Neural Embeddings for Mining Online Reviews

Cagatay Ozese

A thesis submitted in part fulfillment of the
degree of MSc. Computer Science by Negotiated Learning
with the supervision of Aonghus Lawlor

School of Computer Science
University College Dublin

August 2017

ACKNOWLEDGEMENTS

I would like to thank Dr. Aonghus Lawlor greatly for his support and guidance throughout several months in this research project. He has shared his great domain knowledge, support, and guidance whose importance can't be stressed enough for conducting this research.

I would like to thank my partner Anna Knap and my family for their great support and understanding during my attendance to this MSc program, it takes a lot of extra effort to study part time while working full time and it comes with a lot of sacrifice from the personal life and therefore their support and motivation are highly appreciated.

Lastly, I would like to thank my employer Google, for partially funding my taught MSc Computer Science program at UCD; and my manager Ed who has provided flexibility in working hours during the course of the program.

DECLARATION

“I hereby certify that this dissertation is entirely my own work. Neither the work nor parts thereof have been published elsewhere in either paper or electronic form unless indicated otherwise through referencing”.

Signed: ________________________________

Date: _________14/08/2017_________

Cagatay Ozese

ABSTRACT

In this thesis, I investigate unsupervised word embedding techniques which generate word vector representations for domain specific review mining purposes. I apply these techniques to a beer review database of 1.5M reviews from BeerAdvocate.com. The main question addressed in this work is how to build profiles of users and items from the reviews by different word embedding techniques. These enhanced user and item profiles are essential for the task of building personalised recommender systems.

We have extracted, preprocessed (stop word removal and stemming) and trained various neural word embedding methodologies (e.g. Word2Vec and Doc2Vec) and other simpler word embedding techniques (e.g. TF-IDF and LSI) to compare their performance based on their evaluation of user-user and item-item similarities.

We also evaluated Word2Vec and Doc2Vec model accuracies using Word2Vec general domain analogies.

CONTENTS

1 Introduction

The field of word embeddings is an important one in natural language processing, and has found many interesting applications, including building rich user and item profiles which can be used for personalised recommendations. There are many flavours and developments of word vectors which mostly stem from work on the Word2Vec technique, which originated at Google Research.

In Word2Vec, a distributed representation of a word is learned by a neural network. Each word is represented by a distribution of weights across the elements of a high dimensional vector. Rather than a one-to-one mapping between an element in the vector and the word, the word representation is spread across all elements of the vector. It is found that word vectors can represent meaningful syntactic and semantic regularities in a simple and relatively straightforward way. The regularities are manifested in vector offsets between pairs of words which share a particular relationship. This property makes the vectors good at answering analogy type questions, and performance on analogy tests is often used as a means of evaluating the quality of the learning vectors.

Many recommender systems rely on reviews and opinion mining to build personalised recommendations (eg. Amazon, TripAdvisor, Yelp, etc). Collaborative filtering techniques, and nearest neighbour methods rely on a means of identifying the most similar user or item in order to make recommendations. Where user-generated reviews are involved, this similarity measure usually involves some natural language processing methods (eg. tf-idf, or topic modeling). In this work we aim to evaluate the quality of user and item profiles constructed by combining word vectors trained in different ways. The reviews come from the BeerAdvocate.com domain.

The expectation is that user and item profiles constructed from word vectors to capture the semantic content the reviews in a more significant way than traditional TF-IDF techniques, and this should ultimately allow a better strategy for building more accurate personalised recommender systems.

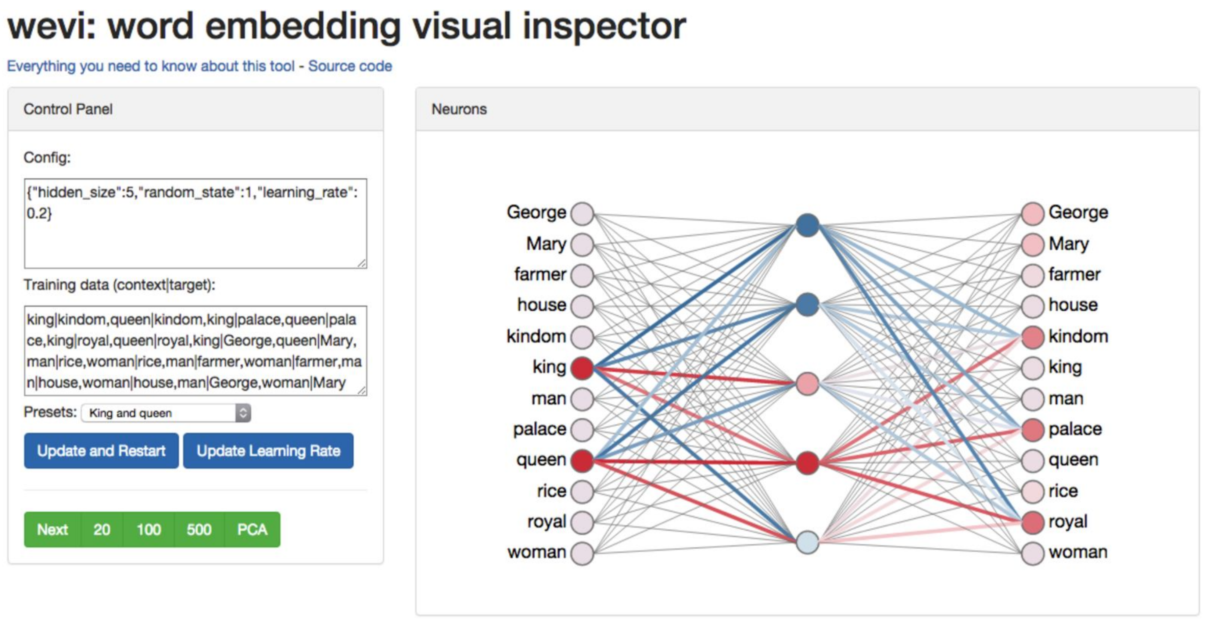
We have extracted, preprocessed (stop word removal and stemming) and trained various neural word embedding methodologies (e.g. Word2Vec and Doc2Vec) and other simpler word embedding techniques (e.g. TF-IDF and LSI) to compare their performance based on their evaluation of user-user and item-item similarities. The Word2Vec and Doc2Vec model accuracies were also evaluated using Word2Vec general domain analogies as well.

2 Literature Review

At the beginning of the work, I had various things in mind to explore such as topic/item modelling, sentiment analysis, Word2Vec and Doc2Vec models, evaluation etc. with various different review data sets such as BeerAdvocate, Amazon, TripAdvisor and Yelp data. However, I pivoted largely to word and item vectors, TF-IDF as a benchmark, evaluation and method comparisons after the first few weeks of exploration and only to the BeerAdvocate data set for the rest of the research. Therefore, a large part of my work involves evaluation and comparison of different methodologies with each other including word and item vectors. Below are some of the work that were done in these areas which I've looked at during my research:

Word vectors

[Rong, 2016] explains Word2Vec and the algorithms that underlies the model which has been optimized computationally and discusses new algorithms that can optimize the computations in a different way. The author also created a visualisation software which shows the layers in the model. The software is available at http://bit.ly/wevi-online. Please see fig. 1 and 2 below with a set of examples:


wevi: word embedding visual inspector

Everything you need to know about this tool - Source code

Control Panel

Config:

```
{"hidden_size":5,"random_state":1,"learning_rate":
0.2}
```

Training data (context|target):

```
king|kindom,queen|kindom,king|palace,queen|pala
ce,king|royal,queen|royal,king|George,queen|Mary,
man|rice,woman|rice,man|farmer,woman|farmer,ma
n|house,woman|house,man|George,woman|Mary
```

Presets: King and queen

Figure 1. WEVI - Word Embedding Visual Inspector with a few chosen words and shows how the layers are embedded in the model

