confused by the visualization? Is the user lost and do they need assistance? How important is applying a specific filter onto the data? Can the interface be made better?

However, despite the growth of visual analytics and the widespread use of visualization tools, mining and modelling these user interactions and applying the knowledge embedded within them continues to be a difficult task. Current approaches are limited by either relying on users to analyze their own thought process [163] or on experts to undergo a laborious process to construct task specific rules [164–166].

To further extract the wealth of knowledge that is contained within the user interaction process, an approach that moves beyond the standard techniques, that can automatically learn by itself, and that can generalize across tasks and domains is needed.

We introduce the application of reinforcement learning onto user interactions within a visual analytic system. Reinforcement learning has the ability to uncover the hidden knowledge and insights embedded within interactions, and most notably can be performed in an automatic manner. In this paper we build and outline the connection of reinforcement learning to user interactions and show its utility through three applications: (i) providing guidance to users towards the completion of a task, (ii) personalizing guidance to match each individual user's behavior and preferences such that the guidance is "polite" and unobtrusive, and (iii) optimizing the visual analytic interface to maximize user performance.

5.2 Related Work

5.2.1 User Interactions

The capturing, modelling, and analysis of user interaction data to understand a user's analytical process in a visualization has been an active research area in the visualization [167–171], human-computer interaction (HCI) [172–174], and worflow mining communities [175, 176]. Multiple systems have been developed that provide users with manual methods of understanding their own interactions, which include: presenting a historical timeline of interaction data to the user and allowing for them to analyze their own analytical process [163], visually encoding a user's interaction history to promote exploration [177], and providing users with the functionality to annotate and share their actions [178]. While these manual techniques allow for users to evaluate their own missteps, the reliance on users to correctly classify their own actions is problematic when users are lost and unsure of how to accomplish their task, or even for new tasks that they have never encountered.

To allow for an external evaluation of where users become lost in a visualization, researchers have attempted to understand interactions by capturing and analyzing the user's analytic provenance by means of eye movement data [179–182], interaction logs [183, 184], and thinking aloud videos [185]. This work has aided the understanding of a user's thought process, but relies on an expert to identify the missteps.

With the goal of building systems that automatically learn and adapt to users,

Gotz and Zhou defined the different tiers of the visual analytic process [186] and Ragan et al. developed a framework of user provenance types to assist researchers in evaluating designs and capturing provenance information [187]. Building on this defined interaction data, Xiao et al. utilized past visualization states and actions to construct an improved and more relevant visualization for future users [188]. Brown et al. used low-level interactions, specifically mouse activity, to classify fast and slow users as well as personality traits [189]. Predicting the next click that a user will make using mouse activity was found to be feasible [190, 191]. Gotz and Wen built a system that analyzes users' actions using regular expressions to suggest alternate visualizations [192]. VisTrails captured user actions in a visualization and analyzed the various branches in activity that users took [193]. And Dabek et al. developed a system that applies a grammar induction algorithm onto a group of user interactions to identify the various paths that users can take and then defined a set of explicit rules that defined when and how to provide assistance to users [194].

5.2.2 Reinforcement Learning

Reinforcement learning is a class of machine learning algorithms that learn how to optimally choose actions to accomplish a task. Recent successful applications have shown utility in a wide range of tasks [195–199]. The underlying principle behind reinforcement learning is that it builds a value structure of the possible actions through learning from experiences of playing a game repeatedly or performing simulations of a task. By learning the value of each action, the algorithm can identify the optimal actions to take at every point of the task.

One area in which reinforcement learning has found tremendous success are in games such as Chess and Go [200–202]. Both of these are historic games that require a lot of thought and future planning of moves. They are regarded as highly intellectual games that while relatively easy to play, are difficult to master. Within both games' communities, an extensive analysis into different strategies and key opening moves to maximize a player's chance of winning have been identified [203]. This analysis required a lot of exploration to be identified, but reinforcement learning models have been able to identify these same insights as well as uncover new ones that had yet to be discovered [204, 205].

A visual analytic interface can be thought of as a game. Similar to chess, the interface can be easy to interact with, but difficult to master. There are also key interactions that users take which lead to their success at accomplishing a task. And similarly, there are many discoveries that are yet to be uncovered.

5.2.2.1 Definitions

Reinforcement learning algorithms are composed of multiple components. The agent is the actor, user, or robot that interacts within the space. The environment is the simulation, scenario, or world that an agent interacts with and changes as actions are performed. Each unique scenario that the agent encounters is referred to as a state. Actions are the finite set of things that can be done within the environment. And rewards are associated with moving between states and tell you what you get for the transition. The agent's goal is to learn a policy that indicates the action to take in each state by maximizing the sum of future discounted rewards.

We relate these key terms directly to a visualization system:

Agent The actor that takes actions and moves. (Ex. User/Robot interacting with the interface)

Environment The world with which the agent interacts. (Ex. Visual Analytic System)

State The current situation of the environment that the agent is in. (Ex. The current configuration of the interface and visualizations: data variables enabled, data manipulations, visualization types, etc.)

Action Possible moves for the agent to take within the environment. (Ex. Clicking a button, filter, visualization, etc.)

Reward A scalar value that indicates how good or bad it is to take specific actions in specific states. (Ex. Assign +10/-10 for arriving a the goal configuration and -1 for each click/interaction taken)

5.3 Research Applications

To help introduce the concept of reinforcement learning on user interactions, we present three applications and areas of exploration using these class of algorithms.

A1 Provide Guidance: Typically when confronted with a new task, visual analytic system, or a particularly challenging dataset users may have a difficult time identifying answers to their questions or new insights in their data. Because of this hurdle, users may take missteps or a path that does not lead them towards completing their task and they may ultimately become frustrated and discouraged. Therefore, it is important to ensure that users stay on track by guiding them when they get stuck or require assistance.

Using a reinforcement learning model and training it within a simulated visual analytic environment, the value structure of each action at each state could be obtained. Through this value structure, the model captures the value of interactions, controls, and visualizations within the interface. Then when a user interacts with the real world system, for each state that the user enters within the system the optimal action could be presented in the form of guidance.

A2 Polite Guidance: While determining what to suggest to a user when guiding them is an initial first step, it has been highly documented that providing assistance to users in a polite manner is important to not impede on already established workflows [206]. Therefore, it is crucial to identify when to provide suggestions to users to avoid being intrusive and widely disliked as Clippy in

Microsoft Office famously was [207]. As shown in the related work, recent methods for user assistance have relied on manually creating rules to determine when to assist a user, but these methods lack the ability to conform to user preferences and dynamically adapt to the interface that they are being presented with.

With a reinforcement learning model the states and actions could correspond to the interaction history and whether or not to make a suggestion, respectively. This model could then receive rewards dependent on if the user follows the prompt or chooses to dismiss it. Through this, a policy of the user's preferences could be learned and thus we can improve how we assist users by determining when to assist them and allowing the model to adapt to their preferences over time with continuous training.

A3 Optimize Interface: The placement of elements within an interface have been shown to be important to the effectiveness of a user accomplishing tasks [208–210]. This can be extended to visual analytic systems in which the placement of buttons, filters, and visualizations are crucial to assist the user in understanding the data and going through their analytic process.

However, identifying the correct arrangement of an interface is a difficult task which typically requires extensive user studies, use of eye tracking devices, and other methods [211, 212]. These methods rely on the feedback of the users, but can be lackluster as users may not themselves know the best method or arrangement [213]. Additionally, studies rely on the statistics of the users'

performance and typically do not place enough emphasis on the interactions performed.

Therefore, utilizing user interactions is a key way to identify which clicks are the most valuable and optimize the interface in terms of the design and layout in order to maximize the users' performance and analytical understanding. With reinforcement learning, the overall value of actions at each state could be analyzed to identify potential areas of the visual analytic system where users may be become stuck or otherwise areas that are of tremendous value.

With these three research applications defined, we formulate hypotheses to test with our approach that we will build:

H1 The use of reinforcement learning will lead to more accurate or faster solutions by users within a visualization system.

H2 We can learn a policy that makes suggestions to users at the correct times, without disturbing them.

H3 Through the presentation of suggestions, users will be more correct.

H4 We will be able to identify the optimal user interaction path for a given task within a visualization system without any user input or data.

5.4 Approach

Our approach uses the Q-Learning algorithm [214] due to the interpretability of the underlying matrix that is built. Therefore, an in depth explanation of the algorithm is explored.

5.4.1 Learning by Playing

Q-Learning is an algorithm that attempts to learn a value for all possible actions within each state. Thus when the algorithm reaches a state, it picks the action with the most value for the given state. For this algorithm, the value is a function such that it is equal to the expected total reward that would be obtained after picking the associated action if the agent was to follow the policy afterwards. The equation for the value can be thought of as:

$$Q^{\pi}(s_t, a_t) \leftarrow E_{\pi}\{R_{s_t,a_t} + \gamma \max Q(s_{t+1}, a)\} \tag{5.1}$$

where the sum of the maximum reward at all future states, up to the goal state, is computed. Through this value function, an agent learns to take actions to maximize their future rewards.

To obtain the associated value for all actions within each state, Q-Learning builds a matrix/table where the rows correspond to states and the columns correspond to actions. Each time that the agent needs to take an action, it looks at the state (row) that it is currently in and identifies the action (column) with the

most value. Then upon taking the action, it receives a reward for taking it. In the simplest of cases it would receive +10 for a win and -10 for a loss, with all other cases resulting in 0. Using this reward, the algorithm then updates the cell of the action that it had just taken. The update rule is defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{5.2}$$

where $Q(s_t, a_t)$ corresponds to the current cell of the action that was selected; r_{t+1} corresponds to the reward that was obtained upon taking the action; $\max_a Q(s_{t+1}, a)$ is the maximum reward possible at the future state that the agent just moved to (by looking at the entire row of the state that it had just moved to after taking the action); and α and γ are two parameters that are set prior to training the model, such that α is the learning rate, how much the newly acquired reward should override the old information, and γ is the discount factor, how much of the future reward should be taken into account. With this update function and states/actions the Q-Learning algorithm performs the number of simulations, n, that are set for it and builds the associated matrix. The pseudocode using the update rule is shown in Alogrithm 4 [214].

5.4.1.1 Exploration

While a model is being trained, there may be instances where the algorithm has learned a specific path of interactions to take in order to reach the goal, but may still not yet have arrived at the optimal interactions to take. This is similar to where

Algorithm 4 Q-Learning

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$,

arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

for `each episode` **do**

Initialize S

for `each step of the episode (until` S `is terminal)` **do**

Choose A from S using policy derived from Q

Take action A, observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

end for

end for

a user follows a set interaction path to arrive at the answer that they are looking for without realizing that there is a more optimal path. This different path could present the data in a more impactful and intuitive manner for the user and thus it would be best for them to follow it. To resolve this issue, ϵ-greedy Q-Learning introduces an instance of randomness into the algorithm: at each point when the algorithm needs to identify the action with the most value for a given state, it occasionally will choose to select a random action. This ensures that the algorithm is able to continue to explore throughout its learning and potentially identify better actions to take than what it had initially learned. The equation for ϵ-greedy Q-Learning is displayed below:

with probability ϵ : choose an action at random

with probability $1 - \epsilon : a = \arg\max_a Q(s, a)$

5.4.2 Reward Structure

While a simple example of a reward structure is to provide a +10 for winning and a -10 for losing, there are alternate structures that could be utilized to achieve different goals. For instance, with the simple reward structure of +10/-10 for a win/loss, the agent would not prioritize the number of actions that it takes to arrive at the goal. This means that within a visualization system, the model could identify an optimal path that consists of extra, unnecessary clicks. This type of example shows the importance of crafting the reward structure such that the desired outcome can be achieved.

