DISSERTATION APPROVAL SHEET

Title of Dissertation: INTERPRETABLE DEEP LEARNING MODELS FOR ELECTRONIC HEALTH RECORDS

Name of Candidate: Peichang Shi
pshi1@umbc.edu
Doctor of Philosophy, 2022
Graduate Program: Information Systems

Dissertation and Abstract Approved:

Aryya Gangopadhyay

Aryya Gangopadhyay
gangopad@umbc.edu
Professor
Information Systems
11/29/2022 | 3:45:01 PM EST

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

Curriculum Vitae

Name: Peichang Shi

Degree and date to be conferred: PhD, 2022

Collegiate institutions attended:

2014-2022 University of Maryland,Baltimore County,PhD in Information Systems

2003-2006 Wright State University, MS in Applied Statistics

2000-2002 Florida Institute of Technology,MS in Computer Information Systems

1991-1995 Jimei University, BS in Fisheries biology

Professional Publications

1.**Conference/journal papers:**

1).Shi, P., Gangopadhyay, A., Yu, P. (2022, June). LIVE: A Local Interpretable model-agnostic Visualizations and Explanations. In 2022 IEEE 10th International Conference on Healthcare Informatics (ICHI) (pp. 245-254). IEEE.

2).Wu, X., Wang, H. Y., Shi, P., Sun, R., Wang, X., Luo, Z., ... Wang, Y. (2022). Long short-term memory model–A deep learning approach for medical data with irregularity in cancer predication with tumor markers. Computers in Biology and Medicine, 144, 105362

3).Shi, P., Gangopadhyay, A., Owens, C., Blunt, B., Grogan, C. (2019, December). A hybrid model using LSTM and decision tree for mortality prediction and its application in provider performance evaluation. In 2019 IEEE International Conference on Big Data (Big Data) (pp. 2773-2781). IEEE.

4).Shi, P., Song, Q., Patwardhan, J., Zhang, Z., Wang, J., Gangopadhyay, A. (2019, September). A hybrid algorithm for mineral dust detection using satellite data. In 2019 15th International Conference on eScience (eScience) (pp. 39-46). IEEE.

2. **Oral presentation /Posters**

1) Shi, P.,Zhang,S.,Yu,P., Shang,Q.,Xiong,X. "An AI Approach for Real-time Provider Performance Evaluation Based on Federated Learning" 2022 Annual Meeting, American Public Health Association , 2022

2) Shi, P., Yu, P., Shang, Q., Levine, E., Zhao, J "Application of Bagging Approach in Program Evaluation with Heterogenous Groups." 2022 Annual Research Meeting. AcademyHealth, 2022

3) Shi, P., Yu, P., Shang, Q., Levine, E., Zhao, J."Application of NLP Word Embedding Model with Irregular Time Series and Sparse Healthcare Data " 2022 Annual Research Meeting. AcademyHealth, 2022

Professional positions: Senior Lead Data Scientist, Booz Allen Hamilton

ABSTRACT

Title of dissertation: INTERPRETABLE DEEP LEARNING MODELS FOR ELECTRONIC HEALTH RECORDS

Peichang Shi, Doctor of Philosophy, 2022

Dissertation directed by: Professor Aryya Gangopadhyay
Department of Information Systems

Analysis of healthcare data could help reduce cost, improve patient outcomes and understand the best practices related to diseases.However,with rapid increase of massive amounts of health-related data, such as Electronic Health Records (EHRs),high dimensionality and large sample size have become challenges for traditional statistical approaches.

Deep learning models have been proved to be powerful tools in computer vision and machine learning in healthcare.However,despite their superior performance to traditional statistical methods,it remains challenging to understand their inner mechanism due to the black box effects.

A variety of interpretability algorithms have been developed to help explain the deep learning models.However due to the trade off between model accuracy and complexity, the current interpretability algorithms have lower performance compared to original deep learning models, which cause some concern for high stakes in healthcare. Also,most of interpretability algorithms focus on correlation interpreta-

tion, highly correlated features may lead to biased causal inference, which may be more important in healthcare.

In this dissertation paper,we proposed a new ensemble approach for deep learning interpretation,Local surrogate Interpretable model-agnostic Visualizations and Explanations (LIVE), where we assumed all the predictions from deep learning model form a mixture of a finite number of Gaussian distributions with unknown parameters.We applied ensemble trees to obtain the mixing coefficients. The rule sets from the trees were used to build an interpretable model through randomized experimental design for interpretation.

Our LIVE algorithm was validated using different types of datasets (image and structured datasets) with different deep learning model structures. Our experiments showed that LIVE algorithm could not only help improve the model accuracy, but also provide visual interpretation.

INTERPRETABLE DEEP LEARNING MODELS FOR ELECTRONIC HEALTH RECORDS

by

Peichang Shi

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
2022

Advisory Committee:
Professor Aryya Gangopadhyay, Chair/Advisor
Professor Jianwu Wang
Professor Sanjay Purushotham
Professor Maryam Rahnemoonfar
Professor Yaacov Yesha

Acknowledgments

I would like to express my deepest appreciation to all the people who have made this thesis possible.

First and foremost I'd like to thank my advisor, Professor Aryya Gangopadhyay for his great patience and feedback. He has always made himself available for help. It has been a pleasure to work with him.

I can not have undertaken this journey without my committee members, Dr. Jianwu Wang, Dr. Sanjay Purushotham, Dr. Maryam Rahnemoonfar and Dr. Yaacov Yesha, who have provided invaluable advice on my dissertation research.

I also have to say a special word of thanks to the fantastic team I have been fortunate to work with over the years at Booz Allen Hamilton, especially Dr. Ping Yu for his encouragement and inspiration.

Finally I would like to express my deepest gratitude to my family who have always stood by me, especially my wife, Xiaochun Mou and my three kids, Chelsea, Ryan and George. Without their tremendous understanding in the past few years, it would be impossible for me to complete my study.

Table of Contents

List of Tables vii

List of Figures viii

1 Introduction 1
- 1.1 Healthcare and healthcare data . . . 3
- 1.2 Deep learning in healthcare . . . 3
- 1.3 What is interpretability . . . 5
- 1.4 What is causal interpretability . . . 6
- 1.5 Current interpretability methods and limitations . . . 7
  - 1.5.1 Major interpretation methods . . . 7
  - 1.5.2 Limitations and challenges for current interpretability algorithms . . . 8
- 1.6 Problem statement . . . 9
- 1.7 Statement of purpose . . . 9
- 1.8 Proposed approaches . . . 10
- 1.9 Contributions . . . 11

2 Background 13
- 2.1 Healthcare data . . . 13
  - 2.1.1 Properties of EHRs . . . 13
  - 2.1.2 Challenges in EHR data . . . 15
- 2.2 Deep learning in healthcare . . . 20
  - 2.2.1 Major techniques in deep learning . . . 21
    - 2.2.1.1 Autoencoders and deep autoencoders . . . 22
    - 2.2.1.2 Recurrent neural network(RNN) . . . 24
    - 2.2.1.3 Restricted Boltzmann Machine (RBM) . . . 25
    - 2.2.1.4 Convolutional Neural Networks(CNNs) . . . 26
    - 2.2.1.5 Transfer learning . . . 27
    - 2.2.1.6 Ensemble learning . . . 28
- 2.3 Interpretability . . . 29
  - 2.3.1 Interpretable models . . . 29

2.3.2 Types of interpretability methods . . . 30
2.3.2.1 Before (pre-model), during (in-model), or after (post-model) . . . 31
2.3.2.2 Intrinsic vs. post hoc . . . 32
2.3.2.3 Model-specific vs. model-agnostic . . . 32
2.3.2.4 Local or global . . . 32
2.3.3 Intrinsic approaches . . . 33
2.3.3.1 Adding interpretability constraints . . . 33
2.3.3.2 Mimic learning . . . 34
2.3.3.3 Attention mechanism . . . 35
2.3.4 Post hoc approaches . . . 35
2.3.4.1 Model-Specific explanation methods . . . 36
2.3.4.2 Model-Agnostic explanation methods . . . 36
2.3.5 Interpretability evaluation . . . 39
2.3.6 Limitations and challenges of current interpretability methods 40
2.4 Decision tree and deep learning models . . . 41
2.4.1 Neural networks to decision trees . . . 42
2.4.2 Decision trees to neural networks . . . 42
2.4.3 Mixing deep learning models and decision trees . . . 42
2.5 Deep learning models and mixture models . . . 43
2.5.1 Gaussian process in deep learning models . . . 43
2.5.2 Mixture density network . . . 44

3 Methodology 46
3.1 Decision tree and regression tree . . . 46
3.2 Clustering . . . 48
3.3 Bagging and random subspace . . . 49
3.4 Deep learning models . . . 49
3.4.1 CNN model . . . 49
3.4.2 Transfer learning models . . . 50
3.4.3 LSTM model . . . 51
3.4.4 Evaluation metrics . . . 51
3.5 Problem definition . . . 52
3.5.1 Mixture distributions . . . 52
3.5.2 Interpretability . . . 54
3.6 Proposed method . . . 54
3.6.1 LIVE algorithm description . . . 57
3.6.2 Properties of LIVE algorithm . . . 59
3.6.2.1 LIVE algorithm is consistent . . . 59
3.6.2.2 LIVE algorithm is an ensemble approach . . . 60
3.6.2.3 LIVE algorithm is causal interpretable . . . 70
3.6.2.4 Complexity of LIVE algorithm . . . 75
3.6.3 Comparison to other interpretability methods . . . 76

4 Results 78
4.1 Results for image data . . . 78
4.1.0.1 Vingroup dataset . . . 78
4.1.0.2 Pneumonia x ray dataset . . . 79
4.1.1 Model structures . . . 79
4.1.1.1 CNN model . . . 80
4.1.1.2 Transfer learning model . . . 82
4.1.2 Experiment design . . . 83
4.1.2.1 Impact of image sizes . . . 84
4.1.2.2 Impact of number of epochs . . . 85
4.1.3 Results of impact of epoches . . . 85
4.1.3.1 Impact of number of decision trees . . . 86
4.1.4 Results of impact of image size . . . 87
4.1.5 Performance comparison . . . 88
4.1.6 Results for transfer learning . . . 89
4.1.7 Performance of LIVE algorithm with smaller image size . . . 91
4.2 Results for visualization . . . 93
4.3 Results for Structured dataset . . . 97
4.3.1 Data source . . . 97
4.3.2 Challenges of this dataset . . . 99
4.3.2.1 Irregular time series . . . 99
4.3.2.2 Hierarchical variables . . . 100
4.3.2.3 Word embedding . . . 103
4.3.3 Models for diabetes dataset . . . 103
4.3.4 Model results for structured data . . . 104
4.3.5 Interpretation and visualization for diabetes dataset . . . 106

5 Conclusion and future work 107
5.1 Conclusion . . . 107
5.2 Future work . . . 111

A Programs codes 114
A.1 Data sources . . . 114
A.2 Codes for Vinbig data . . . 114
A.2.1 CNN model for Vinbig . . . 115
A.2.2 VGG16 model for Vinbig data . . . 122
A.2.3 Visualization codes using LIME,SHAP and GRAD CAM . . . 128
A.2.4 Codes for LIVE algorithm . . . 131
A.2.4.1 SAS code for LASSO regression model . . . 141
A.2.4.2 Visualization for LIVE algorithm . . . 142
A.3 Codes for pneumonia data . . . 144
A.3.1 SAS code for data creation . . . 144
A.3.2 Codes for LIVE algorithm and visualization . . . 151
A.4 Codes for diabetes data . . . 152
A.4.1 data preprocessing . . . 152

A.4.2 Deep learning model and LIVE method 157

Bibliography 170

List of Tables

3.1 Sample node information with features in the rule path 70
3.2 Differences between LIVE algorithm and 4 other interpretability methods 77

4.1 Some transfer learning models [1] 83
4.2 Impact of different image sizes for Pneumonia data 87
4.3 Impact of image sizes for Vingbig data 88
4.4 Performance with smaller image size for Vinbig data 92
4.5 Performance with smaller image size for Pneumonia data 92
4.6 Region of interest for Figure 4.11 95
4.7 Region of interest for Figure 4.12 95
4.8 Some regression coefficients for interpretation using Vinbig data . . . 98
4.9 Frequency of irregular hospital visits 100
4.10 Sample diagnosis codes 101
4.11 Performance comparison between LSTM and LIVE 105

List of Figures

1.1 Deep learning in healthcare [2] . . . 4

2.1 Types of deep learning methods. . . . 23
2.2 Types of interpretability methods [3] . . . 31
2.3 Neural network and Gaussian process . . . 44
2.4 Mixture density network, figures from [4] . . . 45

3.1 Regression tree . . . 47
3.2 Decision tree for clustering . . . 48
3.3 Mixture distribution . . . 53
3.4 LIVE algorithm . . . 56
3.5 Random trees . . . 59
3.6 LIVE is consistent . . . 60
3.7 Two curves without overlapping . . . 63
3.8 Two curves without overlapping . . . 64
3.9 Induction step for proof of LIVE . . . 67
3.10 ICU model structure in literature [5] . . . 70
3.11 LIVE algorithm is interpretable (1) . . . 71
3.12 LIVE algorithm is interpretable (2) . . . 71

4.1 Sample images from Vingroup big data . . . 79
4.2 Sample image from Pneumonia data . . . 80
4.3 CNN model structure . . . 81
4.4 VGG 16 model structure [6] . . . 84
4.5 Pneumonia images with different sizes . . . 85
4.6 Impact of number of epochs . . . 86
4.7 Impact of number of decision trees . . . 86
4.8 Performance for Pneumonia data CNN vs LIVE . . . 89
4.9 Performance for Vinbig data CNN vs LIVE . . . 90
4.10 Performance for VGG16 vs LIVE . . . 91
4.11 Visualization for Vinbig data 1 . . . 94
4.12 Visualization for Vinbig data 2 . . . 96
4.13 Frequency of diagnosis code . . . 102

4.14 LSTM model with embedding layer 104
4.15 Impact of number of decision trees for Diabetes data 105
4.16 Interpretation and Visualization for diabetes data 106

Chapter 1: Introduction

Due to exponential growth in healthcare data, more and more deep learning models have been used in healthcare study with high performance compared to traditional approaches.Lack of transparency of deep learning models are barriers for the wide adoption in medical research and clinical decision making. A variety of traditional interpretable models have been developed where a second model is built to explain the deep learning model.There is a trade off between explainability and accuracy, thus generally the current explainable models which target to mimic the deep learning models have less accuracy. This low performance in interpretable models may lead to biased decisions especially in high staking healthcare industry. Also, traditional interpretable machine learning approaches focus on the association instead of the causality, little has been done for causality inference analysis, which tried to answer questions why the model made such decisions.

The aim of this thesis is to develop an innovative approach with both high interpretablilty and high accuracy compared to deep learning models alone.The new model is supposed to have statistical theory support to show its superior performance and better interpretability with less correlations among the features. The proposed model is evaluated on a variety of real world datasets.

In Chapter 1, we talk about the motivations why we seek to develop a model with both high interpretability and high accuracy. We briefly cover deep learning models in healthcare, some challenges between accuracy, interpretability and causal inference, then we discuss our research questions,research approaches and our contributions.

The rest of this dissertation is organized as follows:

In Chapter 2, we introduce the definition of EHRs, the properties of EHRs, and some challenges in EHRs. We will also discuss the benefits of application of deep learning models in healthcare where we will briefly talk about some major techniques in deep learning. Next we will go over the definition of interpretability and major explanation methods. Also, we will review some specific approaches about decision trees and deep learning models, which are highly related to our proposed algorithm.

In Chapter 3, we propose our LIVE algorithm and describe the implementation details. We also talk about some properties of our LIVE algorithm, such as reducing bias, consistency and visualization etc. We would also explain why our model reduce correlations and improve interpretability.

In Chapter 4, we mainly talk about some experiments to validate our LIVE algorithm. We use different datasets including both images data and structured data, and different deep learning models to show that our LIVE algorithm could consistently improve the model performance compared to deep learning methods alone.

Finally in Chapter 5, we review some limitations of LIVE algorithm and discuss some future work.

1.1 Healthcare and healthcare data

Healthcare is an important sector in US, which costs approximately 18% of the GDP. Rising health care cost is a serious issue. According to McKinsey, healthcare expenses have been growing consistently for 20 years, and reache 17.6 percent of GDP. McKinsey's estimates also show that US health care expenses are nearly 600 billion dollars more than expected and 30% of the healthcare spending is considered a waste [7,8]. The waste includes failure of care delivery, failure of care coordination and over treatment etc [9–11]. Today, approximately 30% of the world's data volume is being generated by the healthcare industry. By 2025, the compound annual growth rate of data for healthcare will reach 36% [12]. The fast-growing and tremendous amount of health data has far exceeded the ability of traditional statistics. We need powerful and versatile machine learning techniques to deal with this big data.

1.2 Deep learning in healthcare

Deep learning models have been received much attention and successfully applied in a variety of task such as classification and prediction, especially in image analysis and natural language processing. In healthcare, medical imaging plays an important role in the disease diagnosis from cancer and appendicitis to stroke and heart disease due to its noninvasive and painless properties. Each year medical imaging saves millions of lives. Images are one of the most challenging sources of data in healthcare. Current image analysis is totally depending on well trained

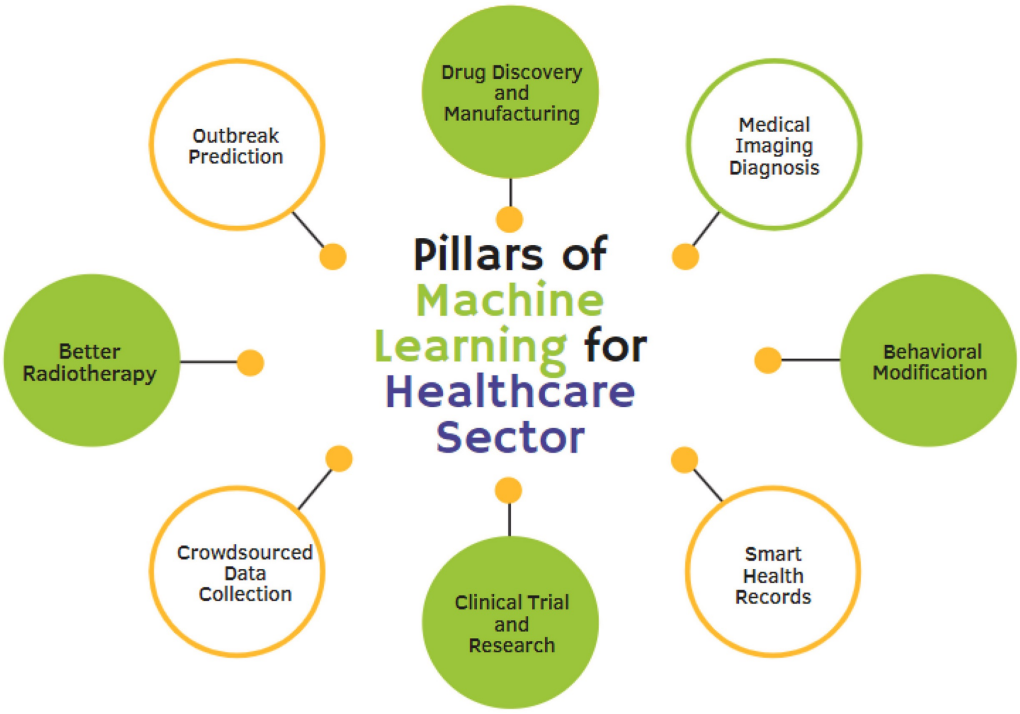

Figure 1.1: Deep learning in healthcare [2]

professionals, which is limited by the complexity of the image, the large variations across different interpreters, and fatigue. Even the best practicing doctor could lead to 25% false positives on image readings, which may produce unnecessary invasive procedures and follow-up scans and increase the cost and stress for patients [13,14]. These incorrect detection of disease may increase treatment costs and reduce survival rates. Artificial intelligence algorithms have been shown to help radiologists improve the speed and accuracy of interpreting medical images. These deep learning techniques include Convolutional Neural Network (CNN), Recurrent neural Network (RNN), Long Short term Memory (LSTM) models, Extreme Learning Model (ELM), Generative Adversarial Networks (GANs) etc. Some research on medical image classification by CNN has achieved better performance than human experts.For

example, CheXNet, a CNN algorithm trained with more than 100,000 chest X-rays, achieved a better performance than the average performance of four radiologists. Moreover, Kermany et al. proposed a transfer learning system to classify 108,309 Optical Coherence Tomography (OCT) images, and the weighted average error was equal to the average performance of 6 human experts [15, 16].

However, one major concern for the deep learning is its black box effect: algorithms could identify an image object as abnormal but cannot explain why it was determined due to the complex architecture of the deep learning models.In healthcare industry, we would not totally rely on machine learning models for medical diagnosis, unless we know how it works [17]. The basic common characteristic of the deep learning models is its multiple layers of hidden neurons, which greatly introduces complexity to the model. For example, VGGNet is a CNN architecture and the 1st runner up of ILSVR2014 in the classification task.Many image classification models are built on top of this architecture. VGGNet has 144 millions of parameters. It is hard to understand the model without help of interpretable models [18].

1.3 What is interpretability

The model interpretability could be defined as the ability to explain the meaning of the model in a way human could understand. Sometimes interpretability and explainability are used interchangeably [19], however it is also reported interpretability and explainability are two different concepts. Interpretability is the intrinsic propriety of the model, for example, in multivariate linear regression model, the

coefficients could be used directly to measure the expected output for different input data. However explainability is kind of interface between the non interpretable models and human understanding, for example, if the black box model is hard for people to understand, you may through another method to help understand the inner mechanics [20]

Interpretability is crucial for several reasons. If researchers don't understand how a model works, they can have difficulty transferring learning into a broader knowledge base. Similarly, interpretability is essential for guarding against embedded bias or debugging an algorithm. It also helps researchers measure the effects of trade-offs in a model. More broadly, as algorithms play an increasingly important role in society, understanding precisely how they come up with their answers will only become more critical [21].

It is also some requirement of regulation. A key example is the new European General Data Protection Regulation, which enforces that the explanation of algorithmic decision must be provided with the data subject [22].

1.4 What is causal interpretability

Causal inference is used to determine the causal relationship between two variables.The traditional interpretability methods focus on correlation analysis and not the causal inference.A causal interpretable model can help us understand the real cause of the decision made by machines. The causal relationship is to study how changing one variable will cause a change in another variable. However traditional

interpretable machine learning models are used for general relationship, where causal relationship can not be verified through interpretable machine learning techniques [23].

1.5 Current interpretability methods and limitations

1.5.1 Major interpretation methods

There are a quite few of review papers which describe a variety of approaches for interpretation methods. Monlnar [24] provides a taxonomy of the approaches for explaining black models, which include intrinsic or post hoc based on when the interpretation happens during the training or after training. Model specific or model agnostic interpretation is based on whether the method is limited to specific model or for any machine learning models. The interpretability could be also local or global where the local interpretation targets the individual prediction, while the global interpretation is for entire model.

One widely popular method is SHapley Additive exPlanations (SHAP) based on cooperative game theory, it is a global agnostic model intepreation. SHAP value could be used to measure the contribution of each feature among the coalition. It is a good way to show how input features contribute to the outcomes either at individual level or population average level [24].

1.5.2 Limitations and challenges for current interpretability algorithms

The interpretablity algorithms have evolved quickly and still suffer some limitations:

1. Accuracy of interpretation models

   One notable limitation is that most of the interpretation algorithms are trying to approximate the black box model, thus they could not precisely catch the inner mechanism of the models. One example is to use tree model to explain the deep learning model using MNIST(Modified National Institute of Standards and Technology) database, the tree model has accuracy of 96.7% compared to 99.2% using deep learning model alone [25]. With simple structure of decision tree, it may be hard to account for the complexity of the deep learning model. Rudin argued that even the explanation model has 90% agreement with deep learning model, the left 10% may still be questionable about trust explanations, especially in high stake decision healthcare area [26].

2. Causal inference due to highly correlation

   Most of current interpretable techniques focus on correlation identification, not causation. Though the interpretation may suggest the relative feature importance in the model, it is not sufficient to draw causal inference from these findings. The challenges for causal inference in observational data include both confounding factors and selection bias. The confounding occurs when

extraneous causes are missing in the model, which is difficult to avoid unless we have substantial domain knowledge to know in advance. The selection bias is due to sampling issue, where the samples are not appropriately selected, which could be avoided through careful experiment design. [27]. Another issue is multicollinearity among features, which could also affect the causal inference. It is already known that SHAP value may serve good tool for feature correlation study, but not be used for causal analysis directly, since during SHAP value calculation, it used all the features regardless their correlations. [24].

1.6 Problem statement

Research indicates that most traditional interpretability models have lower performance than the original deep learning models, also the interpretation is about correlation analysis, and not causal inference due to highly correlated features. All these may lead to biased interpretation.

1.7 Statement of purpose

The purpose of this study was to explore a novel research that could combine the accuracy from deep learning models with statistical casual analysis.To shed light on the problem, the following key research questions will be addressed:

1. Could we develop an interpretation algorithm which is interpretable without sacrificing the accuracy of original deep learning models?

2. If so, could the algorithm reduce biased interpretation due to highly correlated features so that it could reveal the potential rational of a model making the decisions ? Does it answer the following questions, like, why the image is predicted as cancer? Which regions make the algorithm think this is a cancer image?

3. Could this algorithm work for different datasets with different model structures, in another word, is this algorithm model agnostic? Is it a local or global interpretation?

4. Could this algorithm help visual interpretation? How is it compared to other visual interpretation methods, such as SHAP etc.

1.8 Proposed approaches

We propose the following proposed approaches to address the above questions, :

To improve the deep learning model performance, we will try to identify the limitation of the current deep learning models,which assume the predictions from deep models have independent Gaussian distributions, however for many applications, the distributions of predictions are not simple Gaussian distributed,mixture distributions could be more flexible and appropriate for the data. We plan to propose an ensemble approach, where decision trees are used to calculate the mixing coefficients for the mixture distribution. Finally, general linear regression model is built to interpret the outputs from our ensemble approach.

Unlike many other interpretation algorithms, which target to interpret the predictions from deep learning models,our proposed algorithm is composed of black box algorithm and white box algorithms. There, the black box algorithm provides the model predictions, the white box algorithms are decision trees and linear regression models, which are used for interpretation. Our interpretable model is general linear regression model, which is based on the rules from decision trees. Our target interpretation outcome is the newly predictions from the mixture distributions. The current gold-standard approach for causal inference is randomized experimentation [28].We can have our randomized experiment design matrices through random subspace approach based on the rule sets from the decision trees so that we can estimate the contribution of each feature and answer questions, like what the model looks like if the specific feature is missing, thus we could reveal the potential causal relationships based on our linear regression model.

We will evaluate our algorithm using different datasets including both images data and structured data with different model structures to show our algorithm is a model agnostic approach. We will also demonstrate visual interpretation based on model coefficients and compare to other visual interpretation techniques.Finally we would report our evaluation of our model performance using Area Under the Curve(AUC), precision, recall and accuracy.

1.9 Contributions

Our major contributions lie in the following areas:

1. We proposed a new frame work for interpretability algorithm, which is an ensemble approach with black box model and white box model. This algorithm is not trying to interpret the predictions from the black box directly, but to explain the results from the ensemble approach.

2. This algorithm could reduce prediction bias, and provide comparable model accuracy as the black box model or even with slight better performance.

3. This algorithm is transparent and potential causal interpretable.Linear regression model is to explain the contribution of each feature based on rule sets from decision trees, which could make the interpretation more transparent.We incorporate game theory into our model design, so that we can also study the potential causal relationships after we deal with the high correlation in our model.

4. Our LIVE algorithm could be for any machine learning models and different types of data, either images or structured data.

5. LIVE algorithm could provide visual explanation and identification of region of interest.

Chapter 2: Background

2.1 Healthcare data

Health information technology, especially EHRs, has been proved to be an efficient way to improve clinical quality and reduce healthcare costs through descriptive statistics, prediction modeling etc [29]. Over the years, EHR data have been used to improve care, increase patient engagement, perform quality improvement, build shared models and standardization across institutions, enable public health surveillance and intervention, and facilitate personalized care and decision-making [30–32].

2.1.1 Properties of EHRs

However, with rapid development of information technology, EHR data has entered the big data era with the following common properties:

1. Volume:The volume of worldwide healthcare data in 2012 was 500 petabytes, which was estimated to grow to 25,000 petabyte, a fiftyfold increase from 2012 to 2020 [8].

2. Variety: Variety refers to the forms of the data. EHRs include structured, unstructured and semi-structured data.Structured data include patient's name,

physician's name, hospital name and address, and treatment reimbursement codes etc. Unstructured or semi-structured data include e-mails, photos, videos, audios, and other health related data such as hospital medical reports, physician's notes, paper prescriptions, medical images and radiograph films [8, 33, 34].

The number of data sources in healthcare is also growing. The Internet of Things and other technology development produced more data sources such as various wearable devices and sensors, and even smart phone applications.

3. Velocity: Most healthcare data are static, such as patient visits, hospital admissions, however there are still many real time data, for example, patient vital signs in the ICU must be updated in real time and displayed immediately [35, 36].

4. Veracity (uncertainty of data): The veracity means whether the data are consistent. Different data sources have different credibility and reliability. Due to the varieties of data sources, missing or incomplete information increases the risk of veracity [8, 33, 34, 37].

5. Values: It refers to how the big data could be valuable to the patients and clinicians, which could contribute to the following areas: identifying at-risk patients, tracking clinical outcomes, performance measurement and management, clinical decision making at the point of care, length of stay prediction, hospital readmission prediction etc [38].

The most common radiology types include plain X-ray, computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), and ultrasound imaging. X-rays (radiographs) are the most common and widely available diagnostic imaging technique. Even if you also need more sophisticated tests, you will probably get an x-ray first.However, these images are often rendered in grayscale [39].

2.1.2 Challenges in EHR data

These unique features of big data differentiate EHRs from the traditional datasets and significantly challenge the traditional statistical approaches. Though the traditional epidemiology approaches are well established in healthcare data analytics, they provide relatively less desirable performance compared to machine learning approaches. For example, simple regression models require such assumptions like multivariate normality, no multicollinearity and auto-correlation, however due to the complexity of the healthcare data, those assumptions may not hold. This requires us to develop more adaptive and robust models [34]. Kruse listed nine categories of challenges such as data structure, security, data standardization, data storage and transfers, lack of skill of data analysts, inaccuracies in data, regulatory compliance, and real-time analysis [30, 40–42], all these could impede the progress of EHR data analysis.