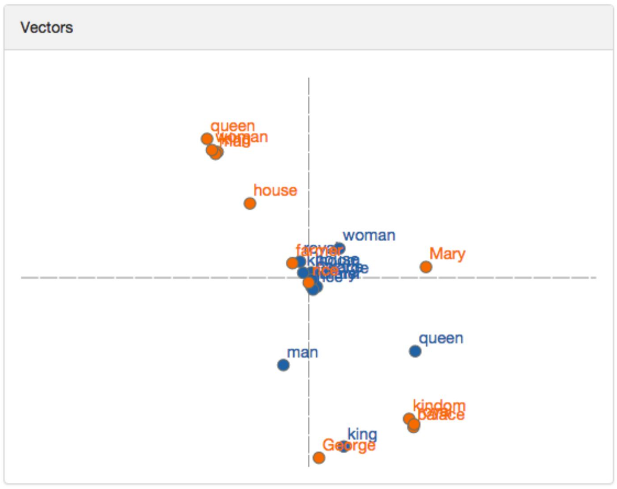
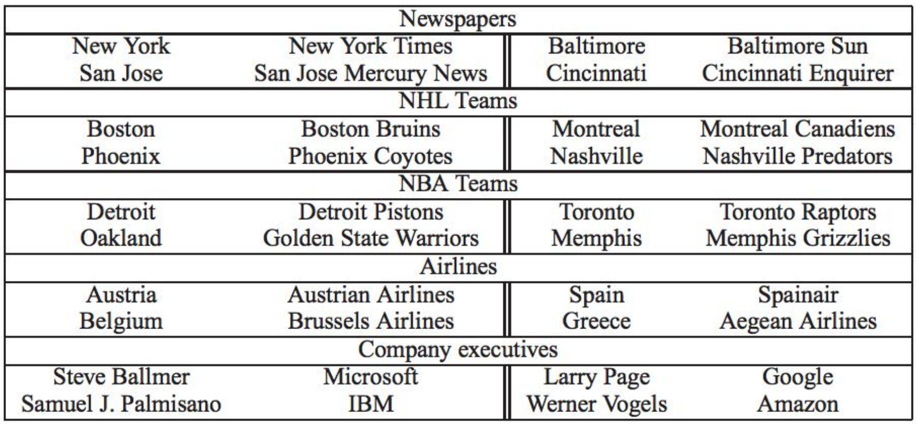


Figure 2. WEVI Vectors with the same set of words as in Figure 1, on 2D visualisation

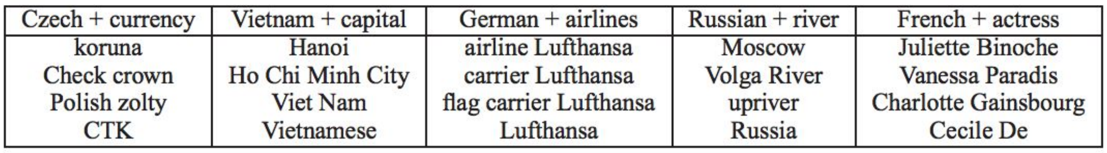
[Tsvetkov et al., 2015] make the following statement about word vectors in their article: “Unsupervised learned word vectors have proven to provide exceptionally effective features in many NLP tasks. Most common intrinsic evaluations of vector quality measure correlation with similarity judgments. However, these often correlate poorly with how well the learned representations perform as features in downstream evaluation tasks.” and they present QVEC, “a computationally inexpensive intrinsic evaluation measure of the quality of word embeddings based on alignment to a matrix of features extracted from manually crafted lexical resources—that obtains a strong correlation with the performance of the vectors in a battery of downstream semantic evaluation tasks.” [Diaz, Mitra, & Craswell, 2016] demonstrate that “word embeddings such as Word2Vec and GloVe, when trained globally, underperform corpus and query specific embeddings for retrieval tasks.”

Word vector applications

[Sienčnik, 2015] adapts Word2Vec to *Named Entity Recognition* and [Lample et al., 2016] work on various neural architectures (including Word2Vec) for a similar purpose - *Named Entity Recognition*. [Kim, 2014] explores the use of convolutional neural networks for *sentence classification*. He was expecting performance gains through the use of pre-trained vectors, however was still surprised with the magnitude of such gains, giving competitive results against the more sophisticated deep learning models. [Mikolov et al., 2013] introduce a simple method to *find phrases in text* by improving their skip-gram methodology which they published a while back and show that “learning good vector representations for millions of phrases is possible”.



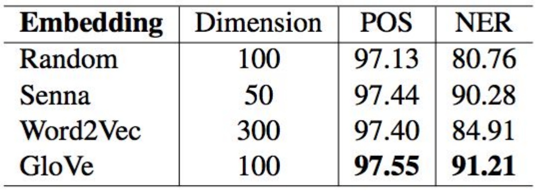
Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Canadiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon



Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Figure 3. (a) Examples of the analogical reasoning task for phrases, their best model accuracy achieved 72% on this data set. (b) Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

[Ma & Hovy, 2016] explore performance of Word2Vec, GloVe, Senna and Random embeddings and get the results in the below fig. 4. We see that Word2Vec produces on par results with other methods on POS but quite poorly on NER tasks.



Embedding	Dimension	POS	NER
Random	100	97.13	80.76
Senna	50	97.44	90.28
Word2Vec	300	97.40	84.91
GloVe	100	**97.55**	**91.21**

Figure 4. Results table comparing different embeddings on POS and NER tasks from [Ma & Hovy, 2016]

[Mikolov, Le, Sutskever, 2013] discuss machine translations using word embeddings across languages achieving a precision of almost 90%. [Levy, Goldberg & Ramat-Gan, 2014] discuss that vector multiplication and division, as opposed to addition and subtraction, also give meaningful analogy results. Some examples from the paper:

- Germany Australia: emigrates, 1943-45
- Yen ruble: devalue, banknote

Word vectors in review context

[Pouransari & Ghili, 2014]; [Shirani-Mehr, 2014] study sentiment analysis on movie reviews using Word2Vec and some ML methods such as Random Forest Classifiers. They've also worked on parameter tuning for Word2Vec (such as impact of Word2Vec dimension and context size on accuracy) and they've shared the plot as in fig. 5. They've chosen 100 as satisfactory dimension as there's quite a steep learning curve for the first 100 and then there's diminishing returns with increasing dimension - as higher values take longer to train.

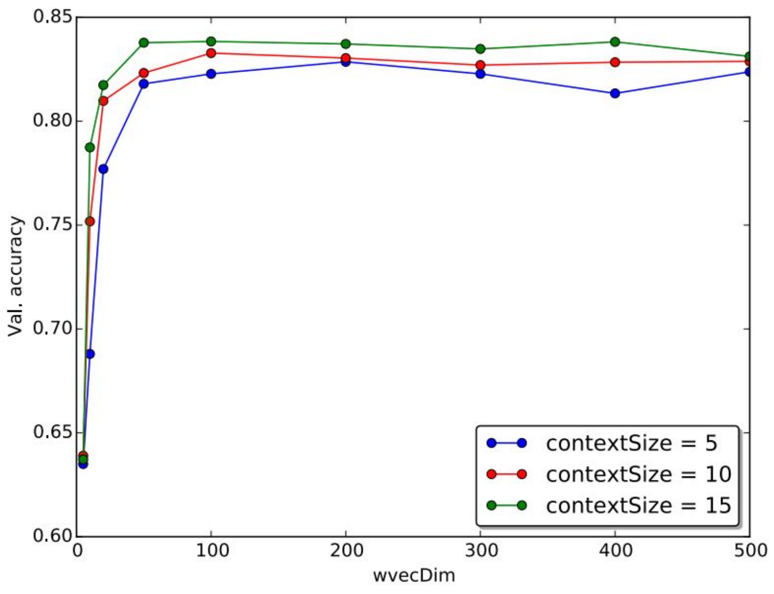


Figure 5. The graph between accuracy vs Word2Vec dimension and context size.

[Xing Margaret & Xiaocheng, 2015] study transfer learning and look into learning word vector representations and sentiment analysis from movie reviews and they transfer the learnings in this domain to the restaurant reviews domain. They claim that their results outperformed the state-of-art model [McAuley and Leskovec, 2013] in mean squared error. [Jiang, Liu & Xu, 2015] focus on similarities between different reviews and different businesses and they achieve successful results calculating these using Word2Vec embeddings based on Yelp user reviews. They generated clustering of these businesses based on word (or document) embeddings and plotted the below graph showing that the model can nicely separate a lot of categories from each other.

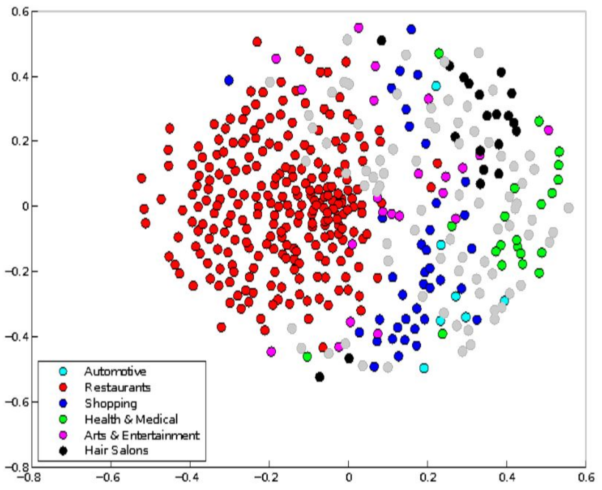


Figure 6. Multidimensional scaling (MDS) plot of the model

Document embeddings and comparisons

[Kusner et al., 2015] measure document distances with Word Mover's Distance (WMD) a novel method that encrypts the distance as a "minimum amount of distance that the embedded words of one document need to 'travel' to reach the embedded words of another document." and they cast this problem similar to the Earth Mover's Distance problem. [Heidarian & Dinneen, 2016] introduce a new document similarity measure - Triangle Similarity-Section Similarity (TS-SS) - improving on the existing similarity measures such as Euclidian distance and cosine similarity. See fig. 7 below for the conceptual comparison of this measure to the others. Their evaluation results show that their new model outperforms the other measures and they claim that "TS-SS clusters documents with better purity and is more reliable for measuring the similarity level."