To overcome this issue of the number of moves it takes, one could add an additional element to the reward structure such that the agent receives -1 for every action that it takes. With this updated structure the agent is still motivated most by a win/loss, but it will also attempt to find the shortest path to arrive at the goal because of the constant penalty it receives from making a move. In another example, the reward structure could be modified to motivate the agent to simplify the visualizations by assigning a +1 reward for each time that an unimportant variable is removed from the data being displayed. Additionally, when the discount factor γ present within the Q-Learning algorithm is less than 1, $\gamma < 1$, then the

agent is motivated to win quickly or put off losing as long as possible. Through these examples of a movement penalty, simplification reward, and discount factor tuning, it is possible to see how a carefully crafted reward structure, through the magnitude and sign of the rewards, is crucial to arrive at the goal set out.

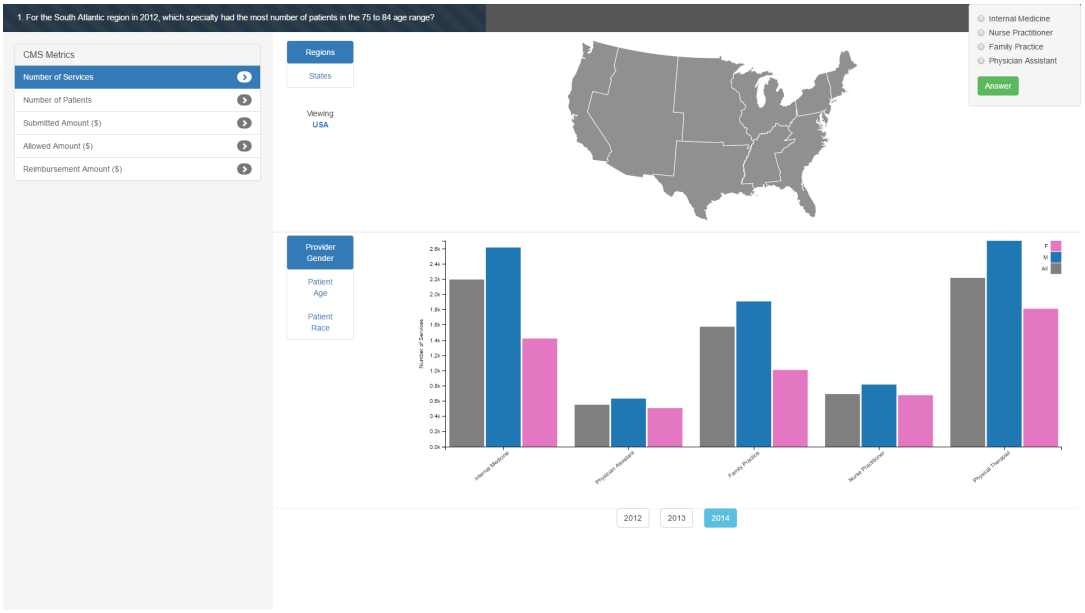


Figure 5.1: A screenshot of our visualization system built for capturing user interactions in a real-world type scenario.

5.5 Visualization System

To evaluate the effectiveness of applying reinforcement learning algorithms within a visualization user interface, a coordinated multi-view visualization system was built. In this system we sought to create a real-world type scenario similar to those visualization frameworks/tools that are created by Tableau [215] and D3 [216]. In these dashboards, users typically look to either explore the data to uncover insights or have a specific question to answer. For both of these tasks, it is necessary for users to click around the interface to manipulate the view and alter the visualization that is being presented.

The visualization system that we built is shown in Figure 5.1. The various components of this system include a list of variables on the left-hand side, a clickable

map in the top center with controls for switching between partitioning the USA into regions or states, a multiple bar chart in the bottom center with a control for switching between three different graphs and the ability to change the year. Finally, at the top of the screen there is a question along with four multiple answer choices at the top right.

5.5.1 Dataset

To mimic a real-world scenario, we chose to represent a dataset in the system that would not easily be understood and thus would require exploration of the data for insights to be discovered. Because of this requirement we focused on healthcare data due to its complexity and inherent lack of understanding compared to standard datasets such as weather, sports, etc. The final dataset chosen consists of medicare provider utilization and payment data taken from the Centers for Medicare and Medicaid Services (CMS) website[1]. The dataset covers the years 2012, 2013, and 2014 and provides an assortment of variables for each physician: the number of services that they performed, the number of patients that they saw, the average amount that they were reimbursed for services, the average age and race breakdown of patients, etc.

[1] https://www.cms.gov/research-statistics-data-and-systems/statistics-trends-and-reports/medicare-provider-charge-data/physician-and-other-supplier.html

5.5.2 Questions

For the study, we devised 10 multiple choice questions that were similar to the types of questions asked by media outlets, doctors, patients, and researchers in the healthcare domain [2] [3]. We ensured that each question required interaction for the answer to be visible in the corresponding visualizations. A listing of the 10 questions can be found below:

1. In 2012, how many services did female nurse practitioners perform in the East South Central region?
2. What specialty submitted the highest medicare charges in Nebraska (NE) for 2014?
3. In 2012, which gender submitted the largest payments in New Mexico (NM)?
4. For all physician assistants in Texas (TX) in 2013, which race had the second highest submitted payments?
5. For the Mountain region in 2013, which age group had the highest reimbursements over all specialties?
6. Over all degrees and genders, which specialty had the highest submitted payments for 2014 in Kansas (KS)?
7. For the South Atlantic region in 2012, which specialty had the most number of patients in the 75 to 84 age range?
8. How much were physical therapists reimbursed for white patients in the New England region in 2012?

[2] Wall Street Journal: http://graphics.wsj.com/medicare-billing/

[3] Washington Post: http://wapo.st/1hAFRFE

9. For physician assistants, which gender was able to submit higher amounts in Nevada (NV) for 2012?
10. For family practice physicians in Florida (FL) in 2013, which age group offered the second highest reimbursement?

5.6 Building Models

With our visual analytic system and associated dataset defined, models can be built to solve the tasks presented by the posed questions. As typical applications of reinforcement learning are performed in an automatic way such that a bot undergoes training runs/simulations through the environment in order to build a model, it is similarly possible to do this with a user interface and treat the bot as simulating a user. For a user interface, a bot would pick the different actions to take to manipulate the interface until it is notified of having reached the goal, which is defined as the state at which the solution to the question/task is displayed within the interface.

Because the bot is simply looking for the state at which the solution is visible and the question can be answered, there exists only a single goal state which we define as being the "winning" state. Thus, after attempting multiple reward structures we arrived at one that optimizes for arriving at the goal state quickly: assigning -1 for each action taken and a 0 once arriving at the goal state. With this structure the agent learns to not explore endlessly, but rather to find the goal state to stop being penalized.

To accomplish this training of a bot we utilized a popular reinforcement learning library, OpenAI Gym [217]. A separate model was built for each question and each was trained by running for a series of 10,000 training runs (an attempt at answering the question). The maximum amount of time that it took to train a model was 30 seconds.

To provide an overview of the models we analyzed the training process that

each model underwent. For this, we gathered the total amount of reward gained for each run, allowing us to see how well the model performed at each point. By analyzing the reward gained we can identify the moment where the model moves from being unintelligent to the point at which it is able to answer the question and receive a maximal reward the majority of the time. To better see this shift in learning, we applied a mean smoothing of size 100 to the reward data.

Figures 5.2a & 5.2b show the training process for two separate models for questions 5 and 10. The maximal reward is negative for each of these cases because in the optimal scenario it would arrive at the answer in a low number of actions which would be summed to be a negative number. The sign (positive/negative) of the number does not matter as simply arriving at the maximum possible value is the key.

Inspecting Figure 5.2a we can see that for answering question 5 the bot became optimal at about 6,000 training runs because of the large vertical spike, while for question 10 in Figure 5.2b it became optimal in the final few training runs before 10,000. This indicates that the bot found question 10 to be more difficult compared to question 5. This difference in difficulty can reveal a lot about the interface and the way in which users interact with it. For example, a task that is constructed with the intention of being relatively simple for an interface, but is found to be difficult by the model would reveal that the interface may need to be refined to have more specific controls for accomplishing the associated task.

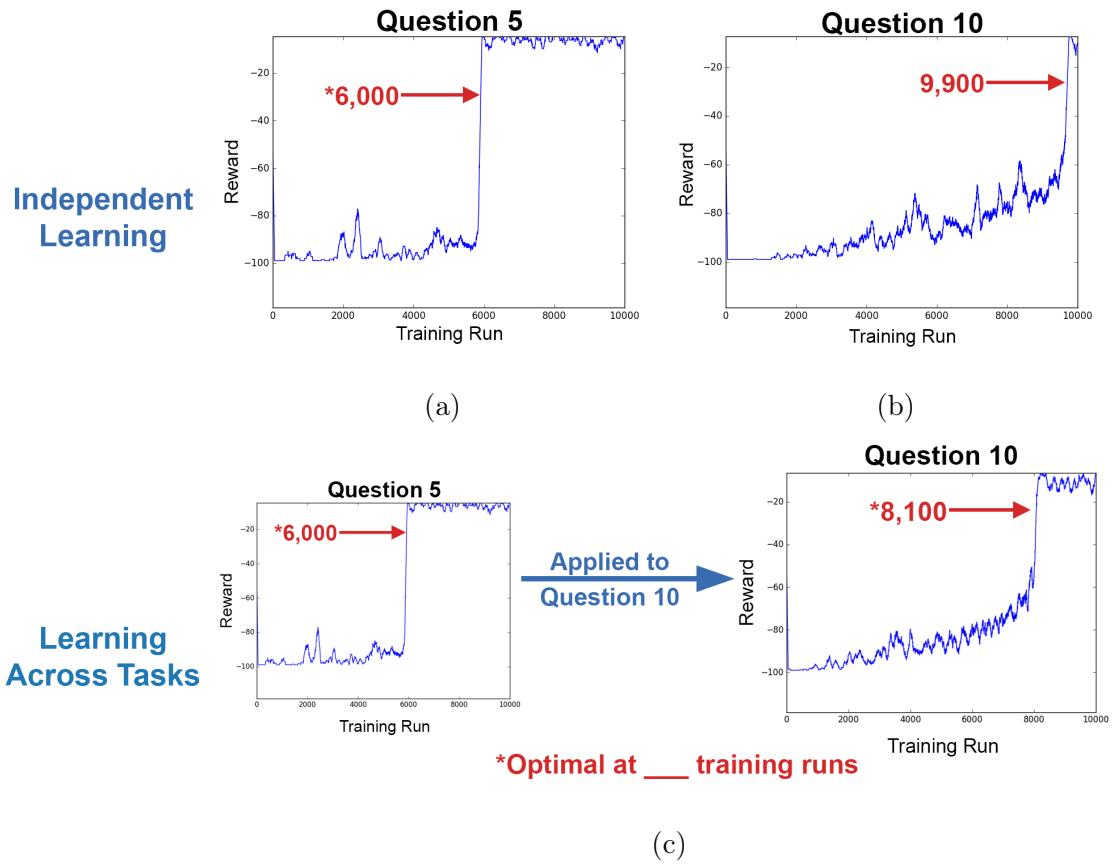


Figure 5.2: (a & b) A separate blank model was trained on (a) question 5 and (b) question 10. It can be seen that question 5 became optimal quicker than question 10 and thus can be concluded to be an easier task. (c) The model trained independently on question 5 is used as the starting point for learning question 10 and produces the resulting graph. This shows that the question 5 model had embedded knowledge about the interface and can be applied across tasks.

5.6.1 Learning Across Tasks

While we trained independent models for each question, we can also utilize a trained model as the starting point for a different question to show how knowledge is embedded and the applicability across tasks (referred to as transfer learning). Figure 5.2c shows this process and the performance on the newly trained task. By analyzing the improved model graph against Figure 5.2b, it can be seen that this model was able to answer question 10 in almost 2,000 less training runs. This indicates that even though the model was trained for a different question, it had learned and embedded information about the system which was then utilized to identify the path to answering question 10 in a quicker manner. This is directly applicable to a real-world scenario where a never before seen task has arisen and the optimal path needs to be discovered.