1. Irregularity

   Irregularity is one of the tough characteristics of EHR data. Irregularity is

due to the fact that EHR data are recorded only when patients visit the hospitals. As a result, patients' EHR data may have irregular samples or patient length variability. The irregular samples means that sequential observations are collected at different times, and the time between two consecutive observations may vary. The patient variability means the number of observations in each data sequence is limited and the duration they span may vary a lot from patient to patient. How to better account for the temporality of EHR data is still an important research question [34, 43–45].

(a) **Use of baseline features**

The first method is to only utilize patients' baseline features, which are collected when patients visit the hospital to perform examinations for the first time [46, 47]. Schulze collected patients' baseline features and then applied a Cox regression model to predict their development of Type 2 Diabetes [48]. However with only baseline features, it may not fully utilize the time related features.

(b) **Data transformation**

The EHR data could also be processed by transforming the time dimension into equally spaced observation data, and then some efficient methods (such as linear regression) will be applied directly, but we need to be aware that the transformation method may possibly lead to the sparsity and missing data problems because there could be no observations during certain time windows. Moreover, by dividing longitudinal data into

windows, the model may be less sensitive to capturing short-time feature patterns [34, 46].

(c) Direct use of irregular data

Long short-term memoery(LSTM) model

The LSTM model is a deep learning model, which could incorporate the time spans between consecutive medical features to handle the irregularity [49, 50]. This category of methods demonstrates the possibility to fully utilize available data. However,when parameterizing time between consecutive medical features, these methods may cause either under-parameterization or over-parameterization [46]. Our previous paper [51]

Data imputation

Methods to deal with missing values can be grouped into three major categories: 1) Missing completely at random (MCAR); 2) Missing at random; 3) Not missing at random. Most data imputation approaches focus on MCAR. There are different data imputation methods, the simplest one involves replacing the missing values with 0, constant values, or mean/median values [52, 53]. This method of data imputation is easy and fast and generally works well with small datasets, however it may not be very accurate and is not recommended for use when imputing categorical variables [54]. The k nearest neighbors (k-NN) approach is another common method for data imputation. The algorithm uses fea-

ture similarity to predict the missing values [55]. The missing values will be replaced based on how closely the present values match to their counterparts in other samples. It is reported that the k-NN method for imputation can achieve much greater accuracy than simple replacement, however it assumes a relationship exists between the various biomarkers, is sensitive to outliers and is computationally expensive since it requires storage of the entire training dataset in local memory. Another popular imputation method is multivariate imputation by chained equation (MICE) [56]. This type of imputation works by filling the missing data multiple times. Multiple Imputations (MIs) are much better than a single imputation as it measures the uncertainty of the missing values in a better way.The chained equations approach is also very flexible and can handle different variables of different data types (ie., continuous or binary) as well as complexities such as bounds or survey skip patterns [57]. Deep Learning also affords an approach to data imputation that generally works very well with both categorical and non-numerical features. In this approach machine Learning models are stored which use Deep Neural Networks to impute the missing values [57].

Generalized Estimating Equation (GEE) model

In statistics, GEE model is popular approach for longitudinal studies with repeated measures through time [58]. The GEE treats each patient as one cluster, and number of visits are within each cluster. The greatest

advantage of GEE is that it could provide robust estimates and does not require distribution assumption as long as there are enough clusters, even the model distribution is seriously in error, it still have valid variance estimate [59–61]. It is reported that GEE is pretty robust as long as you specify the correct correlation structure is much less prone to efficiency loss [62].

2. Hierarchical variables

In EHR data, International Classification of Diseases (ICD-9 or ICD-10) for diagnoses codes and Healthcare Common Procedure Coding System (HCPCS) for procedures are two important variables to evaluate patient's condition. These codes have a hierarchical structure, for example, 250 is for general diabetes, 250.1 means diabetes with ketoacidosis. How to fully utilize that information in the model is still a challenging task [32, 34, 44]. In traditional analysis, people use grouper software to convert ICD codes into different hierarchical condition category variables (HCC), which is recommended by the Centers for Medicare and Medicaid Services (CMS). The HCCs only have 86 categories, which significantly reduce the number of ICD codes. Clinical Classifications Software (CCS) could categorize the ICD diagnosis codes into 283 categories [63] However, such aggregated features may lose some important detailed information and ignore the sequence relationships among the feature elements, for example, pneumonia and bronchitis are clearly more related than pneumonia and obesity [64].

2.2 Deep learning in healthcare

Recently, deep learning has gained great interests in images, natural language processing etc. Different from traditional machine learning, deep learning focuses on how representation could be learnt from the raw data. The core parts of the deep learning are composed of multiple processing layers based on neural networks, which are learnt from data, not from human beings. The deep learning is turned out to achieve remarkable performances in discovering intricate structures in high dimensional data. However, the use of deep learning for healthcare has been not thoroughly explored [65]. In deep learning models, data are processed through a series of connected hidden layers to yield the results. The accuracy of the deep learning models will become better and better with the increase of the data size, especially the deep learning could make correlations and connections from previous results [29, 65, 65, 66].

A simplified version of DNN could be represented as a hierarchical neuron structure with input layer, two hidden layers and output layer. These hidden layers perform the mathematical functions that could convert raw inputs into meaningful output. This multi-layered architecture allows deep learning models to complete classification tasks such as identifying subtle abnormalities in medical images, clustering patients with similar characteristics into risk-based groups [67, 68].

One greatest challenge for DNNs is how to link the processes to their outcomes. Not like the traditional machine learning methods, such as for linear regression, the parameter coefficients suggest how much the outcome will increase per unit

increase for the numerical features. However, for DNNs, it is hard to interpret by simply checking the inference process. The deeper the network architecture, the more parameters need to be estimated. It is common for a DNN to have millions of parameters within the network structure. Furthermore, network architecture is composed of various components (unit type, activation function, connectivity pattern, gating mechanisms) and the interactions between these components. Due to these complications, DNNs are often called black box models. This becomes a big barrier for deep learning models to be widely applied in healthcare areas [69–72].

In healthcare domain, people are more concerned about how machine learning methods make the decisions, and whether the decisions have a solid foundation. For example, Wang reported that though they applied deep learning techniques for diagnosis of breast cancer, the human doctors still needed to check the results to make sure the diagnosis was correct [73]. Especially when the model is more complicated, without interpretation, the outcome prediction will be suspected. If the model can explain itself, it could convince the users to adopt the model [74]. Thus, there is urgent need for developing explainable machine learning methods [29, 69].

2.2.1 Major techniques in deep learning

Until the last few years, most of the techniques for EHR data analysis were based on traditional machine learning and statistical techniques such as logistic regression, support vector machines (SVM), and random forests etc. Recently, deep learning techniques have achieved great success in many domains through deep

hierarchical feature construction and capturing long range dependencies in data in an effective manner [75]. There has also been a number of publications applying deep learning to EHR data for clinical informatics tasks, which achieved better performance than traditional methods. The deep learning was proposed for the first time in 2006. Deep learning becomes more powerful since 2006 when Hinton proposed a novel deep structured training model named as deep belief network (DBN). The general concept of deep learning stems from the artificial neural network (ANN),which has become very popular research area during the past few decades [76, 77].

For many years, hardware limitations have made DNNs impractical due to high computational demands for both training and processing, especially for applications that require real time processing. Recently, advances in hardware and GPU acceleration, cloud computing and multi-core processing, have enabled DNNs to be recognized as a significant breakthrough in artificial intelligence [77]. Several major DNNs architectures are introduced below:

2.2.1.1 Autoencoders and deep autoencoders

An autoencoder is an unsupervised neural network which is designed for features extraction using data driven learning through backpropagation algorithm. Basically, an autoencoder has the same number of input and output nodes. The network is trained to learn an approximate function, so the output is similar to input. Usually, the number of hidden units is smaller than the input/output layers, which

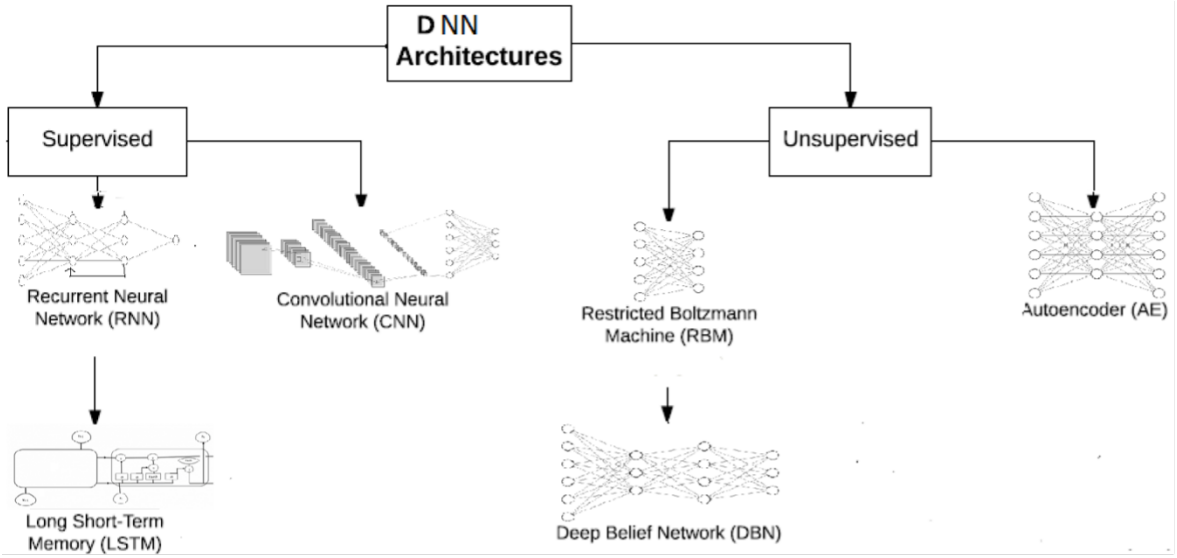

Figure 2.1: Types of deep learning methods.

achieve encoding of the data in a lower dimensional space and extract the most discriminative features. When placing constraints on the network, such as by limiting the number of hidden units, we can discover interesting structure about the data.

However, when the input data have high dimensions, a single hidden layer may not be enough to represent all the data. Alternatively, many autoencoders can be stacked on top of each other to create a deep autoencoder architecture. Deep autoencoder structures also face the problem of vanishing gradients during training. A common solution is to use some unsupervised greedy training algorithms for each layer to complete the pretraining of the hidden layer, and then use the backpropagation algorithm to optimize and adjust the system parameters of the entire neural network., which could avoid localized optimization [72, 78].

Deep autoencoders have been found to predict patient's future disease classification which may help clinical decision making using about 700,000 patients from the Mount Sinai data warehouse. Their results showed that autoencoders performed significantly better than other approaches, such as Principal component analysis, Gaussian mixture model, K- means [66]. Fakoor proposed an autoencoder architecture based on gene expression data from different types of cancer from the same microarray dataset to detect and classify cancer [36].

2.2.1.2 Recurrent neural network(RNN)

RNN is a deep neural network which is capable of analyzing time series data where the output depends on the previous computations. RNN is commonly found in the analysis of text, speech and DNA sequences. RNN usually takes a series of input and remembers things learnt from prior inputs while generating outputs. In this way RNNs combine inputs from both the present and the recent past to produce the outputs. For this reason, RNNs are said to have memories [10, 34, 35, 65].

However traditional standard RNNs suffer from vanishing and exploding gradient problems, an important variation of RNN is LSTM, which was proposed in 1997 and could solve these issues through introducing news gates, such as in- put and forget gates. Specifically, LSTM is particularly suitable for applications with very long lags or unknown sizes in a time series. During the training, the network learns what to store and when to allow reading/writing to minimize the classification errors. Unlike other types of DNNs, the weights of the network of a RNN or a

LSTM are fixed and do not change over time. They only differ in the output and the internal states. These will greatly reduce the total number of parameters to estimate [29, 45, 79].

Several papers applied recurrent neural networks with LSTM to electric health records. When modeling longitudinal EHR data, LSTM were used to establish relationships between mixed effect observations and future events. Rajkomar used 114,003 patient records from University of California, San Francisco, from 2012 to 2016, and the University of Chicago Medicine from 2009 to 2016 for prediction tasks [80]. They tried three deep learning models: one based on recurrent neural networks, one on an attention-based time-aware neural network model, and one on a neural network with boosted time-based decision stumps. They discovered that deep learning methods were capable of accurately predicting multiple medical events (eg., the prediction of in-hospital mortality, readmission, length of stay, and discharge diagnoses) from multiple centers. Jo combined LSTM and latent topic modeling for mortality prediction using MIMIC-ii data set and showed their model significantly outperformed prior models [81].

2.2.1.3 Restricted Boltzmann Machine (RBM)

A RBM was initially proposed in 1986 and is a variant of the Boltzmann Machine, which is a type of stochastic neural network and can learn a probability distribution over the inputs through stochastic optimization such as Gibbs sampling [36, 45]. One important characteristic is that RBMs have undirected nodes

and no connections between any two visible units or any two hidden units, which implies that values can be propagated in both the directions. The stacked hierarchical RBM could form a deep belief network for a supervised learning task. Bayesian networks are probabilistic graphical models to simulate causality in human reasoning in the form of a directed acyclic graph, which is mainly used to describe conditional dependencies between random variables [65,68,77]. Khademi combined a DBN and Bayesian network to extract features from Microarray data to overcome missing attributes and noise in a breast cancer genetic detection study. Deep learning approaches have also outperformed SVM in predicting splicing code and understanding how gene expression changes by genetic variants [82].

2.2.1.4 Convolutional Neural Networks(CNNs)

CNNs have been widely used to analyze image data. CNNs are regularized multilayer perceptrons. CNNs do not use predefined kernels, but instead learn locally connected neurons through a mathematical linear operation. The basic building blocks of CNNs include convolution layer, activation layer, pooling layer and fully connected layer. The convolution layer, also called kernel, has some mathematical operations to filter the images into few pixels (such as 3X3, or 5X5), and then sum up the results into one number to represent all the pixies the filter observed. The activation layer will generate non-linear function to allow the network to train itself through backpropagation. The activation function is normally ReLu. The pooling layer is the process of further reducing the matrix size. The final fully

connected layer is a traditional multiplayer perception structure, which will convert the fully connected layer into a list of probabilities of each label [29, 77].

Shin presented a combined text-image CNN to identify semantic information that links radiology images and reports from a typical Picture Archiving and Communication System (PACS) hospital system [83]. Che highlighted that although DNNs outperform conventional machine learning approaches to predict and classify clinical events, but the DNNs suffer from the issue of model interpretability, which is important for clinical adaptation [84].

2.2.1.5 Transfer learning

Transfer learning is a method that a developed model is reused as the starting point for new model development. The rational is that in some cases, data is difficult to get or training a model may take a huge amount of time. The basic steps for transfer learning is first to create a base model and freeze layers so that those layers will not be changed during the training, then add new trainable layers so that it could be used for predictions on the new dataset. Generally, transfer learning is used between similar domains, which is about homogeneous transfer learning, where we assume the source and target domains have the same feature spaces. Heterogeneous transfer learning is for those source and target are nonequivalent or no overlapping domains. The heterogeneous learning is relatively recent area in research, there are still challenges regarding the model performance [85, 86].

Medical images are one of the most important topics in medical sciences. How-

ever due to the strict Health Insurance Portability and Accountability Act and other rule regulations, it is difficult to collect a massive amount of labeled data. Although transfer learning has proved to be one of the most efficient ways to solve the issue of insufficient data, since most of the base transfer learning models are based on public image data, such as as CIFAR-10, which is about 10 classes of images for airplane, automobile, bird, etc. or imagenet database, which is different from medical domain. The transferring between two distant related domain could lead to negative results. For example, the dog classification model may not be beneficial to COVID image classification. In this case, the transfer learning is not necessary better than the performance by a model from scratch [87].

38 different methods for heterogeneous transfer learning methods have been reviewed and compared. The challenges are that there is little commonalities among these methods. Each author proposed their methods for their specific problems, which is of little value to be applied to other study [85].

2.2.1.6 Ensemble learning

Ensemble learning is a special type of learning approach, where it uses multiple algorithms to obtain better performance than using the individual model alone [88, 89]. Bagging is also called bootstrap aggregating, which is designed to improve the accuracy and reduce the variance of the modes. The process of the bagging is like random subspace method except that bagging randomly selects observations from the original datasets with replacement [90]. Random forest adopted both bagging

and random subspace approaches. When random forests select the bootstrapped set of samples, random foresters also select a small but consistent number of unique features in the decision tree models. These enable random forests to have two elements of variability. The final results will be based on the function of all trees in the random forests. Due to these properties, random forests are considered one of the most accurate data mining algorithms [91]. Random forests also show that the predictions from random forest are asymptotically normal [92]. Also, it has been approved that tree models have universal consistency properties, including random forests, bagging and other classifiers which use average as a means of obtaining the rules [93].

2.3 Interpretability

Most predictive machine learning systems in healthcare just provide predictions, however in practice, you have to convince the medical practitioners to accept the outcomes from such models. Thus, there is a need to integrate interpretable models with predictions from the medical facilities. Most predictive models are not prescriptive or causal in nature. In many healthcare applications, explanations are not sufficient [74, 94–96].

2.3.1 Interpretable models

Two most interpretable models are linear regression and decision trees. The biggest advantage of linear regression models is linearity, which makes estimation

procedure simple and easy to understand through its regression coefficients. The coefficients may show the relationship between input variables and outcome.Beyond linear regression, some extensions of the regression are also popular as explainable models, such as generalized linear regression model and generalized additive models(GAM), which enhances the ability of linear regression models to deal with nonlinear or non Gaussian distributed data. This is one of the main reasons why the linear model and all similar models are so widespread. The drawback for linear model is that it only deals with linear relationship, and could oversimply the complex situation [24].

Decision tree is also considered one of the best models for model interpretability due to its nature to mimic the human thinking by creating decision rules based on previous result. Since decision tree has a graphical structure, the hierarchical trees structure could provide information about the relative importance of attributes.Decision tree also has some disadvantages, such as it lacks of ability to deal with linear relationships, and quite unstable. When there are many features, and the decision tree may be time consuming for large number of features and data inputs. [24, 97, 98]. In this section, we will briefly review some popular techniques to build interpretable deep learning models(see figure2.2.

2.3.2 Types of interpretability methods

Generally, interpretability techniques could be classified into different groups based on different criteria:

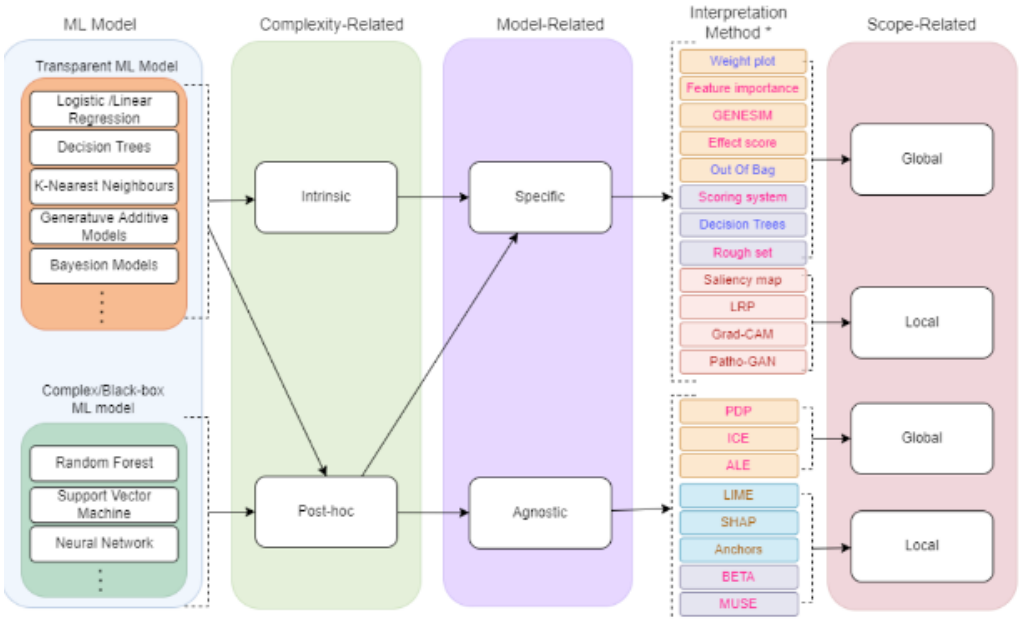

Figure 2.2: Types of interpretability methods [3]

2.3.2.1 Before (pre-model), during (in-model), or after (post-model)

This type is based on when these interpretation methods are applicable [24].

Pre-model interpretability usually happens before model selection, which is more about data interpretability. Through data exploration, we could have a good understanding of the data before thinking of the model.In-model interpretability is about how the machine learning methods have inherent interpretability in the models. The post-model interpretability means how to improve the interpretability after building a model [24, 99].

2.3.2.2 Intrinsic vs. post hoc

Depending on when to obtain the interpretability, interpretable machine learning techniques can generally be grouped into two categories: intrinsic interpretability and post hoc interpretability. Intrinsic interpretability refers to integrating interpretability directly to the model structures such as decision tree, rule-based model, linear model etc. In contrast, the post hoc one needs to create a second model to explain an existing model. Generally, intrinsic approach could provide more accurate and undistorted explanation, while post hoc could maintain the model accuracy with less transparent nature [24, 99, 100].

2.3.2.3 Model-specific vs. model-agnostic

Model-specific interpretation is unique to a single model or group of models. For example, the interpretation through regression parameter coefficients in a linear model is a model-specific interpretation, in contrast, model-agnostic tools can be used for any machine learning models no matter how complicated. These agnostic methods usually work by analyzing feature input and output and applied after the model has been trained [99].

2.3.2.4 Local or global

The interpretability could also be classified into global or local based on whether to explain part of prediction or the entire model. Global interpretability means inspecting the whole structures and parameters of a complex model, while

local interpretability tries to figure out why the model makes the decision through identifying the contributions of each feature in a specific input of the partial model. Global interpretability could explain the inner working mechanisms to increase their transparency. Local interpretability could help uncover the causal relations between a specific input and its corresponding model prediction. Those two could complement each other to improve the interpretability [99].

2.3.3 Intrinsic approaches

2.3.3.1 Adding interpretability constraints

Adding monotonic constraints in the models could improve the interpretability, since fewer features and monotonicity could make simple and straightforward relations between features and the prediction. Similarly, pruning the decision tree could reduce the size of the trees and decrease the complexity to increase the model's interpretability [101–103].

Adding meaningful constraints to the model could further improve interpretability. Zhang proposed an interpretable CNN through adding a regularization loss to higher convolutional layers of CNN. Thus, each filter in the conventional layer may have distinct representations, for example, one filter may represent left eye and another filter could represent the right eye [104].

2.3.3.2 Mimic learning

Mimic learning is another alternative to increase model interpretability [105]. The rationale behind mimic learning is to approximate a complex model using an easily understandable model such as rule-based model, or linear regression etc. As long as the approximation is close, the statistical properties of the complex model will be kept in the interpretable model. Finally, you can get a model with comparable prediction performance, but it is much easier to understand. For instance, the tree ensemble model could be represented by a single decision tree.

Che applied mimic learning frameworks to a healthcare data with 398 patients in ICU and predicted their mortality. The interpretable model showed similar or better performance than the baseline models including SVM, Logistic regression, Gradient boosting trees and several DNN approaches such as LSTM, LR-SDA etc. While the mimic learning model had the desired feature transparency. In their process, they first trained a deep neural network with raw patient data, which produced a vector of class probabilities from last layer. Then they trained a gradient boosting tree (GBT) model on the raw patient data, and use the deep network's probability prediction as the target label. Since GBTs are interpretable linear models, they were able to assign feature importance to the raw input features while harnessing the power of deep networks [84, 105].

2.3.3.3 Attention mechanism

Attention mechanism is another popular method to explain predictions made by sequential models such as RNN, especially in NLP and image areas [106]. Attention mechanism has the ability to interpret which parts of the input are attended through looking at the attention weight matrix.

In machine translation, attention can turn two sentences into a matrix where the words of one sentence form the columns, and the words of another sentence form the rows, and then it will correlate the matrix and identify relevant correlation. Attention mechanism has been used to solve the problem of generating image caption. In this case, a CNN was first used to extract feature vectors from an input image, and then applied LSTM with attention mechanisms to generate descriptions through a stochastic attention model and deterministic attention. visualization of the attention weights could add more interpretability when generating a word. Similarly, attention mechanism has been incorporated in machine translation [107].

2.3.4 Post hoc approaches

Post hoc explanation tries to provide understanding about the models through a second model. Usually, the second model relies on sampling and labeling to approximate the decision function of the deep learning models.

2.3.4.1 Model-Specific explanation methods

Many model-specific methods are designed for Deep Neural Networks. Two interesting methods are class activation mapping (CAM) and gradient-weighted class activation mapping" (Grad-CAM). Both of them could be used for visual interpretation.

CAM is to identify the region of interest. Basically, CAM requires a CNN architecture at one specific layer and then identify which the given input image belongs [108]. Another visualization method—"gradient-weighted class activation mapping" (Grad-CAM) is a generalization version of CAM, as it can be applied to any CNN-based deep learning models [109], these two visualization methods rely on only one prediction score from a particular output node. As a gradient-based method, Grad-CAM uses the class-specific gradient information flowing into the final convolutional layer of a CNN in order to produce a coarse localization map of the important regions in the image when it comes to classification, making CNN-based models more transparent.

2.3.4.2 Model-Agnostic explanation methods

1. Feature importance

   Feature importance is one agnostic model explanation, which is not limited to one type of model, but can be widely applied to many different machine learning models. The features are normally easy to explain and extracted from raw data through data mining techniques based on domain knowledge. The

features could map the representation of the raw data to output. The feature importance is to identify the statistical contribution of each feature to the model.

For some tree-based ensemble models, such as gradient boosting machines, random forests and XGBoost, there are several approaches to measure the contribution of each feature. The first approach is to calculate the accuracy gain when a feature is added in tree branches. The second approach measures the feature coverage, i.e., calculating the relative quantity of observations related to a feature. The third approach is to count the number of times that a feature is used to split the data [105, 110, 111].

2. Local interpretable model-agnostic explanations (LIME)

   LIME is proposed to train local surrogate models to explain individual predictions. The method tries to understand the model by perturbing the input samples to understand how the predictions change. It starts to change a single data point and tweak the features values to identify the possible impact on the output and which features cause the prediction. The output of LIME is a list of explanations, which suggest the contribution of each feature. LIME is one of the few methods which could be used for text, image and tabular data. The advantages of LIME is that it is pretty flexible even you change the machine learning model, it is still working for the local model. The LIME could use the features which is not in the machine learning model. The disadvantages is how to correctly define the neighborhood. Most of the time, the sampling

data points follow Gaussian distribution and ignore the correlation among the features, which may lead to biased estimation [76, 112].

3. SHAP (SHapley Additive exPlanations)

   Shapley value stems from cooperative game theory, which refers to the average marginal contribution of a feature across all possible combination of features and is an interaction-based method. The Shapley value is fairly distributed among feature values and may avoid missing the interaction among the features, but the computation is very expensive. SHAP is based on the SHAP values and extends further. Compared to Shapley value, SHAP represents an additive feature attribution method and assigns weights to the features based on their weights in the coalition. The advantages of SHAP is that they have two alternative estimation approaches, kernel-based and tree based, thus SHAP could be calculated from more complex nonlinear models. The SHAP has a solid theoretical foundation that it could provide consistent. For tree-based model, SHAP implemented a fast algorithm to keep track of what proportion of all possible subsets, which could reduce the exponential time to polynomial time [24, 112, 113].

4. Representation Explanation

   There has been a growing interest to understand the functions at different layers of CNNs through finding the preferred inputs for neurons at each layer [113–115]. Since the CNN learns the high level features in the hidden layers, the role of each neuron or layer could be identified from activation

maximization frame work. Starting from random initialization and optimizing an image to maximally activate the neuron, what individual neuron is represented could be identified through iterative optimization. However sometimes, the feature visualization may contain very abstract features or there are too many neurons to check. Lower layer neurons focus on small and simple patterns, such as object corners. Mid layer neurons could help identify object parts, such as faces, legs. Higher layer neurons may detect whole objects [105]. Transfer learning is another method to use a layer from one network to solve a new problem [116].

Feature visualization technique has also been applied to RNNs or LSTM models. some work examines the representations of the last hidden layer of RNNs and study the function of different units at that layer, by analyzing the real input tokens that maximally activate a unit [99]

2.3.5 Interpretability evaluation

Though there is increasing trend in the growing body of research methods for improving machine learning interpretability, there seems to be very little research on developing measures and approaches for machine learning interpretability assessment.

One most important component for interpretability assessment is accuracy, which refers to the actual connection between the given explanation model and the predictions from the deep learning models [117].

It is reported that if the explanation could not have completely faithfully to the original deep learning models, it could lead to the danger that the explanation could be an inaccurate representation of the original model. For high-stakes decision model, even an explainable model with 90% agreement with the original model, it still means the explanation is wrong 10% of the time [26, 117].

Another important measurement is stability, which means similar objects much have similar explanations. Thus, slight perturbations in the input data should result in small changes in the predictions [118]. Trust is also one important measure for interpretability, which is to capture the mutual agreement between the deep learning model and explanation model [96, 119].

2.3.6 Limitations and challenges of current interpretability methods

The basic logic behind the current explainable machine learning methods is to find a function which could closely approximate the outputs from the black-box model. The predictions from the black box models are normally the target of explainable models, not the original data since generally black box model could achieve the best performance and we try to understand the deep learning model through explanation models. The explanation models could always have poor performance compared to deep learning model, otherwise, there is no need to use deep learning models.The explanation model may calculate which attributes of the input data in the black box models are most important regarding correlation, or it could offer some interpretable models such as linear regression or decision tree to mimic the

behavior of the black box.

Most of current explainable models are designed to identify the correlations between the input features and the predicted outcome,which are unable to to answer the casual inference questions. For example, smoking could lead to high risk of cancer, yellow finger is one side effect of smoking. These two features are highly correlated. These correlation will not affect the prediction accuracy, but could be an serious issue for causal inference. SHAP value is a popular explanable method which could show the contributions of each input feature for the outcome, however it uses all features in the SHAP value calculation regardless the collinearity.The collinearity could affect the SHAP value calculation. If two variables are highly correlated in the model, one variable could be less precise, this will lead to underestimation of parameter [120]. The causal interpretable models could explain the decisions how the model could be if we change some feature values.