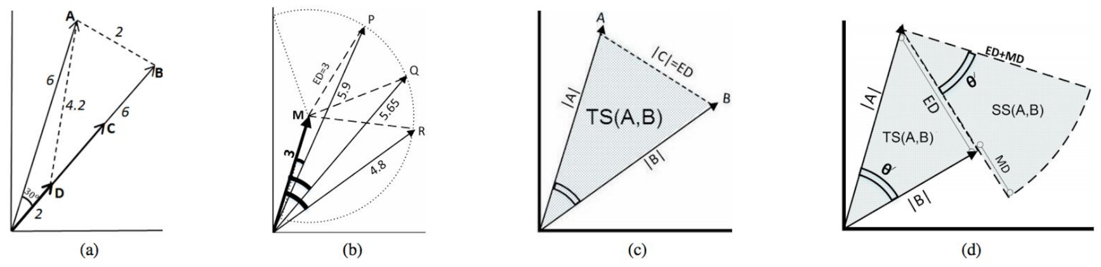


Figure 7. (a) Cosine drawback. (b) Euclidian Distance drawback. (c) Triangle Similarity (TS). (d) Triangle Similarity-Section Similarity (TS-SS)

Sentiment analysis

[Tang et al., 2014] work on Twitter sentiment analysis using word embeddings. They argue that most of the existing algorithms only model syntactic relations and some meanings of words but not really sentiments, e.g. *good* and *bad* are similarly represented however often do not give clue on sentiment polarity, which is the opposite (good - positive vs bad - negative) in this case. They try to tackle this issue with learning sentiment specific word embedding (SSWE) and they achieve positive results. [Dickinson & Hu, 2015] explore the relationship between stock prices compared to the sentiments on Twitter about companies. They use n-gram and Word2Vec representations as well as Random Forest Classification to predict sentiment of tweets. They find strong correlations between the stock prices and Twitter sentiments of users for certain companies and some of these correlations are positive e.g. for Microsoft and Walmart (largely consumer facing companies) and some are negative e.g. for Cisco and Goldman Sachs. [Bhingardive et al., 2015] propose "an unsupervised method for MFS (Most Frequent Sense) detection from the untagged corpora, which exploits word embeddings. We compare the word embedding of a word with all its sense embeddings and obtain the predominant sense with the highest similarity. " [Trask, Michalak & Liu, 2015] present "a novel approach modeling multiple embeddings for each word based on supervised disambiguation, which provides a fast and accurate way for consuming an NLP model to select a sense-disambiguated embedding."

Recommender systems

[Barkan, & Koenigstein, 2016] explore Word2Vec performance for Collaborative Filtering (CF) and conclude by stating that they observed that "Item2Vec produces a better representation for items than the one obtained by the baseline SVD (singular value decomposition) model, where the gap between the two becomes more significant for unpopular items. We explain this by the fact that Item2Vec employs negative sampling together with subsampling of popular items". They have plotted the 2-dimension t-SNE plot which can be found in fig. 8:

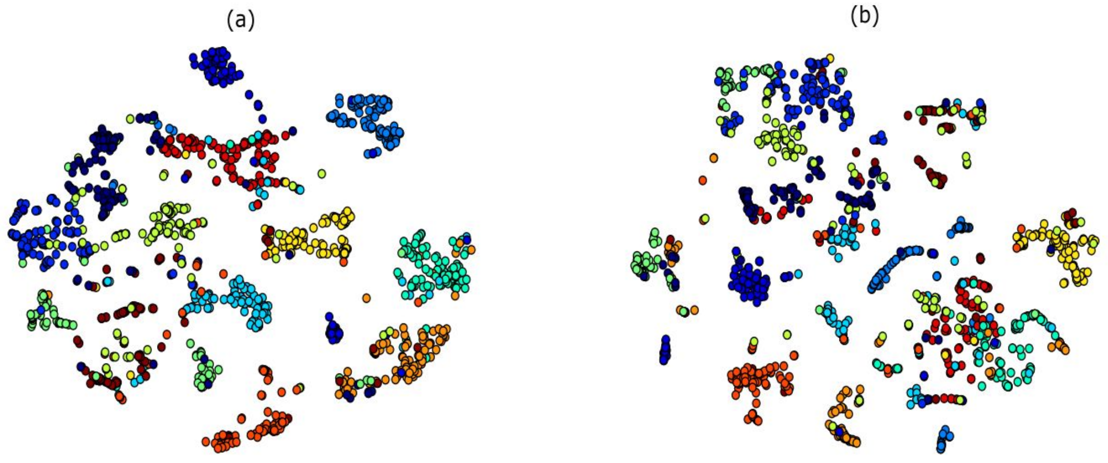


Figure 8. t-SNE embedding for the item vectors produced by (a) Item2Vec and (b) SVD. The items are colored according to a web retrieved genre metadata.

[Phi, Chen & Hirate, 2016] work on recommendation systems based on Word2Vec and Doc2Vec and compare the hit rate of these systems to Matrix Factorization and item similarity. They find that item-vector based Doc2Vec outperforms the others. See fig. 9 below for the hit-rates of different systems.

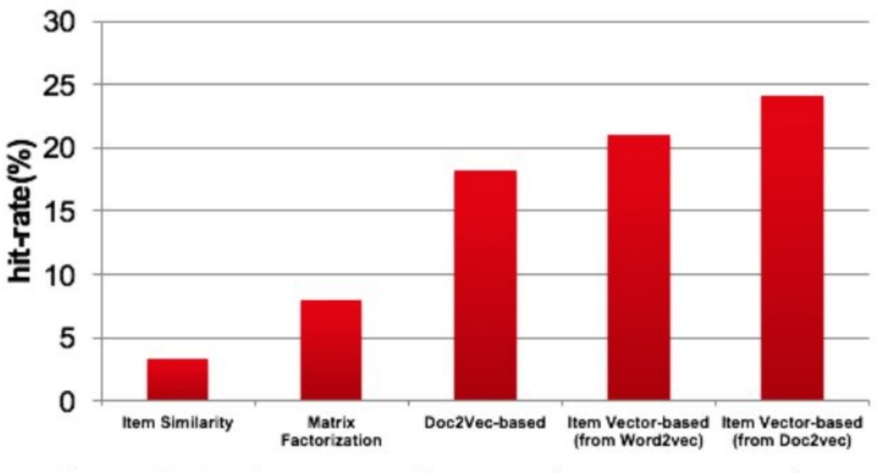


Figure 9. Performance of different user to item recommendation systems

LSA

[Wiemer-Hastings, Wiemer-Hastings, & Graesser, 2004] discuss on LSA concept, how it works, and its pros and cons in depth. They define and describe LSA as "LSA (originally known as Latent Semantic Indexing) was developed for the task of Information Retrieval, that is, selecting from a large database of documents a few relevant documents which match a given query". One of the remarks they make is that "there is little hard evidence on what the "ideal" size of an LSA corpus might be" which was a challenge to decide in our work as well.

Evaluation

[Lau & Baldwin, 2016] work on Doc2Vec released by [Le & Mikolov, 2014] and Word2Vec averaging performances, remarking that "For Word2Vec, the document embedding is a centroid of the word embeddings, given the simple word averaging method. With Doc2Vec, on the other hand, the document embedding is clearly biased towards the content words, and away from the function words." They also explore the use of Doc2Vec for forum question duplication evaluation by pairing documents, calculating their similarities and creating an ROC curve. [Minarro-Giménez et al., 2015] create a system to compare it to a matching system with a widely-accepted medical source by querying Word2Vec analogy and distance tools. They calculate accuracy scores based on analogies and they've given the plot in fig. 10 that analyzes the Skip-Gram vs Continuous Bag of Words (CBOW) methods with changing window size. Window size around 10 seems optimal with SG outperforming CBOW.

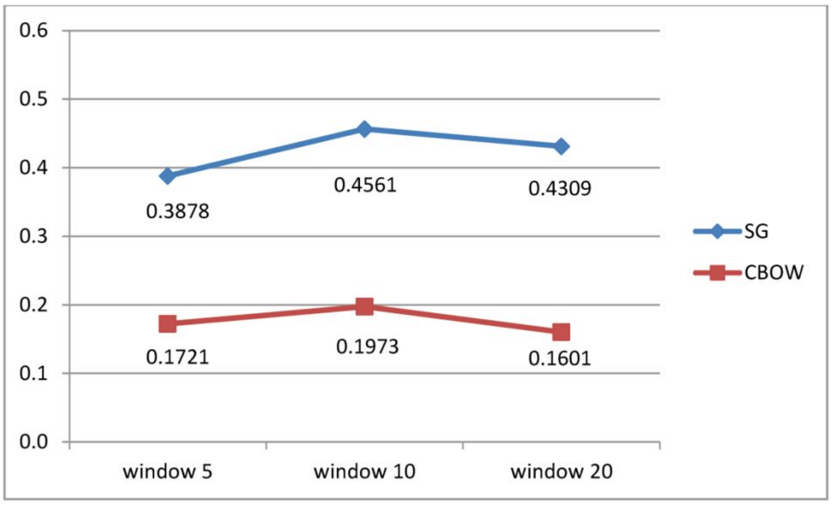


Figure 10. Skip-Gram (SG) vs CBOW accuracies with varying window size

[Wu et al., 2015] use neural word embeddings to disambiguate clinical abbreviations. [Schnabel et al., 2015] evaluate word frequencies and model accuracy through different methods such as CBOW and GloVe etc.