5.6.2 Extracting Optimal Path

Using the built models it is possible to extract the optimal path to complete a task. This can be achieved by starting at the state that corresponds to the beginning of the system, the "start state", then choosing the action with the most reward, and continuing to move through the states until the goal state has been reached. The sequence of actions that were taken is then considered the optimal path for answering the question and what the agent would look to take.

Looking at question 3 ("In 2012, which gender submitted the largest payments in New Mexico (NM)?"), the extracted path from the model is [*states, NM, submitted*

amount, 2012]. This path corresponds to a user clicking on the "States" button to switch the map view from regions to states, clicking on the state of New Mexico, changing the displayed metric to "Submitted Amount" by selecting it, and then changing the year of interest by clicking the "2012" button. This shows that the model learned the correct path to arriving at the goal state because the actions align directly with the keywords of the question and is the logical shortest path to arriving at the answer. Similar patterns were seen across the different questions.

It is also possible to see the optimal paths by inspecting the model's matrix. Using question 6 ("Over all degrees and genders, which specialty had the highest submitted payments for 2014 in Kansas (KS)?") as an example, Figure 5.3 contains a heatmap representation of the model learned. To construct the heatmap, rows and columns were removed if they contained all white, or nearly white, cells as this indicated almost no reward. This removal constrained the heatmap from thousands of rows down to ten.

This heatmap clearly shows that the model learned that the actions "Kansas" and "Submitted Amount" were the most important and provided the most reward to answer the question. Additionally, very light green cells can be seen which indicate that the model had explored other pathways that eventually led to the answer, but that it learned that these were not valuable to answering the question.

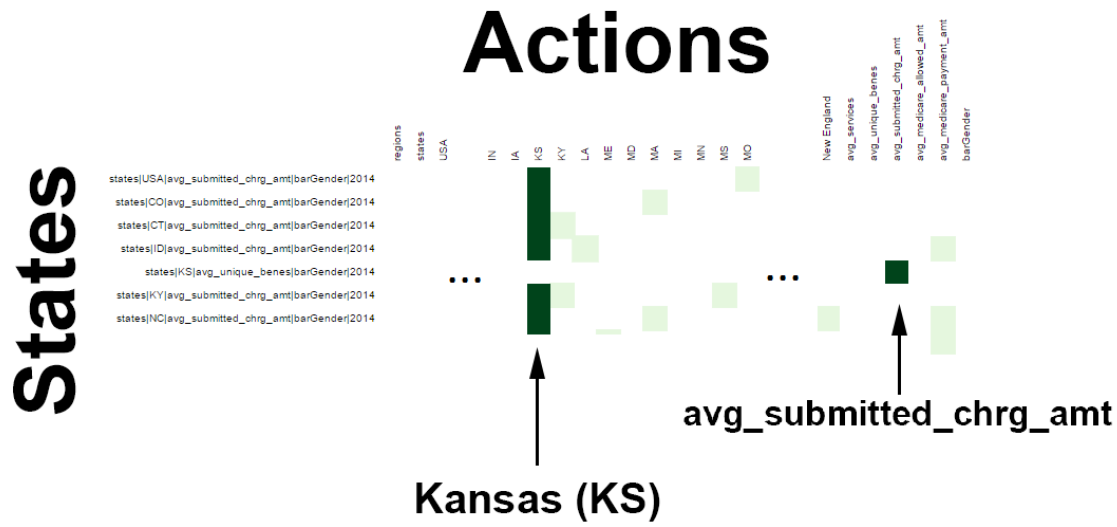


Figure 5.3: A heatmap visualization of the trained model for question 6 where rows and columns that had mostly white for the entire span were removed. Inspecting the heatmap, it shows that the model learned that clicking on "Kansas" and the variable "Submitted Amount" were the most important.

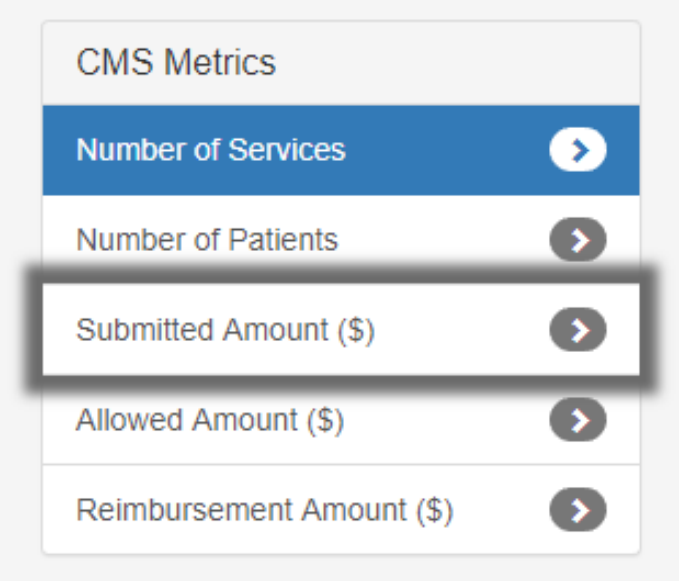


Figure 5.4: An example of a suggestion being presented to a user. In this example the user has the "Number of Services" variable selected, but the "Submitted Amount" variable is highlighted to suggest for them to take the associated action.

5.7 Providing Guidance

With the ultimate goal of providing guidance, we outline how we provide the suggestions and then how these suggestions are personalized/tailored to the user automatically using reinforcement learning. Figure 5.4 shows an example of how a suggestion will be presented to a user. In this example the "Number of Services" variable is selected, but the "Submitted Amount" variable is highlighted to suggest for the user to switch to. We chose to keep the suggestion unobtrusive to ensure that we follow the expectation of being polite to the user [207]. Even with this subtle suggestion, our results will show that users did follow the suggestions.

5.7.1 What to suggest?

As shown in Section 5.6, we build a model for each question by training it on simulations within the visualization system. Then the value of all possible actions in each state of the system can be identified. Using this value structure, guidance can be provided by presenting the user with a tooltip for the action that would provide the user with the most value/lead them towards answering the question correctly. Therefore, we utilize this knowledge in order to present suggestions to users with the goal of improving their accuracy as well as time and the number of actions taken.

5.7.2 When to suggest?

While we will be able to utilize our interaction models to provide guidance to users in the form of suggestion tooltips, this only solves the challenge of determining what to suggest and does not solve the problem of when to suggest. As it is has been highly documented that providing assistance to users in a polite manner is important to not impede on already established workflows [206], it is crucial to identify when to provide suggestions to users to avoid being intrusive and widely disliked as Clippy in Microsoft Office famously was [207]. Recent methods for user assistance have relied on manually creating rules to determine when to assist a user, but these methods lack the ability to conform to user preferences and dynamically adapt to the interface that they are being presented with. Based on this criteria reinforcement learning algorithms can improve how we assist by exploring how the user is interacting, determine when to assist them, and over time adapt to their

preferences.

Therefore, to personalize this guidance for each user we built a separate model called a "Preference Model" that also utilizes reinforcement learning, but instead focuses on determining the most opportune time to present the user with a suggestion. Some users prefer to receive suggestions frequently, others less so. The model can learn that certain users prefer to always get a suggestion after performing certain interactions while others may not like a suggestion because they find it too distracting to their analysis process or they already know what they would like to do.

5.7.3 Preference Model

The preference model has a different state, action, and reward structure than the one defined in Section 5.4, but follows the similar principle of utilizing reinforcement learning and a matrix learned through Q-Learning. To keep this model simple and interpretable we constrained it to two possible actions: "suggest" or "not suggest" which indicate whether or not the user should be presented with a guidance suggestion. Furthermore, when devising the reward structure we sought one that would encourage the model to only present suggestions at the most opportune time and ensure that it remains polite to the user. Thus, we wanted to make sure that the model would choose not to present a suggestion if it was unsure of the user's preference rather than potentially interrupting and agitating the user. With this the reward was structured such that if a suggestion is presented and followed then the

model receives a +5, but if the suggestion is not followed then it is penalized with a -5. And if the model chooses not to present a suggestion then it receives a -1. With the high values of +5 and -5 compared to the -1, the model is able to learn that suggestions are very valuable, but risky; and with the constant -1 it is encouraged to identify moments that the user would follow the suggestion and receive a +5.

As we sought for the preference model to generalize across tasks we defined the states of the model to correspond to the history of actions that a user had taken. Using a history parameter, h, the states are built such that each state corresponds to the last h actions that were taken. For example, given a user creating a chart in Tableau a set of actions within the software could include: $Upload, SetChart, SetColor, FilterVariable$ and with $h = 2$, such that the last two actions make up the states, then possible states for this model would include $SetChart - SetColor$, $SetChart - FilterVariable$, $SetColor - FilterVariable$, etc. Thus a user at the state $SetChart - SetColor$ would have changed the chart type followed by changing the color scheme, resulting in the model learning over all times that the user had performed these two actions whether or not they prefer to have a suggestion displayed to them.

An extra action is also included in the preference model's possible set of actions called "time". This action represents a user not having taken an action for the last s seconds and allows the model the ability to evaluate users that may have become lost or unsure about where to click next and act upon this.

With this model structure the parameters, h and s, can be configured based on the system it is contained within. For our system we initialized the parameters

to $s = 5$, representing a "time" action to be triggered after 5 seconds of inactivity, and $h = 2$, representing a history look-up of two actions.

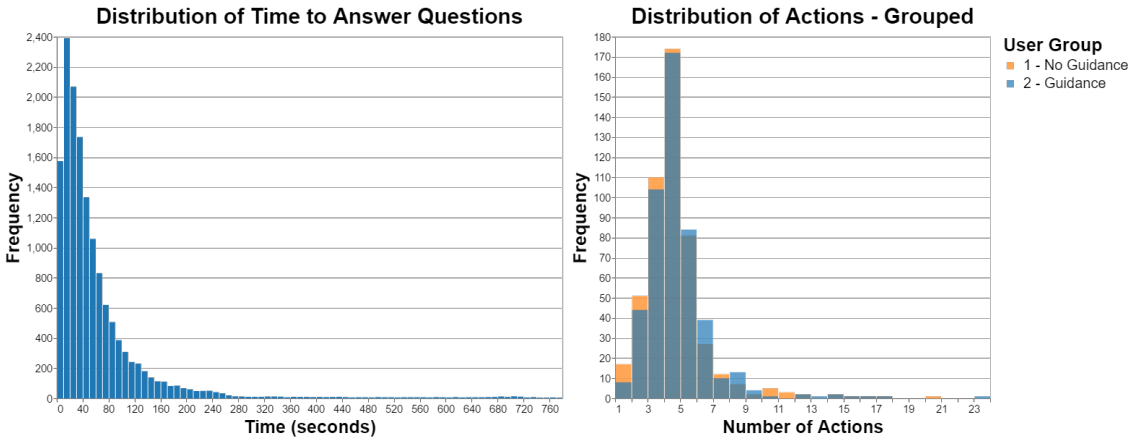
5.8 User Study

To apply our models and system, we conducted an IRB approved (Y17TO37149) user study in which participants were presented with our system and asked to answer a series of questions related to the data. The participants were recruited through Amazon Mechanical Turk, presented with a tutorial of the system, and provided a monetary reward of $1 upon completion of the study.

5.8.1 User Groups

We divided the participants into two groups. The first group consisted of 50 users that received no guidance, while the second group consisted of 50 users that received guidance, in the form of suggestions, for their last 5 questions. All users received questions in a random order.

We identified 100 users as being a sufficient amount for this user study because after running through the two groups of 50 users, we analyzed the data and were able to find insights immediately. Additionally, the 100 users mimics a real world scenario in which a large cohort of users may not be available. By showing an effective approach on a relatively small cohort we can show that our model is able to learn even with limited data and can extrapolate to what could be learned from a large population that has a vast amount of information and pathways.