2.4 Decision tree and deep learning models

As deep learning-based techniques is more popular than other learning methods, more and more work is trying to explore the combination of deep learning and decision trees. There are different types of models based on how they utilize the decision trees [121].

2.4.1 Neural networks to decision trees

Recent work uses distillation is trying to train a decision tree to mimic a neural network's input-output function.The author used the predictions from the neural network and also the data with the true labels to train the soft decision tree. However, the tree model has accuracy of 96.7% compared to 99.2% using deep learning model alone [25].

2.4.2 Decision trees to neural networks

Some work proposed to initialize a deep neural network with decision trees, where the weights of the neural network were provided by decision trees.Basically the neural network was built based on the structure of decision tree, so that neural network had similar structure as the decision tree. However, their model accuracy was only 96% compared to 98.5% using deep learning models [**?**].

2.4.3 Mixing deep learning models and decision trees

The Deep Neural Decision Forest was an ensemble approach which utilizes the vectors from fully connected layer to provide the inputs and internal nodes for decision trees.The final output was not from deep learning model, but from the forest. The results showed better performance than benchmark machine learning models [122] However, they applied very complicated algorithm to embed the full layer functions to compute the weights and probability distribution in decision trees, which sacrificed interpretability of the model.

In a recent paper, neural backed decision trees were proposed to achieve comparable neural network accuracy. They first used the weights from a pretrained network's fully connected layer to build a hierarchy tree, then fine tuned the network and then construct features for each image using the neural network, which were feed into the decision tree to get the final prediction. The final model performance matched the neural network performance with a 0.05% margin [121].

Che applied mimic learning frameworks to a healthcare data with 398 patients in ICU and predicted their mortality. The interpretable model showed similar or better performance than the baseline models including SVM, Logistic regression. In their process, they first trained a deep neural network with raw patient data, which produced a vector of class probabilities from last layer. Then they trained a gradient boosting tree (GBT) model on the raw patient data and use the deep network's probability prediction as the target label, they they try to use feature importance to the raw input features for explanation [5].

2.5 Deep learning models and mixture models

2.5.1 Gaussian process in deep learning models

Deep neural networks with fully connected layers share a common network structure which include input layer, several hidden layers and one output layer, which are composed of the similar core components, like neurons, synapses, weights, biases and activation functions.Even they may have different hidden layers, for example, recurrent neural network has different layer structure as CNN.

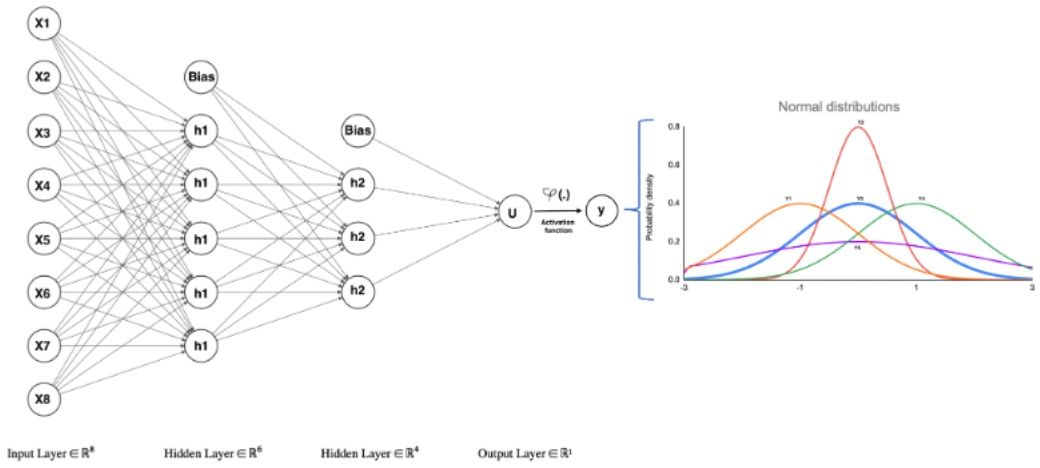

Figure 2.3: Neural network and Gaussian process

For a single-hidden layer neural network, since the weights and bias parameters are i.i.d terms, and activation functions are also independent. The final output is sum of i.i.d terms, thus based on the Central Limit Theorem (CLT), the output is Gaussian distributed [123]

Lee showed that any infinite wide deep neural network with fully connected layers including CNN, RNN etc. could be considered as neural network Gaussian process, which means the output of the neural network can be expressed as a Gaussian process in terms of its input [124].

2.5.2 Mixture density network

In neural network models, the last layer is trained to predict the values. For each specific input value, its prediction has its own Gaussian curve, for example, when input value is x_0, it is normal curve. The predictions of all predicted values

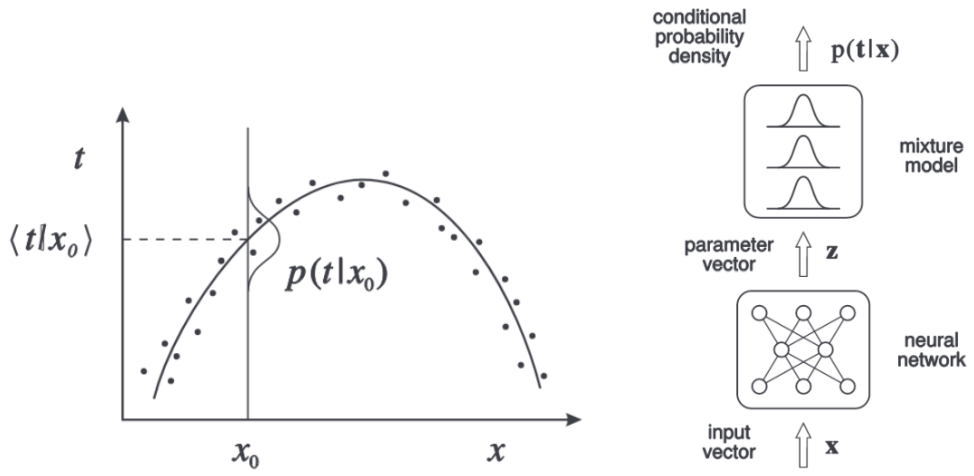

Figure 2.4: Mixture density network, figures from [4]

X forms another distribution. To estimate this distribution of overall input values, Bishop proposed mixture density network [4]. The mixture density network is from two component: a neural network and a mixture model. In his method, the neural network could be any structure which takes input and covers it into outputs. The mixture model is just a model of probability distributions with weighted sum of simple distributions, where Gaussian kernel is used to explain that any probability density function. The mixture structure could allow more flexibility for uncertainty of the outcomes.

Chapter 3: Methodology

In this section, we first describe our LIVE algorithm and show how to implement it, then we discuss its properties and why it could help improve the model performance. Finally we summarize the differences between our LIVE algorithm and several other popular interpretability methods, such as GRAD-CAM, LIME, and SHAP values.

3.1 Decision tree and regression tree

Decision tree is the foundations for many classical machine learning algorithms and widely used in many applications for predictive modeling. This tree-based algorithm is a popular family of non-parametric and supervised methods.

The basic structure of the decision tree includes root node and leaf nodes. The inputs are passed through the root node and further divided into sets of leaf nodes.

Compared to other algorithms, the decision tree requires less effort for data preparation and no need for normalization. The most important is that decision tree is very intuitive and easy to explain.

There are multiple ways to interpret decision tree. See figure 3.1,starting from the root node, the edge is the rule from one node to another node.We can have the

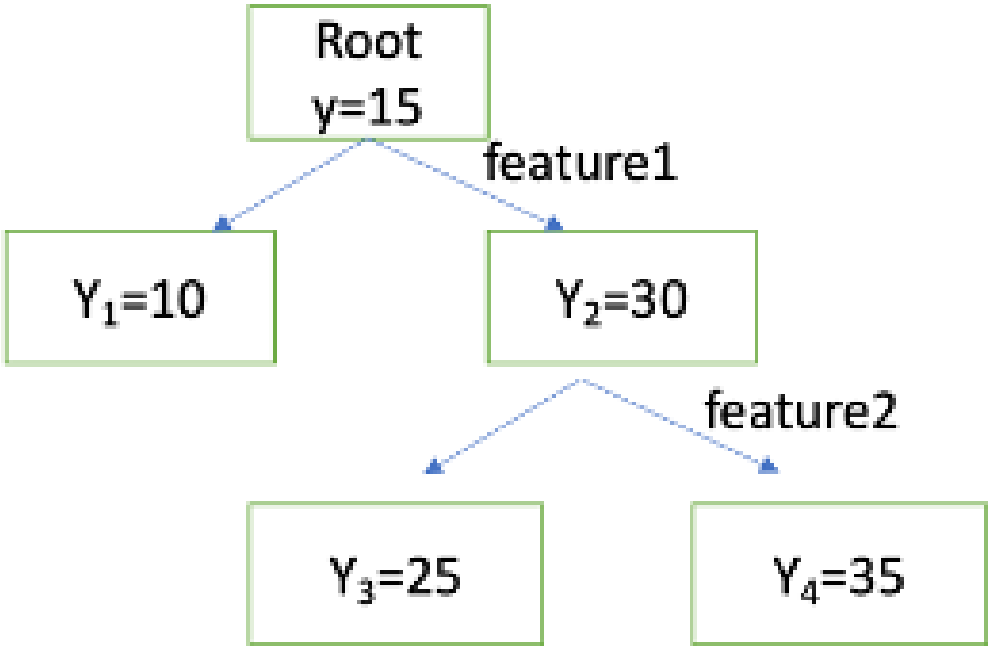

Figure 3.1: Regression tree

overall average value for the outcome, The values in each node will change after new node is added to the rule path, the difference is the effect of newly added feature. For example, the overall average is 15, after adding feature 1, the value of right node becomes 30, which means feature 1 contributes to the increase of 15. Similarly, feature 2 for right node is 5. We can use a simple formula to decompose the feature contributions for each leaf node.

$$y_i = \bar{y} + \sum_{i=1}^{n} \beta_i \times feature_i \tag{3.1}$$

$\bar{y}$ is the average values without any features in the tree.

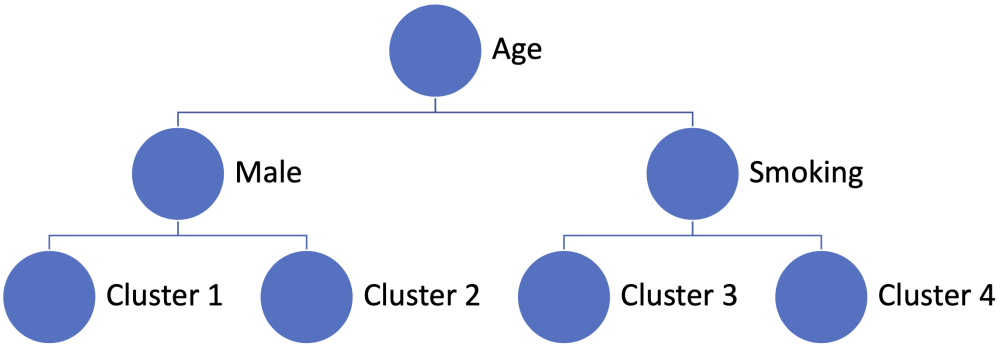

Figure 3.2: Decision tree for clustering

For example,

$$y_2 = \bar{y} + feature_1 + feature_2 \tag{3.2}$$

Through solving the above equations, we could easily interpret the contributions of each feature, such as feature 1 could increase 15, and feature 2 could help increase 5. We could use either generative addition model or linear regression model.

3.2 Clustering

Clustering aims to find and group the data into similarity groups. Generally, clustering is an unsupervised learning as there is no prior cluster information. Decision tree could also be considered as one weak clustering approach, each leaf node could be treated as a cluster,see figure 3.2.Compared to k-means, k nearest neighbor algorithm, decision tree clustering is a supervised clustering approach [125].

3.3 Bagging and random subspace

Bagging is one ensemble learning method where a random sample of data is repeatedly taken from the training data, the final results are aggregated from these samples.The purpose of the bagging is to reduce variance.

Random subspace method is also called attribute bagging, which is to randomly select some features from the entire feature set [126]. Random subspace method is an attractive choice for high dimensional problems, which has been applied to gene expression data and fMRI data [127]. It is reported that random subspace approach could give consistently better results than bagging in both accuracy and stability [128]. Suppose we have D features in the dataset, the basic process for random subspace is that for each model, randomly choose d (d << D) features from all D features with replacement and train the model.

3.4 Deep learning models

3.4.1 CNN model

Image classification is one of the core problems in computer vision filed. CCN model is probably the most widely used model for image classification, which shows excellent success in the field of pattern recognition and classification. There are two common strategies using CNN model. One is to design and train CNN model from scratch, which may need a lot of efforts and time to construct an appropriate model. Transfer learning is another approach that you could utilize the existing CNN model

structure and decrease the training time and may result in lower generalization error.

Since most of existing transfer learning models are based on non-medical images and many transfer learning models rely on assumption that the source domain and the target domain have a strong connection. It is reported that transfer learning models may not significantly improve performance. The models trained from scratch perform nearly as well as the transferred models [129, 130].

Some challenges exist for transfer learning, such as overparameterization, the expensive computations etc. Most transfer learning models require minimum image size, such as MobileNet requires at least 224x224, which apparently increases the running time. In our analysis, we did pilot studies using pneumonia xray data to compare the performance between transfer learning model and CNN model from scratch.

3.4.2 Transfer learning models

Transfer Learning is a machine learning method where we could reuse a pretrained model as the starting point for a model on a new task. Especially in image analysis, very few people like to train an entire CNN from scratch. Instead, it is common to pretrain an existing model and then use it either as an initialization or a fixed feature extractor for the task of interest. When there is a mismatch in the domain between the dataset for pretext tasks and the downstream task, the transfer learning may not work. The pretrained models may converge but it will be stuck in a local minimum. Thus, the performance will not be better than training from

scratch [87].

3.4.3 LSTM model

LSTM is a type of recurrent neural network capable of dealing with longitudinal or time series data problems. LSTM is particularly suitable for applications with very long lags or unknown sizes in a time series, which has demonstrated superior performance for modeling sequential data.

Compared to CNN structure, LSTM has several LSTM layers, which includes a bunch of LSTM cells. The cell includes several important gate structures, which could control information flow from input to output [131].

3.4.4 Evaluation metrics

AUC, Accuracy, precision, and recall are four popular metrics in evaluation of model performance. Accuracy is defined as the number of true positives divided by the total number of true positives and false negatives. Hence, recall shows what percentage of the actual positives you were able to identify positives.

Recall calculates the number of actual positives the model was able to capture after labeling it as positive.

Precision is defined as the number of true positives divided by the total number of positive predictions. Precision is to measure how correct your model's positive predictions were, it could measure how accurate the model is in terms of those which were predicted positive. AUC represents the degree of separability and to measure

the extent to which an information filtering system can successfully distinguish between signal and noise. Similar to Precision and Recall measures, AUC Curves make an assumption of binary relevance and the ordering among relevant items has no consequence on the ROC Curves metric [132].

3.5 Problem definition

In this section, we will describe some common problems, then we will propose our research questions.

3.5.1 Mixture distributions

For single deep learning model, the inputs are assumed i.i.d, thus all the outputs are also independently distributed. However, as we know that there are some correlations among some of input data points, which indicate some common elements among the outputs, see figure 3.3.In another word, the population has some sub populations. For example, within 10,000 images data, there should be some images which share common characteristics, such as both cancer regions are in upper corners.The key part is how to find the latent cluster information among the data points. The traditional approach to calculate the mixing coefficients is through numerical integration. However, given the huge number of input data points, especially for image data, it will be cumbersome to do such calculation. Other than that, the issue of interpretation still exists.

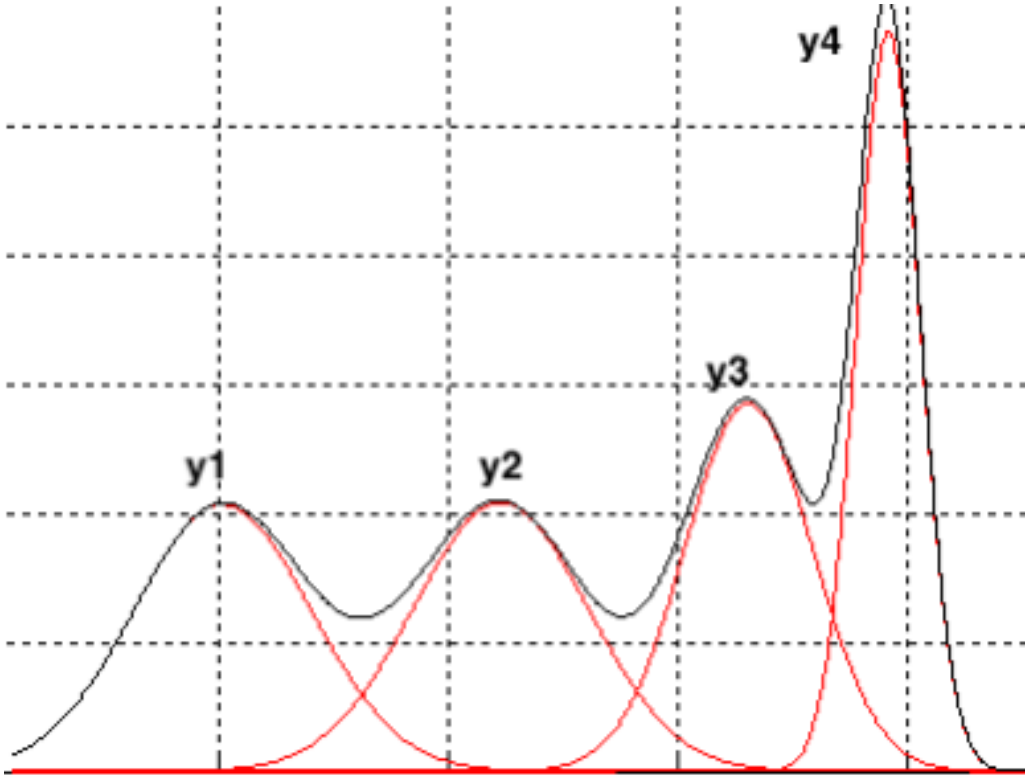

Figure 3.3: Mixture distribution

3.5.2 Interpretability

It is widely known that deep learning models are black box and lack of interpretability, we need to think of ways to link the feature values to the predicted values and make it an interpretable model.

3.6 Proposed method

In this section, we describe our proposed algorithm-LIVE to address the above issues. We start by mathematically formulating the algorithm, then we explain our approach, then we conclude this section with complexity analysis.

Our algorithm is described in figure 3.4. The core idea of LIVE algorithm is that we first train a deep learning model and get predicted values for all input data points, meanwhile, we also train multiple decision trees with raw target values to assign all input data into different nodes(clusters), then feed the predicted values from the deep learning models into the leaf nodes of the different trees. For each tree, we take the average of predicted values in each node, then average the values from different trees to get the final output for each input data point.

Our approach is similar to random forest regression except that random forest tree is using the same set of outcomes for clustering and predictions, while our LIVE algorithm uses two sets of data.The raw outcomes are used for clustering, and the predictions from deep learning models are used for predictions.

Algorithm 1 LIVE algorithm: Part 1: Get newly predicted probabilities

Require: Four inputs

Training dataset $D_1 = (x_i, y_i), i = 1, 2, 3, .., n_1$

Test dataset $D_2 = (x_j, y_j), j = 1, 2, 3, .., n_2$

Deep learning model $f(x)$

Decision tree models $g(x)$

Ensure: Two outputs

a. Newly calculated predicted probabilities for both datasets

b. Rules for all the paths of all decision trees

1: Train data to build deep learning model $D_1 \xrightarrow{train} f(x)$

2: Get predicted probabilities for training data $D_1 \xrightarrow{f(x)} p_1$

3: Get predicted probabilities for test data $D_1 \xrightarrow{f(x)} p_2$

4: **for** $b = 1$ `to m trees` **do**

5: `Randomly select p features from all features in` D_1

6: `Build decision trees using training data` D_1

7: `Get rules from the decision trees` $D_1 \xrightarrow{train} g(x)$

8: `Recalculate leaf node probabilities ,` $p_{1,2} \xrightarrow{D_{1,2}} (D_{1,2}, C_1, p_{1,2}, p^*_{1,2})$

`;where` $p^*_{1,2} = \frac{1}{c}\sum_{i=1}^{c} p_{1,2,j}$

9: **end for**

$LIVE^m(p^{**}_{1,2}; z_1, z_2, .., z_m) = \frac{1}{m} = \sum_{b=1}^{m} p^*_{1,2}$

return $(D_2, R_1 1, R_1 2, ..., R_{j,m}, p_1 1^*, p_1 2^*, ..., p^*_{j,m}$

`where` $R_{j,m}$`, rule for each testing data point` x_j

$p^*_{j,m}$`, recalculated predicted values for each testing data point`

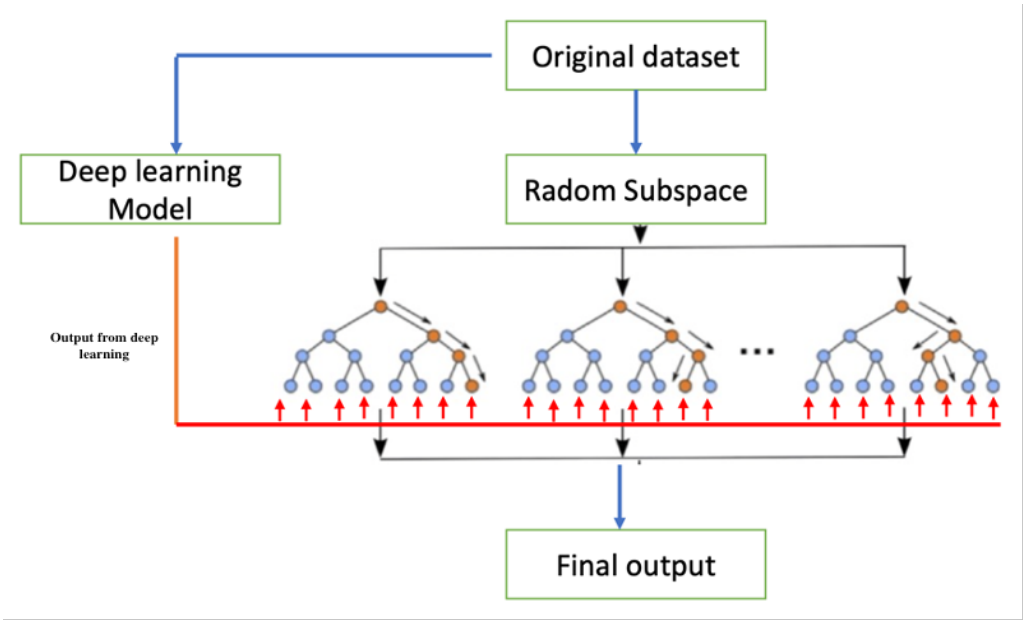

Figure 3.4: LIVE algorithm

Algorithm 2 LIVE algorithm: Part 2: Get interpretable model

Require: Rules and recalculated predicted values from algorithm 1

$(D_2, R_1 1, R12, ..., R_{j,m}, p_1 1, p_1 2, ..., p_{j,m})$

Ensure: Coefficients for each feature for each test data point

1: **for** $j = 1$ `to n data points` **do**

2: **for** $b = 1$ `to m trees` **do**

3: `Collect features in each rule path and predicted values in that node` $(p^*_{j,b}; f_{j,b})$

4: **end for**

`Regression model for data point` $x_j, p_{j,b} = \beta_0 + \beta + \epsilon$

5: **end for**

return Coefficients for each feature

3.6.1 LIVE algorithm description

Definition 1: Suppose we have two i.i.d random variables X, Y, which has n observations , X $\in R^d$, d is the number of features in X, and Y is the outcome variable.

Definition 2: A deep learning model function $\widehat{f}$(x) , the outcomes of input X are

$$\widehat{f}(x; D) \sim \mathcal{N}(\mu, \sigma^2)$$

Definition 3: Suppose we have B number of trees, for each tree we have a function $\widehat{g}(x; D)$. We will randomly take fixed z features from total d features, which are called randomized classifiers. We use these features to build decision trees.

Decision tree is considered as a weak cluster method.In a decision tree, each leaf node is treated as a cluster, all the values in that cluster are taken average as the node value. If we build B decision trees, within each decision tree, we could always find the node which has data point X_1.

We can get the following formulas for each tree regarding data point x_1

For tree k_1, we have :

$$\widehat{g}(T_{k1}; x_1) = \frac{y_1 + y_2 + y_5 + ... + y_n}{n_1} \tag{3.3}$$

For tree k_2, we have :

$$\widehat{g}(T_{k2}; x_1) = \frac{y_1 + y_3 + y_5 + ... + y_n)}{n_2} \tag{3.4}$$

...

For tree k_B, we have :

$$\widehat{g}(T_{kb}; x_1) = \frac{y_1 + y_5 + ... + y_n}{n_b} \tag{3.5}$$

The final average value for data x_1 from all above trees will be:

$$\begin{aligned}
\widehat{g}(x_1) &= \frac{1}{b}\sum_{t=1}^{n} \widehat{g}(T_{kt}; x_1) \\
&= \frac{1}{b}((\frac{1}{n_1} + \frac{1}{n_2}.. + \frac{1}{n_{t1}})y_1 + ... + (\frac{1}{n_1} + \frac{1}{n_2}.. + \frac{1}{n_{tb}})y_n) \\
&= a_{11}y_1 + a_{12}y_2 + ... + a_{1n}y_n \\
&= a_{11}f(x_1) + a_{12}f(x_2) + ... + a_{1n}f(x_n) + \epsilon \\
1 &= a_{11} + a_{12} + ... + a_{a1n}
\end{aligned} \tag{3.6}$$

Similarly, we can expand this equation to all data points.

$$\begin{aligned}
\widehat{g}(x_n) &= \frac{1}{b}\sum_{t=1}^{n} \widehat{g}(T_{kt}; x_n) \\
&= \frac{1}{b}((\frac{1}{n_1} + \frac{1}{n_{12}}.. + \frac{1}{n_t 1})y_1 + ... + (\frac{1}{n_1} + \frac{1}{n_{22}}.. + \frac{1}{n_{tb}})y_n) \\
&= a_{n1}f(x_1) + a_{n2}f(x_2) + ... + a_{nn}f(x_n)
\end{aligned} \tag{3.7}$$

We can see that all the data points have the similar formula as (3.5), which are a mixture distribution. The sum of weights for each y_i is equal to 1,and the coefficients are the mixing rates.

To calculate the overlapping between normal curves, one simple and intuitive way is to randomly draw sample data points from the distributions and count how many data points fall in the overlapping regions and non-overlapping areas. See the figure 3.5, we just need to count how many brown, green and blue data points

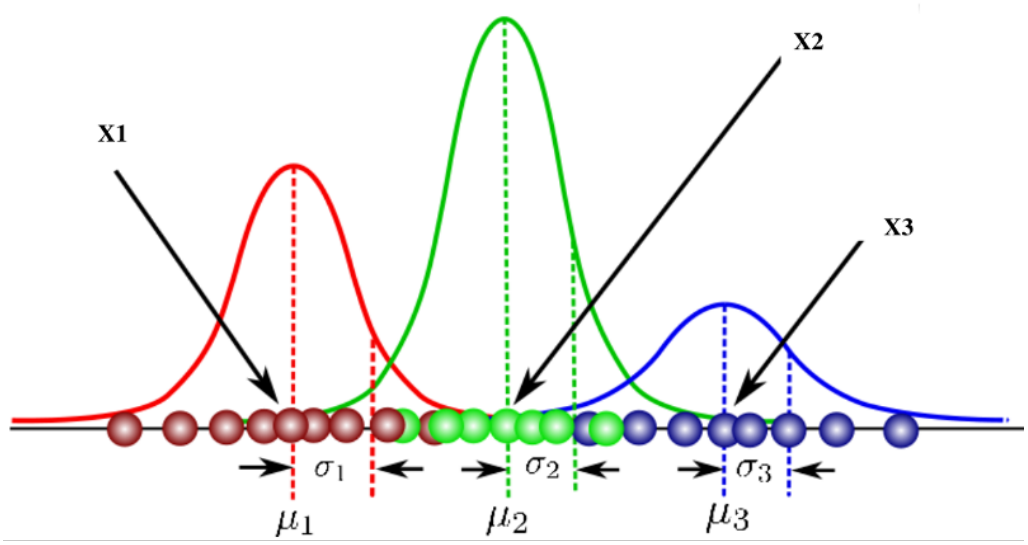

Figure 3.5: Random trees

within each area and divided by total time points, then we could get the proportion of overlapping area.

3.6.2 Properties of LIVE algorithm

3.6.2.1 LIVE algorithm is consistent

As mentioned above, we have $\widehat{f}(x; D)$ from deep learning model for dataset D, we also have decision tree function $\widehat{g}(x; D)$, suppose we have m decision trees, $m \to \infty$, and for each tree we randomly select the features, see figure 3.6, so that decision trees could cover all the combinations of different features.Within these trees, we can identify the population samples for each data point. The law of large number states that if you repeat an experiment independently a large number of times, the average of the results should be close to the expected mean.In our LIVE

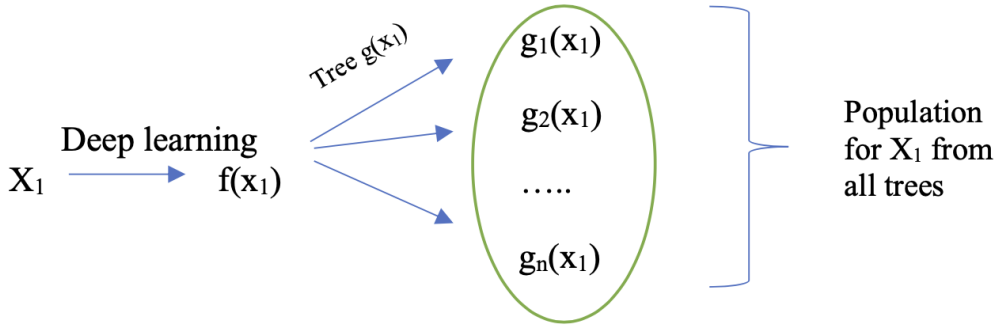

Figure 3.6: LIVE is consistent

algorithm, each tree could be considered one experiment, and the multiple trees are different samples from the overall population.This suggests that estimates from LIVE algorithm are consistent and close to the population mean.

$$\lim_{m\to\infty} g(x, T^m, D_n) = \widehat{f}(x) \tag{3.8}$$

where T is the decision tree, m is the number of trees, and D_n is number of randomly selected features.

