APPROVAL SHEET

Title of Thesis: Cognitive Intelligence in Relational Databases

Name of Candidate: Sushant Athley
Master of Science, 2017

Thesis and Abstract Approved: _____

Dr. Tim Oates
Professor
Computer Science and Electrical Engineering

Date Approved: _____

ABSTRACT

Title of Document:	COGNITIVE INTELLIGENCE IN RELATIONAL DATABASES. Sushant Athley, M.S. Computer Science, 2017.
Directed By:	Professor Tim Oates, Computer Science and Electrical Engineering

We evaluate the applicability of distributed language embedding techniques from the domain of natural language processing to relational data. Relational data is typically stored in SQL databases. We apply modern distributed representations of words (Tomas Mikolov 2013c) and paragraph (Quoc V. Le 2014) techniques to this structured data and attempt to unlock the potential of enhanced *cognitive* querying. The research intention is to be able to perform queries which are non-trivial to perform using the SQL dialect alone.

We tokenize the IMDB 5000 movie dataset to generate embeddings using *word2vec* and a modified version of *doc2vec* that we term as *row2vec*. We discuss the effects of various hyperparameter choices and tokenization techniques. We visualize these embedding using PCA and present the results for certain queries.

Keywords: *Word embedding, databases, word2vec, cognitive querying*.

COGNITIVE INTELLIGENCE IN RELATIONAL DATABASES.

By

Sushant Athley.

Thesis 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
Master of Science, Computer Science

2017

Dedicated to my loving parents Lt. Col. Ganesh and Sunila Athley.

Acknowledgements

I would like to thank my advisor and mentor Dr. Tim Oates for guiding me throughout this thesis. His smart mind and wisdom were very crucial to the completion of this thesis. I would also like to thank Dr. Konstantinos Kalpakis and Dr. Charles Nicholas for agreeing to be part of my thesis defense committee. A special thank you to Ashwinkumar Ganesan for all his help.

Table of Contents

List of Figures

Chapter 1: Introduction

Research Problem

In recent years, researchers have shown a tremendous amount of interest in language embedding and distributed representations of words. The origins can be traced back to (Tomas Mikolov K. C., 2013) with the skip-gram model for word embedding. These algorithms did not involve dense matrix multiplications and were thus very efficient. One could train 100 billion words in a day on a single optimized machine. Since then, these models have been applied to a variety of settings such as language translation (Jansen, 2017), sentiment analysis (Liu, 2017) (Yushi Yao, 2017), DNA sequencing (Ng, 2017) and product recommendations (Christophe Van Gysel, 2016) (Oren Barkan, 2016) (Ozsoy, 2016).

This thesis aims to evaluate the applicability of word embedding to the domain of relational data. Most data online are stored in traditional relational databases. This can be credited to their robust ACID compliance, ease of set based querying, industry grade uptime and security, and widespread availability of experts. SQL tables are employed to persist all kinds of data, from numerical sales figures to textual product reviews. Whereas distributed representations of words models have been known to perform well on large textual corpora such as news articles and Wikipedia entries, we try to understand the feasibility of word representations on SQL tables and the potential of unlocking enhanced querying capabilities. We consider various techniques of transforming SQL tables into a suitable input format for word embedding models (tokenizing) and shed light on the hyperparameter settings that should allow us to yield better models for our intended tasks.

Organization

This document is organized into 5 sections. The next section will discuss the background and state-of-the-art in word embedding. We will talk about the popular word2vec model and the newer doc2vec model as our choices for training on SQL tables. We develop our hypothesis and scope of our research. In section 3, we go deeper into analyzing the different choices for tokenization and hyperparameter options and their subsequent effects. In section 4, we present our findings using visualizations and sample query results. We end in section 5 by discussing our results and proposing some future research directions.

Chapter 2: Theory

Relational model

The relational database is based on the relational model. This relational model was first proposed by E.F. Codd in 1970 (Codd, 1970). The premise is very simple. Data for each business entity and measurement is organized into a collection of tables. Each table has horizontal *rows* (tuples) and vertical *columns* (attributes). A row and column intersect at a *cell*. A row represents one instance of a business entity or measurement. For example, in a table that stores employee information, you would have rows for Bob and Susie. The column stores a subset of the information for all the rows. All the cells for a column conform to a single data type (string, number, date etc.). One column (or a combination of columns) is typically designated as the *primary key* for the table. This primary key uniquely identifies each row in the table.

A software implementation of such a design is called a Relational Database Management System (RDBMS). Structured Query Language (SQL) has been the gold standard for interacting with these systems for many years.

Distributed Language Embeddings

The term **"distributed language embedding"** was originally coined by (Bengio, 2003). Language embedding is the approach of representing each token in a language vocabulary $\mathcal{V}$ as a vector in a low-dimension vector space $\mathbb{R}^n$, where *n* is the dimensionality of the resultant vectors. A token can be a single word like *'eat'* or *'Paris'*. It can be a phrase like *'machine learning'* or *'Baltimore Ravens'*. It can also be sentences, paragraphs or entire documents. For now, we decouple the grain from

the model and call these individual meaningful entities as *tokens*. Language embedding assigns a fixed-dimension numerical vector to each token, such that these vectors capture certain intrinsic properties of the tokens that they represent.

The framework design, algorithm and hyperparameters that are used to generate these token vectors are influenced by the qualities that are desired in the embeddings and by their transfer applications (Omer Levy Y. G., 2015). Also, the dimensionality of these output vectors is often dictated by the corpus size. Typically, we see that researchers strive to capture the semantic and syntactic relationships between word tokens. By this we mean that similar token vectors should be located closer to each other in a vector space than dissimilar tokens. For example, the vector for '*chair'* should be closer to the vectors for '*table'* and '*furniture',* than it is to the vector of '*kangaroo'*. Cosine-distance is popularly used as a measure of the distance between two vectors. These vectors also conform to linear transformations for capturing semantic relationships. They should be well formed to perform analogy queries such as: if '*King'* is to '*man'*, then '*Queen'* is to what? (answer: '*woman'*) See Figure 1. More recently, researchers have proposed that embeddings should better align with their transfer applications by maximizing the amount of *easily* accessible and *useful* information (Stanisław Jastrzebski, 2017).

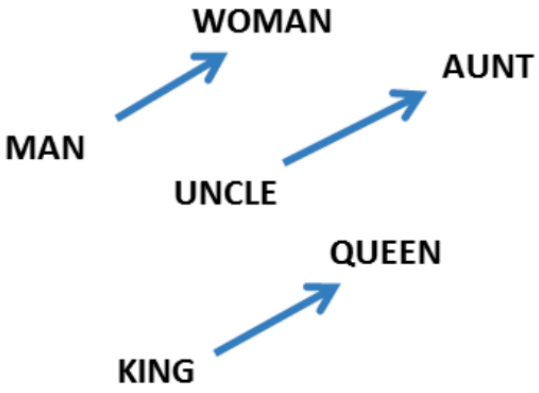


Figure 1: Gender relation captured by vector offsets.

Word2Vec

Skip Gram with Negative Sampling (SGNS), more commonly known as *Word2vec* (Tomas Mikolov I. S., 2013c), is one of the most popular and widely adopted distributed language embedding models. The origins of *word2vec* can be traced to the introduction of the skip-gram model (Tomas Mikolov K. C., 2013). It involves using a probabilistic feed-forward neural network (Y. Bengio, 2003) with stochastic gradient descent (SGD) and backpropagation (D. E. Rumelhart, 1986). The basic premise behind *word2vec* (Tomas Mikolov I. S., 2013c) and its various incarnations (Quoc V. Le, 2014) is that "*a word is characterized by the company it keeps*", as popularized by Firth. What this means is that words in similar contexts have similar meanings, which is the foundation of the distributional hypothesis. The model picks a *target* word in a sentence, and then looks at its nearby *context* words. It then picks one of these nearby context words at random and predicts the probability of every word in our *vocabulary* of being this nearby context word. See Figure 2. The model learns the co-occurrence statistics from the number of times each *target-context pair* shows up. The vector space word representations are implicitly learned by the input layer weights as part of the prediction task. This model is efficient because it does not involve dense matrix multiplications. This context-based prediction model far outperforms previous count-based models (Marco Baroni, 2014). The authors of word2vec proposed two models, namely the continuous *bag-of-words* model and the *skip-gram* model. See Figure 3.