(a) (b)

Figure 5.5: (a) The distribution of the time spent answering each question by all users. (b) A layered distribution graph showing the distribution of actions for both user groups.

5.8.2 Overall Statistics

On average users scored 78% over all ten questions and took 64.82 (±62.53) seconds while performing 4.24 (±2.13) actions per question. Users that correctly answered a question performed 4.32 (±2.12) actions while users that incorrectly answered a question performed 3.68 (±2.3) actions. Figure 5.5a shows the distribution of time to answer a question where users took about 30 seconds to answer a question. Figure 5.5b shows a layered distribution graph in which the number of actions for each user group is shown.

5.9 Evaluation

To evaluate the effectiveness of our approach on providing personalized guidance to users we will perform two different methods of analysis: (i) we will inspect the Preference Models to identify whether individual user preferences were able to be captured and learned by the models and (ii) we will evaluate the statistical impact that the suggestions had on the users.

5.9.1 Model Inspection

Using the matrix learned by our Preference Model we can examine what was learned about the user. Figure 5.6 shows a heatmap of the model for two different users. In the first row of the heatmap, which is the initial load of the question before any actions are taken, the model learned a larger value for not providing suggestions to user 1 while the opposite occurred for user 2. This indicates that user 1 preferred not to receive suggestions at the beginning of the task while user 2 did. Looking at the entirety of users as a whole, we found that the majority aligned similarly with user 1.

Furthermore, looking at subsequent rows it is easy to identify, based on the checkmarks and different shades of green, moments that the model learned to provide suggestions and moments where it learned to stay dormant. Places at which the shades of green differ significantly indicate that the model learned a definitive decision to make. On the other hand, when the shades of green are close in shade then this indicates that either the model requires more experience to better learn

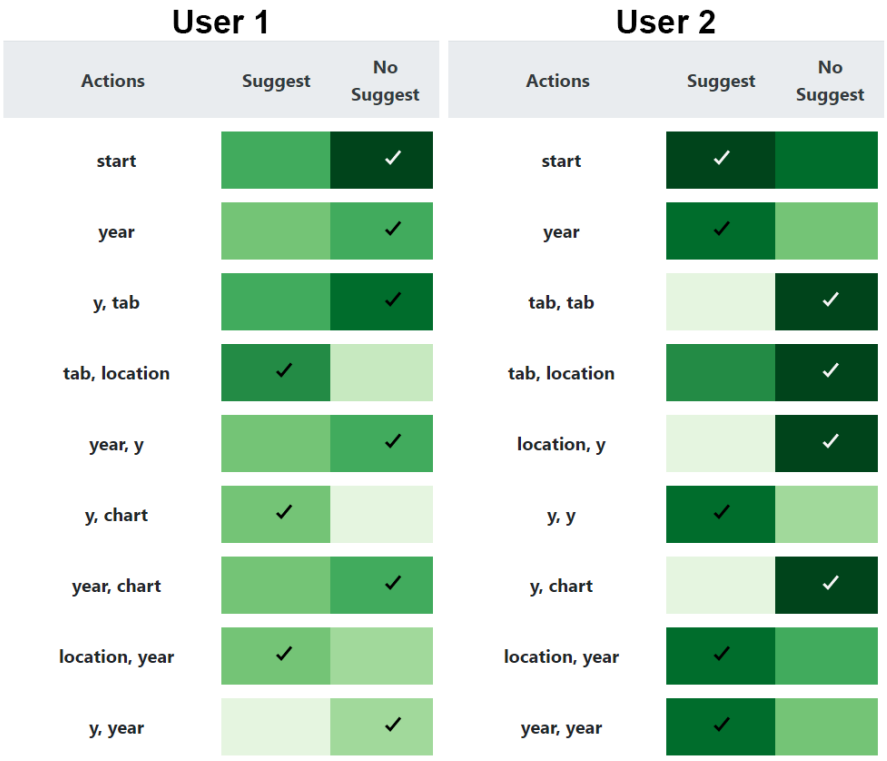


Figure 5.6: A heatmap of the preference model built and trained on two different users. The rows correspond to the last two actions that a user took and the block with the darker shade of green (and checkmark) indicates whether or not the user should be provided guidance by means of presenting a suggestion.

this state or that the user is unsure of which way they prefer the most.

5.9.2 Guidance Impact

To understand the statistical impact that guidance had on users, we analyzed the effect that suggestions had on users and identified two key discoveries when analyzing the first five questions that users saw versus the last five questions that they saw. Because group 2 received suggestions for their last five questions and group 1 did not, we will name these two sets of questions as "pre" and "post" where "pre" corresponds to the first five questions and "post" corresponds to the last five.

For the first key discovery, we found that the groups did not differ significantly between each other when comparing their pre and post average completion times (t-statistic$_{pre}$=0.4918, t-statistic$_{post}$=1.0471). However, comparing all user's pre against all user's post did prove to be significant at an $\alpha = 0.05$ (t-statistic=2.31,p-value=0.0229). This is something that we thought could be expected due to the user learning the system, even though we attempted to mitigate it with a tutorial at the beginning of the session. However, through further analysis we found that this was not the case. Looking at only group 1's pre versus post, we found that their time difference was not significant(t-statistic=1.06,p-value=0.295); and looking at only group 2's pre versus post, we found that their time differences were significant (t-statistic=2.66,p-value=0.011). Therefore, we can rule out any learning effect in the post section and can conclude that group 2's that the suggestions made group 2 fast enough that they were able to shift the entire sample population of users as

Table 5.1: Pre vs Post Number of Questions Correctly Answered

	Pre	*Post*
Group 1	193	198
Group 2	180	209

a whole to be significantly different.

For the second key discovery, we found similar results as before: proportionally, users in group 2 had a higher accuracy rate in the post section than in the pre section (χ^2=9.08,p-value=0.0026). Looking at group 1 with the same parameters shows that they proportionally did not exhibit the same behavior (χ^2=0.188,p-value=0.665). This can be seen in Table 5.1 where a McNemar χ^2-test of the number of questions answered correctly by both groups shows a significant difference (χ^2=69.25,pvalue<0.0001).

To validate these discoveries, we ensure that both groups (first 50 users vs last 50 users) are similar. To accomplish this we ran a t-test on the amount of time spent on all ten questions as a whole and found that the two groups were not significantly different at an $\alpha = 0.05$ (t-statistics=0.903,pvalue=0.249). We also ran a t-test for time taken on each question and once again did not find a significant difference between the two groups (t-statistic=1.15,pvalue=0.369). Next, we also ran χ^2-test for the number of actions that it took a user to complete a question and due to the relatively low number of actions taken and thus low variances, we did not find any significant differences (χ^2=0.277,pvalue=0.599). Furthermore, we ran a final test on the accuracy of group 1 versus group 2 and did not also find any

significance (χ^2=0.0058,pvalue=0.939). Using these tests, we can conclude that the two groups of users behaved similarly and that despite some outliers we can treat them as similar users.

With the two key discoveries and confirmed similarity of the user groups, we can conclude that the guidance provided to users significantly improved their performance, in terms of both accuracy and time. We were not able to identify any significance in the number of actions that they took due to the small variance within the data. Also, due to a small sample size with randomized question order we were not able to identify significance on a per question basis.

5.9.3 Suggestion Accuracy

Additionally, we sought to verify if the suggested action chosen by our interaction model was correct. To accomplish this we aggregated each time that a user was presented a suggestion and the associated reward value from the model. Then we split the rewards into the times that the user followed the suggestions versus the times that it did not and ran a t-test on these rewards. With this we found that the rewards were significant at an $\alpha = 0.05$ level such that the suggestions that users followed had a significantly higher reward value than those that users did not (t-statistic=-3.13,p-value=0.002). From this we can gather that when our interaction model was confident in its suggestion (had a high reward value) the user followed it.

We also assessed the effect of the number of suggestions a user received to their accuracy and we found that as users received more suggestions their accuracy

increased (t-statistic=3.88,p-value=0.0002), indicating that the presence of suggestions was of assistance to them.

5.10 Optimize Interface

With **A1** and **A2** accomplished, we will explore how **A3** could similarly be accomplished. First, we will outline the different methods of optimization that could be implemented for a visual analytic system. Then, we will verify that potential avenues of optimization can be identified in our reinforcement learning models using a manual method of inspection. Because the application of these models to automatically optimizing an interface without manually inspection requires substantial research effort we seek for future work in this area to be performed to extend the introduction of reinforcement learning to user interactions that we have shown.

5.10.1 Methods of Optimization

Interface Layout Users utilize visual analytic interfaces for a wide range of tasks and follow different sets of analytic steps to accomplish the same task. Additionally, tasks across domains may require or highlight different controls. Thus, it is necessary to customize the layout of the tool and its controls for each user and the domain/task that they utilize it for.

Predictive Tasks: A user may commonly perform a series of tasks. For instance, a financial analyst may always apply a green color scheme upon activating a specific chart, while a healthcare analyst may always look to a blue color scheme for the same chart. Rather than requiring the user to make the extra click each time, an interface that can predict common tasks that a specific user performs could automatically change the color scheme to the one that the specific user prefers.

Therefore, even though the color scheme cannot be applied as a general rule for all users or cohorts of users, it can be identified for the individual.

Learn Cohorts: While it is typical to personalize to a singular user, it is also possible to use knowledge learned and apply it towards certain cohorts of users. For example, analysts across different domains utilize visual analytic systems differently, focusing on different controls and performing different domain-specific tasks. Because of this, a separate model could be built for each cohort of users to identify the differences. Using these separate models, a new user that has never interacted with the system and is being introduced into a cohort could be provided with a model learned from the overall cohort and through the user's interactions with the system the model would learn and tune itself to match the unique behaviors of this individual user.

Visualization: Users interpret and understand each type of visualization and interface differently [218]. Some may prefer one over another. For example, in some domains a specific visualization, such as a heatmap, may be widely used causing experts within the domain to understand a heatmap over other visualizations. Therefore, being able to identify the visualizations that a user prefers or understands the most as well as the variations of the visualization can ensure that users are properly supported by a tool.

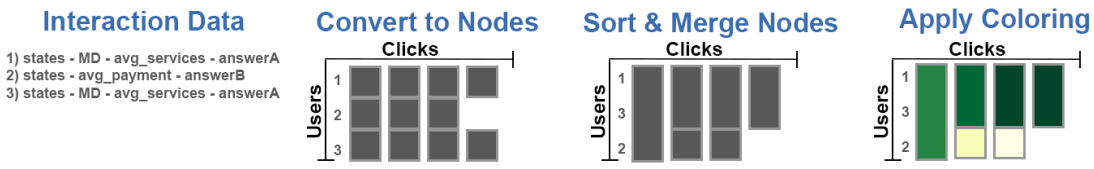
5.10.2 Model Inspection for Optimization

With different methods of optimization defined, we build the same models as previously built using a bot, but this time we use the user interaction data from the user study. Then we inspect these models built from users to identify knowledge that was gained from the reinforcement learning algorithms. The insights into optimization are broken down into three methods: (1) analysis of interaction pathways, (2) user comparison, and (3) difficulty of the assigned task.

Of note: for each of these insights, due to the size of the visualizations caused by the large matrix learned by the Q-Learning model, we extract and choose to only display parts of the visualizations into the figures. These extractions are insights that we gained from an analysis of our models and their visualizations. We refer the reader to the Appendix for the full visualizations.