3.6.2.2 LIVE algorithm is an ensemble approach

The purpose of ensemble algorithm is to combine multiple weak learners to iteratively improve the model results.We believe one simple Gaussian distribution may not be sufficient for all the outputs from deep learning models. The mixture distribution should be more appropriate and flexible for the unknown distribution.We try to use decision trees as our weak learners to classify the input data into different clusters to mimic the mixture distributions to reduce the bias.

1. Bias and variance

For a given dataset $D = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, suppose we have a true function f(x) such that

$$y = f(x) + \epsilon, \tag{3.9}$$

where ϵ is the noise, and f(x) is the true function of the data, which is unknown. We need to find a function to approximate the true function.

$$\widehat{f}(x, D) \xrightarrow{approximate} f(x) \tag{3.10}$$

where $\widehat{f}(x, D)$ is the model we build based on the training data, and our purpose is to find a function which is close to the true function.

The mean squared error is defined as:

$$E_{x,y,D}[(f_D(x) - y)^2] = \underbrace{E_{x,D}[(f_D(x) - \widehat{f}_{x,D})]^2}_{Variance} + \underbrace{E_{x,y}[(\widehat{f}_{x,D} - f(x))]^2}_{Bias} + \underbrace{E_{x,y}[(f(x) - y)]^2}_{Noise} \tag{3.11}$$

Here variance is defined to measure how much the model output will changes with different training sets. Bias is to measure the difference between the built model and the true function, which is the inherent error. Noise is the error to measure ambiguity due to the data distribution and feature representation [133–135].

Since in our LIVE algorithm, we only use one training dataset, so only one prediction is from our deep learning model, thus we don't have variance in our equation. The noise term is nothing we can do about it. What we can improve is the bias using our LIVE algorithm.

The bias from deep learning model is defined as

$$Bias_{DL} = \sum_{i=1}^{N} (\widehat{f}_{x,D} - f(x))^2 \tag{3.12}$$

In our LIVE algorithm, we replaced $\widehat{f}_{x,D}$ with $\widehat{g}_{x,D} = a_{11}\widehat{f}(x_1) + a_{12}\widehat{f}(x_2) + ... + a_{1n}\widehat{f}(x_n)$. Thus, the true function will be changed to equation 3.13 by simple linear transformation.

$$f(x, \widehat{g}(x, D)) = a_{11}f(x_1) + a_{12}f(x_2) + ... + a_{1n}f(x_n) \tag{3.13}$$

we use $f(x)^*$ to represent the new true function, so

$$Bias_{DL} = \sum_{i=1}^{N} (\widehat{g}_{x,D} - f(x)^*)^2 \tag{3.14}$$

To show our LIVE algorithm could reduce bias, we need to prove that $Bias_{LIVE} \leq Bias_{DL}$, and we will use induction proof method.

2. Proof

$$Bias_{DL} = \sum_{i=1}^{N} (\widehat{f}_{x,D} - f(x))^2 \tag{3.15}$$

Base case:

- When we have n=2 data points and there is no overlapping

  The bias from deep learning model is

$$Bias_{DL} = \sum_{i=1}^{2} (\widehat{f}_{x,D} - f(x))^2 = (\widehat{f}(x_1) - f(x_1))^2 + (\widehat{f}(x_2) - f(x_2))^2 \tag{3.16}$$

  In this case, since no overlapping between two curves, then for data point x_1, it has a_1=1, and a_2=0, and for data point x_1, it has a_1=0, and a_2=1, where a_1 and a_2 are mixing coefficients;

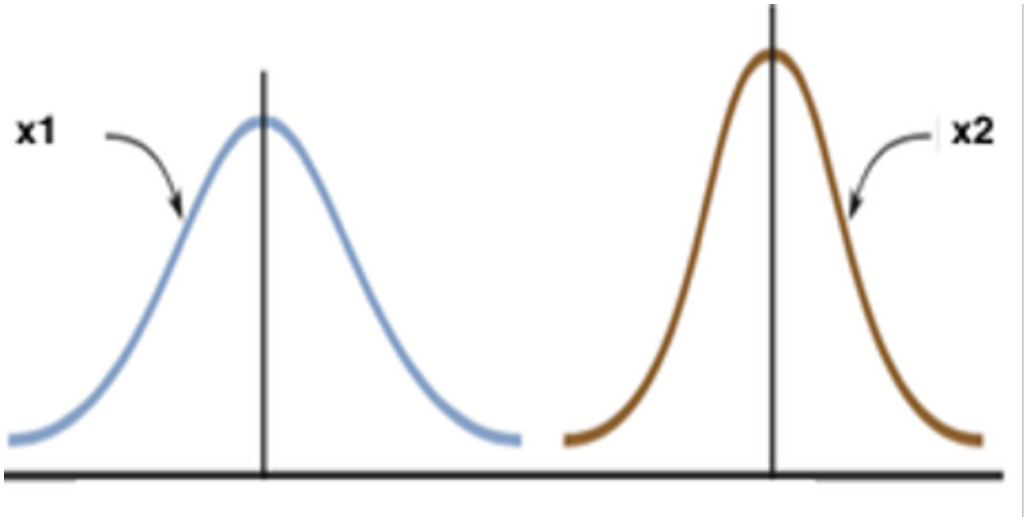

Figure 3.7: Two curves without overlapping

for data point 1:

$$\begin{aligned} \widehat{g}(x,(x_1) &= a_1\widehat{f}(x_1) + a_2\widehat{f}(x_2) = 1\widehat{f}(x_1) + 0\widehat{f}(x_2) = \widehat{f}(x_1) \\ f(x,(x_1)) &= a_1 f(x_1) + a_2 f(x_2) = 1f(x_1) + 0f(x_2) = f(x_1) \end{aligned} \tag{3.17}$$

for data point 2:

$$\begin{aligned} \widehat{g}(x,(x_2)) &= a_1\widehat{f}(x_2) + a_2\widehat{f}(x_1) = 1\widehat{f}(x_2) + 0\widehat{f}(x_1) = \widehat{f}(x_2) \\ f(x,x_2) &= a_1 f(x_2) + a_2 f(x_1) = 1f(x_2) + 0f(x_1) = f(x_2) \end{aligned} \tag{3.18}$$

- When we have n=2 data points and there is overlapping

  For deep learning model, since we assume there are no correlations between these two curves, so it has the same bias as the one without overlapping. But for LIVE algorithm, we need to consider the overlapping part. Since we have two curves, the mixing coefficients should be the same for these two data points.When we build the decision trees,we use all observations, so each data point will appear in each tree, so we have m samples for each data point.

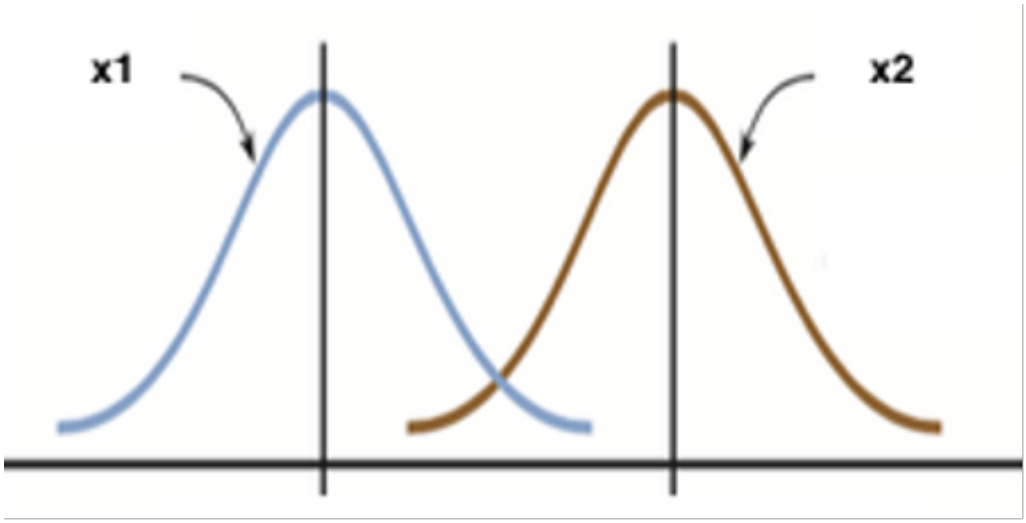

Figure 3.8: Two curves without overlapping

For data point 1,

$$\begin{aligned} \widehat{g}(x, x_1) &= a_1 \widehat{f}(x_1) + a_2 \widehat{f}(x_2) \\ g(x, x_1) &= a_1 f(x_1) + a_2 f(x_2) \end{aligned} \tag{3.19}$$

For data point 2:

$$\begin{aligned} \widehat{g}(x, x_2) &= a_1 \widehat{f}(x_2) + a_2 \widehat{f}(x_1) \\ g(x, x_2) &= a_1 f(x_2) + a_2 f(x_1) \end{aligned} \tag{3.20}$$

$$
\begin{aligned}
Bias_{LIVE} &= \sum_{i=1}^{2} (\widehat{g}_{x,D} - f(x))^2 \\
&= (a_1 \widehat{f} x_1 + a_2 \widehat{f} x_2 - (a_1 f(x_1) + a_2 f(x_2)))^2 \\
&+ (a_2 \widehat{f} x_1 + a_1 \widehat{f} x_2 - (a_2 f(x_1) + a_1 f(x_2)))^2 \\
&= (a_1(\widehat{f}(x_1) - f(x_1)) + a_2(\widehat{f}(x_2) - f(x_2)))^2 \\
&+ (a_2(\widehat{f}(x_1) - f(x_1)) + a_1(\widehat{f}(x_2) - f(x_2)))^2 \\
&= (a_1(\widehat{f}(x_1) - f(x_1)))^2 + (a_2(\widehat{f}(x_2) - f(x_2)))^2 \\
&+ 2a_1a_2(\widehat{f}(x_1) - f(x_1))(\widehat{f}(x_2) - f(x_2)) \\
&+ (a_2(\widehat{f}(x_1) - f(x_1)))^2 + (a_1(\widehat{f}(x_2) - f(x_2)))^2 \\
&+ 2a_1a_2(\widehat{f}(x_2) - f(x_2))(\widehat{f}(x_1) - f(x_1)) \\
&= (a_1^2 + a_2^2)(\widehat{f}(x_1) - f(x_1))^2 + (a_1^2 + a_2^2)(\widehat{f}(x_2) - f(x_2))^2 \\
&+ 4a_1a_2(\widehat{f}(x_1) - f(x_1))(\widehat{f}(x_2) - f(x_2))
\end{aligned}
\tag{3.21}
$$

$$
\begin{aligned}
Bias_{DL} - Bias_{LIVE} &= (\widehat{f} x_1 - f(x_1))^2 + (\widehat{f} x_1 2 - f(x_2))^2 \\
&\quad - [(a_1^2 + a_2^2)(\widehat{f}(x_1) - f(x_1))^2 + (a_1^2 + a_2^2)(\widehat{f}(x_2) - f(x_2))^2 \\
&\quad + 4a_1a_2(\widehat{f}(x_1) - f(x_1))(\widehat{f}(x_2) - f(x_2))] \\
&\quad = (1 - a_1^2 - a_2^2)(\widehat{f} x_1 - f(x_1))^2 + (1 - a_1^2 - a_2^2)(\widehat{f} x_2 - f(x_2))^2 \\
&\quad - 4a_1a_2(\widehat{f}(x_1) - f(x_1))(\widehat{f}(x_2) - f(x_2))
\end{aligned}
\tag{3.22}
$$

We know that a_1+a_2 =1, so we replace 1 with $(a_1+a_2)^2$ in above equation,

$$
\begin{aligned}
&Bias_{DL} - Bias_{LIVE} = ((a_1+a_2)^2 - a_1^2 - a_2^2)(\widehat{f}x_1 - f(x_1))^2 \\
&+ ((a_1+a_2)^2 - a_1^2 - a_2^2)(\widehat{f}x_2 - f(x_2))^2 + 4a_1a_2(\widehat{f}(x_1) - f(x_1))(\widehat{f}(x_2) - f(x_2)) \\
&= 2a_1a_2(\widehat{f}x_1 - f(x_1))^2 + 2a_1a_2(\widehat{f}x_2 - f(x_2))^2 - 4a_1a_2(\widehat{f}(x_1) - f(x_1))(\widehat{f}(x_2) - f(x_2)) \\
&= 2a_1a_2((\widehat{f}x_1 - f(x_1))^2 + (\widehat{f}x_2 - f(x_2))^2 - 2(\widehat{f}(x_1) - f(x_1))(\widehat{f}(x_2) - f(x_2)) \\
&= 2a_1a_2((\widehat{f}x_1 - f(x_1)) - (\widehat{f}x_2 - f(x_2)))^2 \geq 0
\end{aligned}
\tag{3.23}
$$

Our equation indicates that bias from deep learning model is always larger or equal to bias from LIVE algorithm. When there is no overlapping, our LIVE algorithm has the same bias as the deep learning model.

- Induction step

  Assume we have n data points from x_1,x_2,...,x_3, and they have smaller bias than the bias from deep learning model, thus we have

$$
\begin{aligned}
Bias_{DL} &= \sum_{i=1}^{N} (\widehat{f}_{x,D} - f(x))^2 \\
Bias_{LIVE} &= \sum_{i=1}^{N} (\widehat{f}_{g,D} - f(x)^*)^2 \\
Bias_{DL} &\geq Bias_{LIVE}
\end{aligned}
\tag{3.24}
$$

  where we treat these n number of curves as on big curve with mixture distributions, which has a new function $\widehat{f}(x, x_n)^*$, and the true function is $f(x, x_n)^*$.

  When we have the n+1 output curve and the overlapping proportion with

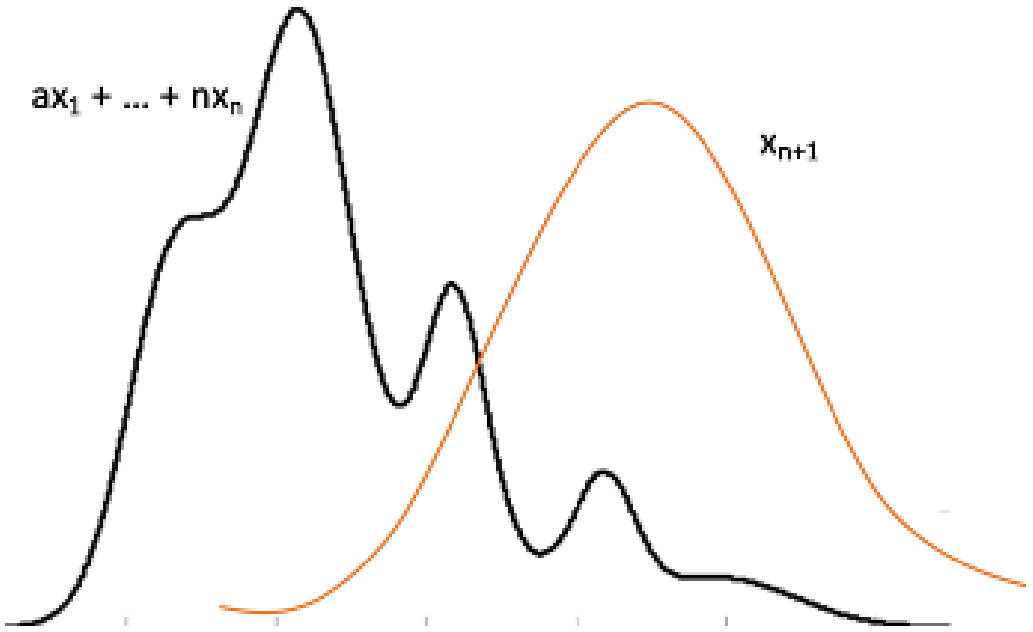

Figure 3.9: Induction step for proof of LIVE

big mixture distribution curve is a_1, thus we convert these n data point curves and n+1 data point into two data points. Similar to our base proof, we could have the following proof.

$$
\begin{aligned}
Bias_{DL} &= \sum_{i=1}^{n+1} (\widehat{f}_{x,D} - f(x))^2 \\
&= \sum_{i=1}^{n} (\widehat{f}_{x,D} - f(x))^2 + (\widehat{f}_{n+1} - f(x_{n+1}))^2 \\
&\leq \sum_{i=1}^{n} (\widehat{g}_{x,D} - f(x)^*)^2 + (\widehat{f}_{n+1} - f(x_{n+1}))^2 \\
&= ((\widehat{f}(x_n^*) - f(x)^*)^2 + (\widehat{f}_{n+1} - f(x_{n+1}))^2 \\
Bias_{LIVE} &= \sum_{i=1}^{2} (\widehat{g}_{x,D} - f(x)^*)^2
\end{aligned}
\tag{3.25}
$$

...

$$
\begin{aligned}
&= (a_1^2 + a_2^2)(\widehat{f}(x_n^*) - f(x_n^*))^2 + (a_1^2 + a_2^2)(\widehat{f}(x_{n+1}) - f(x_{n+1}))^2 \\
&- 4a_1 a_2 (\widehat{f}(x_1) - f(x_1))(\widehat{f}(x_2) - f(x_2))
\end{aligned}
$$

Similarly, we replace 1 with $(a_1 + a_2)^2$, and use the same approach, then we can get the bias difference between deep learning model and LIVE algorithm.

$$
Bias_{DL} - Bias_{LIVE} = 2a_1 a_2((\widehat{f}x_n^* - f(x_n^*)) - (\widehat{f}x_{n+1} - f(x_{n+1})))^2 \geq 0 \tag{3.26}
$$

The equation could be further expressed as:

$$
Bias_{DL} - Bias_{LIVE} = 2\sum_{i=1}^{m} \sum_{j=1, j\neq i}^{m} a_i a_j (\widehat{f}(x_i) - f(x_i) - \widehat{f}(x_j) - f(x_j))^2 \tag{3.27}
$$

Where a_i,a_j are the overlapping between any two data points, which could be zero.

Thus, we prove that our LIVE algorithm always provides smaller bias compared to the bias from deep learning models along, which means that our LIVE could

potentially achieve better performance. The magnitude depends on the overlapping part among data points and the difference between model function and the true function.

Similarity and difference between LIVE algorithm and existing literature

Che proposed an interpretable deep models for ICU outcome prediction in 2017 [5]. Their model structure is in Fig 3.10 . They first train a deep learning model to get the predictions, then use these predictions as outcomes, and build a Gradient Boosting tree model. Their interpretation algorithm achieved comparable model performance as the original deep learning models.

LIVE algorithm and their approach share some commonalities that :

1. Both of them use an ensemble approach to combine deep learning model and tree models.

2. The average predictions for each instance from tree models could be expressed as a mixture distribution, which is a consistent estimator.

The differences between these two methods are:

1. The tree model structure in their approach is based on predictions from deep learning model, ours is based on raw outcomes.

2. They try to interpret the results from deep learning models, LIVE targets the outcomes from tree models.

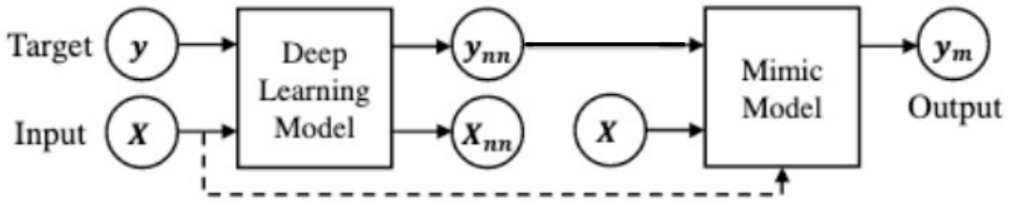

Figure 3.10: ICU model structure in literature [5]

Table 3.1: Sample node information with features in the rule path

Node	Features in the rule path
737	[p82, p95, p70, p153, p82, p37, p160, p198, p73, p214]
673	[p77, p126, p90, p132, p126, p186, p195, p23, p51, p232, p51, p0, p186, p187, p90]
740	[p97, p132, p188, p97, p150, p59, p244]
78	[p94, p85, p153, p64, p134, p70, p230, p232, p234, p59, p217, p182, p181, p52, p165, p59]

3.6.2.3 LIVE algorithm is causal interpretable

From LIVE algorithm part 2, we extract the rules from each tree for each data point.Sample rules could be found in table 3.1.

Suppose we have m decision trees and we would like to know the interpretation for input data point x1, which could help us explain how we get the predicted values for x1.

In decision tree 1, following rule path, see figure 3.11, for x_1, we can see that:

$root \rightarrow feature1 \rightarrow feature2 \rightarrow x_1$, so we have $y_{1x_1} = f_1 + f_2$, where y_{1x_1} is the average predicted scores within that node for x1 in tree 1, f1 is the feature 1, which is a binary value to show whether the feature exists in the rule path or not,f2

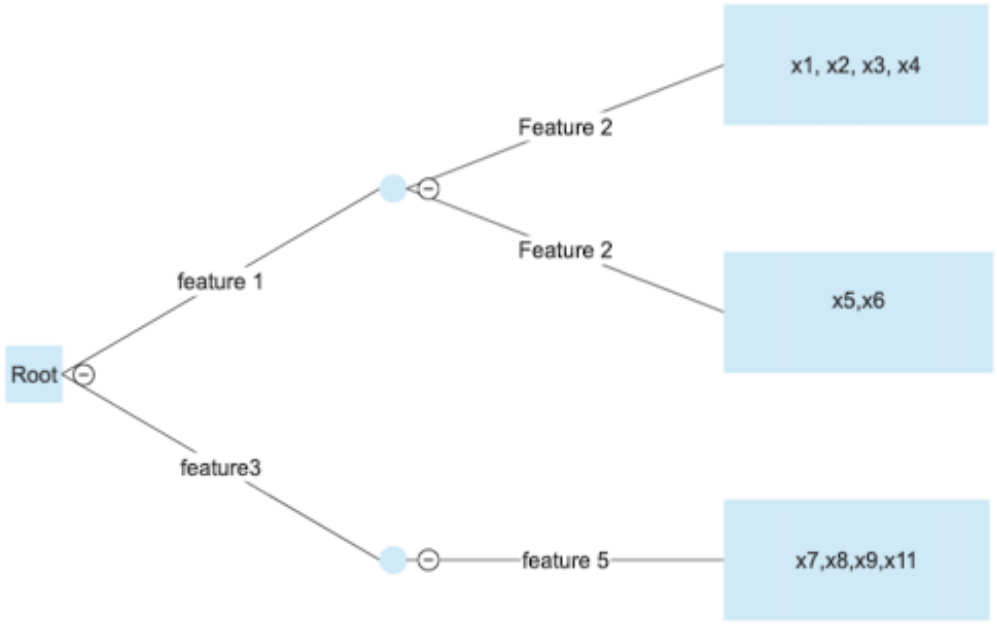

Figure 3.11: LIVE algorithm is interpretable (1)

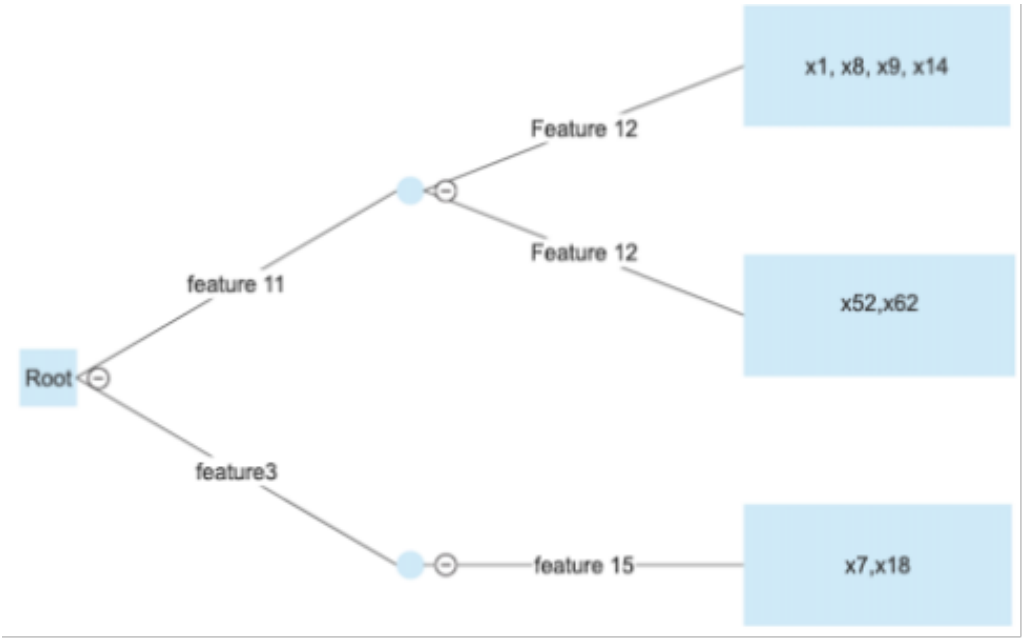

Figure 3.12: LIVE algorithm is interpretable (2)

is the feature 2 in the rule path.

In decision tree 2, see figure 3.12, we have another rule:

$root \rightarrow feature11 \rightarrow feature12 \rightarrow x_1$, so we have another equation $y_{bx_1} = f_{11} + f_{22}$. For each input data point, after we collect the formulas from all trees, we could have a sparse linear regression model for that data point. We call it sparse regression model since for each tree, we only have limited number of features in the rule path, all other features are set to 0.

If we repeat this process for all b decision trees, we will get a linear regression matrix described below:

$$\begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_b \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ ... \\ 1 \end{bmatrix} \beta_0 + \begin{bmatrix} f_1 & f_2 & f_3 & ... \\ f_1 & f_2 & f_3 & ... \\ ... & & & \\ f_1 & f_2 & f_3 & ... \end{bmatrix} \beta = \begin{bmatrix} 1 \\ 1 \\ ... \\ 1 \end{bmatrix} \beta_0 + \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & ... \\ 0 & 0 & 1 & 1 & 0 & ... \\ ... & & & & & \\ 1 & 0 & 0 & 1 & 1 & ... \end{bmatrix} \beta \quad (3.28)$$

For dataset with large number of features, for example, image data may have thousands of features (pixels) in the image, we may not have all combinations of the data points. Sparse data is common in this situation.

Most of current interpretability methods are correlation interpretation, not causal inference. The challenges are due to confounding factors and highly correlated features. The potential solutions in our LIVE algorithm are :

1) randomization through random subspace method,where randomization could reduce the probability to have two correlated features in the same model;

2) tree models, where decision tree is robust to multicollinearity. If two features are highly correlated, only the one with good information gain is kept.

3) LASSO approaches.LASSO is the Least Absolute Shrinkage and Selection Operation, a widely used method for feature selection.

To address the issue of multicollinearity, we use decision tree for variable selection. Decision tree is immune to multicollinearity.Based on an impurity measure like Gini or entropy,if features A, B are heavily correlated,only one is kept in the rules, so multicollinearity is not an concern for decision tree [136].

To answer questions like what the model would be if we remove a feature, we use random subspace approach to mimic the random clinical trial experiment design. We code the feature in the rule set as 1 or 0 , which is similar to treatment variable in clinical trial study. Since we randomly choose features, thus each feature has the same probability to be selected in the tree.Overall,the number of equations with and without that feature should be similar, which is more like a randomized experiment.These steps could enable us to draw potential causal inference conclusions. Suppose we have 3 features (a,b,c), we can demonstrate our experiment design in different scenarios:

1. a,b,c are independent

   When we randomly select 2-3 features, we could have the selection sets bc, ac, ab, abc, we will have four equations below:

$$y_{bc} = f_b + f_c; where f_b, f_c \in 0, 1 \tag{3.29}$$

$$y_{ac} = f_a + f_c; where f_a, f_c \in 0, 1 \tag{3.30}$$

$$y_{ab} = f_a + f_b; where f_a, f_b \in 0, 1 \tag{3.31}$$

$$y_{abc} = f_a + f_b + f_c; ; where f_a, f_b, f_c \in 0, 1 \tag{3.32}$$

Apparently, from the above equations, we could easily derive what is the effect for each feature.

2. a and b are highly correlated, c is independent of a,b

   Since decision tree only selects one feature if two features are highly correlated, so a,b should never be in one set, thus our selection set will be ac,bc,a,ac

$$y_{bc} = f_b + f_c; where f_b, f_c \in 0, 1 \tag{3.33}$$

$$y_{ac} = f_a + f_c; where f_a, f_c \in 0, 1 \tag{3.34}$$

$$y_a = f_a; where f_a \in 0, 1 \tag{3.35}$$

Also, from the above equations, we could calculate the model effect for each feature, which is not affected by multicollinearity issue since those highly correlated features are not in the same model.

In our regression data, there are a lot of zeros in each observation, especially for image data. There are multiple ways for sparse data analysis. Regularization

is one of the mostly widely used methods for feature selection through adding an additional penalty term for extra more covariates. Least Absolute Shrinkage and Selection Operator (LASSO) is a popular regularization approach, where LASSO uses L1 regularization. L1 function is used to remove less important variables from the model entirely. So generally, if the features on the edges could not contribute to the model identification, these features will be forced to have 0s for their coefficients. L2 regularization does not result in any elimination of sparse models or coefficients. Thus, Lasso Regression is easier to interpret as compared to the ridge regression [137]. The LASSO approach could filter out unimportant features.

3.6.2.4 Complexity of LIVE algorithm

The LIVE algorithm includes both deep learning model and decision tree model part. The complexity of deep learning models relies on the deep learning model structure, which varies based on the chosen model. Our focus is the complexity of decision trees. Our tree models are very similar to random forest model, thus they have the similar complexity, which could be generalized as big O(mxsxlog(n)) , where m is number of trees, s is the feature size used in the decision tree, n is the number of samples.For image data, if we treat each pixel as one feature, the feature size will be the image size (width x height in pixels). The complexity will be squared, supposing the width=height, for example, from 32x32 to 128x128, the big O will be increased 16 times. This could be a critical challenging part for large image data using our LIVE algorithm. Besides the complexity of space and time,

in order to get more accurate clustering information, we need as many samples as possible.

There are two potential ways to solve this issue:

- Parallel computing.

  There are already some existing packages for parallelization of decision tree algorithms or random forest. We can adapt their functions to make LIVE algorithm parallel for the decision tree part.

- Using smaller image size for clustering step.