3 Methodology

I have extensively used various NLP (Natural Language Processing) methods in this work including simpler methods such as TF-IDF (Term Frequency - Inverse Document Frequency), LSI (Latent Semantic Indexing) and more complex and more recent techniques that use Machine Learning/Deep Learning algorithms such as Word2Vec (Word vectors) and Doc2Vec (Document Vectors) that use document vectors such as user vectors or items vectors which can be called 'Item2Vec' and 'User2Vec'. This section contains a brief explanation of those methods to give more context and some background for in each of them.

A. Text preprocessing

Text preprocessing of the corpus (corpus is the complete text data in a data set) is a critical task in any NLP text processing methodology. This process helps normalize the words in a certain way to eliminate anomalies, it can also be used to add more information to the sentence structure (e.g. POS, in other words Part-Of-Speech tags which tag words by tags such as adjective, noun, verb etc.), extract entities, stemming, fix misspellings etc.

It's important to know the context and structure of the corpus to decide which of the above and beyond approaches to use for a particular corpus. For example, a tweet database preprocessing and book text preprocessing certainly differ from each other. Database used for this thesis is based on beer reviews and resembles natural text without any certain notation such as Twitter's hashtags and mentions, instead, it is just a bulk of text containing users' reviews of beers containing their sentiments, likes and dislikes of the subject beer.

I have selected to do very basic text preprocessing for this data set and only removed stop words (words that don't add much information to the text such as 'I', 'am', 'are', 'the' etc.) and punctuation marks. I have then stemmed them using PorterStemmer. Stemming is a preprocessing technique to get the words' stems and remove suffixes in words, such as 'apples' to 'apple or 'rye' to 'ri' or 'effective' to 'effect' and 'beer' to 'beer' (the last one has no change as it's the root itself). 'Rye' to 'ri' might appear as a somewhat awkward stemming but as long as we treat all the words the same way, this type of anomalies are okay in the processed text. This helps us see through similar words and use 'apples' and 'apple' as the same word which in fact have the same meaning, one is just the plural of the other.

Once we do these processes on the corpus, we pass this data onto the following techniques as explained below, such as TF-IDF, LSI, Word2Vec, and Doc2Vec.

B. TF-IDF (Term Frequency - Inverse Document Frequency)

TF-IDF is a simple method which uses word counts in a bag of words approach. Bag of words approach means that the word order doesn't matter in this representation as this information is discarded in TF-IDF, it only stores the word counts in each document, such as sentences, paragraphs or reviews. Therefore, the words are thought to be in a bag for each document, without any particular order.

[Ramos, 2003] explains that TF-IDF works by determining the relative frequency of words in a specific document compared to the inverse proportion of that word over the entire document corpus. Given a document collection D, a word w, and an individual document d ϵ D, TF-IDF calculates

$$w_d = f_{w,d} \log(\frac{|D|}{f_{w,d}})$$

where $f_{w,d}$ equals the number of times w appears in d, |D| is the size of the corpus, and $f_{w,D}$ equals the number of documents in which w appears in D [Ramos, 2003].

Therefore the words which are frequent in a document but rare in the overall corpus have a high TF-IDF score while common words across the corpus (such as 'I', 'am', 'are', 'the' etc.) have a lower TF-IDF scores as these would be observed across documents and would be present in virtually all of the documents. However, in our preprocessed data, there won't be any 'I', 'am', 'are', 'the' etc. as these are stop words which we removed before feeding the corpus into the system since they don't provide much information about the opinions or sentiment in the review text. Having said that, there will surely be other such words that have a low TF-IDF score similar to the potential TF-IDF scores of the above mentioned stop words which might be adding meaning to the text, thus not a stop word, or perhaps some words that don't really add much meaning but are not in the stop word list (in nltk library) as there might be stop words specific to this text/domain that are not in the general domain stop word list.

C. LSI (Latent Semantic Indexing)

[Deerwester et al., 1990] describe LSI as a method for indexing and retrieval of documents. It categorizes the documents into a number of categories that describe its context and it is designed to match queries to documents based on their context, not the actual words used in the documents. This is because a different set of words can describe the same thing and thus context matching might perform better than TF-IDF which only measures whether one document matches the other based on the presence and count of words and is unaware of the context, synonyms etc.

[Bengio et al., 2003] describe how LSI works - the model learns simultaneously a distributed representation for each word along with the probability function for word sequences, expressed in terms of these representations. Generalization is obtained thanks to a sequence of words that have never been seen before gets high probability if it is made of words that are similar (in the sense of having a nearby representation) to words forming an already seen sentence.

D. Word2Vec

Vector representations of words have been around for a number of years. One of the first pieces of work considered in this area was published by [Harris, 1954] and the same topic was later discussed by [Hinton, 1986] and some others. The previous techniques (TF-IDF, LSI etc.) also create vector representations from words and documents in a way, however, these type of novel word embedding techniques has gained more traction recently with the papers published by [Mikolov et al., 2013], in which a new architecture (in fig. 11) was proposed and implemented to compute the vector representations in a short period of time from a very large corpus which would have taken a long time with the previous algorithms. This is a fast and efficient algorithm that can deal with large corpus fast and creates sensible results that somehow learn word relationships and meaning based on the context.

Word2Vec creates word vector embeddings (a vector containing floating point numbers and has a certain dimension which is set with a parameter when calling the Word2Vec method) and representations using a neural network with a hidden layer between the input and output letters. There are two main algorithm structures mentioned in the paper by [Mikolov et al., 2013]:

- Continuous Bag of Words (CBOW) in which a word is predicted based on the surrounding words whose order doesn’t have any influence on the output, therefore, it’s a bag of words approach.

- Skip-gram in which the surrounding window of the words are predicted based on the word in the middle.

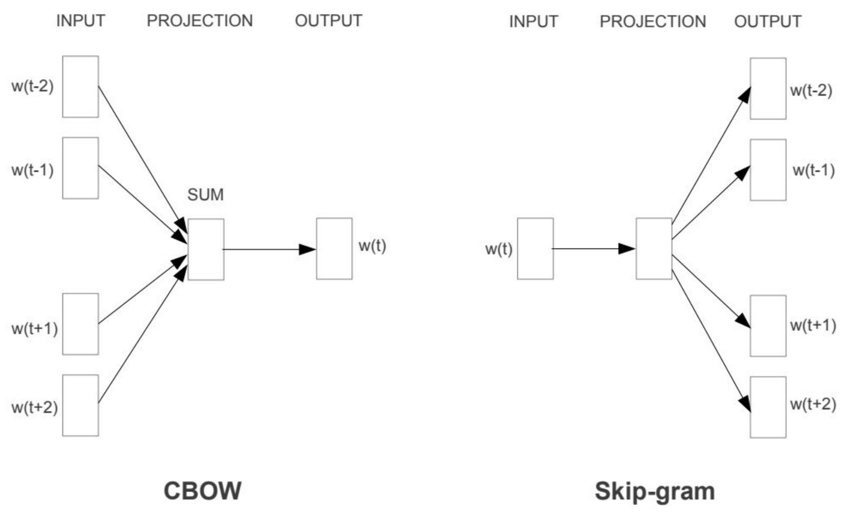


Figure 11. Graphical representation of CBOW and Skip-gram architectures

These concepts are also explained in Tensorflow Word2Vec page. Please see fig. 12 below from the same page to have a better understanding of how these vectors look like in 3-dimensional space. [“Vector Representations of Words”, 2017]

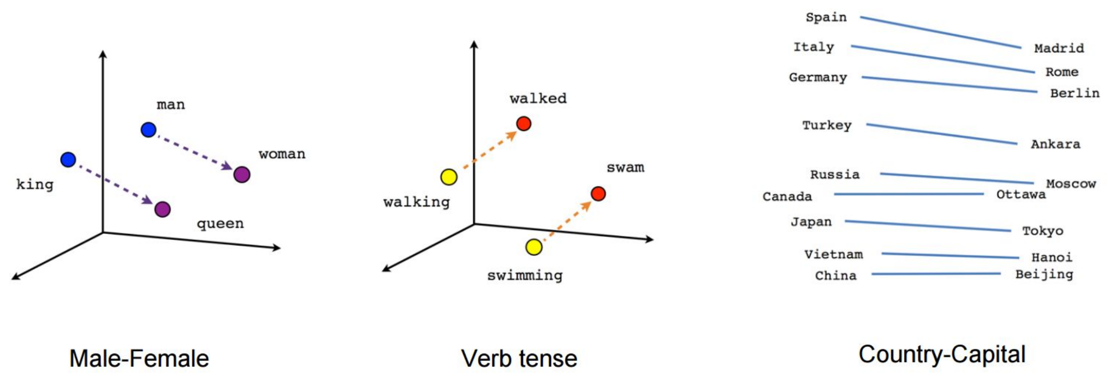


Figure 12. Embedded words in 2 or 3-dimensional space

As can be seen in fig. 12, the vector embeddings trained on a large enough corpus learn certain syntactic, meaningful and analogous relationships between words, in such a way that the words of similar syntax or meanings are represented close to each other and the relationships between words are learned and embedded into the word vectors.