5.10.3 Path Analysis

Evaluating the interaction pathways that users took can reveal differences between users as well as provide information into the types of interactions that are taken within an interface. To understand these pathways, we built a matrix-like visualization where the rows correspond to the users and the columns correspond to the actions that users took, as shown in Figure 5.7. This allows for being able to follow the path of each user by following along horizontally. Furthermore, we sort the rows of users such that similar users are placed next to each other, allowing for interactions to be grouped together. The sorting is accomplished by adapting the



(a) (b) (c) (d)

Figure 5.7: An example of how we construct our interaction pathway visualization. (a) The interaction data is compiled, (b) a node is placed for each action, (c) the users/rows are sorted and similar actions are merged vertically, (d) a coloring scheme from yellow to green based on reward is applied.

radix sort [219] algorithm to to this task. Through the grouping that results from sorting, the amount of users that took each type of interaction is evident by the size of the node.

Figure 5.8 contains a visualization of the final pathway visualization for the sequences of actions that users took to answer question 9. The color scheme was taken from the reward values learned from the models. In addition, red vertical bars were placed on the rows of users that had answered the question incorrectly.

Analyzing this visualization there are two dominant paths to highlight: the large section of users that correctly answered the question at the top, denoted as "Group A", and the large section of users that answered the question incorrectly towards the bottom, denoted as "Group B". These two sections immediately jump out because of the similarity that exists within a large block of users and the fact that the majority of Group A users answered the question correctly whereas the majority of Group B answered incorrectly.

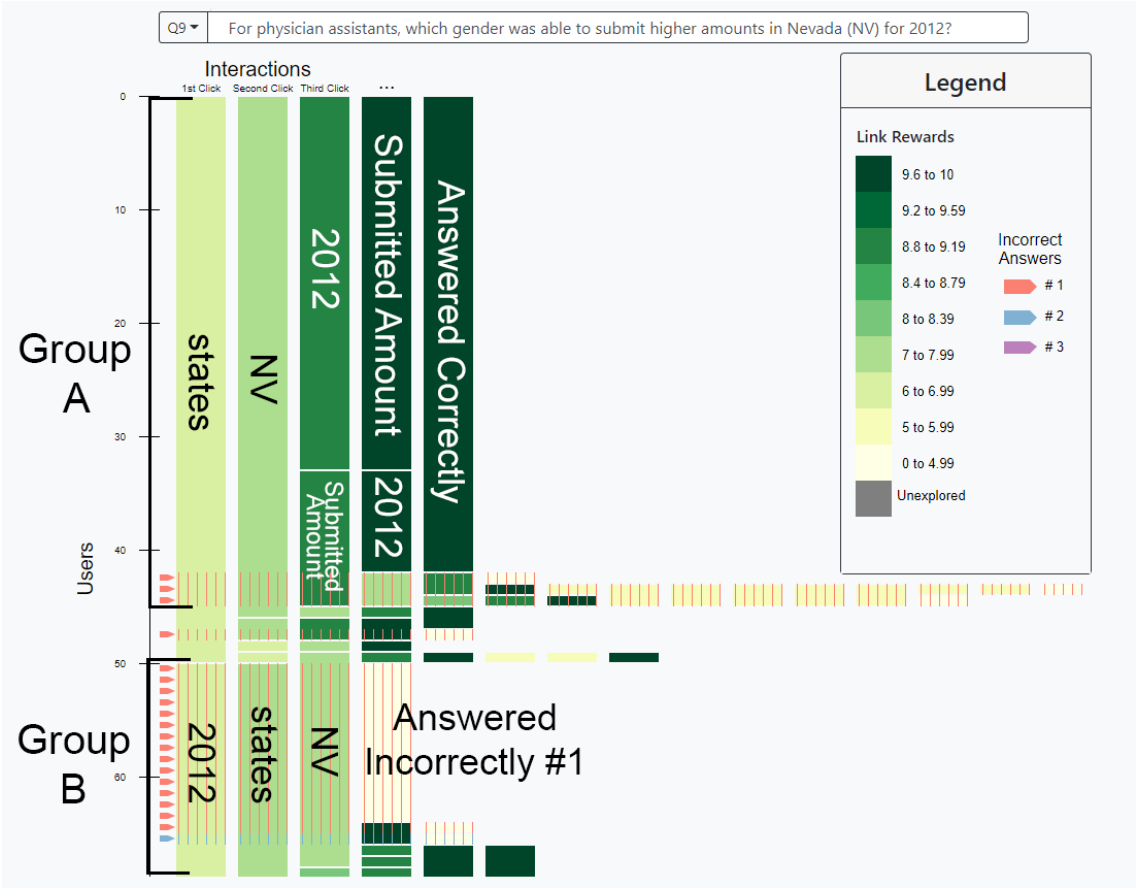


Figure 5.8: A visualization of the interaction pathways for question 9, as built by Figure 5.7. Users in Group B took a subset of the actions taken by users in Group A, presumably forgetting to change to the "Submitted Amount" variable. Additionally, the trained model identified that the Group A actions were more valuable, as evidenced by the darker shades of green for the third column. With this, the data suggest to optimize the interface to ensure that users take the path that leads them towards answering correctly.

As indicated in black text on top of each node, users in Group A took the path [“states”, “NV”, “2012”, “Submitted Amount”, “Answered Correctly”] whereas the majority of the users in Group B took the path [“2012”, “states”, “NV”, “Answered Incorrectly #1”]. The interesting realization between these two paths is that the there exists an intersection in the actions, such that Group B had taken a subset of the moves made by Group A. However, Group B had not taken all of the actions that Group A did, presumably forgetting to switch to the “Submitted Amount” variable. Additionally, the model had identified, without any user data, that the path taken by Group A was more valuable than the path taken by Group B. This can be seen in the third action of the path by comparing the darker shade of green for “2012” against the light green for “NV” within the same column.

Based on this knowledge, it can be learned that it is advantageous for users to follow the path of clicking on the “states” button first rather than first clicking on the year “2012”. Therefore, the data suggests that the interface could be optimized to ensure that this aspect is presented first. A potential layout alteration that could be made would be to place each set of controls vertically such that the controls could be treated as steps, thus forcing users to follow this path for a specific question. While this layout alteration may be most applicable to Amazon Mechanical Turk users, it still shows that our model is able to identify these kinds of insights.

5.10.4 User Comparison

Another important aspect of understanding user interactions is to understand how users differ. Identifying where users took missteps and the reasons behind the different actions that they took is paramount in having a deep and rich understanding of where the interface could be optimized. Common techniques for performing user comparisons consist of using statistical methods in which the amount of time, number of actions, and correctness are taken into account. However, these techniques are limiting in that they are not able to identify the interactions at which users differed. For example, if two cohorts of users both took about the same amount of time and clicks, how did they differ? With this, we show our interaction model applied to this problem on two different comparisons.

5.10.4.1 Correct vs Incorrect

The first type of common analysis that is performed is to compare users that answered a question correctly against those that did not. For this, we utilize question 6 ("Over all degrees and genders, which specialty had the highest submitted payments for 2014 in Kansas (KS)?") and split the users into two groups: those that answered the question correctly (blue) and those that did not (red). A model is trained for each group and then the two models are subtracted to produce a heatmap included within the Appendix. We extracted from the heatmap a single state into Figure 5.9a and a single action over many states into Figure 5.9b. For both of these representations, a more blue cell indicates that the blue group took

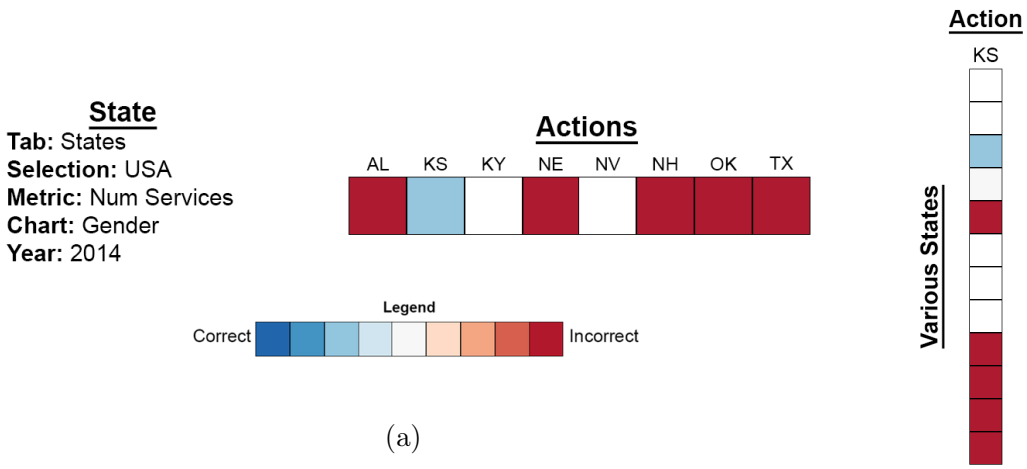


Figure 5.9: A comparison of users that answered a question correctly versus those that did not. (a) For a single state it can be seen that the incorrect users struggled to identify the state Kansas. (b) For the action Kansas (KS), it can be seen that correct users took the action at a singular state while the incorrect users took the action from an assortment of states.

those actions more frequently and a more red cell indicates the same but for the red group. Additionally, the values of the cells correspond to the reward learned by the model.

In Figure 5.9b it can be learned that both sets of users had taken the correct action, but that correct users seemed to have always been in the same state of the system when they had taken it compared to incorrect users that had taken it from a variety of system configurations. To understand why incorrect users had clicked on Kansas from different states, Figure 5.9a shows that for a single state incorrect users had clicked on a wide variety of US states while correct users always clicked

on the correct one. Additionally, the light shade of blue within the figure indicates some of the incorrect users had also clicked on "KS" and that it wasn't limited to only correct users, which would correspond to a dark shade of blue.

The knowledge extracted from this figure provides two helpful insights: (1) incorrect users struggled to identify the state of Kansas and thus the interface should be updated to assist these users that require assistance and (2) some incorrect users had taken the correct action of Kansas and thus further inspection into other states within the model is required to understand where those users had deviated. Both of these insights could be utilized to help optimize the interface, such as displaying the names of the states directly onto the map to make them easier to find for users that are lost.

5.10.4.2 How do correct users differ?

Another interesting comparison that is difficult and sometimes even impossible to perform is to understand how users that correctly answered a question differ. The reason that this task is typically very difficult is because even when a divergence of paths is able to be found, identifying the reason for the divergence requires immense research: did the user diverge because they made a mistake or did they diverge because they had extra knowledge? In the latter case, it is near impossible for standard techniques to make a concrete conclusion as two users may have taken a similar amount of actions, but one of the users may have identified a more intelligent manner to arrive at the answer. For our interaction model, this analysis is simple.

To accomplish this we built a model utilizing only users that correctly answered a question. We then normalized each row (state) of the model such that its reward values would be between 0 and 1, and plotted it as a heatmap. An extraction of a singular row/state within the heatmap for question 4 is shown in Figure 5.10.

Because this heatmap only contains correct users and is not a subtraction of two cohorts as we saw in the correct versus incorrect users, the analysis on this heatmap is different. To evaluate this heatmap, we must look for rows (states) where the variance between actions is high. These variances can reveal the extra knowledge that expert users may possess and thus would provide information into alternate use cases or benefits of a system that were not intended or identified upon its construction. While these types of insights would be more possible and easier to identify within a more complex system where alternate pathways have the potential to be present, we provide a simple example from our user study on a question where the insight was able to be found.

Inspecting Figure 5.10 and the different rewards for each action for the given state, we can see that correct users had clicked on three different variables: "Number of Patients", "Submitted Amount", and "Allowed Amount". The dominant and most rewarding path consisted of "Submitted Amount" which is what the question had asked for. However, other correct users had also taken alternate paths and upon inspection of these variables we found that the user would still arrive at the same answer even if they had taken these wrong paths. This insight reveals to us that the dataset had some commonalities which for this question had allowed for users to take a multitude of paths.