  As we can see that the complexity is heavily affected by the feature size, so we may use smaller feature size. For image data, we may reduce the image size, which may sacrifice the accuracy of the LIVE algorithm, it could significantly reduce the time and space complexity. For example, we may use 32x32 to cluster the data, but use 128x128 image size for deep learning models.

- For image data, we may also try image segmentation to group small pixels to large areas, thus it could reduce image complexity.

3.6.3 Comparison to other interpretability methods

We would like to compare our LIVE algorithm to other popular interpretability methods. In table 3.2, we summarize the differences among these methods.

First, for model accuracy, we mathematically prove that our LIVE could reduce model bias which could lead to better model accuracy. On the contrary, SHAP and

Table 3.2: Differences between LIVE algorithm and 4 other interpretability methods

Properties	Our LIVE	SHAP	LIME	GRAD CAM
Model accuracy	Better	Same	Same	NA
Interpretation	Causal	Correlation	Correlation	Correlation

LIME methods are local accurate, which means they maintain the same accuracy as deep learning model. Grad CAM does not contribute to the model accuracy, and only provides visualization tools to identify important regions. It is claimed that ICU model could achieve comparable deep learning models.

Second, for model interpretation methods, such as SHAP, LIME and GRAD CAM , they utilize traditional correlation interpretations, where they ignore the correlations among the features.In our LIVE algorithm,we try different ways to reduce the correlated features through randomization, tree models and LASSO approach. Recoding the feature value to 1 or 0 could explain what the model could change if we have the feature in the model.These could lead to the interpretation to potential causal inference.

Chapter 4: Results

We use two types of datasets to validate our LIVE algorithm: image data and structured data. Since those two types of data are quite different in the structures and need different models, we would like to discuss the results in two separate sections.

4.1 Results for image data

Our image experiments include 2 public image datasets from Kaggle (www.kaggle.com).

4.1.0.1 Vingroup dataset

Vingroup big data institute organized a Kaggle competition using their 18,000 chest radiographs. They released 15,000 images and labeled the regions where the images have diseases. The labels have 14 classes, which include 13 different diseases and normal class. To simplify the task, we recode the labels into normal and diseased. We randomly take 9,312 images from the total images. Among these images, there are 5,788 normal images and 3,524 images with disease. The image size is 2836x2336.

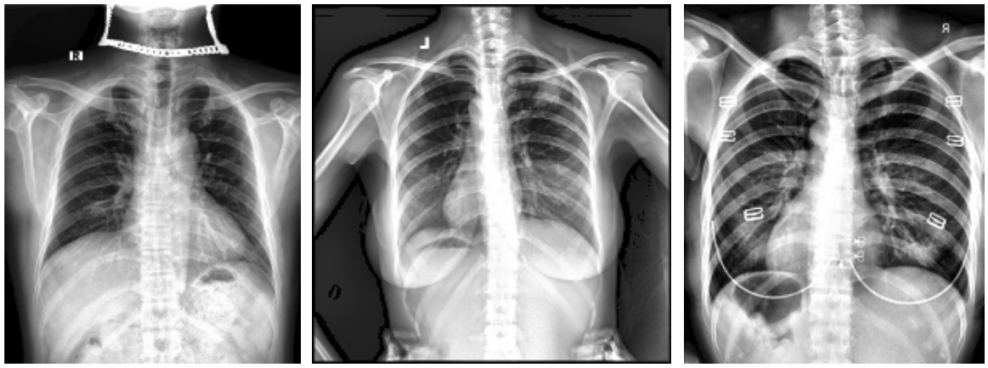

Figure 4.1: Sample images from Vingroup big data

4.1.0.2 Pneumonia x ray dataset

This image data is provided by a research group from University of California, San Diego, where they collected and labeled a total of 5,232 chest X-ray images from children, including 3,883 characterized as depicting pneumonia (2,538 bacterial and 1,345 viral) and 1,349 normal from a total of 5,856 patients. In the test dataset, there are 234 normal images and 390 pneumonia images (242 bacterial and 148 viral) from 624 patients. The image size is 1024x1024.

4.1.1 Model structures

Since our LIVE algorithm is based on model output,not inner structures of the model,we would like to test whether our algorithm works for different model structures.

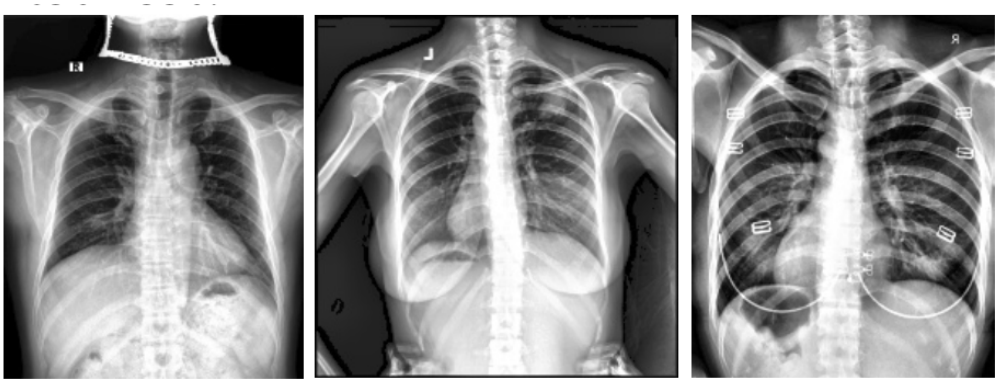

Figure 4.2: Sample image from Pneumonia data

4.1.1.1 CNN model

In our two image analysis, we use a 7-layer CNN model with dropout rate 0.2. The general procedures to construct the CNN model from scratch are listed below:

The network structure is illustrated in figure 4.3. There are 7 convolution layers in our network structure. After each convolution layer, there is a normalization layer. After that there is max pooling layer and drop out layer. The last layer of our model is a fully connected layer. Our self defined model has the comparable performance as the one reported in literature. It is reported by Vingroup institute that their average AUC for all 14 diseases is 0.93. The highest AUC of our own CNN model is 0.929 [39].

For pneumonia dataset,it is reported that their AUC is 0.968, our AUC is 0.949. However their model is based on a much larger dataset with 108,312 images and retrained with pneumonia model [16]. In a recent study they achieved AUC 0.95 using pneumonia image data alone with a deep learning model, which is the

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, 16, 16, 1)	0
conv2d_1 (Conv2D)	(None, 16, 16, 32)	160
batch_normalization_1 (Batch	(None, 16, 16, 32)	128
conv2d_2 (Conv2D)	(None, 16, 16, 32)	4128
batch_normalization_2 (Batch	(None, 16, 16, 32)	128
conv2d_3 (Conv2D)	(None, 16, 16, 64)	8256
batch_normalization_3 (Batch	(None, 16, 16, 64)	256
conv2d_4 (Conv2D)	(None, 16, 16, 64)	16448
batch_normalization_4 (Batch	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 64)	0
conv2d_5 (Conv2D)	(None, 8, 8, 128)	32896
batch_normalization_5 (Batch	(None, 8, 8, 128)	512
conv2d_6 (Conv2D)	(None, 8, 8, 128)	65664
batch_normalization_6 (Batch	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 128)	0
conv2d_7 (Conv2D)	(None, 4, 4, 256)	131328
batch_normalization_7 (Batch	(None, 4, 4, 256)	1024
final (Conv2D)	(None, 4, 4, 256)	262400
batch_normalization_8 (Batch	(None, 4, 4, 256)	1024
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 256)	0
dropout_1 (Dropout)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
Dense_10nb (Dense)	(None, 1)	1025

Total params: 526,145

Figure 4.3: CNN model structure

same as ours [138].

4.1.1.2 Transfer learning model

There are two major ways to customize a pretrained model:

1) Extract the fixed features

Generally, it removes the last fully connected layer, then treats the rest of layers as fixed features for the new dataset. You simply add a new classifier on top of the pretrained model.

2) Fine-tuning the pretrained model

In this approach, you not only replace and retrain the classifier on top of the pretrained model, but also fine-tune the weights of the pertained network by continuing the backpropagation. This will allow us to fine tune the feature representation in the base model.

Although there are different types of transfer learning models, they have different limitations. In table 4.1, we can see that InceptionV3 and MobileNet require the minimum image size at least 75x75 and 128x128, which may limit our analysis when we have smaller image size like 32x32 and 16x16. VGG16 and VGG19 both work for image size starting from 32x32.Since VGG16 has relative less parameters but has the same accuracy as VGG19, we chose VGG16 for our transfer learning model.We select the Adam optimizer from Keras with the learning rate of 0.001. We also tried ResNet50, which worked with image size from 32x32, but it showed poorer performance. We did not show here.

Table 4.1: Some transfer learning models [1]

Model	Size (MB)	Top-1 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Minimum image size needed
InceptionV3	92	77.90%	23.9M	189	42.2	75x75
ResNet50	98	74.90%	25.6M	107	58.2	32x32
VGG19	549	71.30%	143.7M	19	84.8	32x32
VGG16	528	71.30%	138.4M	16	69.5	32x32
MobileNet	16	70.40%	4.3M	55	22.6	128x128

VGG16 is a convolutional neural network model for large image recognition. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It is one of the famous models. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's (see figure 4.4 for the structure of VGG16) [18]. For VGG 16, we import all model weights and obtained the last layer of the pretrained model, then we add our own dense layer.

4.1.2 Experiment design

We categorize our experiment methods into the following groups and try to answer the following questions.

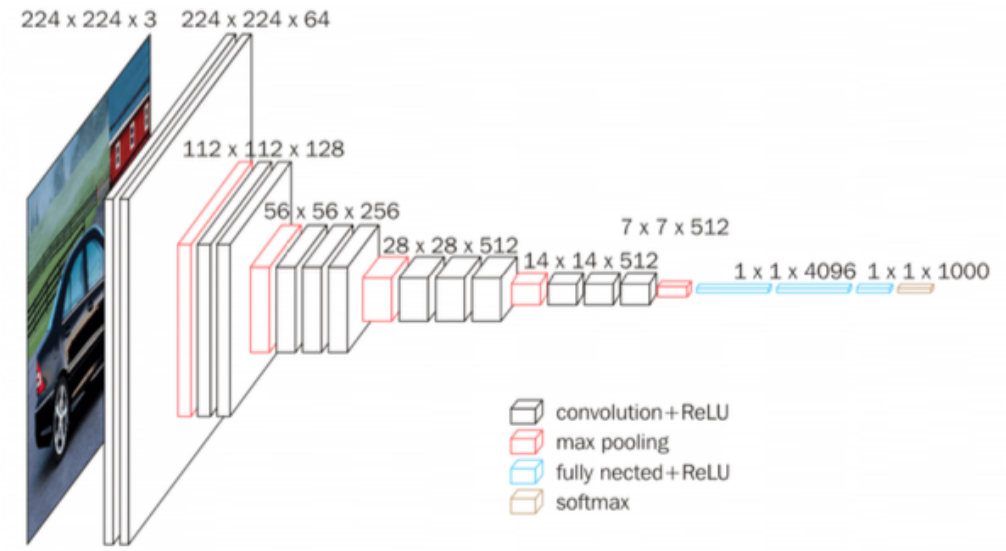

Figure 4.4: VGG 16 model structure [6]

4.1.2.1 Impact of image sizes

It is reported that larger image size may not obtain the optimal model performance. In their experiments, image size with 256x256 achieved the best performance. 32x32 image size may get close performance compared to 320x320 image size [139]. Our figure 4.5 shows one chest image with 6 different resolutions under consideration for deep learning inputs (16x16,32x32,64x64,128x128,224x224 and 512x512). Visually from these images, the low resolution samples at 16x16 and 32x32 have limited observable quality, the other images have similar quality.

The image sizes could have a large impact on our algorithm. Since decision trees are the core part of our algorithm, and it is already known that decision trees require more memory and more time. Due to the graphics processing unit memory constraints, we only evaluate the impact of image sizes from 16x16 to 512x512 using

Figure 4.5: Pneumonia images with different sizes

pneumonia dataset and Vinbig data based on AUC performance and running time.

4.1.2.2 Impact of number of epochs

Similar to random forest, we would like to determine the number of decision trees we need to use. In order to do that, we test the effect of different number of epochs on the model performance from 1 to 100 epochs.

4.1.3 Results of impact of epoches

We use the our own CNN model for both datasets. For Vinbig data, we use 70% as training and 30% as testing data, for pneumonia data we use the default training data and testing data. Figure 4.6 shows that after 30 epochs, the accuracy for training data and testing data is stable. We choose 30 epochs for all our models.

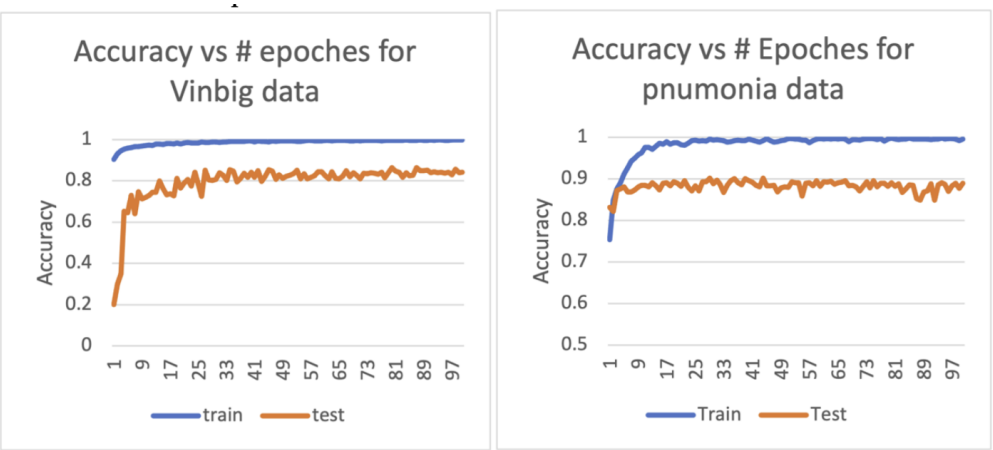

Figure 4.6: Impact of number of epochs

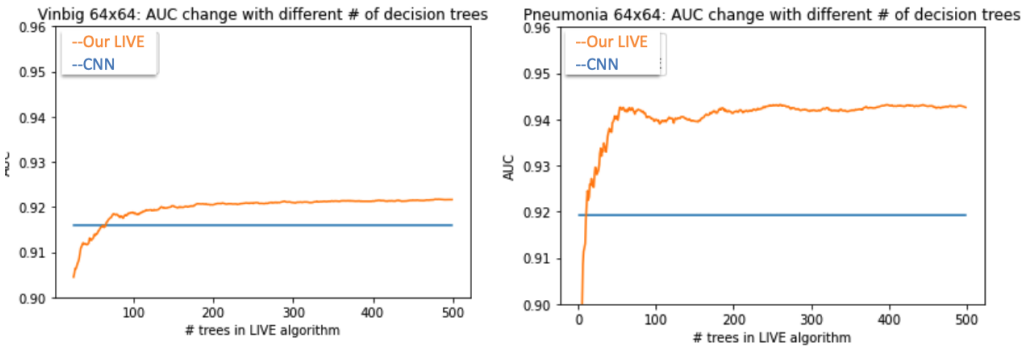

Figure 4.7: Impact of number of decision trees

4.1.3.1 Impact of number of decision trees

We use both pneumonia data and Vingbig data to check the number of decision trees needed for the analysis (image size 64x64). We find the model performance of AUC is pretty stable after 300 decision trees for both models. We chose 300 trees for all the models.

Table 4.2: Impact of different image sizes for Pneumonia data

	Our LIVE algorithm		
Image size	AUC	CNN training	Decision trees
Pneumonia		time(mins)	time (mins)
16x16	0.923	1.7	1
32x32	0.927	1.6	3
64x64	0.936	1.8	25
128x128	0.949	3.2	107
224x224	0.937	8.3	420
512x512	0.923	15.6	NA

4.1.4 Results of impact of image size

From table 4.2, we can find the training time for image size below 128 is pretty similar, however it is almost doubled after image size 128x128. For the running time of our LIVE algorithm, the whole process is extremely slow after image size 64x64,because we significantly increase the number of features after image size 32x32,so the running time is almost exponentially increased from several minutes to several hours. However,the AUCs show an increasing pattern from image size 16x16 to image size 128x128, then a decreasing trend after 128x128. Considering the trade off between running time of our LIVE algorithm and AUC of deep learning model, we decide not to pursuit LIVE algorithm for image size 512x512. Similar pattern is found in our Vinbig dataset, see table 4.3. The best performance occurred at image

Table 4.3: Impact of image sizes for Vingbig data

Image size	AUC	CNN training
Vinbig		time(mins)
16x16	0.887	1.6
32x32	0.915	2.8
64x64	0.926	1.9
128x128	0.929	3.3
224x224	0.874	7
512x512	NA	NA

size 128x128, and after that the performance decreases.

4.1.5 Performance comparison

From figure 4.8 , we can see LIVE algorithm consistently shows better performance than deep learning models alone, especially for image size below 128x128 with about 0.02-0.03 improvement in AUCs. However, this improvement is not obvious for vintage dataset with image size above 128. There are several possible reasons:

1. For Vintage images, since CNN already achieves a higher AUC around 0.95, there is not so much improvement room for LIVE algorithm.

2. Due to the running time concern, we follow the default rule from random forests, where the squared root of total features are used by default. This may not be the optimal number of features used in the model.

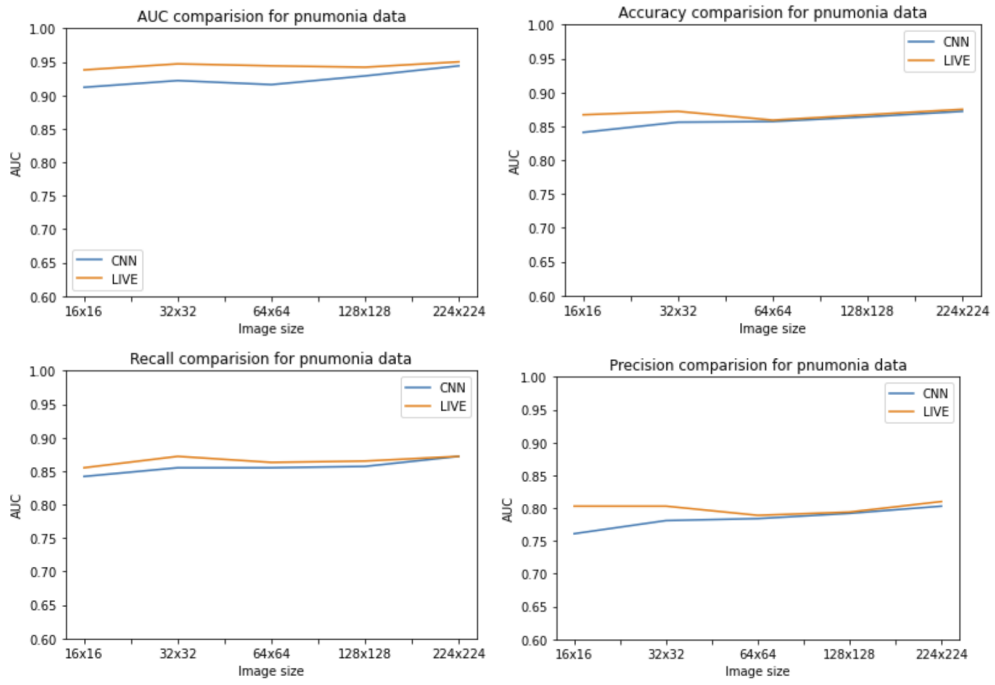

Figure 4.8: Performance for Pneumonia data CNN vs LIVE

3. Also with increased image size, we may not have enough data for clustering in decision trees. For example, the image size 128x12 has 128x128=16,384 features. However, we only have 5000 images to train the clustering algorithm, this may limit the performance of LIVE algorithm.

4.1.6 Results for transfer learning

The performance of transfer learning models with different structures shows the similar result.Our LIVE algorithm consistently performs better than transfer learning models alone. We only look at images with sizes from 32 to 128 since 32x32 is the minimum size required for transfer learning. Also, since transfer learning models require 3 channel images,we need to convert our grayscale images to color

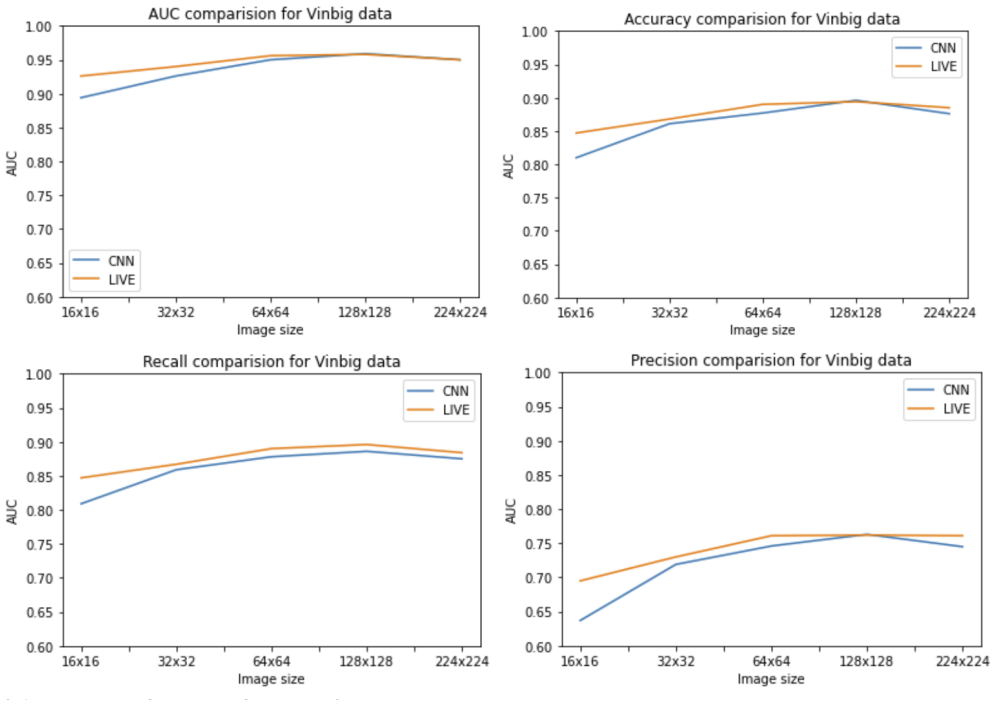

Figure 4.9: Performance for Vinbig data CNN vs LIVE

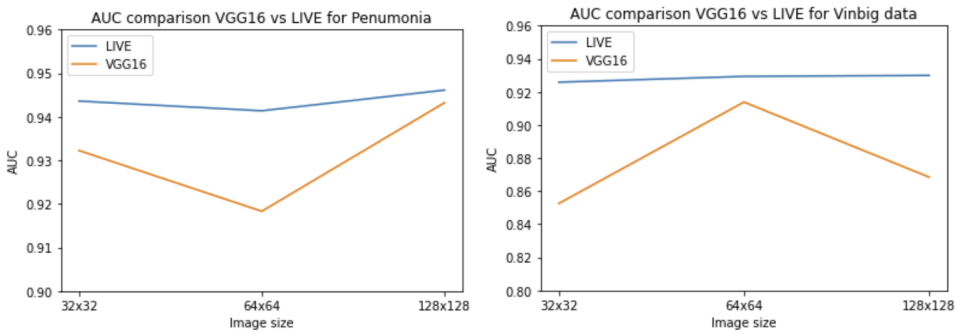

Figure 4.10: Performance for VGG16 vs LIVE

ones, which substantially increases the data size.Due to the limited computing resources on Kaggle platform, our transfer learning model were crashed at image size 224x224.

4.1.7 Performance of LIVE algorithm with smaller image size

As we see previously, the running time is almost 7 hours for image size 224x224. To overcome this issue, we explore whether we could use a smaller image size for clustering, but still use the large image size for deep learning model. In table 4.4 and table 4.5, we compare the performance of the CNN models with image size 128x128, and LIVE algorithm with different smaller image sizes. We can see even we use a much smaller image size 16x16, our LIVE algorithm could still achieve comparable performance or better performance than the deep learning models with larger image size. This is very important since slow process could be a key factor to prevent the application of our LIVE algorithm to large image size.

Table 4.4: Performance with smaller image size for Vinbig data

		CNN model with image size 128x128			
Dataset	Image size	AUC	Accuracy	Recall	Precision
Vinbig	128x128	0.959	0.896	0.886	0.763
		LIVE algorithm with small image sizes			
	Image size	AUC	Accuracy	Recall	Precision
	16x16	0.951	0.888	0.888	0.767
	32x32	0.952	0.887	0.887	0.765
	64x64	0.958	0.894	0.894	0.777

Table 4.5: Performance with smaller image size for Pneumonia data

		CNN model with image size 128x128			
Dataset	Image size	AUC	Accuracy	Recall	Precision
pneumonia	128x128	0.929	0.864	0.857	0.792
		LIVE algorithm with small image sizes			
	Image size	AUC	Accuracy	Recall	Precision
	16x16	0.939	0.861	0.859	0.788
	32x32	0.946	0.861	0.859	0.788
	64x64	0.945	0.861	0.859	0.788

4.2 Results for visualization

In figure 4.11, figure 4.12, we plot the identified regions from different algorithms. In these figures, the rectangle regions are labeled by experienced doctors, which are used to compare the accuracy of the identified regions from the different algorithms. In table **??**,we can see the coefficients from our LIVE algorithm, where the coefficients could be easily to be projected to the raw image to show the regions with interest of region. The images show our LIVE algorithm has slightly higher accuracy compared to LIME, SHAP and Grad-CAM. Our LIVE algorithm could limit the ROI to few data points, though SHAP has similar performance,SHAP values also have many more wrong points which may lead to biased directions since SHAP values use all features without considering the correlations among features. Especially in image data, the neighbor pixels are generally highly correlated.These multicollearity may lead to biased estimates,however our LIVE algorithm removes high correlated variables from the model.

Similar to SHAP values, our coefficients show how each feature contributes to the outcome.Table 4.8 shows some coefficients from the sparse regression model.The variable column is the list of some pixels in the image. The negative estimate value may suggest that pixel is not an interested pixel.The pixels with positive estimates would increase the predicted risk scores.The coefficients also show the magnitude effect on the predicted probabilities.For example, p84 has coefficient 0.09687, which means pixel 84 is our region of interest, it could increase the probability by 0.0987. Unlike feature importance, our model is really explainable by giving the contribution

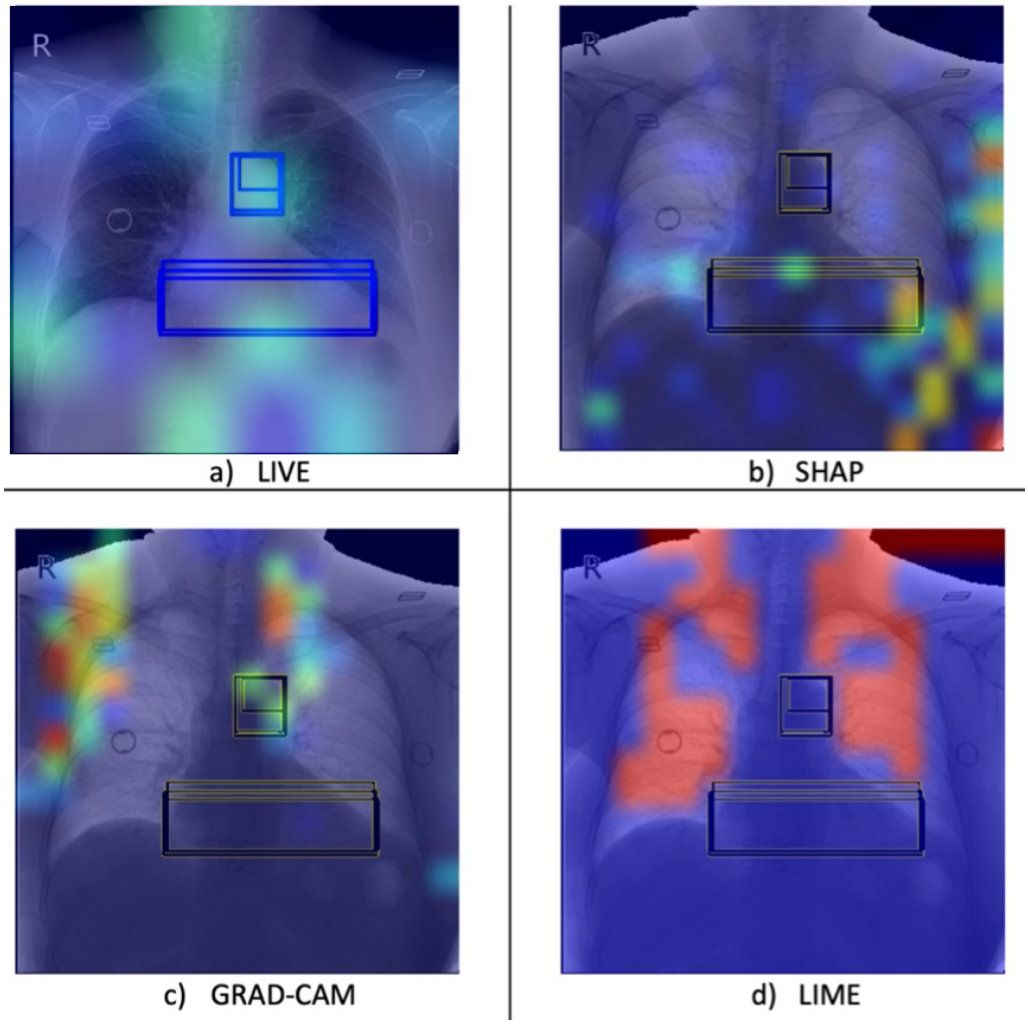

Figure 4.11: Visualization for Vinbig data 1

Table 4.6: Region of interest for Figure 4.11

0	0.40237	0	0	0
0.17272	0.28849	0	0	0.2356
0	0	0.35878	0	0
0	0.14884	0	0.17832	0
0.10174	0	0.38078	0.03525	0
0	0.4215	0.10384	0.3171	0

Table 4.7: Region of interest for Figure 4.12

0.50907	0	0.26608	0.20648	0.28083	0.07385
0	0	0	0	0	0.18528
0.018	0	0.17499	0.4316	0.26412	0.36887
0.17184	0	0	0	0.08599	0
0.43065	0	0	0	0	0.01536
0.48194	0.55527	0	0	0	0

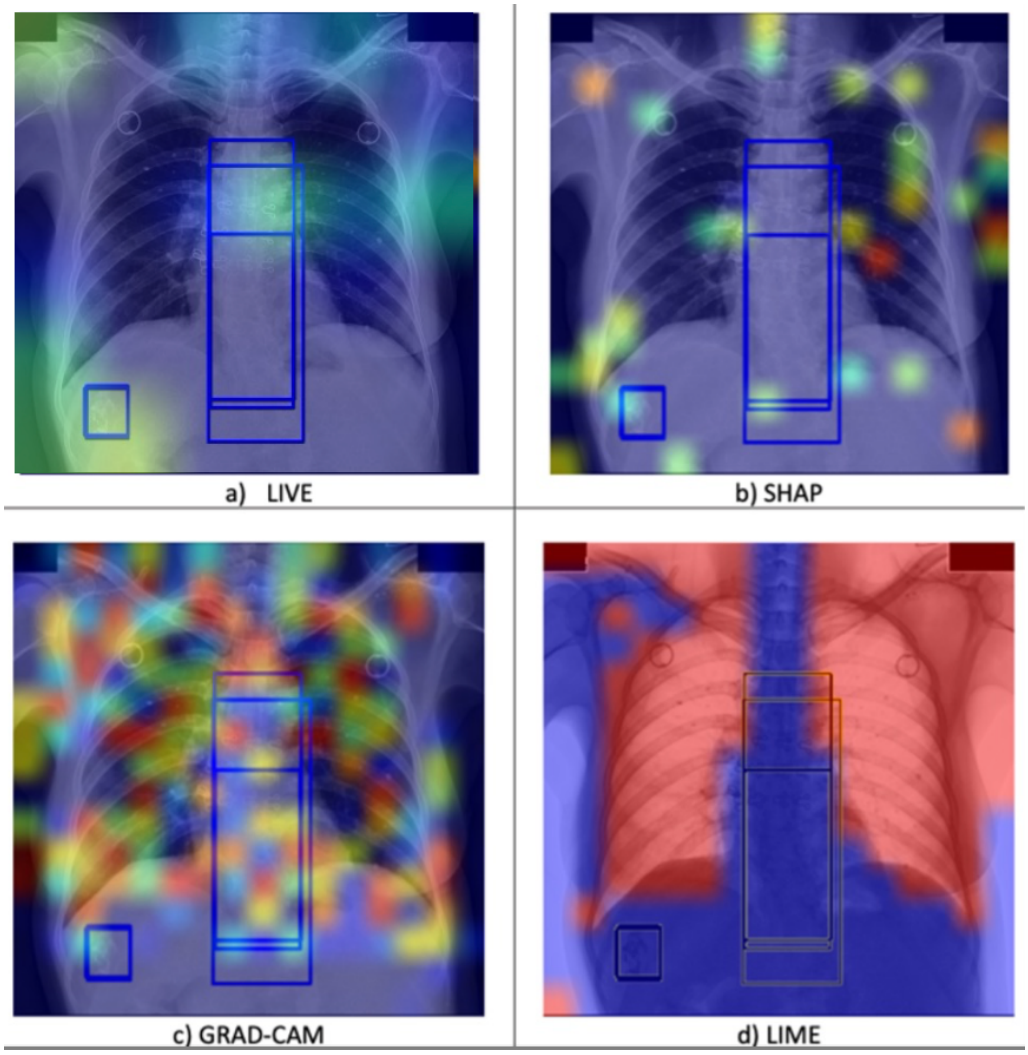

Figure 4.12: Visualization for Vinbig data 2

score for each feature.