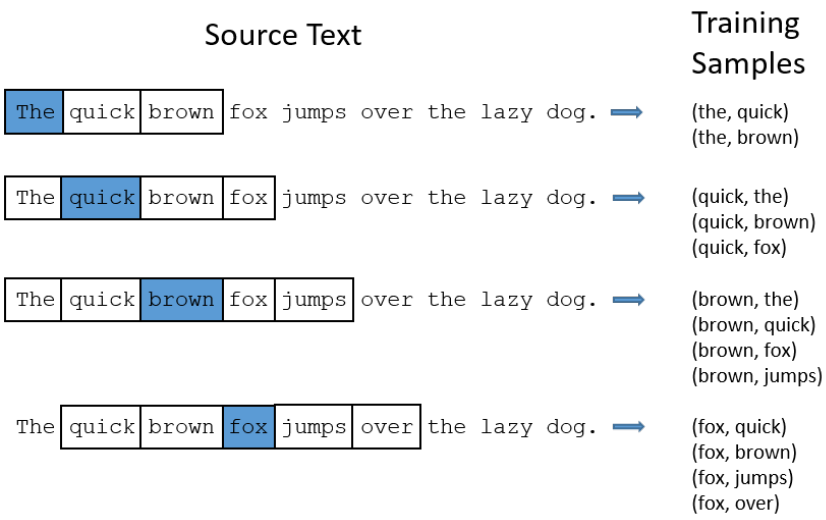


Figure 2: Forming *target-context* pairs from a sentence with a window size of 2

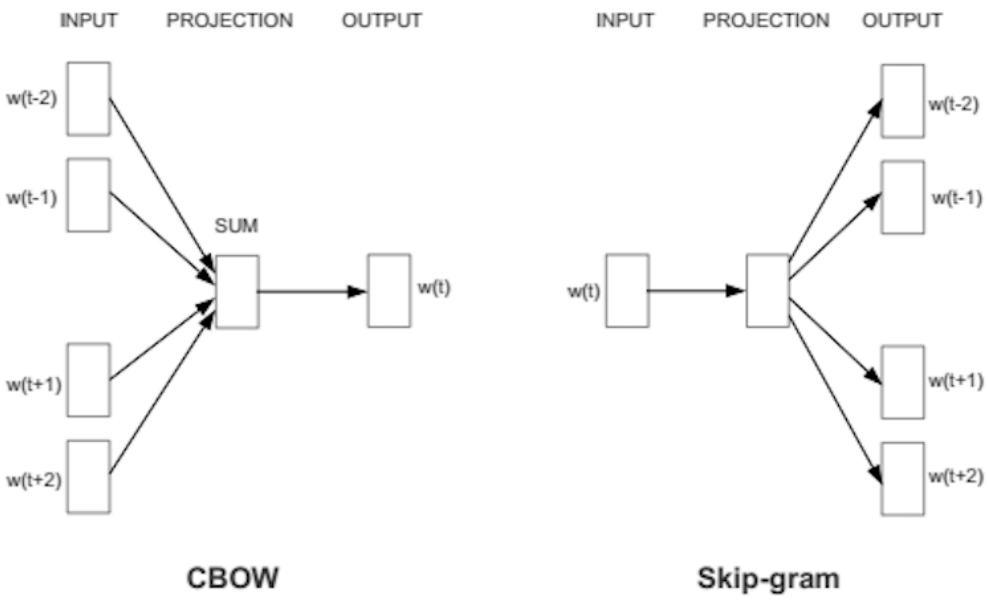


Figure 3: Model log-linear architectures for continuous bag-of-words and skip-gram

(Omer Levy Y. G., 2014) describes word2vec as implicitly factorizing a word-context matrix whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global context. (Rong, 2016) and (Yoav Goldberg, 2014) have done a remarkable job at explaining some of the more nuanced and complex rationales and techniques behind the model. Therefore, we will limit ourselves to a short summary here.

Paragraph Vectors (PV)

Generating paragraph vectors (Quoc V. Le, 2014) uses an unsupervised algorithm that learns fixed-length feature representations for variable-length pieces of texts. These texts can be sentences, paragraphs, or entire documents. These models are an extension of the models for learning word vectors and have achieved state-of-the art performance results on sentiment-analysis and text-classification tasks.

A *context* is defined as a fixed length sliding window over a sentence. This model learns a unique vector representation for every unique word, as well as every paragraph in the corpus. The prediction task involves averaging or concatenating the word vectors and the paragraph vector to predict the next word in the context. See Figure 4.

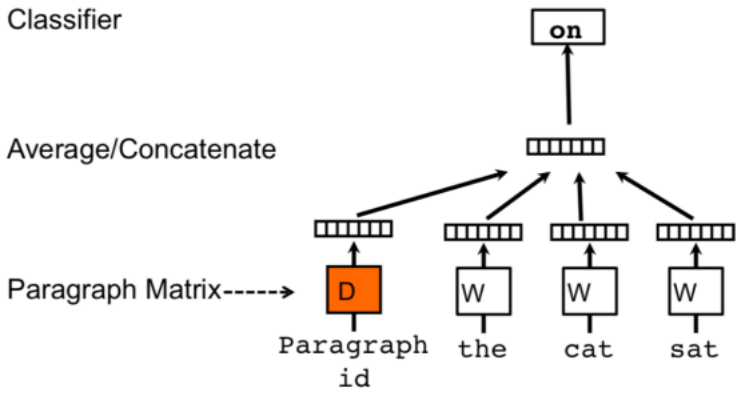


Figure 4: Distributed Memory Model of Paragraph Vectors (PV-DM)

The paragraph vector is common to all contexts generated from the same paragraph. The word vectors are shared across all paragraphs in the corpus. The paragraph vector appears in every context and its corresponding prediction tasks, and thus, it learns information about the paragraph, and becomes a general-purpose identifier vector for the paragraph. Paragraph vectors share their advantages with word vectors such as being able to work on unlabeled data. They too learn semantic meanings of words and capture relationships in a multidimensional vector space. They both scale gracefully on large corpora.

Paragraph vectors come in a second simpler flavor called the *Distributed Bag of Words version of Paragraph Vectors* (PG-DBOW) See Figure 5.

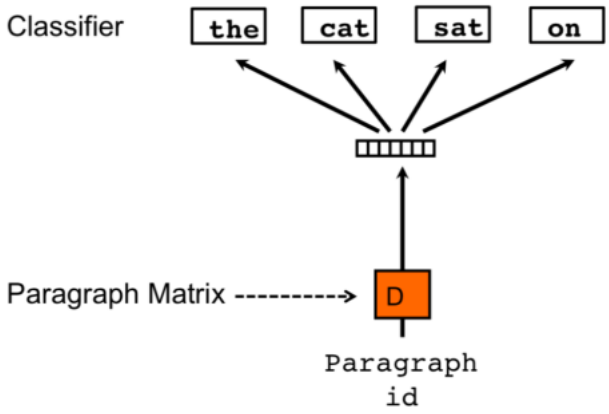


Figure 5: Distributed Bag of Words version of Paragraph Vector (PV-DBOW)

This variation forces the model to predict randomly sampled words from the paragraph. This model generates only paragraph vectors and no word vectors. The prediction task involves predicting words randomly sampled from the paragraph in the output. This makes the model conceptually simpler and reduces the storage overhead. PV-DM has shown to outperform PV-DBOW on many different tasks. But

the combination of PV-DM and PV-DBOW work consistently better and is therefore recommended.

Others

GloVe

GloVe, (Manning, 2014) short for **Glo**bal **Ve**ctors, is modeled around exploiting the *global word occurrence statistics* in the corpus to learn representations of words. The intuition behind GloVe is that there is semantic meaning captured in the *ratio of word-word co-occurrence probabilities*. As an example, *'steam'* frequently co-occurs with *'gas'*, while not so much with *'solid'*. In the same way, *'ice'* frequently co-occurs with *'solid'* a lot more than it does with *'gas'*. Both *'steam'* and *'ice'* co-occur equally with related terms like *'water'* and unrelated terms like *'fashion'*. While word2vec implicitly learns word representations in its hidden embedding layer as a by-product of a prediction task, GloVe does so explicitly. The training objective of GloVe is to learn word representations in a way such that the dot-product of two word vectors is equal to the *log* of the probability of the words' co-occurrence. A thorough comparison between GloVe and word2vec is difficult because of the range of possible hyperparameter choices such as vector length, sliding window size, training iterations/epochs and corpus size (Omer Levy Y. G., 2015).