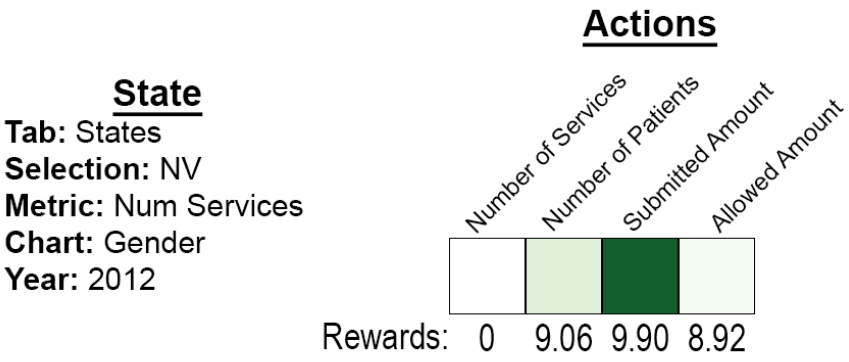


Figure 5.10: A comparison of correct users for question 9 in which for a single state there was variance in the action reward between the users. While the "submitted amount" correctly led users to the answer, it was discovered that the other actions would present a visualization that would also provide users with the correct answer.

5.11 Discussion

With this work, we have laid the foundation for a more intelligent and in depth learning of user interactions. The model and framework that we have outlined utilizes reinforcement learning to automatically model and learn, thereby removing the previous limitation of experts constructing rules or users analyzing their own interactions via historical timelines. This newly applied technique on interaction data allows for analyses and applications that would not be possible with standard techniques, thus opening a new window into future work to build upon this work.

5.12 Limitations

Due to the new introduction of reinforcement learning to visualization, we acknowledge that our approach contains limitations but we recognize this as being a necessary first step. First, the task provided to users was simple in nature compared to many visualization tasks that users perform on a daily basis. This constraint in the difficulty was necessary in order to evaluate the effectiveness of our modelling without needing to account for the noise and challenges associated with complex tasks. However, because the theory and concept of reinforcement learning have been proven to be successful for this task, scaling to more complex tasks can easily be accomplished.

Second, while one of the exciting applications of such automatic models is towards exploration tasks where a user enters a visualization system without a

predefined question or task, our user study did not tackle such tasks. We believe that this type of research is interesting and should be a long-term goal, but there exists work that is still required to be completed before we can reach this goal and accomplish such tasks.

Chapter 6: Conclusion

Within this dissertation, we presented methods for operating on irregular events within healthcare across three tasks: (i) predicting the probability of a future event occurring, (ii) modeling the processes that create events, and (iii) summarizing large event datasets. Our approaches do not bias the dataset by altering it, but rather keep it in its original irregular format. Through operating on the raw dataset, we have shown a boost in prediction performance metrics and more representative visualizations.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[2] Agency for Healthcare Research and MD Quality, Rockville. Clinical classifications software (CCS) for ICD-9-CM, 2018.

[3] Agency for Healthcare Research and MD Quality, Rockville. Beta Clinical Classifications Software (CCS) for ICD-10-CM/PCS, 2018.

[4] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[5] Centers for Disease Control, Prevention, et al. National hospital ambulatory medical care survey: 2011, 2011.

[6] P Rui and K Kang. National hospital ambulatory medical care survey: 2015. *Emergency Department Summary Tables Available from: https://www. cdc. gov/nchs/data/nhamcs/web_ tables/2015_ ed_ web_ tables. pdf. Accessed Sept*, 13, 2018.

[7] Prevention CfDCa. National hospital ambulatory medical care survey: 2016. *Emergency Department Summary Tables*, 2016.

[8] Wullianallur Raghupathi and Viju Raghupathi. Big data analytics in healthcare: promise and potential. *Health information science and systems*, 2(1):3, 2014.

[9] Sean P Keehan, Gigi A Cuckler, John A Poisal, Andrea M Sisko, Sheila D Smith, Andrew J Madison, Kathryn E Rennie, Jacqueline A Fiore, and James C Hardesty. National health expenditure projections, 2019–28: Expected rebound in prices drives rising spending growth: National health expenditure projections for the period 2019–2028. *Health Affairs*, 39(4):704–714, 2020.

[10] John Cristian Borges Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*, 2017.

[11] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

[12] Graham W Taylor and Geoffrey E Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *Proceedings of the 26th annual international conference on machine learning*, pages 1025–1032. ACM, 2009.

[13] Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

[14] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[15] Sharat C Prasad and Piyush Prasad. Deep recurrent neural networks for time series prediction. *arXiv preprint arXiv:1407.5949*, 2014.

[16] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[17] Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*, 2017.

[18] Andreas Eckner. A framework for the analysis of unevenly spaced time series data. *Preprint. Available at: http://www. eckner. com/papers/unevenly_ spaced_ time_ series_ analysis*, 2012.

[19] Andreas Eckner. Algorithms for unevenly-spaced time series: Moving averages and other rolling operators. In *Working Paper*, 2012.

[20] Gilles Zumbach and Ulrich Müller. Operators on inhomogeneous time series. *International Journal of Theoretical and Applied Finance*, 4(01):147–177, 2001.

[21] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

[22] Volker Tresp and Thomas Briegel. A solution for missing data in recurrent neural networks with an application to blood glucose prediction. *Advances in Neural Information Processing Systems*, pages 971–977, 1998.

[23] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016.

[24] Egil Martinsson. Wtte-rnn : Weibull time to event recurrent neural network. Master's thesis, Chalmers University Of Technology, 2016.

[25] Waqas Javed, Bryan McDonnel, and Niklas Elmqvist. Graphical perception of multiple time series. *IEEE transactions on visualization and computer graphics*, 16(6):927–934, 2010.

[26] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, and Christian Tominski. Visual methods for analyzing time-oriented data. *IEEE transactions on visualization and computer graphics*, 14(1):47–60, 2008.

[27] Ming C Hao, Umeshwar Dayal, Daniel A Keim, and Tobias Schreck. Multi-resolution techniques for visual exploration of large time-series data. In *EUROVIS 2007*, pages 27–34, 2007.

[28] Taowei David Wang, Amol Deshpande, and Ben Shneiderman. A temporal pattern search algorithm for personal history event visualization. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):799–812, 2012.

[29] Milos Krstajic, Enrico Bertini, and Daniel Keim. Cloudlines: Compact display of event episodes in multiple time-series. *IEEE transactions on visualization and computer graphics*, 17(12):2432–2439, 2011.

[30] Marc Weber, Marc Alexa, and Wolfgang Müller. Visualizing time-series on spirals. In *Infovis*, volume 1, pages 7–14, 2001.

[31] James Walker, Rita Borgo, and Mark W Jones. Timenotes: a study on effective chart visualization and interaction techniques for time-series data. *IEEE transactions on visualization and computer graphics*, 22(1):549–558, 2016.

[32] Katerina Vrotsou, Jimmy Johansson, and Matthew Cooper. Activitree: interactive visual exploration of sequences in event-based data using graph similarity. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):945–952, 2009.

[33] Jessica Lin, Eamonn Keogh, Wei Li, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107, 2007.

[34] Jessica Lin, Eamonn Keogh, and Stefano Lonardi. Visualizing and discovering non-trivial patterns in large time series databases. *Information visualization*, 4(2):61–82, 2005.

[35] Nitin Kumar, Venkata Nishanth Lolla, Eamonn Keogh, Stefano Lonardi, Chotirat Ann Ratanamahatana, and Li Wei. Time-series bitmaps: a practical visualization tool for working with large time series databases. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 531–535. SIAM, 2005.

[36] Ronald R Yager. A new approach to the summarization of data. *Information Sciences*, 28(1):69–86, 1982.

[37] Janusz Kacprzyk and Ronald R Yager. Linguistic summaries of data using fuzzy logic. *International Journal of General System*, 30(2):133–154, 2001.

[38] Janusz Kacprzyk, Ronald R Yager, and S Zadrożny. A fuzzy logic based approach to linguistic summaries of databases. *International Journal of Applied Mathematics and Computer Science*, 10(4):813–834, 2000.

[39] Janusz Kacprzyk, Anna Wilbik, and S Zadrożny. Linguistic summarization of time series using a fuzzy quantifier driven aggregation. *Fuzzy Sets and Systems*, 159(12):1485–1499, 2008.

[40] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman. Lifelines: visualizing personal histories. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 221–227. ACM, 1996.

[41] Catherine Plaisant, Rich Mushlin, Aaron Snyder, Jia Li, Daniel Heller, and Ben Shneiderman. Lifelines: using visualization to enhance navigation and analysis of patient records. In *Proceedings of the AMIA Symposium*, page 76. American Medical Informatics Association, 1998.

[42] Taowei David Wang, Catherine Plaisant, Ben Shneiderman, Neil Spring, David Roseman, Greg Marchand, Vikramjit Mukherjee, and Mark Smith. Temporal summaries: Supporting temporal categorical searching, aggregation and comparison. *IEEE transactions on visualization and computer graphics*, 15(6), 2009.

[43] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal Event Sequence Simplification. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2227–2236, December 2013.

[44] Mike Sips, Patrick Köthur, Andrea Unger, Hans-Christian Hege, and Doris Dransch. A visual analytics approach to multiscale exploration of environmental time series. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2899–2907, 2012.

[45] Cong Xie, Wei Chen, Xinxin Huang, Yueqi Hu, Scott Barlowe, and Jing Yang. Vaet: A visual analytics approach for e-transactions time-series. *IEEE transactions on visualization and computer graphics*, 20(12):1743–1752, 2014.

[46] Conglei Shi, Weiwei Cui, Shixia Liu, Panpan Xu, Wei Chen, and Huamin Qu. Rankexplorer: Visualization of ranking changes in large time series data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2669–2678, 2012.

[47] Susan Havre, Beth Hetzler, and Lucy Nowell. Themeriver: Visualizing theme changes over time. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 115–123. IEEE, 2000.

[48] Aleks Aris, Ben Shneiderman, Catherine Plaisant, Galit Shmueli, and Wolfgang Jank. Representing unevenly-spaced time series data for visualization and interactive exploration. *Human-Computer Interaction-INTERACT 2005*, pages 835–846, 2005.

[49] Myoungsu Cho, Bohyoung Kim, Hee-Joon Bae, and Jinwook Seo. Stroscope: Multi-scale visualization of irregularly measured time-series data. *IEEE transactions on visualization and computer graphics*, 20(5):808–821, 2014.

[50] Michael Schulz and Karl Stattegger. Spectrum: Spectral analysis of unevenly spaced paleoclimatic time series. *Computers & Geosciences*, 23(9):929–945, 1997.

[51] Web-based injury statistics query and reporting system (wisqars), Jul 2020.

[52] Key substance use and mental health indicators in the united states: Results from the 2018 national survey on drug use and health (hhs publication no. pep19-5068, nsduh series h-54), Jul 2018.

[53] Janet Kemp and Robert Bossarte. *Suicide data report: 2012*. Department of Veterans Affairs, Mental Health Services, Suicide Prevention . . . , 2013.

[54] Department of Veterans Affairs et al. Suicide among veterans and other americans 2001-2014. *Washington, DC: Office of Suicide Prevention*, 2016.

[55] World Health Organization: Mental Health and Substance Use. Preventing suicide: A global imperative. *World Health Organization*, 2014.

[56] Brian K Ahmedani, Gregory E Simon, Christine Stewart, Arne Beck, Beth E Waitzfelder, Rebecca Rossom, Frances Lynch, Ashli Owen-Smith, Enid M Hunkeler, Ursula Whiteside, et al. Health care contacts in the year before suicide death. *Journal of general internal medicine*, 29(6):870–877, 2014.

[57] Diane L Frankenfield, Penelope M Keyl, Andrea Gielen, Lawrence S Wissow, Lisa Werthamer, and Susan P Baker. Adolescent patients—healthy or hurting?: Missed opportunities to screen for suicide risk in the primary care setting. *Archives of pediatrics & adolescent medicine*, 154(2):162–168, 2000.