To get region of interest for large area, we could aggregate the neighbourhood into one big areas, see table **??**, which is for figure 4.11. There are couple of methods for those interpolation, such as nearest neighbor, bilinear etc. Since the coefficients are individual contributions of the features, we can either sum them together to get the region contribution to the predictions or use the average values, or even randomly take one value from that region.We can see the value 0.358 in red color means this region contributes 0.35 to the whole predicted disease probability. The light regions could show how LIVE algorithm makes decision through the coefficients. The negative coefficients are coded as 0. Some identified regions by LIVE algorithm are beyond the labels by doctors, which may be due to the noise region in the raw images. However the two ROI are consistent with doctor's labeled region, which is better than traditional GRAD-CAM or SHAP algorithms. Similar to table 4.8, our LIVE algorithm could also identify the key regions for figure 4.12.

The ROI from GRAD-CAM and LIME is too large and may not provide much useful information for doctors.

4.3 Results for Structured dataset

4.3.1 Data source

A research group made a synthetic dataset based on the health facts database available through UC Irvine Machine Learning Repository, which could also be found in kaggle website. This dataset represented 10 years of clinical data with 130

Table 4.8: Some regression coefficients for interpretation using Vinbig data

Variable	DF	Parameter Estimate	Standard Error	t Value	P value
Intercept	1	0.56795	0.02125	26.73	<.0001
p243	1	-0.09944	0.02822	-3.52	0.0005
p84	1	0.09687	0.02891	3.35	0.0009
p134	1	0.09435	0.02817	3.35	0.0009
p118	1	0.08269	0.02886	2.87	0.0043
p125	1	0.08305	0.02957	2.81	0.0052
p223	1	0.07264	0.0285	2.55	0.0111
p228	1	0.0627	0.02826	2.22	0.027
p96	1	0.05471	0.02816	1.94	0.0526
p28	1	0.0538	0.02892	1.86	0.0635
p225	1	0.05207	0.02806	1.86	0.0641
p127	1	0.0535	0.0292	1.83	0.0675
p205	1	0.04849	0.02749	1.76	0.0783
p172	1	0.04881	0.02811	1.74	0.0831
p207	1	0.04838	0.02922	1.66	0.0985
p95	1	0.04193	0.02882	1.45	0.1464
p250	1	0.03238	0.02889	1.12	0.2629
p241	1	0.00789	0.02873	0.27	0.7839

hospitals, 100,000 encounters, and 71,518 unique patients. The study outcome is whether the patient has readmission within 30 days post hospital discharge and 56 variables were selected in the models.

4.3.2 Challenges of this dataset

4.3.2.1 Irregular time series

Irregular time series data is very common in healthcare, which may be due to different number for hospital visits for different patients, or different number of tests for each hospital visits. Often, conventional methods used for handling time series data could introduce bias and make strong assumptions on the underlying data generation process. In table 4.9, we can see the number of visits could range from 1 to 40, and 98% have less than 6 visits. The unequal numbers of visits are one type of data irregularity. GEE model and LSTM model are used for this data analysis.

- GEE model

  GEE model allows different number of visits for different patients. We used autocorrelation to control the relationship of different visits within the same patient.

- LSTM

  The LSTM model could incorporate the time spans between consecutive medical features to handle the irregularity. This category of methods demonstrates

Table 4.9: Frequency of irregular hospital visits

# visits	Total Patient	Percent	Cumulative Frequency	Cumulative Percent
1	7242	57.03	7242	57.03
2	3345	26.34	10587	83.38
3	1054	8.3	11641	91.68
4	467	3.68	12108	95.35
5	253	1.99	12361	97.35
6	337	2.65	12698	100

the possibility to fully utilize available data.

4.3.2.2 Hierarchical variables

In EHR, there are several diagnosis codes with hierarchical structures, such as ICD and CPT codes. ICD 10 codes have more than 70,000 numerical codes and CPT codes have 15,000 codes. These codes are hierarchical in nature, see table 4.10. We can see some sample ICD codes,which have different structures from the traditional variables.

Several efficient representations for ICD code using various types of neural networks have been reported. Through deep learning, high-dimensional data can be transformed to continuous real-valued concept vectors that efficiently capture their latent relationship from data. Such representations have been shown to improve the performance of various complex tasks [140].

Table 4.10: Sample diagnosis codes

diagnosis code 1	diagnosis code 2	diagnosis code 3
78	250	414
	719	438
435		
38	599	250
780	577	E942
599	250.02	41
784	782	250
250.13	276	276
522	682	

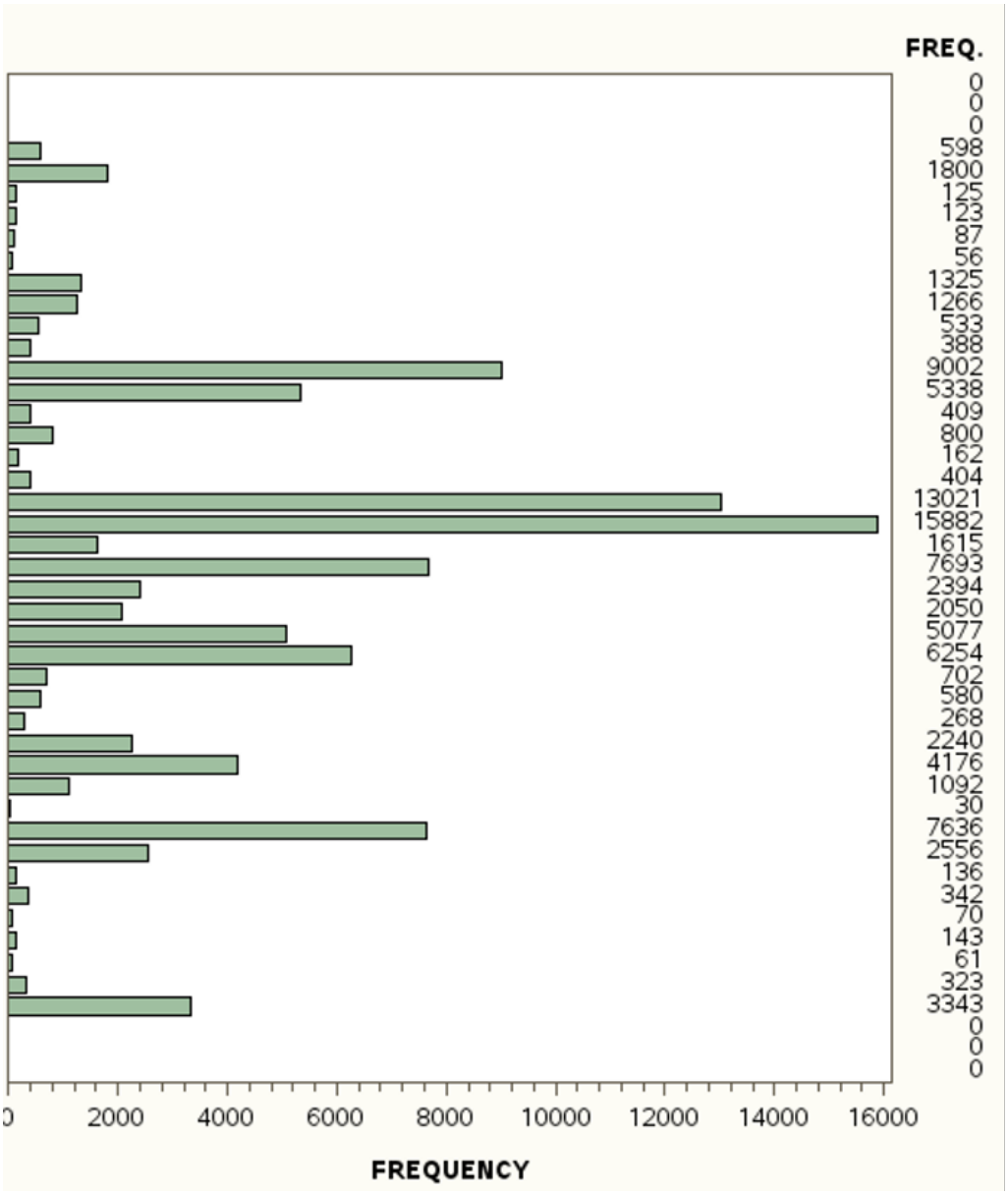

Figure 4.13: Frequency of diagnosis code

4.3.2.3 Word embedding

In this dataset, we have 769 different diagnosis codes, and each code has different frequency, see figure 4.13.This sparse variable creates a challenge for traditional model approach. To overcome this issue, we borrow some ideas from word embedding technique to address this question. Word embedding is an efficient way for dense representation in which similar words have a similar encoding.

In natural language processing (NLP), we generally use word2vec or similar packages to convert the word into numbers, then use the word embedding to get dense representation.Also, NLP could easily handle sentences with different length [141].For our data, we already have the numerical values for most of the variables,and their values represent the similarity of each number.For example, age 10 is more similar to age 20 than age 90.We first use the padding technique to fill the missing with 0 if the patient does not have that visit. After this, all patients have the same length. We treat each variable as a unique word and transform their original values by adding a different constant value so their original values could have different meanings. For example, age 10 and number of visits 10 are different. We can simply add 1000 to variable number of visits, thus they could have different values with different meanings.

4.3.3 Models for diabetes dataset

We try a LSTM model with one extra embedding layer before the LSTM layer to account for the sparsity of the diagnosis codes (figure **??**). GEE model is used

Layer (type)	Output Shape	Param #
(Embedding)	(None, 131, 8)	400000
(LSTM)	(None, 20)	2320
flatten_6 (Flatten)	None, 20)	0
dense_20 (Dense)	(None, 1)	21

Total params: 402,341
Trainable params: 402,341
Non-trainable params: 0

Figure 4.14: LSTM model with embedding layer

as the baseline model.

We also plot the model performance against the number of trees for LIVE algorithm, and find after 100 trees, the model performance is pretty stable,see figure 4.15.

4.3.4 Model results for structured data

Our baseline GEE model could achieve AUC with 0.76. Our result shows that after adding word embedding,the model performance increases from 0.76 to 0.793 (see table 4.11). The LIVE algorithm increases the model AUC from 0.793 to 0.813.

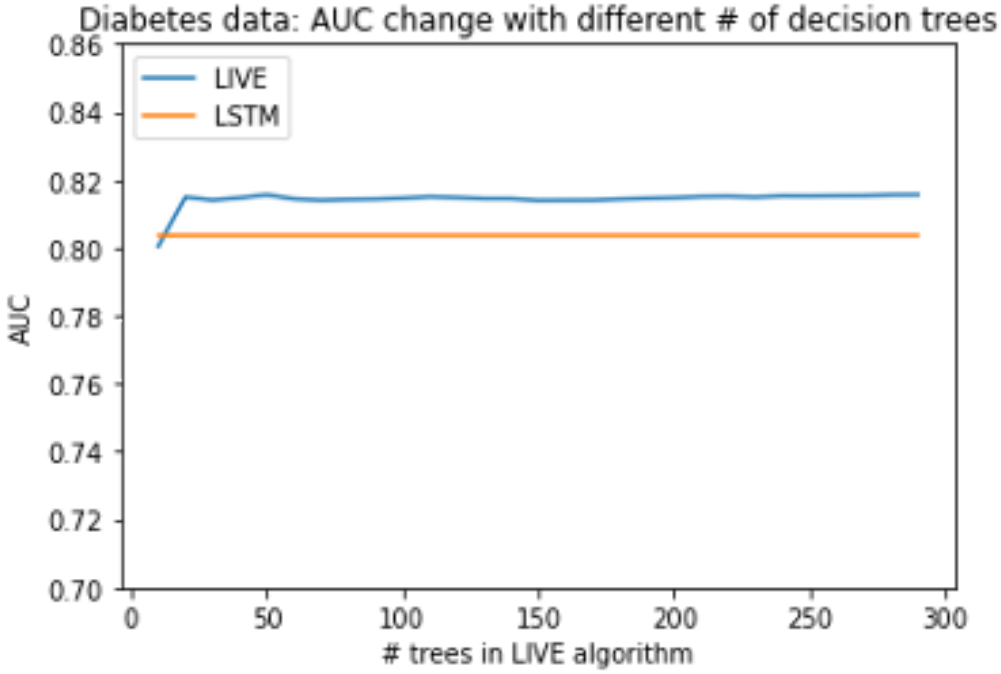

Figure 4.15: Impact of number of decision trees for Diabetes data

Table 4.11: Performance comparison between LSTM and LIVE

Algorithms	AUC	Accuracy	Recall	Precision
Repeated LR (GEE)	0.76	0.724	0.724	0.716
LSTM with embedding layer	0.793	0.745	0.745	0.737
LIVE	0.813	0.763	0.762	0.756

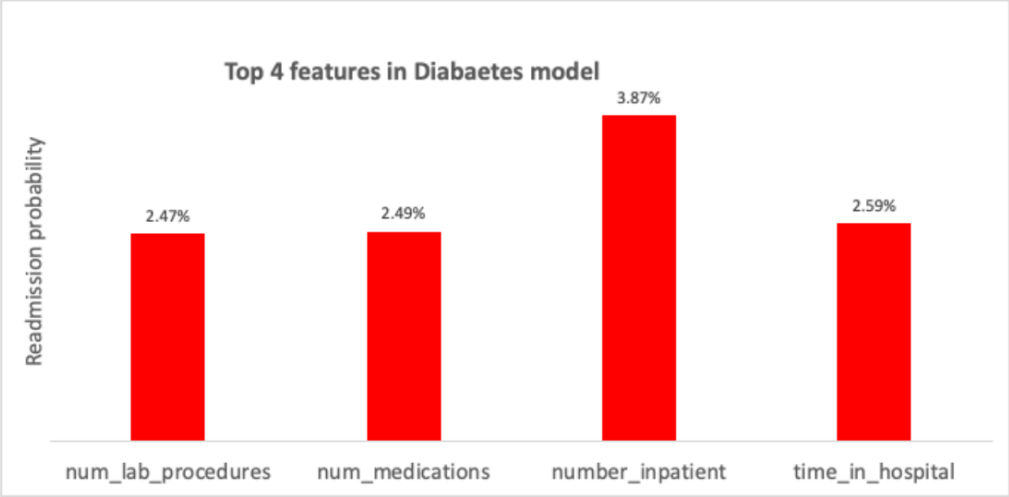

Figure 4.16: Interpretation and Visualization for diabetes data

4.3.5 Interpretation and visualization for diabetes dataset

The following figure 4.16 could demonstrate how our LIVE algorithm could help interpret the model and also the visualization explanation. Based on LASSO regression, the coefficients could represent how each variable contributes to the probability score. For example, the number of lab procedures, number of inpatients and historical time in hospitals could significantly increase the chance of readmission in the next 30 days. Since this is a linear model, the coefficients also suggest the magnitude of the risk. For example, number of inpatients could increase 0.039 in overall risk probability.

Chapter 5: Conclusion and future work

5.1 Conclusion

In this paper, we briefly reviewed the deep learning models and healthcare. We also discussed the black box property of deep learning models. Interpretability is an essential part for knowledge discovery and model justification in critical applications. We summarized the taxonomy of the existing machine learning interpretability methods, and also discussed their limitations and challenges.The accuracy of the interpretable models and the causal analysis due to high correlations among the features from these models are two key issues in the current approaches.To address these above issues, we proposed our LIVE algorithm.

We started from the predictions of deep learning model, which are assumed to be independent normal distributions. We believe there are always some subgroups within the population. Samples within one group are more similar that other samples in other groups. The mixture distribution could be better to address the heterogeneous issues in the whole samples. We treated each individual prediction as one normal distribution,and tried to find the property of the mixture distribution among the individual predictions.To calculate the mixing coefficients, we applied the random subspace method to calculate the mixing coefficients, which

could lead to a consistent estimates. We also attempted to mathematically prove that our LIVE algorithm could reduce the bias.

To reduce the bias of interpretation due to high correlated features, we tried multiple approaches to reduce the correlation issues in the data. In multivariate linear model, the features are based on the process to calculate the mixing coefficients for new predictions. Our explanation model is faithful to the new predictions from our LIVE algorithm. In our LIVE algorithm, decision tree models could avoid feature collinearity issue which is one key challenge question in traditional interpretation methods, such as SHAP and LIME. Also, to extract valid feature values, we used random clinical trial experiment design, each time we randomly selected the data features. Through this randomization procedure, we could always have a data point with a specific feature and another data point without that specific feature, thus this could answer the question what the model output could be without certain features.

Our LIVE algorithm is based on the prediction results from the deep learning models, not anything in the structure or the layers, thus our explanation is model agnostic.Our multivariate regression model is based on each individual instance and we don't try to interpret all instances, so it is a local interpretation approach.

Our algorithm could also be used for visualization interpretation and ROI. Since the coefficients from the linear model are indicators of the contribution of each feature. We can provide a saliency map to indicate the contributions of each feature base on their coefficients. We may use the features with biggest coefficients and significant p values as the ROI.

To validate our algorithms, we used different datasets including two image datasets and one structured data.We also tried different model structures. In our structured data model, we proposed a new method for hierarchical variable like ICD codes, where we applied word embedding technique to deal with it.

Our preliminary experimental results show that:

- Our proposed LIVE algorithm could achieve consistent better performance compared to deep learning model alone, even slightly, especially for small image size under 128x128. It also show modest improvement for structured data.

- Our proposed LIVE algorithm could be potentially for causal interpretation, the coefficients could indicate the magnitude effect on the predicted values. Also, reducing multicollinearity among features and randomization of the features in the model could lead to better causal inference.

- Our proposed LIVE algorithm could be applied to different deep learning models and different types of data.

- Our proposed LIVE algorithm could be used for identification of region of interests and provide comparable or better performance than the traditional visualization approaches, such as LIME, SHAP and Grad-CAM.

However, we also realized the disadvantages of our LIVE algorithm, which became our goals in the future work.

1. Expand our LIVE algorithm to different types of outcomes and data types

Currently we focus on binary outcomes since binary outcome is one of most important research in healthcare.The outcomes with continuous data or multiclass outcome could be converted to dichotomous variable.

2. The bottleneck of our algorithm is the running time for large image size

   The running time could be one important measure for evaluation of interpretable models. Our LIVE has a complexity with O(mxsxlog(n)) , which could be a big disadvantage. We should either develop an approach using parallel computing or think of ways to reduce the dimension of the image size. For example, we may preprocess the image to remove some regions which has no information, or just extract the primary part,which may significantly reduce the running time.

3. High dimensional data

   In our experiments with image data, we noticed that our LIVE algorithm performance is very similar to deep learning model for large image sizes. This could be due to small datasets. Since for image size 128x128, we have 15,000 features, which may need at least 150,000 images if we use rule of thumb that one feature needs ten samples. Currently we only have 5000 in the training dataset. in order to make accurate estimates of the coefficients, we need large number of datasets.

4. Uncertainty in deep learning model

   Though deep learning models have been widely used in several research ares,

however uncertainty is also one challenge question, which may prevent the deep learning model to have reliable estimates for a deep learning model's decision. The most common method to estimate the uncertainty is to use a different model to predict the uncertainty. For example, estimating the distribution over deep learning functions could be applied to estimate uncertainties in neural networks [142, 143].

In our LIVE algorithm, we also tried to apply mixture distribution approach to make the model more robust and flexible for the uncertainty, where we built a tree model to estimate the data uncertainty.However we are lack of data to evaluate our estimates of the uncertainty. For example, in our pneumonia experiment, LIVE suggested several potential ROI for infected region,some of them may be false segmented areas,however due to lack of data support, we may not verify whether our findings are true or false.

5. Lack of ground truth causal inference

   Although we theoretically proposed several solutions to make the interpretation more casual interpretation, we are lack of true data to support our conclusion, though we can demonstrate that we could reduce the correlations among the features.

5.2 Future work

Based on the above limitations of LIVE algorithm, in the future we would like to focus on following areas:

1. Image segmentation to reduce complexity

   Image segmentation is a process to partition the image into linked homogeneous regions, which could be a potential way to reduce the complexity of our LIVE algorithm.The image segmentation could simplify the representation of an image. The image segmentation could reduce an input image from millions of pixels to a few thousand clusters without losing model performance. There are different methods in the field of image segmentation, such as hierarchical clustering-based methods could deal with dataset with arbitrary shape and attribute type [144].

2. Uncertainty analysis Since in healthcare, when the algorithm is used for diagnostic assistance, people need to know the confidence of the algorithm, which includes uncertainty estimates. We would like to extend our current method for uncertainty estimates. We would like to seek more data or collaborate with medical doctors for further analysis.

3. Data preprocessing and data augmentation

   Currently we use the raw image data without further preprocessing. There is a lot redundant regions in the image data which may not be related to the outcome, for example,head or neck part in the images may not be helpful for lung disease prediction.

   In our experiments, we find due to the limited training dataset, for image size above 128, our LIVE algorithm does not perform very well since the trees don't have enough data to find the related images or overlapping images. We may

adopt the data augmentation technique to increase our training data pool.

4. Text data

   Text data is another major area which need deep learning models, however due to the data limit, it is hard to find a public healthcare data with text. We may try to obtain some text data to see whether our algorithm is applied to natural language processing data.

Appendix A: Programs codes

This section includes the data source, program codes for Vinbig data , pneumonia data and diabetes dataset.

A.1 Data sources

VinBig data:

https://www.kaggle.com/code/xhlulu/vinbigdata-process-and-resize-to-jpg/notebook

Chest X-Ray Images(Pneumonia):

https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

Diabetes data

https://www.kaggle.com/datasets/brandao/diabetes

A.2 Codes for Vinbig data

The following codes include image preprocessing for image size 16x16,32x32,64x64,128x128,224x224 and 512x512.

A.2.1 CNN model for Vinbig

```
# function to convert greyscale to 3 channel, x is the image data,
and si is the size you want to convert;
def to_rgb(x,si):
    x_rgb = np.zeros((x.shape[0], si, si, 3))
    for i in range(3):
        x_rgb[..., i] = x[..., 0]
    return x_rgb


###### read raw image files;
import keras
import os
train=pd.read_csv('/kaggle/input/lunglabel/train.csv')
train=train[['image_id','class_id']].groupby('image_id')
train=train.max().reset_index()
path = '../input/lungpred/train_cropped_224/'
files = [os.path.splitext(filename)[0] for filename
in os.listdir(path)]
import pandas as pd
file1=pd.DataFrame(files,columns=['image_id'])
train=pd.merge(file1,train,on='image_id')
```

```
#### recode outcome to binary values;
train['cancer']=np.where(train['class_id']==14,0,1)

#### get the data and labels from file;
labels=[]
data=[]
data_dir='../input/lungpred/train_cropped_224/'
for i in range(train.shape[0]):
    data.append(data_dir + train['image_id'].iloc[i]+'.png')
    labels.append(train['cancer'].iloc[i])
df=pd.DataFrame(data)
df.columns=['images']
df['cancer']=labels

train=df
train['cancer'].value_counts()

## split the files into training and testing;
X_train, X_val, y_train, y_test = train_test_split(train['images'],\\
train['cancer'], test_size=0.3, random_state=1234)
import time
```

```
train=pd.DataFrame(X_train)
train.columns=['images']
####replace the raw labels with binary;
train['isup_grade']=y_train

validation=pd.DataFrame(X_val)
validation.columns=['images']
####replace the raw labels with binary;
validation['isup_grade']=y_test

train['isup_grade']=train['isup_grade'].astype(str)
validation['isup_grade']=validation['isup_grade'].astype(str)

batch_size=32
#### reshape the images into different image sizes for different
experiments;
ls=[16,32,64,128,224]
comp=[]
#### convert image into array values for different image sizes.
for i in range(5):
    tr=[]
```

```
    img_size=ls[i]
    start_time=time.time()
#### training data;
    for img in train['images']:
        img_arr = cv2.imread(img, cv2.IMREAD_GRAYSCALE)
        resized_arr = cv2.resize(img_arr, (img_size, img_size))
        tr.append([resized_arr])
    x_train=np.array(tr)


#### testing data;
    test=[]
    for img in validation['images']:
        img_arr = cv2.imread(img, cv2.IMREAD_GRAYSCALE)
        resized_arr = cv2.resize(img_arr, (img_size, img_size))
        test.append([resized_arr])
    x_test=np.array(test)


    x_train=np.array(tr)
    x_test=np.array(test)
    x_train = np.array(x_train)/255
    x_test = np.array(x_test)/255
```

```
x_train = x_train.reshape(-1,img_size,img_size,1)
x_test = x_test.reshape(-1,img_size,img_size,1)



y_train=np.array(y_train)
y_test=np.array(y_test)
input_shape=(img_size,img_size,1)
#### define CNN model
input = Input(input_shape, name='input')

layer=Conv2D(img_size, kernel_size=(2, 2), activation='relu',\\
padding='same')(input)
layer=BatchNormalization()(layer)
layer=Conv2D(32, kernel_size=(2, 2), activation='relu',\\
padding='same')(layer)
layer=BatchNormalization()(layer)

layer=Conv2D(64, kernel_size=(2, 2), activation='relu',\\
padding='same')(layer)
layer=BatchNormalization()(layer)
layer=Conv2D(64, kernel_size=(2, 2), activation='relu',\\
padding='same')(layer)
```

```
layer=BatchNormalization()(layer)
layer=MaxPooling2D(pool_size=(2, 2))(layer)

layer=Conv2D(128, kernel_size=(2, 2), activation='relu',\\
padding='same')(layer)
layer=BatchNormalization()(layer)
layer=Conv2D(128, kernel_size=(2, 2), activation='relu',\\
padding='same')(layer)
layer=BatchNormalization()(layer)
layer=MaxPooling2D(pool_size=(2, 2))(layer)

layer=Conv2D(256, kernel_size=(2, 2), activation='relu', \\
padding='same')(layer)
layer=BatchNormalization()(layer)
layer=Conv2D(256, kernel_size=(2, 2), activation='relu',\\
padding='same',name='final')(layer)
layer=BatchNormalization()(layer)
layer=MaxPooling2D(pool_size=(2, 2))(layer)


layer = Dropout(0.5)(layer)
layer = Flatten(name='flatten')(layer)
```

```
output = Dense(1, name="Dense_10nb", activation='sigmoid')(layer)


model = Model(inputs=[input], outputs=[output])


model.compile(loss='binary_crossentropy',\\
optimizer=keras.optimizers.Adam(lr=0.0001,\\
decay=1e-7),metrics = ['accuracy'])
x_train1, x_test1, y_train1, y_test1 = \\
train_test_split((x_train),(y_train), test_size=0.2,
random_state=i)
model.fit(x=x_train1, y=y_train1,batch_size=batch_size,
epochs=30,verbose=1,validation_data=(x_test1, (y_test1)))


###### output datasets for decision tree models in LIVE algorithm; \\
dtx_train=pd.DataFrame(np.asarray(x_train).ravel().reshape\\
(x_train.shape[0],img_size*img_size))
dtx_test=pd.DataFrame(np.asarray(x_test).ravel().reshape\\
(x_test.shape[0],img_size*img_size))
dtx_train['yhat']=model.predict(x_train).ravel()
dtx_test['yhat']=model.predict(x_test).ravel()
end_time=time.time()
print(end_time-start_time)
```

```
from sklearn.metrics import roc_auc_score
roc=roc_auc_score(y_test,model.predict\\
(np.asarray(x_test)).ravel())
diff=end_time-start_time
comp.append([i,diff,roc])
print(roc,diff)
dtx_test['y_test']=y_test
dtx_train['y_train']=y_train
fn1="/kaggle/working/vincroptest_" + str(ls[i]) + ".csv"
dtx_test.to_csv(fn1,index=False)
fn2="/kaggle/working/vincroptrain_" + str(ls[i]) + ".csv"
dtx_train.to_csv(fn2,index=False)
```