Because vectors are mathematical representations of words, it's possible to do mathematical operations (addition, subtraction, multiplication etc.) with those embeddings. Using them, we can test the relationships between words as above, or create document vectors, such as creating sentence vector from word vectors by averaging the word vectors in a sentence.

The length of the word vectors depend on the dimension on which we train the Word2Vec algorithm, so it's a 1-D array of length 300 if the dimension is 300. The word vectors are initiated with some random values and then trained on the corpus until they learn and converge until a certain point and then the algorithm stops and gives the final word vectors.

There are other parameters passed to Word2Vec algorithm other than the dimension:

- *Minimum word count:* To discard words that appear in the corpus lower than a threshold. This can help get rid of uncommon misspellings and irrelevant words to the corpus.
- *Context window size:* To determine the context window size to look for each word, looking at an X number of words before and after the word.
- *Downsampling:* To downsample very frequent words to avoid over-training the model for them
- *Number of workers:* To make training faster by using more resources

Some examples for operations of word vectors are below:

Meaning relationships:

$v_{king} - v_{man} + v_{woman} = v_{queen}$ or $v_{man} - v_{woman} = v_{king} - v_{queen}$
'king is to man similar to queen is to woman'

$v_{Spain} - v_{Madrid} + v_{Italy} = v_{Rome}$ or $v_{Spain} - v_{Madrid} = v_{Italy} - v_{Rome}$
'Madrid is to Spain similar to Rome is to Italy'

Syntactic relationships:

$v_{walking} - v_{walked} = v_{swimming} - v_{swimmed}$
'walked is to walking similar to swimmed is to swimming'

Word2Vec library has certain very useful methods to automatically calculate the above, given 3 words to calculate the 4th word (most_similar() function) or give out the one word that's the most different than the others (doesnt_match() function) in a list of words, an example from our beer data set is:

```
[In] -> most_similar(positive=[u'heineken'])
[Out] -> (u'budweis', 0.7306), (u'amstel', 0.7214), (u'beck', 0.7112)

[In] -> doesnt_match(['lager', 'vodka', 'ale', 'beer', 'wall'])
[Out] -> 'wall'

# or to make this harder:
[In] -> doesnt_match(['lager', 'vodka', 'ale', 'beer'])
[Out] -> 'vodka'

# and even harder:
[In] -> doesnt_match(['lager', 'ale', 'beer', 'cider'])
[Out] -> 'cider'
```

In the next section, we will discuss Doc2Vec which is an additional algorithm that was also developed by [Le & Mikolov, 2014] to represent documents as vectors.

E. Doc2Vec

Doc2Vec is an addition on top of the Word2Vec algorithm. It was formalized and published by [Le & Mikolov, 2014] in which they proposed a *Paragraph Vector*, an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. Documents are represented by a dense vector which is trained to predict words in the document.

Paragraphs are just units consisting of a number of words that can vary in their length, therefore the same method can apply to sentences, articles, chapters, books and even other language and entity (user, item, product etc.) units. Therefore these methods are generally referred to as Doc2Vec representing document vectors whatever the documents might be. Some useful documents for this work are review documents, user documents, and item documents which represent reviews, users, and items (such as beers).

There are two potential approaches to getting document vectors:

- Averaging word vectors to get document vectors or
- Concatenating and then calculating word vectors.

In their paper [Le & Mikolov, 2014] chose to concatenate which is a different algorithm and gives a different model than averaging word vectors. In gensim library which is largely used in this work, Doc2Vec method has a parameter dm_concat which is defaulted to 0 for averaging as opposed to concatenating. dm_concat should be set to 1 to use concatenation instead of averaging. However, when thought about it, it can be deducted that averaging vectors makes more sense than concatenating as concatenating introduces word context that is not naturally present in the corpus data set.

F. GloVe

GloVe stands for *"Global Vectors for Word Representation"* and the approach is similar to Word2Vec in the sense that it creates word vector embeddings, however, there are some differences between these algorithms. GloVe model efficiently leverages statistical information by training only on the non-zero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows in a large corpus. [Pennington, Socher, & Manning, 2014]; [Shi & Liu, 2014]; [Baroni, Dinu & Kruszewski, 2014] discuss the comparison between Word2Vec and Glove in length, but in short, their objective functions are similar but their difference comes from different cost functions and weighting strategies.

G. t-SNE (t-distributed Stochastic Neighbor Embedding)

t-SNE is a technique introduced by [Maaten & Hinton, 2008] which visualizes high dimensional data by giving each point a location in a two or three dimensional map. Using t-SNE, one can reduce the dimensionality of the data keeping the characteristics and distances between data points to still keep the cluster structure. t-SNE makes the visualization of the data points much easier by projecting n-dimensional space to 2 or 3 dimensional space which is much easier for humans to understand.

We have used sklearn's manifold.TSNE method to project our word vectors (which consist of 300 dimensions) to 2 dimensions and plot it on a bokeh plot to visualize the clusters and similarities.

Why have we investigated these areas?

I've focused in our work to use Word2Vec and Doc2Vec to generate user and item vectors as well as using simpler approaches such as TF-IDF and LSI as a baseline and then measure user and item similarities and compare how the above algorithms compare to each other. There has been some work on using Word2Vec and deep learning in review data sets such as [Pouransari & Ghili, 2014]; [Shirani-Mehr, 2014]; [Xing Margaret & Xiaocheng, 2015]; [Jiang, Liu & Xu, 2015], however, I've felt that there is much more to research and experiment in this area. We wanted to explore and get item similarities based on the way users talk about them or user similarities based on how they use the language and describe a certain set of items which in our case are beers.

Improvements of simple methods with Word2Vec

In the work by [Turney & Pantel, 2010] which was published before Word2Vec-like word embedding algorithms and representations have become popular, it discusses several different Vector Space Models (VSMs) very well. Word2Vec and Doc2Vec family have some improvements over simpler methods such as TF-IDF, some of these are:

- TF-IDF embeddings are just based on a simple word count/frequency while Word2Vec can learn deeper meanings in the corpus and language.
- Word2Vec can pick up various relationships between words which is similar to how humans use and perceive the words and the language such as syntactical connections, synonym/antonym, masculine/feminine, country/capital, artist/album or even movie/actor type of relationships etc. while TF-IDF doesn't build these relationships in its embedding
- TF-IDF cannot identify and combine synonyms and antonyms while Word2Vec can pick these up based on context in which these words are used
- Doc2Vec can be worse off in very short documents as discussed by [De Boom et al., 2015]
- Word2Vec/Doc2Vec performs better with lots of data and doesn't do as well with the limited amount of data while TF-IDF is not directly affected by the corpus size.

Word embeddings evaluations

One of the unanswered questions and open research topic areas is the evaluation of word embeddings. There are a couple of evaluation methods, one of the most common is using word analogy that's curated manually having lists with different syntactic and meaningful relationships that currently consists of the following groups:

Categories	**Examples**
capital-common-countries	Athens Greece Berlin Germany
capital-world	Berlin Germany Lisbon Portugal
currency	Brazil real USA dollar

city-in-state	Seattle Washington Portland Oregon
family	boy girl brother sister
gram1-adjective-to-adverb	amazing amazingly happy happily
gram2-opposite	acceptable unacceptable certain uncertain
gram3-comparative	bad worse big bigger
gram4-superlative	bad worst big biggest
gram5-present-participle	code coding fly flying
gram6-nationality-adjective	Austria Austrian Greece Greek
gram7-past-tense	falling fell going went
gram8-plural	car cars man men
gram9-plural-verbs	find finds search searches

Other hand-curated more complicated but currently uncategorized analogies can also be added to this list of analogies such as ‘man woman king queen’

Looking at these categories, we can see right away that these are very specific analogies that would appear in corpuses like encyclopedia (such as Wikipedia or Google News) with many contexts and domains but would not work well for specific domains, such as the beer reviews domain. Beer reviews data most likely wouldn’t have currencies and nationalities or US states as prevalent (if any) in the database compared to the general Wikipedia corpus, therefore these analogies wouldn’t work. However, I’ve still tried and used it with the words that were in our database as a baseline metric.