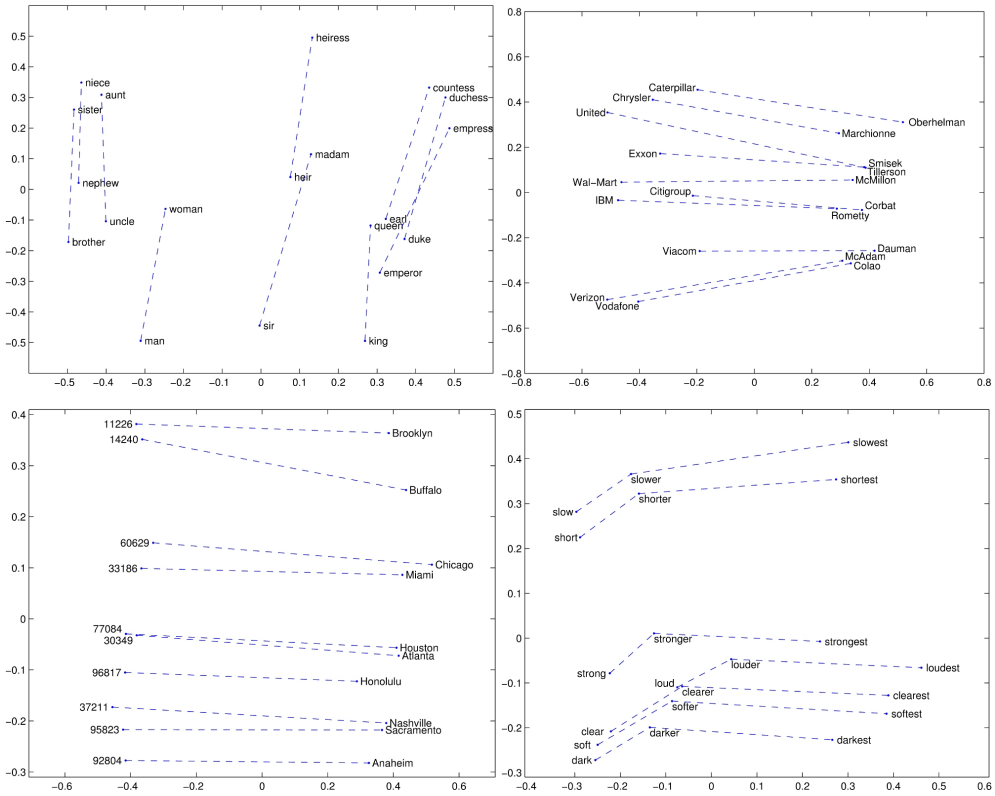
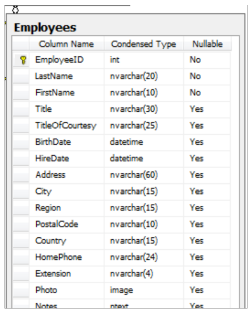


Figure 6: Vector relations as captured by GloVe (a) man - woman (b) company - ceo (c) city - zip-code (d) comparative – superlative (GloVe: Global Vectors for Word Representation)

Hypothesis

Language Embedding and relational data

As we've seen until now, word embedding models have received a lot of attention from researchers from a vast landscape of applicable areas. This thesis aims to evaluate the applicability of this newfound spirit to unlock the potential of enhanced cognitive querying on data stored in traditional relational databases.



Employees

Column Name	Condensed Type	Nullable
EmployeeID	int	No
LastName	nvarchar(20)	No
FirstName	nvarchar(10)	No
Title	nvarchar(30)	Yes
TitleOfCourtesy	nvarchar(25)	Yes
BirthDate	datetime	Yes
HireDate	datetime	Yes
Address	nvarchar(60)	Yes
City	nvarchar(15)	Yes
Region	nvarchar(15)	Yes
PostalCode	nvarchar(10)	Yes
Country	nvarchar(15)	Yes
HomePhone	nvarchar(24)	Yes
Extension	nvarchar(4)	Yes
Photo	image	Yes
Notes	ntext	Yes

Figure 7: Employee table from the Northwind database

For explaining this proposition, let us run with an example of the typical Employee database in Figure 7. This table (or view) represents the details of all the employees at an organization. Most data warehouses would deem this is a *dimension table*[1] and as such it would be wide and descriptive. Commonly found attributes would be *id, name, age, title, region, evaluation, salary,* etc. These attributes collectively describe the employee in the system. The values for these fields are expressed in dates, numbers and words (or combinations of words) borrowed from the English language. But because of the structured nature of the table, a row of tabular data does not read like any other grammatically correct (or consistent) sentence found in an English text. A row in the table to describe an employee called Bob, would look something like Figure 8.

EmployeeID	LastName	FirstName	Title	Salary
511	Elliot	Sam	Chief Marketing Officer	60000
512	Thorton	Bob	Area Sales Manager	45000
513	Selleck	Tom	Region Head	55000

Figure 8: An example employee SQL database table

1 https://en.wikipedia.org/wiki/Dimension_(data_warehouse)#Dimension_table

On the other hand, a text corpus, like the employee directory magazine, one that more closely resembles the ideal input dataset to a word embedding model, would have one or more sentences like this to describe Bob,

"… . Bob Thorton is an enthusiastic and hardworking employee. He holds the title of Area Sales Manager for the district of Baltimore County. He lives at 1101, Elm Ridge Avenue. …".

Now when this sentence, as part of a very large textual corpus, is fed into a language embedding model, the corresponding output word vectors for *'Bob'*, *'Manager', 'works',* etc. will capture their natural language semantics and relationships. We will start to see parts of speech such as the nouns, adjectives and verbs cluster together in the restricted vector space. This is because of their positions in sentences relative to other types of words, as dictated by the principles and structure of the English grammatical system. They will repeatedly appear in context windows with certain other words that will afford the model to generalize on the meaning of these words.

Now if we feed the rows of data from the employee table into this model, the model can again capture meaning and relationships for the cell values based on the colocation of tokens in a row and the probability of their presence in any given row. We begin by tokenizing the table into a list of sentences. There are many ways of doing this that involve many tradeoffs which we will discuss later. A simple example of tokenizing the Bob tuple from the table into a textual sentence would be,

"EmployeeID 512 FirstName Bob LastName Thorton Title Area Sales Manager Region Baltimore County Evaluation High…".

Next, we generate our input textual corpus by placing such sentences for all the rows in the table one after the other and feed this into a language embedding framework. The same language embedding model that could learn vector representations for words and paragraphs by leveraging the consistencies in the

English grammatical system, can now be used to learn meaningful information about tuple values which are expressed in the structure and system of the relational language. We can learn vectors for tuple values *'Bob'*, '*Manager'* etc. and capture relationships between tokens in a row and across tuples. We can also learn vectors for entire rows, tables or databases, using only the raw row data stored in the database. This provides us with a vector space model of the relational data. This thesis is inspired by, and extends the work done in *"Enabling Cognitive Intelligence Queries in Relational Databases using low-dimensional word embeddings"* (Rajesh Bordawekar, 2016). They implemented a prototype system on Apache Spark[2] to demonstrate the power of Cognitive Intelligence (CI) queries. The authors of the paper describe enhanced querying capability by having each database text entity associated with a low dimensional vector that captures its semantic and syntactic qualities. The paper outlines a tokenization process and feeds the *textified* input into word2vec. This made possible a new orthogonal view over the database in a vector space as compared to the traditional SQL mechanisms. The authors then defined SQL user defined functions (UDFs) that can leverage these vectors along with the SQL API when querying. We extend the premise in this thesis in a few different directions,

1. We delve deeper into the choices and their effects of different tokenization approaches and hyperparameter choices.
2. We propose the superiority of paragraph vectors (PV) over vanilla word2vec to capture row and token representations simultaneously.
3. We describe how word embedding models, which are run on structured relational data, capture meaning and relationships of tokens differently than running the same models on text corpora.

2 http://spark.apache.org

Scope

This thesis aims at describing the potential of enhanced querying on relational databases using distributed word embedding models from the domain of natural language processing. We apply these techniques to data stored in structured relational tables. The type of data undertaken by this process is typically dimensional data with low to moderate numerical measurement values. A primarily *factual table*[3] that is largely used to store continuous numerical values will lead the model to overfit aggressively. Although the intention is to slightly overfit the model as there are few to none outside participants involved in the execution of the query, we still want to generate models that can generalize well and work in tandem with external inputs and information in the future. We also omit BLOB data types in tables as this will be part of another research direction of vectorizing textual tokens and binary data in a common vector space.

3 https://en.wikipedia.org/wiki/Fact_table

Chapter 3: Methods

Introduction

In this section, we describe our process pipeline and discuss the decisions taken regarding the choices in models and hyperparameter. We begin with our input relational structure (Figure 9). This can be a standalone table in a relational database, like the Employee table discussed earlier. This can also be a subset, superset or computed view of a single table. We pick a view of the database that captures a *domain of interest* that we wish to explore and query. For example, we could use a joined view (via foreign keys) of all the tables related to the HR department of an organization (superset). We could also use a smaller portion of a single larger table by limiting the number of rows and/or columns based on some heuristic (subset).

Once we have our input table view, the next step is to *tokenize* and *textify* the relational data. Technically, a token is simply a discernible unique combination of characters strung together without any spaces between them. A sentence is an ordered collection of such tokens that is complete and adheres to the structure and rules of some overarching system. A text corpus is a set comprised of many such sentences. Tokenization is the process of extracting *word-like* constructs from the table. We discuss different tokenization approaches and their tradeoffs in the following few sections.