[58] Jason B Luoma, Catherine E Martin, and Jane L Pearson. Contact with mental health and primary care providers before suicide: a review of the evidence. *American Journal of Psychiatry*, 159(6):909–916, 2002.

[59] Anna Pearson, Pooja Saini, Damian Da Cruz, Caroline Miles, David While, Nicola Swinson, Alyson Williams, Jenny Shaw, Louis Appleby, and Navneet Kapur. Primary care contact prior to suicide in individuals with mental illness. *British Journal of General Practice*, 59(568):825–832, 2009.

[60] Ayal Schaffer, Mark Sinyor, Paul Kurdyak, Simone Vigod, Jitender Sareen, Catherine Reis, Diane Green, James Bolton, Anne Rhodes, Sophie Grigoriadis, et al. Population-based analysis of health care contacts among suicide decedents: identifying opportunities for more targeted suicide prevention strategies. *World Psychiatry*, 15(2):135–145, 2016.

[61] Eve K Mościcki. Identification of suicide risk factors using epidemiologic studies. *Psychiatric Clinics of North America*, 20(3):499–517, 1997.

[62] Ronald C Kessler, Robert M Bossarte, Alex Luedtke, Alan M Zaslavsky, and Jose R Zubizarreta. Suicide prediction models: a critical review of recent research with recommendations for the way forward. *Molecular psychiatry*, pages 1–12, 2019.

[63] John F McCarthy, Robert M Bossarte, Ira R Katz, Caitlin Thompson, Janet Kemp, Claire M Hannemann, Christopher Nielson, and Michael Schoenbaum. Predictive modeling and concentration of the risk of suicide: implications for preventive interventions in the us department of veterans affairs. *American journal of public health*, 105(9):1935–1942, 2015.

[64] Ronald C Kessler, Irving Hwang, Claire A Hoffmire, John F McCarthy, Maria V Petukhova, Anthony J Rosellini, Nancy A Sampson, Alexandra L Schneider, Paul A Bradley, Ira R Katz, et al. Developing a practical suicide risk prediction model for targeting high-risk patients in the veterans health administration. *International journal of methods in psychiatric research*, 26(3):e1575, 2017.

[65] Joseph C Franklin, Jessica D Ribeiro, Kathryn R Fox, Kate H Bentley, Evan M Kleiman, Xieyining Huang, Katherine M Musacchio, Adam C Jaroszewski, Bernard P Chang, and Matthew K Nock. Risk factors for suicidal thoughts and behaviors: a meta-analysis of 50 years of research. *Psychological bulletin*, 143(2):187, 2017.

[66] JD Ribeiro, JC Franklin, Kathryn Rebecca Fox, KH Bentley, Evan M Kleiman, BP Chang, and Matthew K Nock. Self-injurious thoughts and behaviors as risk factors for future suicide ideation, attempts, and death: a meta-analysis of longitudinal studies. *Psychological medicine*, 46(2):225–236, 2016.

[67] Colin G Walsh, Jessica D Ribeiro, and Joseph C Franklin. Predicting risk of suicide attempts over time through machine learning. *Clinical Psychological Science*, 5(3):457–469, 2017.

[68] Jan Horsky, Elizabeth A Drucker, and Harley Z Ramelson. Accuracy and completeness of clinical coding using icd-10 for ambulatory visits. In *AMIA Annual Symposium Proceedings*, volume 2017, page 912. American Medical Informatics Association, 2017.

[69] Icd - icd-10-cm - international classification of diseases,(icd-10-cm/pcs transition, Nov 2015.

[70] Surveillance case definitions, Dec 2015.

[71] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.

[72] Bum Chul Kwon, Min-Je Choi, Joanne Taery Kim, Edward Choi, Young Bin Kim, Soonwook Kwon, Jimeng Sun, and Jaegul Choo. Retainvis: Visual analytics with interpretable and interactive recurrent neural networks on electronic medical records. *IEEE transactions on visualization and computer graphics*, 25(1):299–309, 2018.

[73] Tim Rosenflanz and Ryan Caldwell. Retain-keras: Keras reimplementation of retain. https://github.com/Optum/retain-keras, 2020.

[74] Phuoc Nguyen, Truyen Tran, Nilmini Wickramasinghe, and Svetha Venkatesh. Deepr: a convolutional net for medical records. *IEEE journal of biomedical and health informatics*, 21(1):22–30, 2016.

[75] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[76] Filip Dabek and Jesus J Caban. A neural network based model for predicting psychological conditions. In *International conference on brain informatics and health*, pages 252–261. Springer, 2015.

[77] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[78] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[79] word2vec, Jul 2013.

[80] Manaal Faruqui, Jesse Dodge, Sujay K Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. Retrofitting word vectors to semantic lexicons. *arXiv preprint arXiv:1411.4166*, 2014.

[81] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. Deep learning for answer sentence selection. *arXiv preprint arXiv:1412.1632*, 2014.

[82] Samuel Gershman and Joshua B Tenenbaum. Phrase similarity in humans and machines. In *CogSci*. Citeseer, 2015.

[83] Tom Kenter and Maarten De Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1411–1420, 2015.

[84] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[85] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018.

[86] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[87] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[88] Pandu Nayak. Understanding searches better than ever before. *Google Blog, October*, 25, 2019.

[89] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[90] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[91] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[92] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[93] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences, 2014.

[94] Peng Wang, Jiaming Xu, Bo Xu, Chenglin Liu, Heng Zhang, Fangyuan Wang, and Hongwei Hao. Semantic clustering and convolutional neural network for

short text categorization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 352–357, Beijing, China, July 2015. Association for Computational Linguistics.

[95] Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks, 2014.

[96] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[97] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

[98] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[99] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.

[100] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3), 2015.

[101] Edward Choi, Cao Xiao, Walter Stewart, and Jimeng Sun. Mime: Multilevel medical embedding of electronic health records for predictive healthcare. In *Advances in neural information processing systems*, pages 4547–4557, 2018.

[102] Jonas Kemp, Alvin Rajkomar, and Andrew M Dai. Improved patient classification with language model pretraining over clinical notes. *arXiv preprint arXiv:1909.03039*, 2019.

[103] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[104] Yikuan Li, Shishir Rao, Jose Roberto Ayala Solares, Abdelaali Hassaine, Dexter Canoy, Yajie Zhu, Kazem Rahimi, and Gholamreza Salimi-Khorshidi. Behrt: Transformer for electronic health records, 2019.

[105] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn:

Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[106] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[107] Emily Alsentzer, John R Murphy, Willie Boag, Wei-Hung Weng, Di Jin, Tristan Naumann, and Matthew McDermott. Publicly available clinical bert embeddings. *arXiv preprint arXiv:1904.03323*, 2019.

[108] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[109] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[110] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.

[111] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *arXiv preprint arXiv:2002.12327*, 2020.

[112] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[113] Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies, 2016.

[114] Christopher J Kelly, Alan Karthikesalingam, Mustafa Suleyman, Greg Corrado, and Dominic King. Key challenges for delivering clinical impact with artificial intelligence. *BMC medicine*, 17(1):195, 2019.

[115] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):91–100, 2016.

[116] Josua Krause, Aritra Dasgupta, Jordan Swartz, Yindalon Aphinyanaphongs, and Enrico Bertini. A workflow for visual diagnostics of binary classifiers using instance-level explanations. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 162–172. IEEE, 2017.

[117] Filip Dabek, Peter Hoover, and Jesus J Caban. Addressing the need for raw-valued dataset exploration in neural network visualization. *Interpreting, Explaining and Visualizing Deep Learning Workshop at NIPS 2017*, 2017.

[118] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[119] Emily Reif, Ann Yuan, Martin Wattenberg, Fernanda B Viegas, Andy Coenen, Adam Pearce, and Been Kim. Visualizing and measuring the geometry of bert. In *Advances in Neural Information Processing Systems*, pages 8594–8603, 2019.

[120] Jesse Vig. A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*, 2019.

[121] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, 2017.

[122] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Jonathan Reynolds, Alexander Melnikov, Natalia Lunova, and Orion Reblitz-Richardson. Pytorch captum. https://github.com/pytorch/captum, 2019.

[123] James C Overholser, Abby Braden, and Lesa Dieter. Understanding suicide risk: Identification of high-risk groups during high-risk times. *Journal of clinical psychology*, 68(3):349–361, 2012.

[124] Chandru Ravindran, Sybil W Morley, Brady M Stephens, Ian H Stanley, and Mark A Reger. Association of suicide risk with transition to civilian life among us military service members. *JAMA network open*, 3(9):e2016261–e2016261, 2020.

[125] David A Wooff and Jillian M Anderson. Time-weighted multi-touch attribution and channel relevance in the customer journey to online purchase. *Journal of Statistical Theory and Practice*, 9(2):227–249, 2015.

[126] D. Gotz and H. Stavropoulos. DecisionFlow: Visual Analytics for High-Dimensional Temporal Event Sequence Data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1783–1792, December 2014.

[127] Adam Perer and David Gotz. Visualizations to support patient-clinician communication of care. In *ACM CHI Workshop on Patient-Clinician Communication*, Paris, France, 2013.

[128] Wolfgang Aigner, Silvia Miksch, Bettina Thurnher, and Stefan Biffl. Planninglines: novel glyphs for representing temporal uncertainties and their evaluation. In *Information Visualisation, 2005. Proceedings. Ninth International Conference on*, pages 457–463. IEEE, 2005.

[129] Steve B Cousins and Michael G Kahn. The visual display of temporal information. *Artificial intelligence in medicine*, 3(6):341–357, 1991.

[130] Ragnar Bade, Stefan Schlechtweg, and Silvia Miksch. Connecting time-oriented data and information to a coherent interactive visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 105–112. ACM, 2004.

[131] Beverly L Harrison, Russell Owen, and Ronald M Baecker. Timelines: an interactive system for the collection and visualization of temporal data. In *Graphics Interface*, pages 141–141. Citeseer, 1994.

[132] Michael Burch, Fabian Beck, and Stephan Diehl. Timeline trees: visualizing sequences of transactions in information hierarchies. In *Proceedings of the working conference on Advanced visual interfaces*, pages 75–82. ACM, 2008.

[133] Doantam Phan, Andreas Paepcke, and Terry Winograd. Progressive multiples for communication-minded visualization. In *Proceedings of Graphics Interface 2007*, pages 225–232. ACM, 2007.

[134] Taowei David Wang, Catherine Plaisant, Alexander J Quinn, Roman Stanchak, Shawn Murphy, and Ben Shneiderman. Aligning temporal data by sentinel events: discovering patterns in electronic health records. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 457–466. ACM, 2008.

[135] Emanuel Zgraggen, Steven M Drucker, and Danyel Fisher. (s|qu)eries: Visual regular expressions for querying and exploring event sequences. *Proceedings of CHI 2015*, 2015.

[136] Jerry Alan Fails, Amy Karlson, Layla Shahamat, and Ben Shneiderman. A visual interface for multivariate temporal data: Finding patterns of events across multiple histories. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 167–174. IEEE, 2006.

[137] Krist Wongsuphasawat and Ben Shneiderman. Finding comparable temporal categorical records: A similarity measure with an interactive visualization. In *Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on*, pages 27–34. IEEE, 2009.

[138] Katerina Vrotsou, Kajsa Ellegård, and Matthew Cooper. Everyday life discoveries: Mining and visualizing activity patterns in social science diary data. In *Information Visualization, 2007. IV'07. 11th International Conference*, pages 130–138. IEEE, 2007.

[139] Krist Wongsuphasawat, Catherine Plaisant, Meirav Taieb-Maimon, and Ben Shneiderman. Querying event sequences by exact match or similarity search: Design and empirical evaluation. *Interacting with computers*, 24(2):55–68, 2012.