A.2.2 VGG16 model for Vinbig data

```
#### transfer learning model using VGG 16

import keras
import os
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.optimizers import SGD
train=pd.read_csv('/kaggle/input/lunglabel/train.csv')
```

```
train=train[['image_id','class_id']].groupby('image_id')
train=train.max().reset_index()
path = '../input/lungpred/train_cropped_224/'
##path = '../input/vintage/train/'
files = [os.path.splitext(filename)[0] for filename
in os.listdir(path)]
import pandas as pd
file1=pd.DataFrame(files,columns=['image_id'])
train=pd.merge(file1,train,on='image_id')

train['cancer']=np.where(train['class_id']==14,0,1)

labels=[]
data=[]
data_dir='../input/lungpred/train_cropped_224/'
##data_dir = '../input/vintage/train/'
for i in range(train.shape[0]):
    ##data.append(data_dir + train['image_id'].iloc[i]+'.png')
    data.append(data_dir + train['image_id'].iloc[i]+'.png')
    labels.append(train['cancer'].iloc[i])
df=pd.DataFrame(data)
df.columns=['images']
```

```
df['cancer']=labels


train=df
train['cancer'].value_counts()


X_train, X_val, y_train, y_test = train_test_split(train['images'],


train['cancer'], test_size=0.3, random_state=1234)
import time


train=pd.DataFrame(X_train)
train.columns=['images']
train['isup_grade']=y_train


validation=pd.DataFrame(X_val)
validation.columns=['images']
validation['isup_grade']=y_test


train['isup_grade']=train['isup_grade'].astype(str)
validation['isup_grade']=validation['isup_grade'].astype(str)


batch_size=32
```

```
ls=[32,64,128]
comp=[]
auc=[]
for i in range(3):
    tr=[]
    img_size=ls[i]
    start_time=time.time()
    for img in train['images']:
        img_arr = cv2.imread(img, cv2.IMREAD_GRAYSCALE)
        resized_arr = cv2.resize(img_arr, (img_size, img_size))
        tr.append([resized_arr])
    x_train=np.array(tr)



    test=[]
    for img in validation['images']:
        img_arr = cv2.imread(img, cv2.IMREAD_GRAYSCALE)
        resized_arr = cv2.resize(img_arr, (img_size, img_size))
        test.append([resized_arr])
    x_test=np.array(test)
```

```
x_train=np.array(tr)

x_test=np.array(test)

x_train = np.array(x_train)

x_test = np.array(x_test)

x_train = x_train.reshape(-1,img_size,img_size,1)

x_test = x_test.reshape(-1,img_size,img_size,1)



y_train=np.array(y_train)

y_test=np.array(y_test)

input_shape=(img_size,img_size,1)


base_model = VGG16(weights="imagenet", include_top=False, \\

input_shape=[img_size,img_size,3])

##base_model = InceptionV3(weights="imagenet", \\

include_top=False, input_shape=[128,128,3])

base_model.trainable = False #### Not trainable weights

from tensorflow.keras import layers, models


flatten_layer = layers.Flatten()

dense_layer_1 = layers.Dense(50, activation='relu')

dense_layer_2 = layers.Dense(20, activation='relu')
```

```
prediction_layer = layers.Dense(1, activation='sigmoid')



model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_2,
    prediction_layer])




model.compile(
    optimizer=SGD(lr=0.01),
    loss='binary_crossentropy',
    metrics=['accuracy'],
)
model.fit(to_rgb(x_train,img_size), y_train, \\
epochs=30, validation_split=0.1, batch_size=32)
##dtx_train=pd.DataFrame(np.asarray(x_train).ravel().reshape\\
(x_train.shape[0],img_size*img_size))
##dtx_test=pd.DataFrame(np.asarray(x_test).ravel().reshape\\
(x_test.shape[0],img_size*img_size))
```

```
    from sklearn.metrics import roc_auc_score
    roc2=roc_auc_score(y_test,model.predict\\
    (to_rgb(x_test,img_size)).ravel())
    auc.append([ls[i],roc,roc2])
    print(auc)
#### The following is to output to csv file for LIVE algorithm;
    ##diff=end_time-start_time
    ##comp.append([i,diff,roc])
    ##print(roc,diff)
    ##dtx_test['y_test']=y_test
    ##dtx_train['y_train']=y_train
    ##fn1="/kaggle/working/vinrestest_" + str(ls[i]) + ".csv"
    ##dtx_test.to_csv(fn1,index=False)
    ##fn2="/kaggle/working/vinrestrain_" + str(ls[i]) + ".csv"
    ##dtx_train.to_csv(fn2,index=False)
```

A.2.3 Visualization codes using LIME,SHAP and GRAD CAM

```
###### here is the GRAD CAM visualization
pos=[10]
```

```
hm=[]
last_conv_layer = model.get_layer('conv2d_11')
for ii in range(1):

    x=x_test[9:10,:,:,:]
    preds=model.predict(np.asarray(x))
    num = np.argmax(preds)
    from keras import backend as K
    from keras.preprocessing import image
    from keras.applications.vgg16 import preprocess_input,
    decode_predictions
    import numpy as np
    ot=model.output[:, 0]
    grads = K.gradients(ot, last_conv_layer.output)[0]
    pooled_grads = K.mean(grads, axis=(0, 1, 2))
    print('pooled_grads:',pooled_grads.shape)
    iterate = K.function([model.input], [pooled_grads, \\
    last_conv_layer.output[0]])
    pooled_grads_value, conv_layer_output_value = iterate([x])
    ## We multiply each channel in the feature map array
    ## by "how important this channel is" with regard to the
    elephant class
```

```
    for i in range(len(pooled_grads_value)):
        conv_layer_output_value[:, :, i] *= pooled_grads_value[i]

    ## The channel-wise mean of the resulting feature map
    ## is our heatmap of class activation
    heatmap = np.mean(conv_layer_output_value, axis=-1)
    ## ##shape:14*14
    heatmap = np.maximum(heatmap, 0)
    heatmap /= np.max(heatmap)  ##shape:14*14
    heatmap
    hm.append([pos[ii],heatmap])

###### LIME we tried 1000 sample points;
import lime
from lime import lime_image
explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(x_test[9:10,:,:,:], \\
model.predict, hide_color=0, num_samples=1000)

#### for SHAP value;
import shap
```

```
## since we have two inputs we pass a list of inputs to the explainer
explainer = shap.GradientExplainer(model, [x_train, x_train])

## we explain the model's predictions on the first 19 samples
of the test set
shap_values = explainer.shap_values((x_test[0:19,:,:,:]))
sh=np.asarray(shap_values).reshape(19,16,16)
#### only keep the positive impact part
sh[sh<0]=0
```

A.2.4 Codes for LIVE algorithm

```
###### Tree models for LIVE algorithms

import numpy as np ## linear algebra
import pandas as pd ## data processing, CSV file I/O (e.g. pd.read_csv)

## Input data files are available in the "../input/" directory.
## For example, running this (by clicking run or pressing Shift+Enter)\
will list all files under the input directory
import os
import cv2
import PIL
```

```
from IPython.display import Image, display
from keras.applications.vgg16 import VGG16, preprocess_input
## Plotly for the interactive viewer (see last section)
import plotly.graph_objs as go
from sklearn.metrics import cohen_kappa_score
from sklearn.model_selection import train_test_split
from keras.models import Sequential, Model, load_model
from keras.applications.vgg16 import VGG16, preprocess_input

from keras.preprocessing.image import ImageDataGenerator,\\
load_img, img_to_array
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Input,\\
Flatten, BatchNormalization, Activation
from keras.layers import GlobalMaxPooling2D
from keras.models import Model
from keras.optimizers import Adam, SGD, RMSprop
from keras.callbacks import ModelCheckpoint, Callback, EarlyStopping
from keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import gc
import skimage.io
```

```
from sklearn.model_selection import KFold
## for dirname, _, filenames in os.walk('/kaggle/input'):
##     for filename in filenames:
##         print(os.path.join(dirname, filename))

## Any results you write to the current directory are saved as output.
import tensorflow as tf
from tensorflow.python.keras import backend as K
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

from sklearn.tree import DecisionTreeClassifier
def find_path(node_numb, path, x):
        path.append(node_numb)
        if node_numb == x:
            return True
        left = False
        right = False
        if (children_left[node_numb] !=-1):
            left = find_path(children_left[node_numb], path, x)
        if (children_right[node_numb] !=-1):
            right = find_path(children_right[node_numb], path, x)
```

```
        if left or right :
            return True
        path.remove(node_numb)
        return False
def get_rule(path, column_names):
    mask = ''
    for index, node in enumerate(path):
        ##We check if we are not in the leaf
        if index!=len(path)-1:
            ## Do we go under or over the threshold ?
            if (children_left[node] == path[index+1]):
                mask += "(df['{}']<= {}) \t ".\\
                format(column_names[feature[node]], threshold[node])
            else:
                mask += "(df['{}']> {}) \ t".\\
                format(column_names[feature[node]], threshold[node])
    ## We insert the & at the right places
    mask = mask.replace("\t", "&", mask.count("\t") - 1)
    mask = mask.replace("\t", "")
    return mask

from sklearn.metrics import roc_auc_score
```

```
from sklearn.metrics import roc_curve
import random
import time

from sklearn.tree import DecisionTreeClassifier
from scipy.stats.mstats import gmean
pi=[16,32,64,128]
rocc=[]
for m in range(0,1):
    fname1='vinbig_cnn/vincroptrain_'+str(pi[m])+'.csv'
    fname2='vinbig_cnn/vincroptest_'+str(pi[m])+'.csv'
    print(fname1)
    train=pd.read_csv(fname1)
    test=pd.read_csv(fname2)
    newdata=pd.DataFrame([])
    regdata=[]
    imp=pd.DataFrame([])
    ru=pd.DataFrame([])
    regdata=pd.DataFrame([])
    start_time=time.time()
    fd=pd.DataFrame([])
    train.columns = ['v'+col_name for col_name in train.columns]
```

```
test.columns = ['v'+col_name for col_name in test.columns]
train.yhat=np.log(train.vyhat/(1-train.vyhat))
test.yhat=np.log(test.vyhat/(1-test.vyhat))
train_new=pd.DataFrame([])
test['vy_train']=test['vy_test']
test=test.drop(['vy_test'],axis=1)


##test=test.iloc[1:5]
##test=train
##ds=train.append(test)


##train=train[((train['vyhat']>0.67) & (train['vy_train']==1)) |\\
((train['vyhat']<0.6) & (train['vy_train']==0)) ]
##test['vy_train']=(test['vyhat']>0.67)*1
##test['vy_train']=test['vy_test']


##test1=test.sample(frac=0.05, replace=True, random_state=2)
y_train=train['vy_train']
y_test=test['vy_train']
##y_test=train['vy_train']
train_yhat=train['vyhat']
##train_yhat=xx2['pred_lstm']
```

```
##train_yhat=train2['yhat']
##train_yhat=xx_raw['pred_lstm']
test_yhat=test['vyhat']
##test_yhat=train['vyhat']
dtx_train=train.drop(['vy_train','vyhat'],axis=1)
##dtx_train=dtx_train[dtx_train.columns[600:2400]]
dtx_test=test.drop(['vy_train','vyhat'],axis=1)
##dtx_test=dtx_test[dtx_test.columns[600:2400]]
##dtx_test=dtx_train
for i in range(2000):
    print(i)
    ##x_train11=dtx_train.sample(random.randint(15, 16),axis=1)
    ##the following is to try randomly observation selection;
    ##x_train11=x_train11.sample(random.randint(9000, 9001),axis=0)
    ## the following is for different feature selections;
    x_train11=dtx_train.sample(random.randint(10, 40),axis=1)
    chi_feature=x_train11.columns
    chi_feature=list(chi_feature)
    ##chi_feature=list(x_train11.columns)+['vyhat']
    tree = DecisionTreeClassifier(random_state=42, \\
    min_samples_leaf=5,criterion='entropy')
    tree.fit(dtx_train[chi_feature], y_train)
```

```
        #### to get node information;
         n_nodes = tree.tree_.node_count
         children_left = tree.tree_.children_left
         children_right = tree.tree_.children_right
         feature = tree.tree_.feature
         threshold = tree.tree_.threshold
         leave_id = tree.apply(dtx_train[chi_feature])
         x_cp1=dtx_test[chi_feature].copy()
         x_cp1['death']=y_test
         x_cp1=x_cp1.reset_index()
         #### add predicted values from deep learning model to tree data
         x_cp1['lstm_pred'] = test_yhat.values
         X=(np.asarray(dtx_test).astype(np.float32))
         x_cp1['id']=x_cp1.index+1
         paths={}
         for leaf in np.unique(leave_id):
             path_leaf = []
             find_path(0, path_leaf, leaf)
             paths[leaf] = np.unique(np.sort(path_leaf))
         rules = []
###### get the rule information which is for LASSO regression;
         for key in paths:
```

```
            rules.append([key,get_rule(paths[key], chi_feature)])
        ##yhat2= tree.predict_proba(x_train[])[:,1]

        df=pd.DataFrame(rules)
        df.columns=['node','rules']
#### to extract features from the rules;
        df['rule']= df['rules'].str.findall(r"(?<=\[)([^]]+)(?=\])")
        x_cp2=dtx_train[chi_feature].copy()
        x_cp2['death']=y_train
        x_cp2=x_cp2.reset_index()

        x_cp2['lstm_pred'] = train_yhat.values
        X=(np.asarray(dtx_train[chi_feature]).astype(np.float32))
        x_cp2['pred_node']=tree.tree_.apply(X)
        ###### reassign Id in case the id numbers in both training \\
        and testing data;
        x_cp2['id']=x_cp2.index+10000
        X2=(np.asarray(dtx_test[chi_feature]).astype(np.float32))
        x_cp1['pred_node']=tree.tree_.apply(X2)
        x_cp=x_cp2
        ##x_cp=pd.concat([x_cp1, x_cp2], sort=False,ignore_index=True).\
        fillna(0)
```

```
        x_cp['pred_lstm'] = x_cp.groupby('pred_node')['lstm_pred'].\\
        transform('mean')
        x_cp['pred_ct'] = x_cp.groupby('pred_node')['lstm_pred'].\\
        transform('count')
        x_cp['pred_dt']=tree.predict_proba(dtx_train[chi_feature])[:,1]
        x_cp_tr=x_cp[['id','pred_lstm','death','pred_node',\\
        'lstm_pred','pred_dt','pred_ct']]
        ##x_cp_tr['order']=i
        x_cp_tr = x_cp_tr.assign(order=i)
        train_new=train_new.append(x_cp_tr)
        x111=x_cp[['pred_node','pred_lstm','pred_ct']].drop_duplicates(
###### merge training and testing data together by node;
        x3=x_cp1.merge(x111, left_on='pred_node', \\
        right_on='pred_node', how='left')
        ds=x3[chi_feature]
        x3['pred_dt']=tree.predict_proba(dtx_test[chi_feature])[:,1]
        x3=x3[['id','pred_lstm','death','pred_node',\\
        'lstm_pred','pred_dt','pred_ct']]
        x3['order']=i
#### get the data from all the trees;
        newdata=newdata.append(x3)
        importance = tree.feature_importances_
```

```
            imp1=pd.DataFrame(list(zip(importance,chi_feature)))
            imp1['order']=i
            imp=imp.append(imp1)
            ##nd=x3['pred_node']
            ##df=df[df['node']==int(nd)]
            ##ru=ru.append(df)
            ds['id']=x3['id']
#### regression data for regression model;
            regdata=regdata.append(ds)
#### get the new auc for predicted values from LIVE;
xx2=pd.DataFrame(newdata[['pred_lstm','id','death','lstm_pred']].\\
groupby('id').mean())
##xx2
roc_auc_score(xx2.death,xx2.pred_lstm)
```

A.2.4.1 SAS code for LASSO regression model

```
#### the following is SAS code for LASSO/Ridge or Elastic net regressior
data regdata;
set regdata;
array feature v:;
do over a;
##* recode all feature values to 0 or 1
```

```
depending whether they are missing or \\
not in the rule path;
a=(a>0)*1;
end;
run;
proc glmselect data=regdata;
model pred_lstm = v_: /selection=lasso(stop=L1 L1choice=ratio L1=.4)
run;
```

A.2.4.2 Visualization for LIVE algorithm

```
## The following is for visualization of LIVE algorithm;
import matplotlib.pyplot as plt
import matplotlib.image as img
import numpy as np
import cv2
im = cv2.imread('id13.png')

from IPython.display import Image, display
import matplotlib.pyplot as plt
import matplotlib.cm as cm
##*coefficients from lasso regression models;
p=[0.0234,-0.28997,...,0.18653]
```

```
original_image = np.array(p).reshape(16,16)
for W in range(16):
    for H in range(16):
        new_width = int( W * width/original_image.shape[0] )
        new_height = int( H * height/original_image.shape[1]
)
        resize_image[new_width][new_height] =\\
        resize_image[new_width][new_height]+original_image[W][H]
resize_image[resize_image<0] = 0
heatmap = np.uint8(255 *np.asarray(resize_image).reshape(6,6) )
import keras
jet = cm.get_cmap("jet")
alpha=0.8
# Use RGB values of the colormap
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]

# Create an image with RGB colorized heatmap
jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((im.shape[1], im.shape[0]))
jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)
```

```
# Superimpose the heatmap on original image
superimposed_img = jet_heatmap * alpha + im
superimposed_img = keras.preprocessing.image.\\
array_to_img(superimposed_img)
display(superimposed_img)
```

A.3 Codes for pneumonia data

A.3.1 SAS code for data creation

```
**** the following is for pneumonia image data
import os
from sklearn.metrics import roc_auc_score
labels = ['PNEUMONIA', 'NORMAL']
def datafunc(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img),\\
                cv2.IMREAD_GRAYSCALE)
```

```
                resized_arr = cv2.resize(img_arr,(img_size, img_size))
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)

ls=[32,64,128,224]
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications import ResNet50
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten,
\\ GlobalAveragePooling2D

auc=[]
from tensorflow.keras.applications.vgg16 import preprocess_input
import time
ti=[]
for i in range(1,3):
    img_size=ls[i]
    print(img_size)
    start_time=time.time()
    train = datafunc('../input/pnunomia/chest_xray/chest_xray/train')
```

```
test = datafunc('../input/pnunomia/chest_xray/chest_xray/test')
x_train=[]
y_train=[]
x_test=[]
y_test=[]
for feature, label in train:
    x_train.append(feature)
    y_train.append(label)
for feature, label in test:
    x_test.append(feature)
    y_test.append(label)
x_train = np.array(x_train)/255.0
x_test = np.array(x_test)/255.0
x_train = x_train.reshape(-1,img_size,img_size,1)
x_test = x_test.reshape(-1,img_size,img_size,1)
y_train=np.array(y_train)
y_test=np.array(y_test)
import keras
input_shape=(img_size,img_size,1)
input = Input(input_shape, name='input')
layer=Conv2D(32, kernel_size=(2, 2), activation='relu', \\
padding='same')(input)
```

```
layer=BatchNormalization()(layer)
layer=Conv2D(32, kernel_size=(2, 2), activation='relu', \\
padding='same')(input)
layer=BatchNormalization()(layer)

layer=Conv2D(64, kernel_size=(2, 2), activation='relu', \\
padding='same')(layer)
layer=BatchNormalization()(layer)
layer=Conv2D(64, kernel_size=(2, 2), activation='relu', \\
padding='same')(layer)
layer=BatchNormalization()(layer)
layer=MaxPooling2D(pool_size=(2, 2))(layer)

layer=Conv2D(128, kernel_size=(2, 2), activation='relu', \\
padding='same')(layer)
layer=BatchNormalization()(layer)
layer=Conv2D(128, kernel_size=(2, 2), activation='relu', \\
padding='same')(layer)
layer=BatchNormalization()(layer)
layer=MaxPooling2D(pool_size=(2, 2))(layer)

layer=Conv2D(256, kernel_size=(2, 2), activation='relu', \\
```

```
        padding='same')(layer)
        layer=BatchNormalization()(layer)
        layer=Conv2D(256, kernel_size=(2, 2), activation='relu',\\
        padding='same',name='final')(layer)
        layer=BatchNormalization()(layer)
        layer=MaxPooling2D(pool_size=(2, 2))(layer)


        layer = Dropout(0.5)(layer)
        layer = Flatten(name='flatten')(layer)
        output = Dense(1, name="Dense_10nb", activation='sigmoid')(layer)
        model = Model(inputs=[input], outputs=[output])
        model.compile(loss='binary_crossentropy',optimizer=SGD(lr=0.0001),\\
        metrics = ['accuracy'])
        model.fit(x=x_train, y=y_train,batch_size=batch_size,\\
        epochs=30,verbose=1,validation_data=(x_test, y_test))
        end_time=time.time()
        tot=(end_time-start_time)/60
        print(ls[i],tot)
        auc=roc_auc_score(y_test,model.predict(x_test).ravel())
        ti.append([ls[i],start_time,end_time,tot,auc])
        print(auc)
    ### output CNN model predictions to csv;
```

```
dtx_train=pd.DataFrame(np.asarray(x_train).ravel().\\
reshape(5216,img_size*img_size))
dtx_test=pd.DataFrame(np.asarray(x_test).ravel().\\
reshape(624,img_size*img_size))
dtx_train['yhat']=model.predict(x_train).ravel()
dtx_test['yhat']=model.predict(x_test).ravel()
dtx_test['y_test']=y_test
dtx_train['y_train']=y_train
fn1="/kaggle/working/pneutest_" + str(ls[i]) + ".csv"
dtx_test.to_csv(fn1,index=False)
fn2="/kaggle/working/pneutrain_" + str(ls[i]) + ".csv"
dtx_train.to_csv(fn2,index=False)
import datetime
base_model = VGG16(weights="imagenet", include_top=False,
input_shape=[img_size,img_size,3])
#base_model = InceptionV3(weights="imagenet", include_top=False,\\
input_shape=[128,128,3])
base_model.trainable = False ## Not trainable weights
from tensorflow.keras import layers, models

flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(50, activation='relu')
```

```
dense_layer_2 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(1, activation='sigmoid')
model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_2,
    prediction_layer
])
from tensorflow.keras.callbacks import EarlyStopping
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'],
)
model.fit(to_rgb(x_train,img_size), y_train, epochs=30, \\
validation_split=0.1, batch_size=20)
auc1=roc_auc_score(y_test,model.predict(to_rgb(x_test,img_size)).\\
ravel())
auc2=roc_auc_score(y_test,model.predict(to_rgb(x_test,img_size)).\\
ravel())
auc.append([ls[i],auc1,auc2])
print(auc)
```

```
#tot=(end_time-start_time)/60
#print(ls[i],tot)
ti.append([ls[i],start_time,end_time,tot])


dtx_train=pd.DataFrame(np.asarray(x_train).ravel().\\
reshape(5216,img_size*img_size))
dtx_test=pd.DataFrame(np.asarray(x_test).ravel().\\
reshape(624,img_size*img_size))
dtx_train['yhat']=model.predict(to_rgb(x_train,img_size)).ravel()
dtx_test['yhat']=model.predict(to_rgb(x_test,img_size)).ravel()
dtx_test['y_test']=y_test
dtx_train['y_train']=y_train
fn1="/kaggle/working/pneutest_" + str(ls[i]) + ".csv"
dtx_test.to_csv(fn1,index=False)
fn2="/kaggle/working/pneutrain_" + str(ls[i]) + ".csv"
dtx_train.to_csv(fn2,index=False)
```

A.3.2 Codes for LIVE algorithm and visualization

The other parts are very similar to Vinbig data, so just use the same code with pneumona data

A.4 Codes for diabetes data

A.4.1 data preprocessing

We adopted a public preprocessed dataset from Github, \\
https://github.com/andrewwlong/diabetes_readmission/blob/master/diabetic
then we used a SAS code to create the dataset for our experiment:

```
## read the above csv file to SAS;
   FILENAME REFFILE DISK 'diabetic_data.csv';
   PROC IMPORT DATAFILE=REFFILE replace
   DBMS=CSV
   OUT=WORK.ds;
   GETNAMES=YES;
   RUN;

 ### only keep the last 6 visits if the patient has more than 6 visits;
proc sort data=ds;by patient_nbr descending encounter_id;
data ds;
retain time;
set ds;
by patient_nbr descending encounter_id ;
if first.patient_nbr then time=0;
```

```
time+1;
run;
data ds1;
set ds;
if time<7;
time=7-time;
run;

### get the patients with the readmission status in the last visit;
## we got 6342 patients;
proc sort data=ds1;by patient_nbr time;
data adm;
set ds1;
by patient_nbr time;
if first.patient_nbr and output_label=1;
run;

## remove those patients from the population to get control groups;
proc sql;create table control as select distinct patient_nbr
from ds1 where patient_nbr not in (select patient_nbr from adm);quit;

## get a balanced control data by simple random selection;
```

```
proc surveyselect data=control out=ctrl sampsize=6342;
run;

data tmp;
set adm(in=a) ctrl;
if a=1 then readm=1;else readm=0;
keep patient_nbr readm;
run;
proc sql;create table phd_ds
as select * ,readm from ds1,tmp where ds1.patient_nbr=tmp.patient_nbr;
quit;

data lstm_long;
set phd_ds;
## most data have values less than 2, so we need to artifically add
some numbers to make sure they have different meanings.

array dd race_Asian     race_Caucasian  race_Hispanic
gender_Male     max_glu_300     max_glu_serum_Norm      A1Cresult_8
A1Cresult_Norm  repaglinide_Steady      repaglinide_Up
nateglinide_Steady      glimepiride_Steady      glimepiride_Up
glipizide_Steady        glipizide_Up    glyburide_Steady
```

```
glyburide_Up     tolbutamide_Steady      pioglitazone_Steady
pioglitazone_Up  rosiglitazone_Steady    rosiglitazone_Up
acarbose_Steady  acarbose_Up     miglitol_Steady  tolazamide_Steady
insulin_Steady   insulin_Up      change_No        diabetesMed_Yes
med_Emergency_Trauma
med_Family       med_InternalMedicine
med_Nephrology   med_Orthopedics med_Other
med_Radiologist  med_Surgery     med_UNK;
do i=1 to dim(dd);
dd{i}=i*3+dd{i};
end;
num_procedures  =       num_procedures+500;
time_in_hospital        =       time_in_hospital+550;
number_diagnoses        =       number_diagnoses+600;
number_inpatient        =       number_inpatient+620;
number_outpatient       =       number_outpatient+680;
number_emergency        =       number_emergency+780;
num_medications =       num_medications+880;
age_group       =       age_group+980;
num_lab_procedures      =       num_lab_procedures+1100;
diag_1  =       diag_1+1500;
diag_2  =       diag_2+1500;
```

```
run;
## here is the list of variables we want to keep in the model;
%let colname=A1Cresult_8
A1Cresult_Norm acarbose_Steady acarbose_Up age_group
change_No diabetesMed_Yes gender_Male diag_1 diag_2
glimepiride_Steady glimepiride_Up glipizide_Steady
glipizide_Up glyburide_Steady glyburide_Up
insulin_Steady insulin_Up max_glu_300 max_glu_serum_Norm
med_Emergency_Trauma med_Family med_InternalMedicine
med_Nephrology med_Orthopedics med_Other med_Radiologist
med_Surgery miglitol_Steady nateglinide_Steady
num_lab_procedures num_medications num_procedures
number_diagnoses number_emergency number_inpatient
number_outpatient pioglitazone_Steady pioglitazone_Up
race_Asian race_Caucasian race_Hispanic
repaglinide_Steady repaglinide_Up rosiglitazone_Steady
rosiglitazone_Up time_in_hospital
tolazamide_Steady tolbutamide_Steady readm;

## we need to convert the data from long format to wide
format;
proc sort data=lstm_long;by patient_nbr time;
```

```
proc transpose data=lstm_long out=out1 ;
var &colname.;
by patient_nbr time;
quit;
proc transpose data=out1 delimiter=_ out=dswide(drop=_name_);
by patient_nbr;
var col1;
id _name_ time;
run;

## replace missing values with 0;
data lstm_wide;
set dswide;
;
array t _numeric_;
do over t;
if t=. then t=0;
end;
run;
```