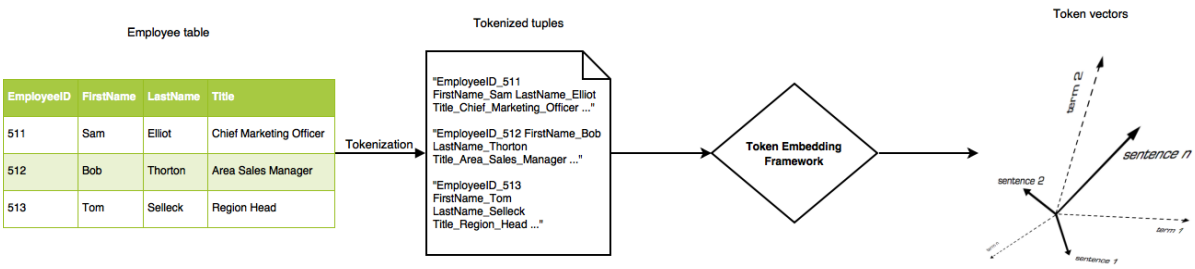


Figure 9: Pipeline to generate token vectors for a table

Upon tokenization, we arrange these tokens into sentences and documents. We feed the resulting dataset as an input to word[4] and document[5] embedding frameworks. With word2vec, we learn word vector representations for every unique token in the table. For doc2vec, we present each tokenized row of the table as a separate document to the model. This way we can generate word vectors for individual tokens as well as row vectors for each individual row in the table. We evaluate the effect of certain hyperparameters on the quality of these vectors. We reduce the dimensionality of the resultant vector representations to plot these token and row vectors in a 2D vector space. This allows us to visually explore the captured semantic qualities by the model. These vectors, along with the SQL API, provide a larger combined query surface over the dataset. We perform novel similarity and analogy queries over this dual view and present the results.

Dataset

For the experiments, the dataset chosen is the IMDB 5000 movie dataset from Kaggle (IMDB 5000 Movie Dataset, n.d.). This dataset captures 27 variables for 5043 movies and 4906 posters spanning across 100 years in 66 countries. The 27 attributes captured are,

S. Number	Column Name	Data Type
1	*movie_title*	STRING
2	*color*	STRING
3	*num_critic_for_reviews*	NUMERIC
4	*movie_facebook_likes*	NUMERIC
5	*duration*	NUMERIC
6	*director_name*	STRING
7	*director_facebook_likes*	NUMERIC
8	*actor_3_name*	STRING
9	*actor_3_facebook_likes*	NUMERIC
10	*actor_2_name*	STRING
11	*actor_2_facebook_likes*	NUMERIC
12	*actor_1_name*	STRING
13	*actor_1_facebook_likes*	NUMERIC
14	*gross*	NUMERIC
15	*genres*	STRING
16	*num_voted_users*	NUMERIC
17	*cast_total_facebook_likes*	NUMERIC
18	*facenumber_in_poster*	NUMERIC
19	*plot_keywords*	STRING
20	*num_user_for_reviews*	NUMERIC
21	*language*	STRING
22	*country*	STRING
23	*content_rating*	NUMERIC
24	*budget*	NUMERIC
25	*title_year*	DATE
26	*imdb_score*	NUMERIC
27	*aspect_ratio*	NUMERIC

There are 2399 unique director names and thousands of actor/actress names in the dataset. This dataset is a combination of dimensional and factual data. It has a healthy proportion of textual data (such as *plot_keywords, genres, movie_title, actor_1_name*) and numerical values, which are both categorical (*title_year, aspect_ratio*) and continuous (*gross, director_facebook_likes*).

Tokenization

Tokenization is the overall process of transforming our relational dataset into a large textual corpus. This corpus does not have to be grammatically well formed sentences. There are several considerations when effectively tokenizing a table. We will discuss them here.

Dealing with textual fields

Textual fields such as *'movie_title'*, *'plot_keywords'*, *'genre'* can be tokenized in several different ways. The value in a table cell can be appended to the column name for the cell to form a single token. This will assist us in recognizing the affiliation of the value to the column. We can use spaces between different columns to identify token boundaries. For example, we may tokenize a row from our movie metadata database as,

"... movie_name_Guardians_of_the_galaxy actor_1_name_Bradley_Cooper actor_2_name_Vin_Diesel …"

It makes sense to combine phrases such as *"Guardians of the galaxy"* and *"Bradley Cooper"* into *"Guardians_of_the_galaxy"* and *"Bradley_Cooper"*. This is because

we are not interested in the individual token vectors for each of these words independently. Researchers have extended models to achieve phrase-level representations (Jeff Mitchell, 2010), but because we are guaranteed that each value stored in a cell is atomic and therefore indivisible (1NF), we can confidently convert such phrases into a single token.

Other approaches to tokenize the table involve completely ignoring the column names altogether. This would yield a textified row as,

"... Guardians_of_the_galaxy Bradley_Cooper Vin_Diesel …"

or separating the column name and value by a space delimiter and treating them as separate tokens.

"... movie_name Guardians_of_the_galaxy actor_1_name Bradley_Cooper actor_2_name Vin_Diesel …"

All these approaches have an impact on the co-occurrence dynamics of the tokens that we are interested in. We can adjust the context window size to some extent and accommodate these approaches. The decision to include/exclude column names from the tokenization process is influenced by the queries that we intend to perform on the resultant vectors. We will discuss this further in the section on Hyperparameter considerations.

Dealing with numerical fields

Continuous numerical values are inherently a bad fit for language embedding and various other machine learning tasks. Numeric column values for *'gross'*, *'budget'* and *'director_facebook_likes'* will cause the model to generalize poorly as these machine learning models do not handle continuous values well and focus on nominal

feature spaces (Dougherty, 1995). The simplest solution is to *discretize* the values into *k* bins of equal interval widths. Here *k* can be a user specified parameter or computed using some heuristic based on the distribution of the values. This technique is not ideal because outliers will drastically skew the range (Catlett J. , 1991). Another approach is to discretize the continuous variables into *k* equal frequency bins where each bin contains an equal number of values. This way we end up with discretized column value tokens like *'budget_high'*, *'gross_low'* and *'director_facebook_likes_medium'*. It is likely that we lose out on information by transforming a continuous value with higher precision into a lower cardinality nominal value. More advanced discretization strategies have been studied which involve decision trees (Holte, 1993) and minimal entropy heuristics (Catlett J. , 1991) (U M Fayyad, 1993) to preserve as much of this information as possible, and allow machine learning algorithms to generalize better. We will leave these more advanced techniques and their effects out of the scope of this thesis.

Other data types

Other attributes such as Booleans and NULLs can also be accommodated using the same techniques as String values. Dates can either be treated as nominal or continuous depending on the domain of the values.

Models

Word2Vec

We start our experimentation by feeding our tokenized table corpus into *word2vec* from the genism software toolkit[4]. We consider each tokenized row as a separate sentence and pass a collection of all such sentences to the model. As a result, we learn vector representations for the entire vocabulary of tokens in the table in a vector space.

Row2Vec

We use a modified version of PV[5] from the genism software toolkit as something we would like to term as *row2vec* from here on. Just as in word2vec, we treat each tokenized table row as a sentence. But we also treat each such sentence as a paragraph. Row2Vec generates vectors for each unique token in the table and each sentence (row) as an implicit by-product of its explicit prediction task. We consider these sentence vectors to be row vectors. These row vectors capture certain intrinsic qualities of the entire row and help us in querying the table. We will discuss the hyperparameter choices for these models in the next section.

Hyperparameter considerations

Much work has been done recently in coming up with robust recommendation lists of hyperparameter choices for word embedding models such as word2vec and PV (Omer Levy Y. G., 2015) (Jey Han Lau1, 2016). We further investigate these choices in the backdrop of applying these models to relational data. This discussion aims to shed some light on the effect of certain hyperparameter choices.

---

4 https://radimrehurek.com/gensim/models/word2vec.html

5 https://radimrehurek.com/gensim/models/doc2vec.html

Column order and Context Windows

The column order for tokenization coupled with the size of the sliding context window for the model can have a significant impact on the output vector representations for the table tokens. The documentation[4] defines the *context window* parameter as the "maximum distance between the current and predicted word *within a sentence*". Note here that a context window does not overlap across sentences. Also, the context window extends on both sides of the target word. Therefore, the effective window size is twice this context window parameter value. Let us run with an example tokenized row from our dataset,

```
"… director_name George_Lucas num_critic_for_reviews_high
duration_high director_facebook_likes_low actor_2_name
Peter_Cushing actor_1_facebook_likes_high gross_high genres
Action Adventure Fantasy Sci-Fi actor_1_name Harrison_Ford
movie_title Star_Wars:_Episode_IV_-_A_New_Hope   plot_keywords
death star empire galactic war princess rebellion language
English country USA content_rating PG budget_medium
actor_2_facebook_likes_high imdb_score_high aspect_ratio_medium
…"
```

In the case of word2vec and row2vec (PV), we treat each tokenized row as a sentence. Additionally, for row2vec we consider each paragraph to be comprised of a single sentence. This way the learned paragraph vectors represent the entire row in the vector space. The recommended context size for these models on English texts is 5-10. But in our case, we have some additional pieces of information. All our sentences will roughly have the same number of words as each row has a fixed number of columns. Also, they will all have a similar structure. For example, the second word in every sentence will be the name of the director, which will always be preceded by the keyword *'director_name',* which is the name of the column. We have a few different options here. We could set the context window to be much smaller (1 or 2) than the average length of our tokenized row sentences (Figure 10). This way any target word like *'Harrison_Ford'* will only have a few nearby context words from the entire row. This is generally recommended to yield more topical vectors. This would lead us to suspect that the order of the columns in the tokenization

process might play a more significant role in the final output representations of the tokens. On the other hand, if we set the context window size to be larger than the largest row sentence that we have, we will have all the tokens from the row appear in the context of every word considered as a target word (Figure 11). This approach is recommended for more generalized representations. With this approach, the column order has little to no impact on the final word representations.