[140] Jian Zhao, Zhicheng Liu, Mira Dontcheva, Aaron Hertzmann, and Alan Wilson. Matrixwave: Visual comparison of event sequence data. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 259–268. ACM, 2015.

[141] Krist Wongsuphasawat and David Gotz. Outflow: Visualizing patient flow by symptoms and outcome. *IEEE VisWeek Workshop on Visual Analytics in Healthcare, Providence, Rhode Island, USA*, pages 25–28, 2011.

[142] Florent Masseglia, Maguelonne Teisseire, and Pascal Poncelet. Sequential pattern mining. *Encyclopedia of Data Warehousing and Mining*, pages 1028–1032, 2005.

[143] Zhicheng Liu, Yang Wang, Mira Dontcheva, Matthew Hoffman, Seth Walker, and Alan Wilson. Patterns and sequences: Interactive exploration of clickstreams to understand common visitor paths. *IEEE Transactions on Visualization and Computer Graphics*, 23(01), 2016.

[144] Bum Chul Kwon, Janu Verma, and Adam Perer. Peekquence: Visual analytics for event sequence data. In *ACM SIGKDD Workshop on Interactive Data Exploration and Analytics (IDEA 2016)*, 2016.

[145] Adam Perer and Fei Wang. Frequence: Interactive Mining and Visualization of Temporal Frequent Event Sequences. In *Proceedings of the 19th International Conference on Intelligent User Interfaces*, IUI '14, pages 153–162, New York, NY, USA, 2014. ACM.

[146] Jian Pei, Jiawei Han, B. Mortazavi-Asl, Jianyong Wang, H. Pinto, Qiming Chen, U. Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, November 2004.

[147] Tao Li. *Event Mining: Algorithms and Applications*, volume 38. CRC Press, 2015.

[148] Mohammed J Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2):31–60, 2001.

[149] Theophano Mitsa. *Temporal data mining*. CRC Press, 2010.

[150] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[151] Greg Hamerly and Charles Elkan. Learning the k in a> means. *Advances in neural information processing systems*, 16:281, 2004.

[152] Duc Truong Pham, Stefan S Dimov, and CD Nguyen. Selection of k in k-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 219(1):103–119, 2005.

[153] Mark Ming-Tso Chiang and Boris Mirkin. Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads. *Journal of classification*, 27(1):3–40, 2010.

[154] James Dougherty, Ron Kohavi, Mehran Sahami, et al. Supervised and unsupervised discretization of continuous features. In *Machine learning: proceedings of the twelfth international conference*, volume 12, pages 194–202, 1995.

[155] P. Riehmann, M. Hanfler, and B. Froehlich. Interactive Sankey diagrams. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*, pages 233–240, October 2005.

[156] Takayuki Itoh, Hiroki Takakura, Atsushi Sawada, and Koji Koyamada. Hierarchical visualization of network intrusion detection data. *IEEE Computer Graphics and Applications*, 26(2):40–47, 2006.

[157] Kwan-Liu Ma. Cyber security through visualization. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation-Volume 60*, pages 3–7. Australian Computer Society, Inc., 2006.

[158] Juhnyoung Lee, Mark Podlaseck, Edith Schonberg, and Robert Hoch. Visualization and analysis of clickstream data of online stores for understanding web merchandising. In *Applications of Data Mining to Electronic Commerce*, pages 59–84. Springer, 2001.

[159] Harry Hochheiser and Ben Shneiderman. Coordinating overviews and detail views of www log data. In *Workshop on New Paradigms in Information Visualization and Manipulation (NPIVM 2000), ACM Press*, 2000.

[160] Juhnyoung Lee and Mark Podlaseck. Visualization and analysis of clickstream data of online stores with a parallel coordinate system. In *International Conference on Electronic Commerce and Web Technologies*, pages 145–154. Springer, 2000.

[161] Ed Huai-hsin Chi. Improving web usability through visualization. *IEEE Internet Computing*, 6(2):64–71, 2002.

[162] Igor V Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. Visualization of navigation patterns on a web site using model-based clustering. In *KDD*, page 280, 2000.

[163] N. Kadivar, V. Chen, D. Dunsmuir, E. Lee, C. Qian, J. Dill, C. Shaw, and R. Woodbury. Capturing and supporting the analysis process. In *IEEE Symposium on Visual Analytics Science and Technology, 2009. VAST 2009*, pages 131–138, October 2009.

[164] Wesley Willett, Jeffrey Heer, and Maneesh Agrawala. Scented widgets: Improving navigation cues with embedded visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1129–1136, 2007.

[165] Marc Streit, Hans-Jorg Schulz, Alexander Lex, Dieter Schmalstieg, and Heidrun Schumann. Model-driven design for the visual analysis of heterogeneous data. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):998–1010, 2012.

[166] Gary M Olson, James D Herbsleb, and Henry H Rueter. Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Human-Computer Interaction*, 9(4):427–472, 1994.

[167] Wenwen Dou, Dong Hyun Jeong, F. Stukes, W. Ribarsky, H.R. Lipford, and R. Chang. Recovering Reasoning Processes from User Interactions. *IEEE Computer Graphics and Applications*, 29(3):52–61, May 2009.

[168] Magdalini Eirinaki and Michalis Vazirgiannis. Web Mining for Web Personalization. *ACM Trans. Internet Technol.*, 3(1):1–27, February 2003.

[169] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of international conference on intelligence analysis*, volume 5, pages 2–4, 2005.

[170] S. Kandel, A. Paepcke, J.M. Hellerstein, and J. Heer. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, December 2012.

[171] T.M. Green, R. Maciejewski, and S. DiPaola. ALIDA: Using machine learning for intent discernment in visual analytics interfaces. In *2010 IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 223–224, October 2010.

[172] Alan J Munro, Kristina Höök, and David Benyon. *Social navigation of information space*. Springer Science & Business Media, 2012.

[173] Ionathan Grudin. History and focus. *interaction*, 1994.

[174] JE Allen, Curry I Guinn, and E Horvtz. Mixed-initiative interaction. *IEEE Intelligent Systems and their Applications*, 14(5):14–23, 1999.

[175] Wil MP Van der Aalst, Boudewijn F van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and Anton JMM Weijters. Workflow mining: a survey of issues and approaches. *Data & knowledge engineering*, 47(2):237–267, 2003.

[176] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[177] Mi Feng, Cheng Deng, Evan M Peck, and Lane Harrison. Hindsight: Encouraging exploration through direct encoding of personal interaction history. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):351–360, 2017.

[178] Adam Perer and Ben Shneiderman. Systematic yet flexible discovery: guiding domain experts through exploratory data analysis. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 109–118. ACM, 2008.

[179] Sung-Hee Kim, Zhihua Dong, Hanjun Xian, Benjavan Upatising, and Ji Soo Yi. Does an eye tracker tell the truth about visualizations?: findings while investigating visualizations for decision making. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2421–2430, 2012.

[180] Gennady Andrienko, Nathaliya Andrienko, Peter Bak, Daniel Keim, Slava Kisilevich, and Stefan Wrobel. A conceptual framework and taxonomy of techniques for analyzing movement. *Journal of Visual Languages & Computing*, 22(3):213–232, 2011.

[181] Mathias Pohl, Markus Schmitt, and Stephan Diehl. Comparing the readability of graph layouts using eyetracking and task-oriented analysis. In *Computational Aesthetics*, pages 49–56, 2009.

[182] RJ Jacob and Keith S Karn. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. *Mind*, 2(3):4, 2003.

[183] Hua Guo, Steven R Gomez, Caroline Ziemkiewicz, and David H Laidlaw. A case study using visualization interaction logs and insight metrics to understand how analysts arrive at insights. *IEEE transactions on visualization and computer graphics*, 22(1):51–60, 2016.

[184] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE transactions on visualization and computer graphics*, 14(6), 2008.

[185] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[186] D. Gotz and M.X. Zhou. Characterizing users' visual analytic activity for insight provenance. In *IEEE Symposium on Visual Analytics Science and Technology, 2008. VAST '08*, pages 123–130, October 2008.

[187] E.D. Ragan, A. Endert, J. Sanyal, and Jian Chen. Characterizing Provenance in Visualization and Data Analysis: An Organizational Framework of Provenance Types and Purposes. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):31–40, January 2016.

[188] Ling Xiao, J. Gerth, and P. Hanrahan. Enhancing Visual Analysis of Network Traffic Using a Knowledge Representation. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 107–114, October 2006.

[189] E.T. Brown, A. Ottley, H. Zhao, Quan Lin, R. Souvenir, A. Endert, and R. Chang. Finding waldo: Learning about users from their interactions. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1663–1672, December 2014.

[190] Tiffany CK Kwok, Eugene Yujun Fu, Erin You Wu, Michael Xuelin Huang, Grace Ngai, and Hong-Va Leong. Every little movement has a meaning of its own: Using past mouse movements to predict the next interaction. In *23rd International Conference on Intelligent User Interfaces*, pages 397–401. ACM, 2018.

[191] Eugene Yujun Fu, Tiffany CK Kwok, Erin You Wu, Hong Va Leong, Grace Ngai, and Stephen CF Chan. Your mouse reveals your next activity: Towards predicting user intention from mouse interaction. In *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*, volume 1, pages 869–874. IEEE, 2017.

[192] David Gotz and Zhen Wen. Behavior-driven visualization recommendation. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, pages 315–324, New York, NY, USA, 2009. ACM.

[193] L. Bavoil, S.P. Callahan, P.J. Crossno, J. Freire, C.E. Scheidegger, C.T. Silva, and H.T. Vo. VisTrails: enabling interactive multiple-view visualizations. In *IEEE Visualization, 2005. VIS 05*, pages 135–142, October 2005.

[194] Filip Dabek and Jesus J Caban. A grammar-based approach for modeling user interactions and generating suggestions during the data exploration process. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):41–50, 2017.

[195] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[196] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[197] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[198] Aniruddh Raghu, Matthieu Komorowski, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. Continuous state-space models for optimal sepsis treatment-a deep reinforcement learning approach. *arXiv preprint arXiv:1705.08422*, 2017.

[199] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[200] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[201] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[202] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[203] Nick De Firmian. *Modern Chess Openings: MCO-15*. Random House Incorporated, 2008.

[204] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[205] Murray Campbell. Knowledge discovery in deep blue. *Communications of the ACM*, 42(11):65–67, 1999.

[206] Eric Horvitz, Andy Jacobs, and David Hovel. Attention-sensitive Alerting. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pages 305–313, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[207] Brian Whitworth. Polite computing. *Behaviour & Information Technology*, 24(5):353–363, September 2005.

[208] Cheri Speier and Michael G Morris. The influence of query interface design on decision-making performance. *MIS quarterly*, pages 397–423, 2003.

[209] David Roldán-Álvarez, Estefanía Martín, Manuel García-Herranz, and Pablo A Haya. Mind the gap: impact on learnability of user interface design of authoring tools for teachers. *International Journal of Human-Computer Studies*, 94:18–34, 2016.

[210] Jörn Hurtienne and Luciënne Blessing. Design for intuitive use-testing image schema theory for user interface design. In *16 th International Conference on Engineering Design*. Citeseer, 2007.

[211] Robert JK Jacob. Eye tracking in advanced interface design. *Virtual environments and advanced interface design*, pages 258–288, 1995.

[212] Claudia Ehmke and Stephanie Wilson. Identifying web usability problems from eye-tracking data. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it-Volume 1*, pages 119–128. British Computer Society, 2007.

[213] Andreas Sonderegger and Juergen Sauer. The influence of design aesthetics in usability testing: Effects on user performance and perceived usability. *Applied ergonomics*, 41(3):403–410, 2010.

[214] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[215] Tableau. http://www.tableau.com/.

[216] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

[217] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[218] Raquel O Prates, Clarisse S de Souza, and Simone DJ Barbosa. Methods and tools: a method for evaluating the communicability of user interfaces. *interactions*, 7(1):31–38, 2000.

[219] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.