A.4.2 Deep learning model and LIVE method

```
## read the data file;
```

```
diab= pd.read_csv("LSTM_WIDE_1126.csv")
y0=(diab["readm_6"])
df0=diab.drop(['readm_6','patient_nbr','readm_1','readm_2','readm_3',\\
'readm_4','readm_5'], axis=1)
auc=[]
app=pd.DataFrame([])
for ii in range(1):
    #df1=np.array(df0).reshape(len(df0),1,294)
    x_train, x_test, y_train, y_test = train_test_split(df0, y0, \\
    test_size=0.3, random_state=ii)
    vocab_size = 50000
    #max_length = x.shape[1]
    #max_length=743
    max_length=df0.shape[1]
    # define our LSTM model with word embedding layer;
    model = Sequential()
    #model.add(LSTM(10),input_shape=(1,294)))
    model.add(Embedding(vocab_size,20, input_length=max_length))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    # compile the mode
    model.compile(optimizer='adam', loss='binary_crossentropy',\\
```

```
    metrics=['AUC','acc'])
    model.fit(np.array(x_train), np.array(y_train), \\
    validation_data=(np.array(x_test),\\
    np.array(y_test)), epochs=30, batch_size=32)
    yhat = model.predict((x_test), verbose=0)
    print(roc_auc_score(y_test,yhat))

### use SAS logistic to check the variable selection;
## we use gee model for repeated logistic regression;
## for gee model if using all variable, the model could not converge,
#so we use logistic model for variable selection;
proc logistic data=lstm_long_1124;
model readm=A1Cresult_8
A1Cresult_Norm

acarbose_Steady
acarbose_Up
age_group
change_No
diabetesMed_Yes
diag_1
diag_2
```

encounter_id

gender_Male

glimepiride_Steady

glimepiride_Up

glipizide_Steady

glipizide_Up

glyburide_Steady

glyburide_Up

insulin_Steady

insulin_Up

max_glu_300

max_glu_serum_Norm

med_Emergency_Trauma

med_Family

med_InternalMedicine

med_Nephrology

med_Orthopedics

med_Other

med_Radiologist

med_Surgery

med_UNK

miglitol_Steady

nateglinide_Steady

num_lab_procedures

num_medications

num_procedures

number_diagnoses

number_emergency

number_inpatient

number_outpatient

pioglitazone_Steady

pioglitazone_Up

race_Asian

race_Caucasian

race_Hispanic

time

repaglinide_Steady

repaglinide_Up

rosiglitazone_Steady

rosiglitazone_Up

time_in_hospital

```
tolazamide_Steady
tolbutamide_Steady/selection=stepwise;
run;
### we used the above selected variables for our GEE model;
proc genmod data=lstm_long_1124;
class    patient_nbr ;
model
readm=A1Cresult_8
acarbose_Steady
age_group
diabetesMed_Yes
diag_1
med_UNK
num_procedures
number_diagnoses
number_inpatient
number_outpatient
number_inpatient
time_in_hospital
tolazamide_Steady
/dist=bin maxiter=1000;
repeated subject=        patient_nbr / type=ind maxiter=1000 ;
```

```
output out=test pred=pred;;

;

run;;


## we should only check the auc for time=6; the last visit;
ods output WilcoxonScores=WilcoxonScore;
proc npar1way wilcoxon data= test ;
class readm;
var  pred;
where time=6;
run;


data AUC;
set WilcoxonScore end=eof;
retain v1 v2 1;
if _n_=1 then v1=abs(ExpectedSum - SumOfScores);
v2=N*v2;
if eof then do;
d=v1/v2;
Gini=d * 2;     AUC=d+0.5;
put AUC=  GINI=;
keep AUC Gini;
```

```
output;
end;
run;
#### our LIVE method, similar to the process in image analysis
    for j in range(1):
        #test1=test.sample(frac=0.05, replace=True, random_state=2)
        train_yhat= model.predict(x_train, verbose=0)
        #train_yhat=xx2['pred_lstm']
        #train_yhat=train2['yhat']
        #train_yhat=xx_raw['pred_lstm']
        test_yhat= model.predict(x_test, verbose=0)
        dtx_train=x_train
        #dtx_train=dtx_train[dtx_train.columns[600:2400]]
        dtx_test=x_test
        #dtx_test=dtx_test[dtx_test.columns[600:2400]]
        newdata=pd.DataFrame([])
        regdata=[]
        imp=pd.DataFrame([])
        ru=pd.DataFrame([])
        regdata=pd.DataFrame([])
        start_time=time.time()
        fd=pd.DataFrame([])
```

```
train_new=pd.DataFrame([])
for i in range(1,200):
    print(i)
    x_train11=dtx_train.sample(random.randint(10, 80),axis=1)
    chi_feature=x_train11.columns
    chi_feature=list(chi_feature)
    #chi_feature=list(x_train11.columns)+['vyhat']
    tree = DecisionTreeClassifier(random_state=42, \\
    min_samples_leaf=5,criterion='entropy')
    tree.fit(dtx_train[chi_feature], y_train)

    n_nodes = tree.tree_.node_count
    children_left = tree.tree_.children_left
    children_right = tree.tree_.children_right
    feature = tree.tree_.feature
    threshold = tree.tree_.threshold
    leave_id = tree.apply(dtx_train[chi_feature])
    x_cp1=dtx_test[chi_feature].copy()
    x_cp1['readm']=y_test
    x_cp1=x_cp1.reset_index()
    x_cp1['lstm_pred'] = test_yhat
```

```
X=(np.asarray(dtx_test).astype(np.float32))



x_cp1['id']=x_cp1.index+1
paths={}
for leaf in np.unique(leave_id):
    path_leaf = []
    find_path(0, path_leaf, leaf)
    paths[leaf] = np.unique(np.sort(path_leaf))
rules = []
for key in paths:
    rules.append([key,get_rule(paths[key], chi_feature)])
#yhat2= tree.predict_proba(x_train[])[:,1]


df=pd.DataFrame(rules)
df.columns=['node','rules']
df['rule']= df['rules'].str.findall(r"(?<=\[)([^]]+)(?=\])"
x_cp2=dtx_train[chi_feature].copy()
x_cp2['readm']=y_train
x_cp2=x_cp2.reset_index()


x_cp2['lstm_pred'] = train_yhat
```

```
#x_cp2['lstm_pred'] = rf.predict(x_train)
#x_cp2['pred_tree']=tree.predict_proba(dtx_train[chi_feature
X=(np.asarray(dtx_train[chi_feature]).astype(np.float32))
x_cp2['pred_node']=tree.tree_.apply(X)
#x_cp2['pred_lstm'] = x_cp2.groupby('pred_node')['lstm_pred
.transform('mean')

x_cp2['id']=x_cp2.index+10000
X2=(np.asarray(dtx_test[chi_feature]).astype(np.float32))
x_cp1['pred_node']=tree.tree_.apply(X2)
x_cp=x_cp2
#x_cp=pd.concat([x_cp1, x_cp2], sort=False,\\
ignore_index=True).fillna(0)
x_cp['pred_lstm'] = x_cp.groupby('pred_node')\\
['lstm_pred'].transform('mean')
x_cp['pred_ct'] = x_cp.groupby('pred_node')['lstm_pred'].\\
transform('count')
x_cp['pred_dt']=tree.predict_proba(dtx_train[chi_feature])[:
x_cp_tr=x_cp[['id','pred_lstm','readm','pred_node',\\
'lstm_pred','pred_dt','pred_ct']]
#x_cp_tr['order']=i
x_cp_tr = x_cp_tr.assign(order=i)
```

```
        train_new=train_new.append(x_cp_tr)
        x111=x_cp[['pred_node','pred_lstm','pred_ct']].drop_duplica


        x3=x_cp1.merge(x111, left_on='pred_node', right_on='pred_noc
        how='left')
        ds=x3[chi_feature]
        x3['pred_dt']=tree.predict_proba(dtx_test[chi_feature])[:,1]
        x3=x3[['id','pred_lstm','readm','pred_node','lstm_pred','pre
        x3['order']=i
        newdata=newdata.append(x3)
        importance = tree.feature_importances_
        imp1=pd.DataFrame(list(zip(importance,chi_feature)))
        imp1['order']=i
        imp=imp.append(imp1)
        #nd=x3['pred_node']
        #df=df[df['node']==int(nd)]
        #ru=ru.append(df)
        ds['id']=x3['id']
        regdata=regdata.append(ds)
#xx=pd.DataFrame(newdata[['pred_lstm','id','readm','lstm_pred']]
.groupby('id').median())
xx=pd.DataFrame(newdata[['order','pred_lstm','id','readm',\\
```

```
    'lstm_pred','pred_node','pred_ct']].groupby('id').agg(['mean',
elapsed_time = time.time() - start_time
xx2=pd.DataFrame(newdata[['pred_lstm','id','readm',\\
'lstm_pred','pred_dt']].groupby('id').mean())
#xx2['pred_l']= new2.groupby('id').pred_lstm.apply(stats.gmean)

auc22=roc_auc_score(xx2.readm,xx2.pred_lstm)
```

Bibliography

[1] tlmodel. https://keras.io/api/applications/.

[2] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, Rajiv Suman, and Shanay Rab. Significance of machine learning in healthcare: Features, pillars and applications. *International Journal of Intelligent Networks*, 2022.

[3] Talal AA Abdullah, Mohd Soperi Mohd Zahid, and Waleed Ali. A review of interpretable ml in healthcare: Taxonomy, applications, challenges, and future directions. *Symmetry*, 13(12):2439, 2021.

[4] Christopher M Bishop. Mixture density networks. 1994.

[5] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. Interpretable deep models for icu outcome prediction. In *AMIA annual symposium proceedings*, volume 2016, page 371. American Medical Informatics Association, 2016.

[6] vgg16structure. https://neurohive.io/en/popular-networks/vgg16/.

[7] Peter Groves, Basel Kayyali, David Knott, and Steve Van Kuiken. Accelerating value and innovation. *The 'big data' revolution in healthcare: Accelerating value and innovation*, (January):1–22, 2013.

[8] Bonnie Feldman, Ellen M. Martin, and Tobi Skotnes. Big Data in Healthcare - Hype and Hope. *Dr.Bonnie 360 degree (Business Development for Digital Health)*, 2013(1):122–125, 2012.

[9] William H. Shrank, Teresa L. Rogstad, and Natasha Parekh. Waste in the US Health Care System: Estimated Costs and Potential for Savings. *JAMA - Journal of the American Medical Association*, 322(15):1501–1509, 2019.

[10] Kee Yuan Ngiam and Ing Wei Khor. Big data and machine learning algorithms for health-care delivery. *The Lancet Oncology*, 20(5):e262–e273, 2019.

[11] David W. Bates, Suchi Saria, Lucila Ohno-Machado, Anand Shah, and Gabriel Escobar. Big data in health care: Using analytics to identify and manage high-risk and high-cost patients. *Health Affairs*, 33(7):1123–1131, 2014.

[12] V Shyamala Susan, K Juliana Gnana Selvi, and Ir Bambang Sugiyono Agus Purwono. Big data in healthcare: Applications and challenges. *Advanced Analytics and Deep Learning Models*, pages 351–363, 2022.

[13] Isaac Bankman. *Handbook of medical image processing and analysis*. Elsevier, 2008.

[14] Justin Ker, Lipo Wang, Jai Rao, and Tchoyoson Lim. Deep learning applications in medical image analysis. *Ieee Access*, 6:9375–9389, 2017.

[15] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017.

[16] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS Valentim, Huiying Liang, Sally L Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131, 2018.

[17] Vasiliki Diamantopoulou, Aggeliki Tsohou, and Maria Karyda. General data protection regulation and iso/iec 27001: 2013: Synergies of activities towards organisations' compliance. In *International Conference on Trust and Privacy in Digital Business*, pages 94–109. Springer, 2019.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE transactions on neural networks and learning systems*, 32(11):4793–4813, 2020.

[20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115, 2020.

[21] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[22] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI magazine*, 38(3):50–57, 2017.

[23] Raha Moraffah, Mansooreh Karami, Ruocheng Guo, Adrienne Raglin, and Huan Liu. Causal interpretability for machine learning-problems, methods and evaluation. *ACM SIGKDD Explorations Newsletter*, 22(1):18–33, 2020.

[24] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2019.

[25] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

[26] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[27] C Mary Schooling and Heidi E Jones. Clarifying questions about "risk factors": predictors versus explanation. *Emerging themes in epidemiology*, 15(1):1–6, 2018.

[28] Amelia Jean Averitt. *Machine Learning Methods for Causal Inference with Observational Biomedical Data*. Columbia University, 2020.

[29] Benjamin Shickel, Patrick James Tighe, Azra Bihorac, and Parisa Rashidi. Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis. *IEEE Journal of Biomedical and Health Informatics*, 22(5):1589–1604, 2018.

[30] M. K. Ross, W. Wei, and L. Ohno-Machado. "Big data" and the electronic health record. *Yearbook of medical informatics*, 9:97–104, 2014.

[31] Liang Hong, Mengqi Luo, Ruixue Wang, Peixin Lu, Wei Lu, and Long Lu. Big data in health care: Applications and challenges. *Data and information management*, 2(3):175–197, 2018.

[32] Pranjul Yadav, Michael Steinbach, Vipin Kumar, and Gyorgy Simon. Mining electronic health records (EHRs): A survey. *ACM Computing Surveys*, 50(6), 2018.

[33] Rebecca Hermon and Patricia Williams. Big data in healthcare: What is it used for? *Proceedings of the 3rd Australian eHealth Informatics and Security Conference*, pages 40–49, 2014.

[34] Chonho Lee, Zhaojing Luo, Kee Yuan Ngiam, Meihui Zhang, Kaiping Zheng, Gang Chen, Beng Chin Ooi, and Wei Luen James Yip. Big Healthcare Data Analytics: Challenges and Applications. pages 11–41, 2017.

[35] Volker Tresp, J. Marc Overhage, Markus Bundschus, Shahrooz Rabizadeh, Peter A. Fasching, and Shipeng Yu. Going Digital: A Survey on Digitalization and Large-Scale Data Analytics in Healthcare. *Proceedings of the IEEE*, 104(11):2180–2206, 2016.

[36] Daniele Ravi, Charence Wong, Fani Deligianni, Melissa Berthelot, Javier Andreu-Perez, Benny Lo, and Guang Zhong Yang. Deep Learning for Health Informatics. *IEEE Journal of Biomedical and Health Informatics*, 21(1):4–21, 2017.

[37] Bogusław Cyganek, Manuel Graña, Bartosz Krawczyk, Andrzej Kasprzak, Piotr Porwik, Krzysztof Walkowiak, and Michał Woźniak. A Survey of Big Data Issues in Electronic Health Record Analysis. *Applied Artificial Intelligence*, 30(6):497–520, 2016.

[38] Anima Singh, Girish Nadkarni, Omri Gottesman, Stephen B. Ellis, Erwin P. Bottinger, and John V. Guttag. Incorporating temporal EHR data in predictive models for risk stratification of renal function deterioration. *Journal of Biomedical Informatics*, 53:220–228, 2015.

[39] Hieu H Pham, Tung T Le, Dat Q Tran, Dat T Ngo, and Ha Q Nguyen. Interpreting chest x-rays via cnns that exploit hierarchical disease dependencies and uncertainty labels. *Neurocomputing*, 437:186–194, 2021.

[40] Tim Hulsen, Saumya S. Jamuar, Alan R. Moody, Jason H. Karnes, Orsolya Varga, Stine Hedensted, Roberto Spreafico, David A. Hafler, and Eoin F. McKinney. From big data to precision medicine. *Frontiers in Medicine*, 6(MAR):1–14, 2019.

[41] Sabyasachi Dash, Sushil Kumar Shakyawar, Mohit Sharma, and Sandeep Kaushik. Big data in healthcare: management, analysis and future prospects. *Journal of Big Data*, 6(1), 2019.

[42] Clemens Scott Kruse, Rishi Goswamy, Yesha Raval, and Sarah Marawi. Challenges and Opportunities of Big Data in Health Care: A Systematic Review. *JMIR Medical Informatics*, 4(4):e38, 2016.

[43] Satya Narayan Shukla and Benjamin M. Marlin. Modeling Irregularly Sampled Clinical Time Series. 2018.

[44] Mohammad Taha Bahadori and Zachary Chase Lipton. Temporal-Clustering Invariance in Irregular Healthcare Time Series. (ii):1–16, 2019.

[45] Gloria Hyun-Jung Kwak and Pan Hui. DeepHealth: Deep Learning for Health Informatics. 2019.

[46] Samee U Khan, Albert Y Zomaya, and Assad Abbas. *Handbook of Large-Scale Distributed Computing in Smart Healthcare*. Springer, 2017.

[47] Kaiping Zheng, Jinyang Gao, Kee Yuan Ngiam, Beng Chin Ooi, and Wei Luen James Yip. Resolving the bias in electronic medical records. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2171–2180, 2017.

[48] Peter EH Schwarz, Jiang Li, Heiko Wegner, Stefan R Bornstein, Joana Lindström, and Jaakko Tuomilehto. An accurate risk score based on anthropometric, dietary, and lifestyle factors to predict the development of type 2 diabetes: response to schulze et al. *Diabetes care*, 30(8):e87–e87, 2007.

[49] Philip B Weerakody, Kok Wai Wong, Guanjin Wang, and Wendell Ela. A review of irregular time series data handling with gated recurrent neural networks. *Neurocomputing*, 441:161–178, 2021.

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

[51] Xiaoxing Wu, Hsin-Yao Wang, Peichang Shi, Rong Sun, Xiaolin Wang, Zhixiao Luo, Fanling Zeng, Michael S Lebowitz, Wan-Ying Lin, Jang-Jih Lu, et al. Long short-term memory model–a deep learning approach for medical data with irregularity in cancer predication with tumor markers. *Computers in Biology and Medicine*, 144:105362, 2022.

[52] Therese D Pigott. A review of methods for missing data. *Educational research and evaluation*, 7(4):353–383, 2001.

[53] Mutamba T Kayembe, Shahab Jolani, Frans ES Tan, and Gerard JP van Breukelen. Imputation of missing covariate in randomized controlled trials with a continuous outcome: Scoping review and new results. *Pharmaceutical statistics*, 19(6):840–860, 2020.

[54] Zhongheng Zhang. Missing data imputation: focusing on single imputation. *Annals of translational medicine*, 4(1), 2016.

[55] Arjun Puri and Manoj Gupta. Review on missing value imputation techniques in data mining. In *Proceedings of the International Conference on Machine Learning and Computational Intelligence, Sydney, Australia*, pages 6–11, 2017.

[56] Anil Jadhav, Dhanya Pramod, and Krishnan Ramanathan. Comparison of performance of data imputation methods for numeric dataset. *Applied Artificial Intelligence*, 33(10):913–933, 2019.

[57] Yanjie Duan, Yisheng Lv, Wenwen Kang, and Yifei Zhao. A deep learning based approach for traffic data imputation. In *17th International IEEE conference on intelligent transportation systems (ITSC)*, pages 912–917. IEEE, 2014.

[58] Ming Wang. Generalized estimating equations in longitudinal data analysis: a review and recent developments. *Advances in Statistics*, 2014, 2014.

[59] James W Hardin and Joseph M Hilbe. *Generalized estimating equations*. chapman and hall/CRC, 2002.

[60] David A Freedman. On the so-called "huber sandwich estimator" and "robust standard errors". *The American Statistician*, 60(4):299–302, 2006.

[61] Alan E Hubbard, Jennifer Ahern, Nancy L Fleischer, Mark Van der Laan, Sheri A Satariano, Nicholas Jewell, Tim Bruckner, and William A Satariano. To gee or not to gee: comparing population average and mixed models for estimating the associations between neighborhood risk factors and health. *Epidemiology*, pages 467–474, 2010.

[62] Zibo Tian, John S Preisser, Denise Esserman, Elizabeth L Turner, Paul J Rathouz, and Fan Li. Impact of unequal cluster sizes for gee analyses of stepped wedge cluster randomized trials with binary outcomes. *Biometrical Journal*, 64(3):419–439, 2022.

[63] Christin Juhnke, Susanne Bethge, and Axel C Mühlbacher. A review on methods of risk adjustment and their use in integrated healthcare systems. *International journal of integrated care*, 16(4), 2016.

[64] Mieke Deschepper, Kristof Eeckloo, Dirk Vogelaers, and Willem Waegeman. A hospital wide predictive model for unplanned readmission using hierarchical icd data. *Computer methods and programs in biomedicine*, 173:177–183, 2019.

[65] Amir Mosavi and Material Engineering. Deep Learning : a Review Deep Learning : a Review. (July), 2017.

[66] Riccardo Miotto, Li Li, Brian A. Kidd, and Joel T. Dudley. Deep Patient: An Unsupervised Representation to Predict the Future of Patients from the Electronic Health Records. *Scientific Reports*, 6(January):1–10, 2016.

[67] Nishita Mehta and Anil Pandit. Concurrence of big data analytics and healthcare: A systematic review. *International Journal of Medical Informatics*, 114(January):57–65, 2018.

[68] Mihalj Bakator and Dragica Radosav. Deep learning and medical diagnosis: A review of literature. *Multimodal Technologies and Interaction*, 2(3), 2018.

[69] Ahmad Al-Aiad, Rehab Duwairi, and Manar Fraihat. Survey: Deep Learning Concepts and Techniques for Electronic Health Record. *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, 2018-November:1–5, 2019.

[70] Fan Yang, Mengnan Du, and Xia Hu. Evaluating Explanation Without Ground Truth in Interpretable Machine Learning. 2019.

[71] Ben Dickson. EXPLAINABLE AI: LEARNING HOW AI MAKES DECISIONS.: EBSCOhost. page 15, 2019.

[72] Luciano Caroprese, Pierangelo Veltri, Eugenio Vocaturo, and Ester Zumpano. Deep learning techniques for electronic health record analysis. *2018 9th International Conference on Information, Intelligence, Systems and Applications, IISA 2018*, pages 1–4, 2019.

[73] Yichuan Wang, Lee Ann Kung, and Terry Anthony Byrd. Big data analytics: Understanding its capabilities and potential benefits for healthcare organizations. *Technological Forecasting and Social Change*, 126:3–13, 2018.

[74] Yi Luo, Huan-Hsin Tseng, Sunan Cui, Lise Wei, Randall K. Ten Haken, and Issam El Naqa. Balancing accuracy and interpretability of machine learning approaches for radiation treatment outcomes modeling. *BJR—Open*, 1(1):20190021, 2019.

[75] Awais Ashfaq. *Predicting clinical outcomes via machine learning on electronic health records*. Number 58. 2019.

[76] Kenneth David Strang and Zhaohao Sun. Hidden big data analytics issues in the healthcare industry. *Health Informatics Journal*, 2019.

[77] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T. Dudley. Deep learning for healthcare: Review, opportunities and challenges. *Briefings in Bioinformatics*, 2017.

[78] Saroj Kumar Pandey and Rekh Ram Janghel. Recent Deep Learning Techniques, Challenges and Its Applications for Medical Healthcare System: A Review. *Neural Processing Letters*, 50(2):1907–1935, 2019.

[79] Rocio Vargas, Amir Mosavi, and Ramon Ruiz. Deep learning: a review. *Advances in Intelligent Systems and Computing*, 2017.

[80] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M. Dai, Nissan Hajaj, Michaela Hardt, Peter J. Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, Patrik Sundberg, Hector Yee, Kun Zhang, Yi Zhang, Gerardo Flores, Gavin E. Duggan, Jamie Irvine, Quoc Le, Kurt Litsch, Alexander Mossin, Justin Tansuwan, De Wang, James Wexler, Jimbo Wilson, Dana Ludwig, Samuel L. Volchenboum, Katherine Chou, Michael Pearson, Srinivasan Madabushi, Nigam H. Shah, Atul J. Butte, Michael D. Howell, Claire Cui, Greg S. Corrado, and Jeffrey Dean. Scalable and accurate deep learning with electronic health records. *npj Digital Medicine*, 1(1):1–10, 2018.

[81] Yohan Jo, Lisa Lee, and Shruti Palaskar. Combining lstm and latent topic modeling for mortality prediction. *arXiv preprint arXiv:1709.02842*, 2017.

[82] Mahmoud Khademi and Nedialko S Nedialkov. Probabilistic graphical models and deep belief networks for prognosis of breast cancer. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 727–732. IEEE, 2015.

[83] Hoo-Chang Shin, Le Lu, Lauren Kim, Ari Seff, Jianhua Yao, and Ronald M Summers. Interleaved text/image deep mining on a very large-scale radiology database. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1090–1099, 2015.

[84] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. Distilling Knowledge from Deep Networks with Applications to Healthcare Domain. pages 1–13, 2015.

[85] Oscar Day and Taghi M Khoshgoftaar. A survey on heterogeneous transfer learning. *Journal of Big Data*, 4(1):1–42, 2017.

[86] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[87] Shuteng Niu, Meryl Liu, Yongxin Liu, Jian Wang, and Houbing Song. Distant domain transfer learning for medical imaging. *IEEE Journal of Biomedical and Health Informatics*, 25(10):3784–3793, 2021.

[88] Lior Rokach. Ensemble-based classifiers. *Artificial intelligence review*, 33(1):1–39, 2010.

[89] Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.

[90] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[91] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[92] Stefan Wager. Asymptotic theory for random forests. *arXiv preprint arXiv:1405.0352*, 2014.

[93] Gérard Biau, Luc Devroye, and Gäbor Lugosi. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research*, 9(9), 2008.

[94] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. *Proceedings - 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA 2018*, pages 80–89, 2019.

[95] Muhammad Aurangzeb Ahmad, Ankur Teredesai, and Carly Eckert. Interpretable machine learning in healthcare. *Proceedings - 2018 IEEE International Conference on Healthcare Informatics, ICHI 2018*, page 447, 2018.

[96] Zachary C Lipton. THe of Model The Interpretability. *Acm*, (june 2018):1–28, 2018.

[97] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 0210–0215. IEEE, 2018.

[98] Xuhong Li, Haoyi Xiong, Xingjian Li, Xuanyu Wu, Xiao Zhang, Ji Liu, Jiang Bian, and Dejing Dou. Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond. *arXiv preprint arXiv:2103.10689*, 2021.

[99] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, 2019.

[100] Arun Das and Paul Rad. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371*, 2020.

[101] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–45, 2018.

[102] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics (Switzerland)*, 8(8):1–34, 2019.

[103] Finale Doshi-Velez and Been Kim. Towards A Rigorous Science of Interpretable Machine Learning. (Ml):1–13, 2017.

[104] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018.

[105] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, 2020.

[106] Joost Bastings, Wilker Aziz, and Ivan Titov. Interpretable Neural Predictions with Differentiable Binary Variables. pages 2963–2977, 2019.

[107] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[108] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[109] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[110] Katherine J Strandburg. RULEMAKING AND INSCRUTABLE AUTOMATED DECISION TOOLS Author (s): Katherine J . Strandburg Source : Columbia Law Review , Vol . 119 , No . 7 , SYMPOSIUM : Common Law for the Age of AI (NOVEMBER 2019), pp . 1851-1886 Published by : Columbia Law Review Association , Inc . Stable URL : https://www.jstor.org/stable/10.2307/26810852 REFERENCES Linked references are available on JSTOR for this article : reference # references _ tab _ contents You may need to log in to JSTOR to access the linked references . RULEMAKING AND INSCRUTABLE AUTOMATED DECISION TOOLS. 119(7):1851–1886, 2019.

[111] Gabriëlle Ras, Marcel van Gerven, and Pim Haselager. Explanation Methods in Deep Learning: Users, Values, Concerns and Challenges. pages 19–36, 2018.

[112] Scott M. Lundberg and Su In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 2017-December(Section 2):4766–4775, 2017.

[113] Harmanpreet Kaur, Harsha Nori, Samuel Jenkins, Rich Caruana, Hanna Wallach, and Jennifer Wortman Vaughan. Interpreting Interpretability: Understanding Data Scientists' Use of Interpretability Tools for Machine Learning. *CHI Conference on Human Factors in Computing Systems Proceedings*, pages 1–14, 2020.

[114] Federico Baldassarre and Hossein Azizpour. Explainability Techniques for Graph Convolutional Networks. 2019.

[115] Cao Xiao, Edward Choi, and Jimeng Sun. Opportunities and challenges in developing deep learning models using electronic health records data: A systematic review. *Journal of the American Medical Informatics Association*, 25(10):1419–1428, 2018.

[116] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[117] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

[118] David Alvarez-Melis and Tommi S Jaakkola. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049*, 2018.

[119] Radwa ElShawi, Youssef Sherif, Mouaz Al-Mallah, and Sherif Sakr. Interpretability in healthcare: A comparative study of local machine learning interpretability techniques. *Computational Intelligence*, 37(4):1633–1650, 2021.

[120] Matthew Ryan Lavery, Parul Acharya, Stephen A Sivo, and Lihua Xu. Number of predictors and multicollinearity: What are their effects on error and bias in regression? *Communications in Statistics-Simulation and Computation*, 48(1):27–38, 2019.

[121] Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E Gonzalez. Nbdt: neural-backed decision trees. *arXiv preprint arXiv:2004.00221*, 2020.

[122] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475, 2015.

[123] Yang Zhao, Zoie Shui Yee Wong, and Kwok Leung Tsui. A Framework of Rebalancing Imbalanced Healthcare Data for Rare Events' Classification: A Case of Look-Alike Sound-Alike Mix-Up Incident Detection. *Journal of healthcare engineering*, 2018(2010):6275435, 2018.

[124] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[125] Bing Liu, Yiyuan Xia, and Philip S Yu. Clustering through decision tree construction. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 20–29, 2000.

[126] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[127] Ludmila I Kuncheva, Juan J Rodríguez, Catrin O Plumpton, David EJ Linden, and Stephen J Johnston. Random subspace ensembles for fmri classification. *IEEE transactions on medical imaging*, 29(2):531–542, 2010.

[128] Robert Bryll, Ricardo Gutierrez-Osuna, and Francis Quek. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern recognition*, 36(6):1291–1302, 2003.

[129] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pretraining. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019.

[130] Jiquan Ngiam, Daiyi Peng, Vijay Vasudevan, Simon Kornblith, Quoc V Le, and Ruoming Pang. Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*, 2018.

[131] G Maragatham and Shobana Devi. Lstm model for prediction of heart failure in big data. *Journal of medical systems*, 43(5):111, 2019.

[132] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[133] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[134] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: a survey and categorisation. *Information fusion*, 6(1):5–20, 2005.

[135] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th international conference on machine learning*, pages 231–238. Morgan Kaufmann Stanford, 2000.

[136] Sotiris B Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013.

[137] R Muthukrishnan and R Rohini. Lasso: A feature selection technique in predictive modeling for machine learning. In *2016 IEEE international conference on advances in computer applications (ICACA)*, pages 18–20. IEEE, 2016.

[138] Kevin Z Xin, David Li, and Paul H Yi. Limited generalizability of deep learning algorithm for pediatric pneumonia classification on external data. *Emergency Radiology*, 29(1):107–113, 2022.

[139] Carl F Sabottke and Bradley M Spieler. The effect of image resolution on deep learning in radiography. *Radiology. Artificial intelligence*, 2(1), 2020.

[140] Fei Teng, Yiming Liu, Tianrui Li, Yi Zhang, Shuangqing Li, and Yue Zhao. A review on deep neural networks for icd coding. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[141] Jose Camacho-Collados and Mohammad Taher Pilehvar. Embeddings in natural language processing. In *Proceedings of the 28th international conference on computational linguistics: tutorial abstracts*, pages 10–15, 2020.

[142] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*, 2021.

[143] Alexander Kurz, Katja Hauser, Hendrik Alexander Mehrtens, Eva Krieghoff-Henning, Achim Hekler, Jakob Nikolas Kather, Stefan Fröhling, Christof von Kalle, Titus Josef Brinker, et al. Uncertainty estimation in medical image classification: systematic review. *JMIR Medical Informatics*, 10(8):e36427, 2022.

[144] Himanshu Mittal, Avinash Chandra Pandey, Mukesh Saraswat, Sumit Kumar, Raju Pal, and Garv Modwel. A comprehensive survey of image segmentation: clustering methods, performance parameters, and benchmark datasets. *Multimedia Tools and Applications*, pages 1–26, 2021.