"… director_name George_Lucas num_critic_for_reviews_high duration_high director_facebook_likes_low actor_2_name Peter_Cushing actor_1_facebook_likes_high gross_high genres Action Adventure Fantasy Sci-Fi actor_1_name **Harrison_Ford** movie_title Star_Wars:_Episode_IV_-_A_New_Hope plot_keywords death star empire galactic war princess rebellion language English country USA content_rating PG budget_medium actor_2_facebook_likes_high imdb_score_high aspect_ratio_medium …"

Figure 10: Context window for 'Harrison Ford' with context size = 1

"… director_name Joss_Whedon num_critic_for_reviews_high duration_high director_facebook_likes_low actor_3_facebook_likes_high actor_2_name Robert_Downey_Jr. actor_1_facebook_likes_high gross_high genres Action Adventure Sci-Fi actor_1_name Chris_Hemsworth movie_title Avengers:_Age_of_Ultron num_voted_users_high cast_total_facebook_likes_high actor_3_name Scarlett_Johansson facenumber_in_poster_high plot_keywords artificial intelligence based on comic book captain america marvel cinematic universe superhero num_user_for_reviews_high language English country USA content_rating PG-13 budget_high title_year_high actor_2_facebook_likes_high imdb_score_high aspect_ratio_medium …

… director_name George_Lucas num_critic_for_reviews_high duration_high director_facebook_likes_low actor_2_name Peter_Cushing actor_1_facebook_likes_high gross_high genres Action Adventure Fantasy Sci-Fi actor_1_name **Harrison_Ford** movie_title Star_Wars:_Episode_IV_-_A_New_Hope plot_keywords death star empire galactic war princess rebellion language English country USA content_rating PG budget_medium actor_2_facebook_likes_high imdb_score_high aspect_ratio_medium …

… director_name Mel_Gibson num_critic_for_reviews_medium duration_high director_facebook_likes_low actor_3_facebook_likes_medium actor_2_name Patrick_McGoohan actor_1_facebook_likes_medium gross_high genres Biography Drama History War actor_1_name Mhairi_Calvey mo

```
vie_title Braveheart num_voted_users_high cast_total_facebook_like
s_medium actor_3_name James_Robinson facenumber_in_poster_low plot
_keywords 14th century legend revolt scotland tyranny num_user_for
_reviews_high language English country USA content_rating R budget
_high title_year_low actor_2_facebook_likes_medium imdb_score_high
aspect_ratio_medium …"
```

Figure 11: Context window for 'Harrison Ford' with context window size > tokenized row length

We should be wary of the fact that some implementations of the algorithm consider a *dynamic context window*. Under this scheme, more *weight* is assigned to closer words, as closer words are considered to have more importance to the meaning of the word. Another thing to keep in mind is that the size of the window is sometimes not fixed either. The actual window size is *dynamic, i.e.* it is sampled uniformly between 1 and a maximum window size parameter during the training phase. Both these schemes affect our hypothesis and care must be taken to understand the consequences of these hyperparameters on the quality of the output vectors.

Min-count

The model ignores all words whose frequency is lower than this *min-count* hyperparameter. This hyperparameter is tuned based on the input dataset's vocabulary statistics. In our case, we will set it to 1. This is because we want our model to generate vectors even for the words that occur only once in the entire table. We cannot afford to **not** have a vector for any table token, as we will not be able to include that term in our querying. This is a particularly important requirement for us because, as we know, every row has at least one unique primary key.

Hierarchical Softmax

Given a sequence of words $w_1, w_2, w_3, \ldots, w_T$, the objective of the word2vec model is to maximize the average log probability that is,

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

Here *c* is our context under consideration. As we can see the size of *W* can be very large for textual corpora. To circumvent this inefficiency, hierarchical softmax (Frederic Morin, 2005) posits that we can obtain a probability distribution by simply using $\log_2 W$ nodes. During our experimentation, we noticed that we achieved our best results by employing hierarchical softmax.

Negative Sampling

This hyperparameter can be used *in lieu* of hierarchical softmax. This integer hyperparameter specifies how many *noise-words* should be drawn in the prediction task. In a typical neural network, all weights are tweaked with each pass over the training sample. As we are generally working with large text corpora and vocabularies, our neural network models can have millions or billions of weights. Negative sampling exists to reduce this computational burden. Negative sampling is a simplified form of Noise Contrastive Estimation (NCE). NCE posits that a good model should be able to differentiate between data and *noise* words using logistic regression. In any given prediction task, we select a small sample of negative noise words which are outside of the given context window, and use them for prediction, along with the positive words from the context. It is for the reader to decide whether this will lead to better representations for token and/or row vectors. In our experiments, we found a low value for this hyperparameter to better help generalize the model and higher values to better cluster the column values.

Sample

This is the *threshold for configuring which higher-frequency words are* ***randomly*** *down sampled*. Basically, the model ignores some instances of the words it sees often. These words that occur very frequently generally carry less meaning. This is an

essential parameter that resists the more popular words from overpowering the model. Recommended values for this hyperparameter are between 0.001 and 0.00001, with the default being 0.001.

Iterations/Epochs

This is the number of times the sliding window goes over the text corpus. The recommended value for this hyperparameter in research papers is between 10 and 20. We feel this is a good starting point for our experiments as this will allow the model to capture the semantics of our row token vectors optimally. We can go higher than this recommended number, but we must be careful that running the models with higher iteration counts takes the model a long time to run.

All the other hyperparameters to the model offer little help to our end goal of efficiently and effectively representing vector representations for the table vocabulary. Some of these hyperparameters are the number of worker threads, and whether to use hierarchical softmax as a more efficient softmax and the learning rate. The final output dimension size of the vector representations is a special case. The recommended value for this hyperparameter is between 200 to 300, but we have seen certain researchers going as high as 800. Because our dataset under consideration is relatively small, we choose to hover around the 100-300 mark.

Measurement

Once we have learned vector representations for our token vocabulary using word2vec and row vectors using Row2Vec, we reduce the dimensionality of the vectors by using Principal Component Analysis (PCA) and plot them in a 2D space as a scatter plot. This gives us a visual glimpse of the quality of the semantic relationships captured by our models.

One of the foundational strengths of relational databases is to be able to identify a row uniquely by its primary key. With our word2vec experiments, there are two possible ways to identify a row uniquely in the vector space. The first one is to average the token vectors for all the tokens in the row and treat this average as the unique row representation. The second way is to consider the vector for the unique primary key as the row vector (in our case the *movie title*). Again, column order and context window size play a non-trivial role when choosing any one of these approaches. In the case of row2vec, we already have a vector representation for every row as a by-product of the prediction task.

We execute certain novel similarity queries on our dataset using the learned vector representations. We build a simple recommendation engine, which can recommend movies to users based on their likes and dislikes.

Chapter 4: Findings

Overview

We successfully ran our Word2Vec and Row2Vec models on our tokenized IMDB dataset. Our goal was to optimize the model on three types of tasks. These tasks are ***similarity*** (what tokens are near *'Leonardo Di Caprio'?*), ***analogy*** (if *'Inception'* is to *'Leonardo Di Caprio'*, then what is to *'Tom Cruise'?*) and ***additive composability*** (what is *'Leonardo Di Caprio' + 'Kate Winslet'?*). To get a better idea of the effects of the hyperparameters on the model, we tested different hyperparameter combinations. Keeping all other things constant, we individually introduced variation in context size, negative sampling, output dimensionality and the number of training iterations. We reduced the dimensionality of the resultant token vectors for each of these experiments using Principal Component Analysis (PCA) and color coded the various column values to better illustrate clustering patterns. In the next two sections, we present these visualizations and sample outputs.

Word2vec Visualizations

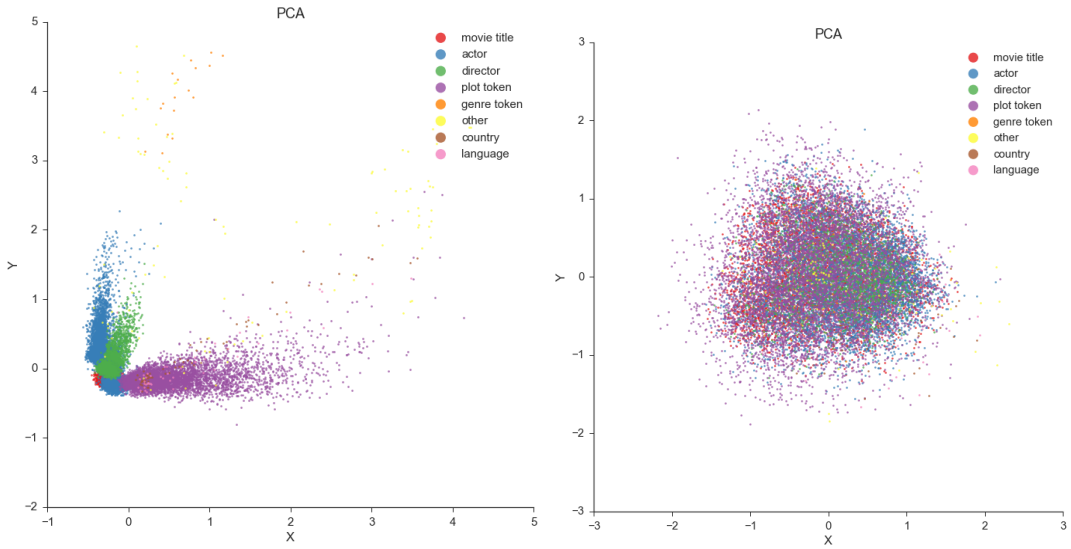


Figure 12: Context size = 1 vs Context size = 30 (300-dimension skip-gram with hierarchical softmax)

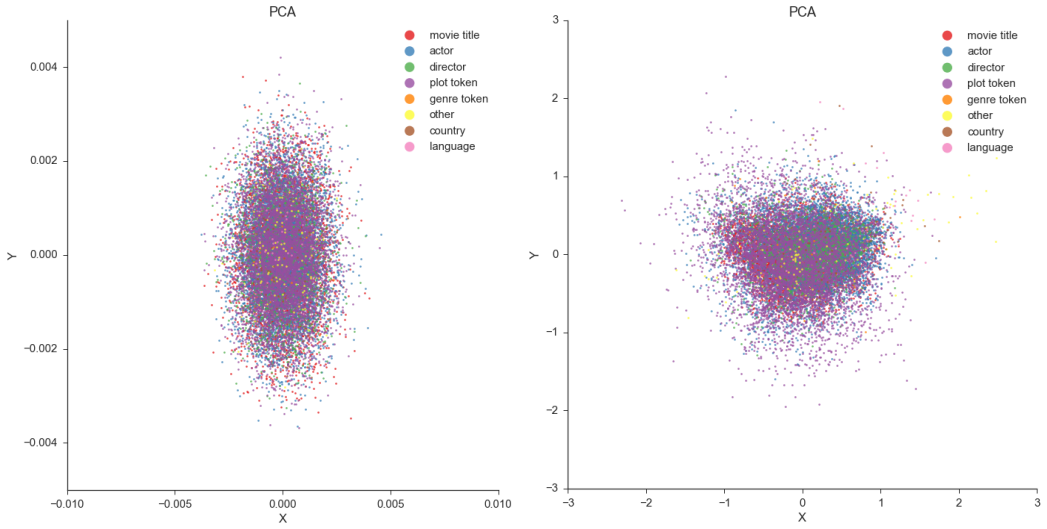


Figure 13: Negative sampling = 0 vs Negative sampling = 25 (300-dimension skip-gram with context size = 30)

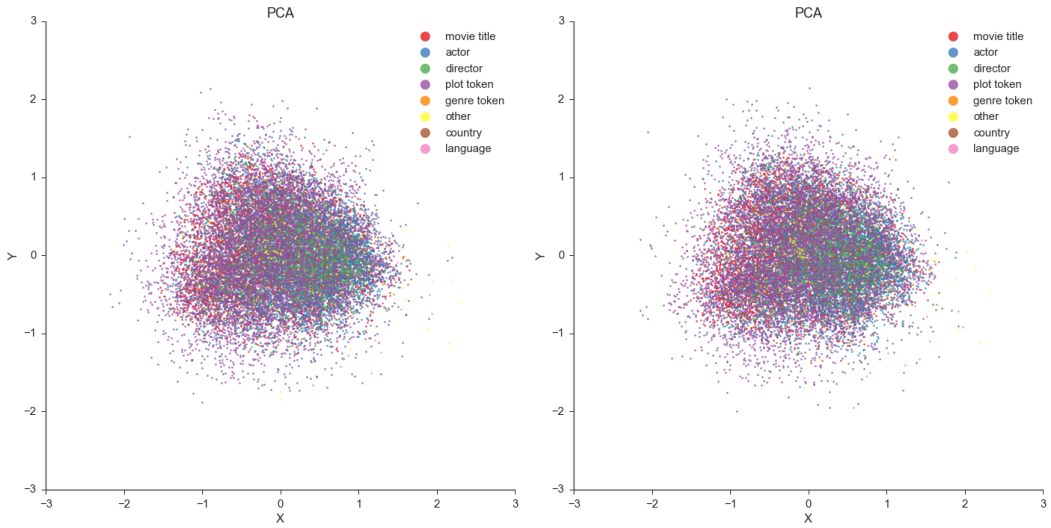


Figure 14: 100-Dimension vs 300-Dimension (skip-gram with context window = 30 and with hierarchical softmax)

Figure 15: 10 iterations vs 20 iterations (300-dimensional skip-gram with context window = 30 and hierarchical softmax)

Our first observation is that context size plays a vital role in the model's ability to capture vector semantics. A smaller context size yields more clustered column values,

whereas a larger context size generalizes over the dataset better. Secondly, negative sampling and hierarchical softmax are also non-trivial for the model to capture relationships well. Recall that negative sampling is a simpler form of Noise Contrastive Estimation that makes learning parameters more efficient. It draws negative samples in the prediction task and optimizes the model to predict correct and *noise* words for a given target word. With hierarchical softmax (or high negative sampling), the PCA plots show the vectors as a giant mess centered around the origin. It is not yet established why we see such patterns. As we shall demonstrate later, using hierarchical softmax or high negative sampling rates yielded better results for our intended tasks. Lower values of negative sampling resulted in absolutely garbage results.

Row2vec Visualizations

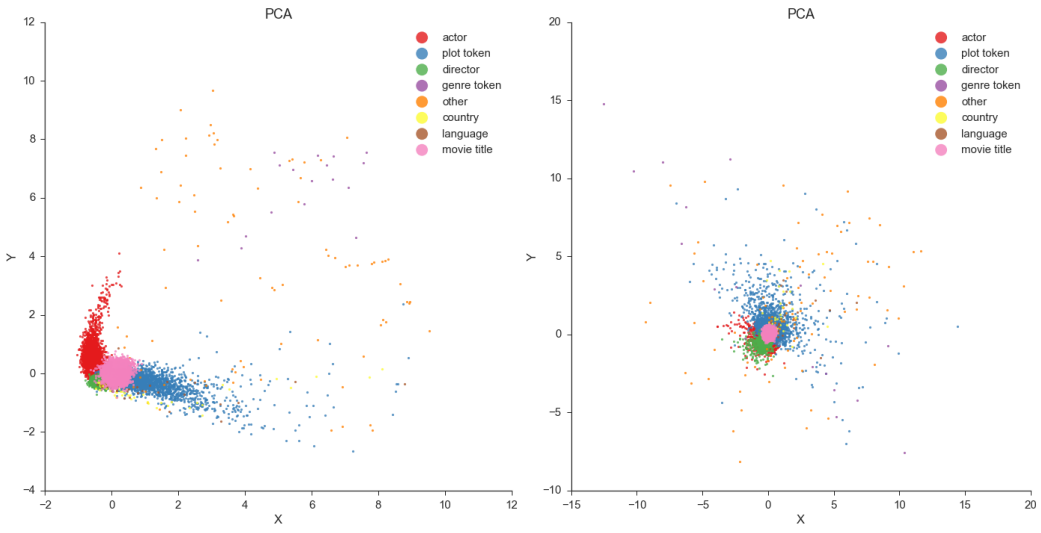


Figure 16: Context size = 1 vs Context size = 30 (300-dimension PV-DM with hierarchical softmax)

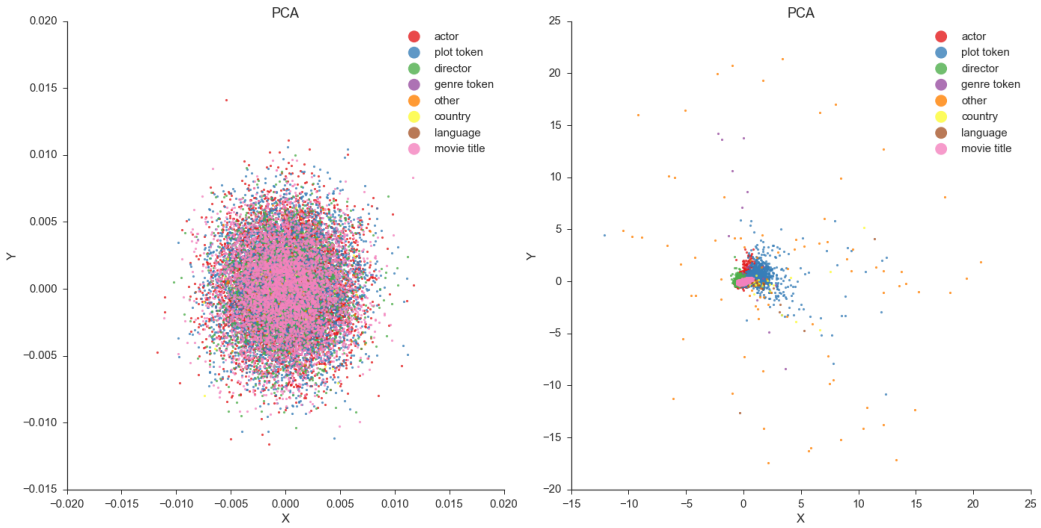


Figure 17: Negative sampling = 0 vs Negative sampling = 25 (100-dimension PV-DM with context size = 30)

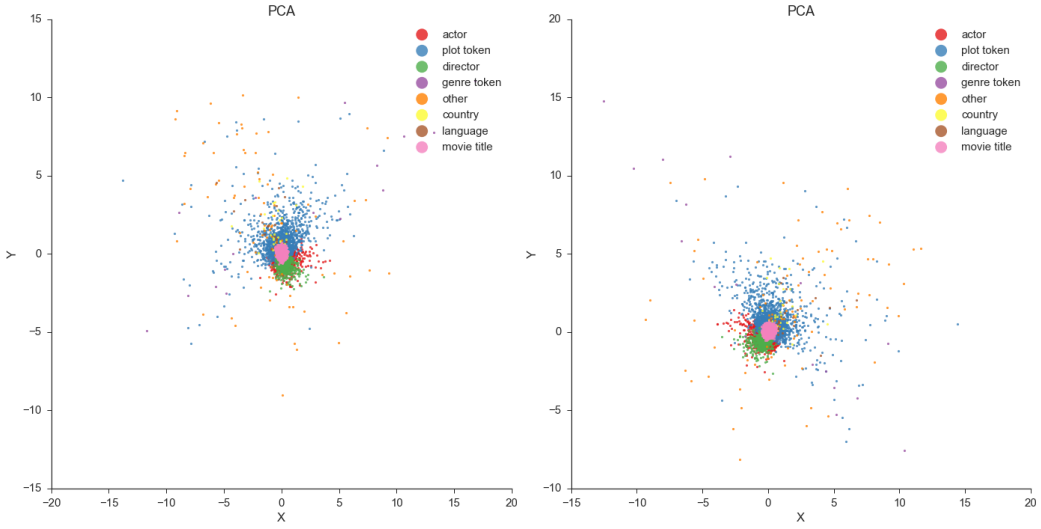


Figure 18: 100-Dimension vs 300-Dimension (PV-DM with context size = 30 and hierarchical softmax)

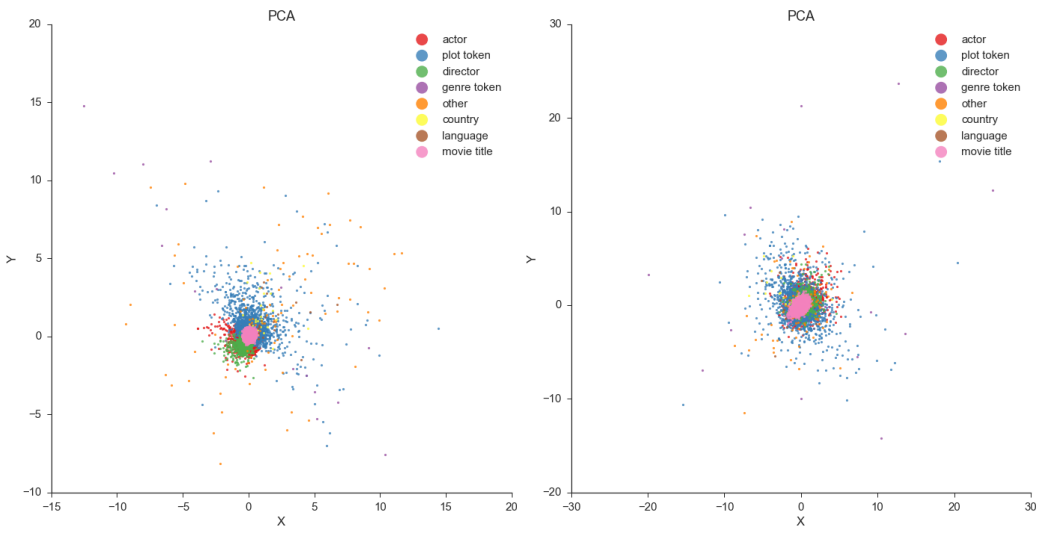


Figure 19: 10 iterations vs 20 iterations (100-dimensional PV-DM with context window = 30 and hierarchical softmax)

Here we see negative sampling and hierarchical softmax both make the model tend to cluster column values to some extent. The situation is exacerbated when we use a small context window as smaller context windows tend to yield more topical vectors (Figure 16). On the other hand, when we switch off both negative sampling and hierarchical softmax (Figure 17), we see no kind of obvious relationship semantics being captured by the model.

Sample Cognitive query results

Word2Vec

The sample query responses shown here are on a word2vec model run with hierarchical softmax. We have taken a context size of 30 and generated output vectors of size 300. We downsampled the popular words with a threshold of 0.001 and ran the model for 10 iterations. We observed similar results when instead of hierarchical sampling, we used a high value for negative sampling.

Similarity

Top 20 vectors closest to *'Leonardo_DiCaprio'*

```
('Leonardo_DiCaprio', 0.9999998807907104),
('The_Departed', 0.662142276763916),
('Titanic', 0.6590770483016968),
('Gangs_of_New_York', 0.6585075855255127),
('Body_of_Lies, 0.6564064621925354),
('Elizabeth_Debicki', 0.6560742855072021),
('Revolutionary_Road', 0.654662549495697),
('Django_Unchained', 0.6534621715545654),
('Joe_Komara', 0.6504959464073181),
('Gloria_Stuart', 0.6503431797027588),
('jordan', 0.6393324136734009),
('Catch_Me_If_You_Can', 0.6392801403999329),
('Peter_Youngblood_Hills', 0.6386548280715942),
('The_Beach', 0.633617639541626),
('The_Revenant', 0.6317786574363708),
('Blood_Diamond', 0.6263160705566406),
('The_Great_Gatsby', 0.6188250780105591),
('The_Wolf_of_Wall_Street', 0.6161619424819946),
('hungary', 0.6137104034423828),
('paradise', 0.6085795164108276)
```

As we can see here, the model did quite well in capturing words that are relevant to our target word *'Leonardo_DiCaprio'*. We can see some of his movies (The Departed, Titanic and Gangs of New York), co-stars (Elizabeth Debicki, Joe Komara) and some of the plot tokens from his movies (Jordan, paradise).

Analogy

Top 10 results for *'Inception'* + *'Tom_Cruise'* – *'Leonardo_DiCaprio'*

```
('War_of_the_Worlds', 0.6809234023094177),
('Mission:_Impossible_II', 0.680419385433197),
('Mission:_Impossible_-_Rogue_Nation', 0.6794121861457825),
('subconscious', 0.6704068779945374),
('Knight_and_Day', 0.6620358824729919),
('Mission:_Impossible', 0.6609631776809692),
```

```
('tripod', 0.6468757390975952),
('Minority_Report', 0.6445044279098511),
('Independence_Day', 0.6434791684150696),
('Jack_Reacher', 0.6304120421409607)
```

In this query, we want to find out analogies of the kind "If Inception is to Leonardo DiCaprio, then what is to Tom Cruise?". We see many of Tom Cruise's movies come up in the top 10 such vectors.

Additive Composability

Top 10 results for *'Leonardo_DiCaprio' + 'Kate_Winslet'*

```
('Titanic', 0.7365461587905884),
('Gloria_Stuart', 0.7282267808914185),
('Revolutionary_Road', 0.708398163318634),
('Joe_Komara', 0.6952486634254456),
('Romeo_+_Juliet', 0.6811110973358154),
('The_Beach', 0.671924352645874),
('Blood_Diamond', 0.6625310778617859),
('Peter_Youngblood_Hills', 0.6582512855529785),
('Tomas_Alfredson', 0.6535829901695251),
('hungary', 0.6524652242660522)
```

Here again we can see that our number 1 and 3 results were movies in which Leonardo DiCaprio and Kate Winslet acted together. Number 2 and 4 are co-stars from movies in which Leo and Kate appeared together.

Row2Vec

For our Row2vec experiments, we used the same hyperparameter settings as our word2vec model. Recall that row2vec learns an additional vector for every movie, and thus we have 2 sets of vectors; one for all the tokens in our table and one for all the movies.

Similarity

Top 10 vectors closest to *'Leonardo_DiCaprio'*

```
('Leonardo_DiCaprio', 0.9999999403953552),
('Brad_Pitt', 0.6535583734512329),
('Tom_Hanks', 0.6237167119979858),
('Benedict_Cumberbatch', 0.6078296899795532),
('Kevin_Spacey', 0.598504364490509),
('Christian_Bale', 0.5830089449882507),
('Christoph_Waltz', 0.5748230814933777),
('Hugh_Jackman', 0.5710854530334473),
('1970s', 0.5472841262817383),
('Denzel_Washington', 0.5426238775253296)
```

Here we can see that row2vec did a lot more clustering of the column values. We can see similar actors to *'Leonardo_DiCaprio'.* These actors don't have a lot of movies in common with Leo, but they are all mostly famous actors.

Top 10 movie vectors closest to *'Leonardo_DiCaprio'*

```
('The_Monuments_Men', 0.5280457735061646),
('Philadelphia', 0.47902488708496094),
('The_Wolf_of_Wall_Street', 0.4776039719581604),
('Catch_Me_If_You_Can', 0.4774860739707947),
('War_Horse', 0.4763687252998352),
('Mission:_Impossible_III', 0.4706520736217499),
('Flags_of_Our_Fathers', 0.46463078260421753),
('Unforgiven', 0.4625704288482666),
('Carnage', 0.4617079794406891),
('The_Book_of_Eli', 0.45993518829345703)
```

Here, we see how row2vec performed poorly in contrast to our word2vec model. We get some Leonardo DiCaprio movies (The Wolf of Wall Street, Catch Me If You Can), but the others are not. Also, the association is a lot weaker than what we saw in word2vec.

Analogy

Top 10 movie results for *'Inception' + 'Tom_Cruise' – 'Leonardo_DiCaprio'*

```
('The_Firm', 0.5307195782661438),
('MacGruber', 0.5178793668746948),
('Man_of_the_House', 0.5129494667053223),
('The_Tailor_of_Panama', 0.44215500354766846),
('Texas_Rangers', 0.4251130223274231),
('White_Chicks', 0.42136460542678833),
('Pale_Rider', 0.4106457531452179),
('Machete', 0.4106242060661316),
('The_Majestic', 0.4104437232017517),
('Runner_Runner', 0.3990623354911804)
```

Additive Composability

Top 10 results for *'Leonardo_DiCaprio' + 'Kate_Winslet'*

```
('Leonardo_DiCaprio', 0.8796911239624023),
('Kate_Winslet', 0.8526569604873657),
('Benedict_Cumberbatch', 0.7082951068878174),
('Meryl_Streep', 0.640968918800354),
('Christian_Bale', 0.6378368735313416),
('Kevin_Spacey', 0.6290881037712097),
('Tom_Hanks', 0.6269365549087524),
('Jennifer_Lawrence', 0.6222625970840454),
('Brad_Pitt', 0.6136409044265747),
('Scarlett_Johansson', 0.6120947599411011)
```

In both these query results we can see that row2vec did not perform at par with word2vec. In the case of analogy query, Tom Cruise starred in only one of the movies. With the additive composability query, we see a clustering effect around column values.

Chapter 5: Discussion, Conclusion and Future Work

In this thesis, we studied two distributed word embedding algorithms (word2vec and doc2vec), along with their bells and whistles and reasoned on their applicability to relational databases. Our primary goal was to introduce to the reader, an enhanced protocol for *cognitively* querying structured data. This pattern of querying is non-trivial to do with the SQL dialect alone. To achieve this, we had to better understand the effects of the hyperparameter choices and tokenization dynamics.

We began our discussion by outlining the models. We presented our hypothesis and proposed extensions to the models (that were originally designed to work on large textual corpora) and how they could be molded for our use case. We discussed tokenization techniques and hyperparameter considerations and presented their effects using visual plots. We also presented the results of sample queries executed on both types of models.

We designed row2vec as a special case of doc2vec that we believed would perform well on our tabular data. We had studied word2vec carefully and had tried to reduce some of its shortcomings that we believed would disallow it from generalizing well on SQL tables. Row2vec performed poorly across the board on our chosen tasks of similarity, analogy and additive composability. It might be possible to squeeze out better results on these same tasks by finding a more appropriate combination of hyperparameters. It is also possible that this same model might perform well on some other task that was concerned with queries involving higher orders of abstractions. More work needs to be done in better understanding the effects of certain hyperparameters on our tasks.

We were surprised to see that vanilla word2vec gave us the best results on our tasks. Negative sampling and hierarchical softmax, which are discussed in the

literature as merely optimization techniques, proved to play vital roles in the overall capturing of relationships between our token vectors. On the other hand, hyperparameters like vector size, iterations, and model type (skip-gram vs CBOW) played smaller roles.

Considering the results presented above, it is evident that we need more information about the effects of hyperparameters on the quality of model output. A better benchmark for assessing the performance of cognitive querying would go a long way in that directions. Such a benchmark will allow researchers to evaluate extensions to the model is a more streamlined fashion.

Bibliography

Bengio, Y. D. (2003). A Neural Probabilistic Language Model. *The Journal of Machine Learning Research, 3*, 1137–1155.

Catlett, J. (1991). *Megainduction: Machine Learning on Very Large Databases.* University of Sydney.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In *Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)* (Vol. 482).

Christophe Van Gysel, M. d. (2016). Learning Latent Vector Spaces for Product Search. *CIKM'16.* Indianapolis, IN, USA.

Codd, E. F. (1970, June). A Relational Model of Data for Large Shared Data Banks. *Commun. ACM, 13*(6).

D. E. Rumelhart, G. E. (1986). Learning internal representations by backpropagating errors. *Nature, 323:533.536*.

Dougherty, J. R. (1995). Supervised and unsupervised discretization of continuous features. *Machine learning: proceedings of the twelfth international conference*, *12*, pp. 194-202.

Frederic Morin, Y. B. (2005). Hierarchical probabilistic neural network language model. *Proceedings of the international workshop on artificial intelligence and statistics*, (pp. 246–252).

GloVe: Global Vectors for Word Representation. (n.d.). Retrieved from The Stanford Natural Language Processing Group: https://nlp.stanford.edu/projects/glove/

Holte, R. C. (1993). Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. In *Machine Learning 11* (pp. 63-91).

IMDB 5000 Movie Dataset. (n.d.). Retrieved from https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset

Jansen, S. (2017). Word and Phrase Translation with word2vec.

Jeff Mitchell, M. L. (2010). Composition in Distributional Models of Semantics. *Cognitive Science, 34*, 1388–1429.

Jey Han Lau1, T. B. (2016). An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. *Proceedings of the 1st Workshop on Representation Learning for NLP* (pp. 78–86). Berlin: Association for Computational Linguistics.

Laurens van der Maaten, G. H. (n.d.). Visualizing Data using t-SNE. *Journal of Machine Learning Research 9 (2008) 2579-2605*.

Liu, H. (2017). Sentiment Analysis of Citations Using Word2vec.

Manning, J. P. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (pp. 1532–1543).

Marco Baroni, G. D. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, (pp. 238–247). Baltimore, Maryland, USA.

Ng, P. (2017). dna2vec: Consistent vector representations of variable-length k-mers.

Omer Levy, Y. G. (2014). Neural word embedding as implicit matrix factorization. *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems*, (pp. 2177-2185). Montreal, Canada.

Omer Levy, Y. G. (2015, May). Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, 211–225.

Oren Barkan, N. K. (2016). ITEM2VEC: NEURAL ITEM EMBEDDING FOR COLLABORATIVE FILTERING.

Ozsoy, M. G. (2016). From Word Embeddings to Item Recommendation.

Quoc V. Le, T. M. (2014). Distributed Representations of Sentences and Documents. *International Conference on Machine Learning*.

Rajesh Bordawekar, O. S. (2016). Enabling Cognitive Intelligence Queries in Relational Databases using Low-dimensional Word Embeddings. *arXiv:1603.07185*.

Rong, X. (2016). word2vec Parameter Learning Explained. *arXiv:1411.2738v4*.

Stanisław Jastrzebski, D. L. (2017). How to evaluate word embeddings? On importance of data efficiency and simple supervised tasks. *arXiv:1702.02170*.

Tomas Mikolov, I. S. (2013c). Distributed Representations of Words and Phrases and their Compositionality. *Advances on Neural Information Processing Systems*.

Tomas Mikolov, K. C. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781v3.

U M Fayyad, K. B. (1993). Multi-Interval discretization of continuous valued attributes for classification learning. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, (pp. 1022-1027).

Y. Bengio, R. D. (2003). A neural probabilistic language model. *Journal of Ma- chine Learning Research*, 1137-1155.

Yoav Goldberg, O. L. (2014). word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. *arXiv:1402.3722*.

Yushi Yao, G. L. (2017). Context-aware Sentiment Word Identification: sentiword2vec.