There are also other evaluation techniques experimented and discussed in literature such as [Tsvetkov et al., 2015]; [Lau & Baldwin, 2016]; [Schnabel et al. 2015]; [Musto et al., 2015]; [Ma & Zhang, 2015]; [Minarro-Giménez, [Marín-Alonso & Samwald, 2015]; however there’s no conclusive evaluation method that is a clear winner just yet that would work across different domains and for all corpuses.

One of the main challenges and themes in this work has also been the evaluation. I’ve tried various things, mainly calculating and comparing user/user or item/item similarities by different algorithms and also the quite conventional analogies evaluation method. I have also not reached any strong conclusion on this and the evaluation of word embeddings still stays as an open research subject and challenge for researchers.

Data and methodologies used in the work:

Data set: BeerAdvocate beer reviews data set. Below are the schemas and some statistics for the data set:

Users/Reviews table	
Schema	{ "beer_id":"Id of the beer, an integer number", "user_id":"Id of the user, a string", "r_appearance":"Rating for appearance, 1-5", "r_aroma":"Rating for aroma, 1-5", "r_overall":"Overall rating, 1-5", "r_palate":"Rating for palate, 1-5", "r_taste":"Rating for taste, 1-5", "r_text":"Review text, string", "r_time":"Review date/time, timestamp"}
# Reviews	1518567
# Unique users	32908
# Unique beers	49005
Avg overall rating	3.83
Avg review length	687.9 characters
Avg reviews/users	46.15
Avg reviews/beer	31
Beers table	
Schema	{ "beer_id":"Id of the beer, an integer number", "user_id":"Id of the user, a string ", "r_appearance":"Rating for appearance, 1-5 ", "r_aroma":"Rating for aroma, 1-5 ", "r_overall":"Overall rating, 1-5 ", "r_palate":"Rating for palate, 1-5 ", "r_taste":"Rating for taste, 1-5 ", "r_text":"Review text, string", "r_time":"Review date/time, timestamp"}
# Beers	49005
# Brewers	5232
# Beer categories	104
# reviews/categories	14600.6
# beers/brewer	9.37
# beers/style	471.2

Methodologies used:
Reading the data: I have used 2 different methodologies for reading in the data.

- First, I have read all the rows and relevant columns of the data into a pandas dataframe and then preprocessed and stored that in another column in the dataframe. This works well and fast enough with a small data set (e.g. less than 1M reviews)

```
# Function to read .db database using sql and place in a dataframe
def pull_data():
  eng = sa.create_engine('sqlite:////home/cagatay/cagatay_thesis/beeradvocate.db')
  sql = 'select * from reviews'
  df = pd.read_sql(sql, eng)
  df = df[df.beer_id != 'null']
  df['review_id'] = df.index
  return df

# Function to preprocess the review_text, removing stop-words, stemming
# and tokenizing into 'processed' column
def preprocess_data(df):
  stemmer=PorterStemmer()
  stop = stopwords.words('english')
  stop += list(string.punctuation)
  df['processed'] = df.apply(lambda row: [stemmer.stem(i) for i in
word_tokenize(row['r_text'].encode('utf-8')) if i not in stop], axis=1)
  return df

# Run the functions in order and pickle the data
df = pull_data()
preprocess_data(df)
df.to_pickle('processed_full_data.pkl')
```

- Second, I have created a generator object for methods like Word2Vec can use and iterate over, which is faster than the first approach for streaming and/or large data set (e.g. over 1M reviews)

```
class ReviewStream(object):
  def __init__(self, dbname, table, column):
    self.dbname = dbname
    self.table = table
    self.column = column
    self.engine = sa.create_engine("sqlite:///{}".format(self.dbname))
    self.stemmer=PorterStemmer()
    self.stop = set(stopwords.words('english') + list(string.punctuation))
    self.sent_tokenizer = nltk.data.load('nltk:tokenizers/punkt/english.pickle')
    return

  def w_tokenize(self, sent):
    return [self.stemmer.stem(word) for word in nltk.tokenize.word_tokenize(sent)
if word.isalnum() and not word in self.stop]

  def sent_process(self, text):
    raw_sentences = sent_tokenizer.tokenize(text.strip())
    return [self.w_tokenize(sent) for sent in raw_sentences if len(sent) > 0]

  def __iter__(self):
    sql = "select beer_id, user_id, r_text from reviews limit
-1".format(self.column, self.table)
```

```
    for i, (beer_id, user_id, text,) in enumerate(self.engine.execute(sql)):
      yield {'beer_id': beer_id, 'user_id': user_id,
           'text': self.w_tokenize(text)}
dbname, table, column  = "beeradvocate.db", "reviews", "r_text"
rs = ReviewStream(dbname, table, column)
```

The next step is to create word embeddings. I have used two approaches- Word2Vec and Doc2Vec. There are few different ways of training the word vector models. The first is to train the model directly on the reviews- this is the traditional approach which treats each review as a document. The context of each word is learned from its context in the review.

The Doc2Vec techniques assign tags to the documents and learn associations with the tags- the tags are often sentence or paragraph id's. In this case I have used either the unique user id or beer (item) id to 'tag' the document. In this way, the vectors are learning in association with the user who wrote the reviews or in association with the item about which the reviews were written. In addition, I have constructed combinations of word vectors (with Word2Vec) to represent an 'average' user vector:

$$\bar{w}^U = \sum_{R_i \in U} \sum_{v \in R_i} \frac{w_v}{|R||v|}$$

and similarly for average item vectors. Our expectation is that these user and item vectors will be somewhat similar to what Doc2Vec is learning with user and item id tags.

Iterator RGen() for gensim:

```
class RGen(object):
  def __iter__(self):
    for user_id, g in df.groupby('user_id', sort=False):
      words = np.hstack(g.text.values).tolist()
      yield words
```

- Word2Vec
    - Word2Vec models based on each row in reviews (W2V-r)
    - Word2Vec models based on each row in users (reviews by user concatenated first) (W2V-c)
    - Word2Vec models based on each row in reviews intersected by the pre-trained vectors trained by [Mikolov et al., 2013] on 100 billion words of Google News which are public.["Word2Vec", 2013] (W2V-ggl)

The code snippet for one of the above models:

```
import gensim
from gensim.models import Word2Vec

num_features = 300
min_word_count = 5
num_workers = 4
context = 10
downsampling = 1e-3

ggl_model = Word2Vec(RGen(), workers=num_workers, size=num_features, min_count =
min_word_count, window = context, sample = downsampling)
ggl_model.intersect_word2vec_format('GoogleNews-vectors-negative300.bin.gz',
binary=True)
```

- Doc2Vec
  - Doc2Vec based on only user tags (user_models)
  - Doc2Vec based on only beer tags (beer_models)
  - Doc2Vec based on both user and beer tags (d2vmodels)

The code snippet for one of the above models:

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument, LabeledSentence

num_features = 300
min_word_count = 5
num_workers = 4
context = 10
downsampling = 1e-3

beer_docs = [TaggedDocument(words=row.text, tags=[row.user_id]) for index, row in
df.iterrows()]
user_models = {}

for size in [10, 50, 100, 300, 500, 1000]:
  print('size', size)
  m = Doc2Vec(beer_docs, workers=8, size=size, min_count = min_word_count, window
= context, sample = downsampling)
  m.save('d2v_beer_model_{}.model'.format(size))
  beer_models[size] = m
```

Information Retrieval/NLP techniques:

- TF-IDF (Term Frequency - Inverse Document Frequency)
  - TF-IDF on review text
  - TF-IDF on beer categories
  - TF-IDF on beer names

The code snippet for one of the above models:

```
from sklearn.feature_extraction.text import TfidfVectorizer

user_groups = df.groupby('user_id')
tfidf_vectorizer = TfidfVectorizer(tokenizer=lambda x: x, analyzer=lambda x: x)
tfidf_mat_u = tfidf_vectorizer.fit_transform((np.hstack(g.text.values) for (row,
g) in user_groups))

def find_similar(tfidf_matrix=None, idx=0, top_n = 5):
  sims = linear_kernel(tfidf_matrix[idx:idx+1], tfidf_matrix).flatten()
  related = [i for i in sims.argsort()[::-1] if i != idx]
  return [(idx, sims[idx]) for idx in related][0:top_n]

def _gen(tfidf_matrix=None):
  for i in user_idx.keys():
    if (i%1000==0):
      print i
    res = find_similar(tfidf_matrix=tfidf_mat_u, idx=i, top_n=1)[0]
    yield {'user_id':user_idx[i], 'tfidf_sim_user':user_idx[res[0]], 'tfidf_sim':
res[1]}

# we create a data frame for the user-user similarities
df_tfidf_sims = pd.DataFrame(_gen(tfidf_matrix=tfidf_mat_u))
```

- LSI (Latent Semantic Indexing)

The code snippet for LSI:

```
from gensim import corpora, models, similarities

dictionary = corpora.Dictionary(RGen())
dictionary.save('/tmp/lsi.dict')  # store the dictionary, for future reference
print(dictionary)

corpus = [dictionary.doc2bow(text) for text in RGen()]

lsi = models.LsiModel(corpus, id2word=dictionary, num_topics=10)
```

And we examine user-to-user similarities along different dimensions:

- Creating user-user similarities:
  - W2V based on reviews - W2V-r user similarities
  - W2V based on users - W2V-c user similarities
  - W2V based on reviews intersected by Google News pre-trained model - W2V-ggl user similarities
  - D2V beer tags - D2V-b beer similarities
  - D2V user tags - D2V-u user similarities
  - D2V beer and user tags - D2V-bu user similarities

The code snippet for one of the above calculations:

```
def _user_wv(model=None, user_id=None):
  for word in np.hstack(df.query("user_id == @user_id",
engine='python').text.values):
    if word in model.wv.vocab:
      yield model.wv[word]
wv_r_mat_300 = np.array([np.mean(list(_user_wv(model=w2vmodels_r[300],
user_id=user_id)), axis=0) for user_id in user_idx.values()])
wv_r_sim_300 = cosine_similarity(wv_r_mat_300, wv_r_mat_300)
```

- t-SNE:
  - Users vs beer categories
  - Beers vs beer categories

The code snippet for t-SNE:

```
from sklearn.manifold import TSNE
tsne_model = TSNE(n_components=2, verbose=1, random_state=0, init='pca')
tsne_d2v_beer = tsne_model.fit_transform(beer_doc_vectors)

top_styles = [u'american ipa', u'american double/imperial ipa',
              u'american pale ale (apa)', u'russian imperial stout', u'american
double/imperial stout']
df_topstyles = df[df.beer_style.isin(top_styles)]

import bokeh.plotting as bp
from bokeh.models import HoverTool, BoxSelectTool
from bokeh.plotting import figure, show, output_notebook

colormap = {u'american ipa': '#0000FF', u'american double/imperial ipa':
'#00CED1', u'american pale ale (apa)': '#6495ED', u'american double/imperial
stout': '#B8860B', u'russian imperial stout': '#F0E68C'}
colors = [colormap[x] for x in df_topstyles['beer_style']]

plot_d2v = bp.figure(plot_width=900, plot_height=700, title='D2V Beer Models by
Beer Category',
  tools='pan,wheel_zoom,box_zoom,reset,hover,previewsave',
  x_axis_type=None, y_axis_type=None, min_border=1)

plot_d2v.scatter(x=tsne_d2v_beer[:,0], y=tsne_d2v_beer[:,1],
          color=colors,
          source=bp.ColumnDataSource({
            'review': df_topstyles.text,
            'beer_id': df_topstyles.beer_id,
            'cat': df_topstyles.beer_style
    }))
hover = plot_d2v.select(dict(type=HoverTool))
hover.tooltips=[('review', '@review'), ('beer_id', '@beer_id'), ('category',
'@cat')]
output_notebook()
show(plot_d2v)
```

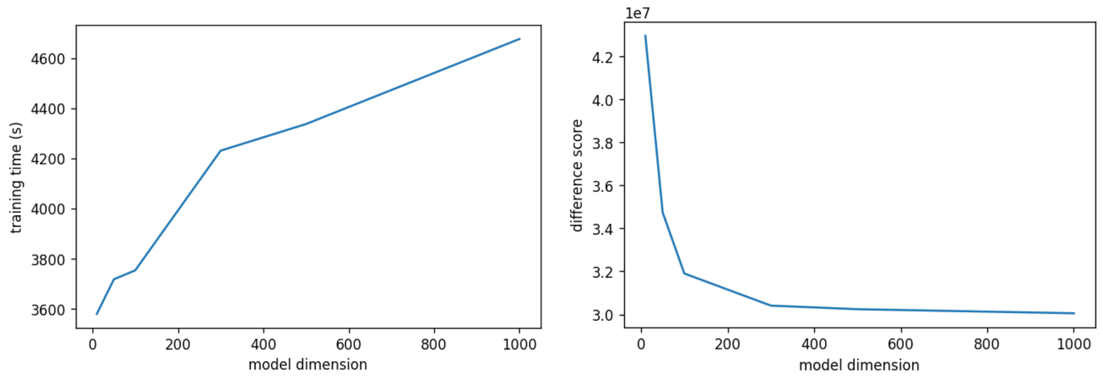
4 Results/Evaluation

Original motivation

The original motivation at the beginning of my work was to explore word vectors for various NLP tasks, specifically on item reviews, primarily use beer reviews for my exploration, do sentiment analysis, topic extraction (to find themes in reviews, such as [taste, feel, color, price] etc.), making recommendations based on user, finding item and user similarities etc.

The course over the period in the research slightly changed its direction and I haven't had a chance to do all of explorations that I've planned and focused on evaluation more than initially planned. The first predictions were that Word2Vec and Doc2Vec would somehow correlate with TF-IDF and LSI which not necessarily proved true and then I've focused on understanding the data further using clustering, dimension reduction with t-SNE and bokeh plots.

One of the first things I've focused on was to understand how model dimension affects model quality and accuracy. For this purpose, I've used TF-IDF pair-wise similarity scores with a subset of user reviews as a proxy and checked the effect with varying model dimensions. I've chosen 10, 50, 100, 300, 500, 1000 as dimensions to test. I've then generated pairwise user similarities in TF-IDF and in Word2Vec separately, took the absolute difference of each element of the matrix and summed all these elements up to get a numeric metric to measure the variation between Word2Vec and TF-IDF.



(a) (b)

Figure 13. Plot (a) shows how the training time changes (in seconds) with increasing dimension and plot (b) shows the difference score (x10^7) as sum of differences of pair-wise use similarity scores in [0, 1] between W2V vs TF-IDF for increasing dimension.

In fig. 13, we can see that the relationship between training time and dimension is linear and we can say that the processing time requirement for W2V is O(n). We see in (b) that the difference between W2V and TF-IDF user similarity scores decrease for increasing dimension, however, the rate of closing the gap decreases with increasing dimension. It can mean that W2V model gets more accurate (although this is just a proxy for accuracy) and it looks like dimension 300 is optimal considering the reduced amount of gain above that point and the increasing training time.

When I started the project, before having the plot (b) above, I predicted that this plot would be like a bell curve, first decreasing with increasing dimension and then increasing after some point (which would be the optimal dimension). This didn't turn out as predicted probably due to that W2V and TF-IDF embed things differently. Potentially W2V could be a better method than TF-IDF and it's not very reasonable to expect a superior method to follow the inferior method exactly to measure its accuracy. However, there still is some kind of an optimal point such as 300 if we optimize both the training time and accuracy parameters.

Additionally, I've explored the effect of context size on the accuracy of the system. The results (based on spot checks, not checked statistically) are similar to the results obtained by [Pouransari & Ghili, 2014], accuracy of the system is higher with 10 as context size vs 5 and 15, however, there isn't a very crisp distinction between these models with varying context sizes. E.g.:

```
[In] -> Model.most_similar('stout')
# for Context Size 5:
[Out] -> (u'porter', 0.7918), (u'ri', 0.7238), (u'stouti', 0.5432)
# for Context Size 10:
[Out] -> (u'porter', 0.7954), (u'ri', 0.7044), (u'guin', 0.4942)
# for Context Size 15:
[Out] -> (u'porter', 0.7840), (u'ri', 0.6810), (u'guin', 0.5016)
```

Having that in mind and given this example, context size can affect the performance significantly for small data sets. Below are some interesting examples of most similar words:

- beer ⇔ brew
- brewery ⇔ (brewer, microbrewery)
- blond ⇔ pale
- black ⇔ mocha
- alcohol ⇔ booze
- cider ⇔ cidar (misspelling)
- heineken ⇔ (beck, heinekin, heini, budweis, corona, grolsch, heiniken, hein, peroni) (misspellings and very similar lager beers)
- abv ⇔ (apv, avb) (seems like misspellings of abv)

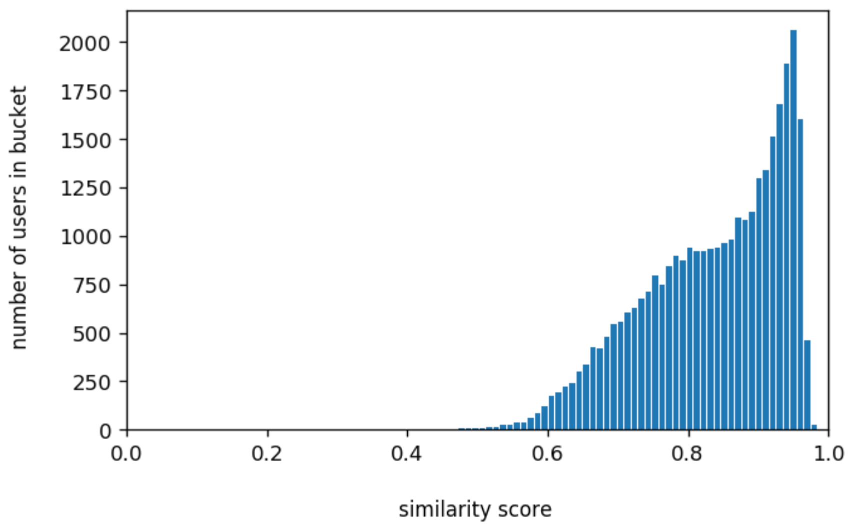


Figure 14. Doc2Vec most similar users similarity scores histogram, number of users for each similarity score bucket vs the similarity scores ranging [0, 1]

In fig. 14, the histogram of user similarity scores for the Doc2Vec model, it's observed that a lot of users have users with high similarity (>0.5), this is because we choose and plot the most similar user's similarity score for each user. Otherwise, it'd be in 0.3 - 0.4 interval, similarly to the average TF-IDF similarity score of two random users. We observe that there are very few users with similarity over 0.97, and they are likely to be duplicates, we could use this data to look into those and deduplicate as necessary.

Evaluation of TF-IDF:

In fig. 15, we see that TF-IDF most similar user similarity scores for each user are quite dense around [0.3, 0.8] interval while very few users are similar over a score of 0.8.

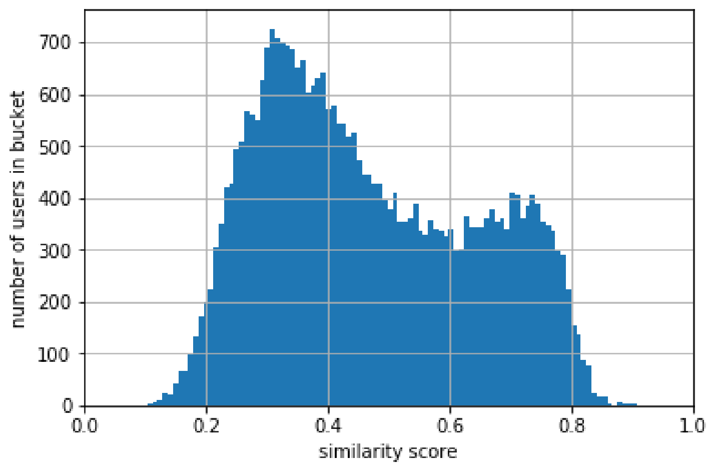


Figure 15. TF-IDF most similar users similarity scores histogram, number of users for each similarity score bucket vs the similarity scores ranging [0, 1]

In general, if we pick 2 random users from the data set and measure their similarity score, we would see that most of them would have a similarity around 0.3 which is the basic similarity score arising from the fact that they use the same language, same words - such as 'beer', 'taste' etc. and it's very unlikely to find users with TF-IDF cosine similarity of 0.2 or lower and also very unlikely to find users with TF-IDF cosine similarity of 0.8 or above. However, as plotted in the histogram of the similarity score for the most similar user for each user in fig. 15, it's not a random selection and we get higher similarity scores, and the histogram has the highest density around 0.3-0.8 interval.

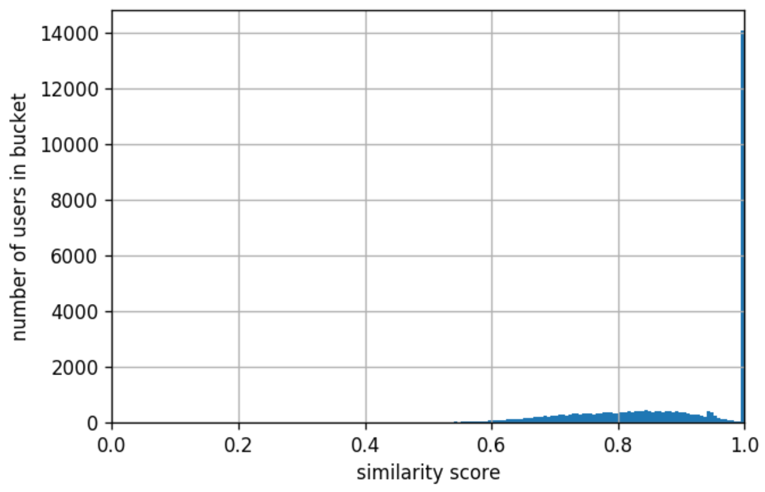


Figure 16. TF-IDF histogram of similarity scores of most similar user for each user based on beer categories, number of users in buckets vs similarity score [0, 1]

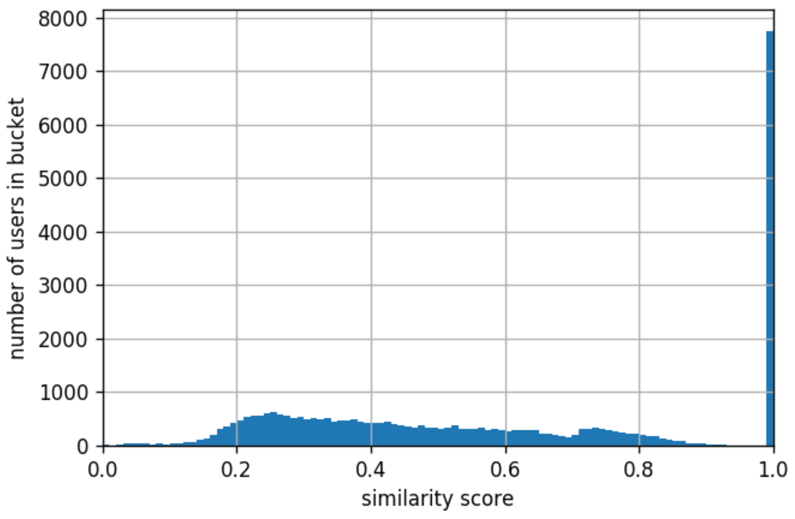


Figure 17. TF-IDF histogram of similarity scores of most similar user for each user based on beer names, number of users in buckets vs similarity score [0, 1]

I was also curious about how TF-IDF behaves not based on what the reviews wrote about beers, but the beers and beer categories they've reviewed, and how the histograms for the similarity scores for most similar users vary based on that. This analysis is designed to inform us whether users mostly review beers of a few distinct types, or if their tastes are very broad. As we can see in fig.16 and fig.17, there's a huge spike around 1.0 similarity. This is because, for most of the users - especially users that reviewed only 1 or 2-3 beers, it's very easy to find at least one other user that reviewed exactly the same beers or beer categories. Beer categories and beer names are much more limited and less diverse than the English language which as they are just a list of predefined items but the English language is very flexible with many permutations of words to use; and less diversity means that the similarity of users tend to converge to each other. Beer categories are less diverse (only a little more than 100 categories) than beer names (a couple of thousands), so we see less diversity in most similar user scores. Most of the users have one other user with a very high similarity score based on beer categories and beer names, and it's very difficult to find any user who doesn't have a most similar user with a score less than 0.5.

Evaluation of word vectors:

As mentioned before, evaluation of word embeddings is an open question and an active research area that has been explored by some of the work that was mentioned earlier, and it still is being researched by other researchers as well.

First, as many researchers do, I've checked for sensible similarities with some words as a sanity check. This is a basic consistency check that the word vectors we have learned are accurately representing the context of the words from the reviews. For each case, we've set the number of dimensions, then train the word2vec model on the reviews, and checked the most similar words. We've changed the dimensions from 10 to 1000 to see how different dimensions (embedding space dimension/resolution) affect the accuracy of similarities and their scores. See below for the similar words to stout and their similarity scores with different dimensions using different model training methodologies (intersection with GoogleNews, training word vectors on review documents vs training word vectors on user documents(by concatenating reviews for each user))

```
[In] -> ggl_model.most_similar('stout', topn=3)
[Out] -> (300, [(u'stouter', 0.56), (u'stingy', 0.53), (u'beefy', 0.46)])
[In]->  w2vmodels_r[s].most_similar('stout', topn=3)
[Out] -> (10, [(u'porter', 0.98), (u'ri', 0.94), (u'oatmeal', 0.92)])
[Out] -> (50, [(u'porter', 0.92), (u'ri', 0.82), (u'stouti', 0.67)])
[Out] -> (100, [(u'porter', 0.85), (u'ri', 0.80), (u'stouti', 0.56)])
[Out] -> (300, [(u'porter', 0.79), (u'ri', 0.70), (u'stouti', 0.50)])
```

```
[Out] -> (500, [(u'porter', 0.77), (u'ri', 0.67), (u'stouti', 0.47)])
[Out] -> (1000, [(u'porter', 0.75), (u'ri', 0.65), (u'stouti', 0.47)])
[In] -> w2vmodels_c[s].most_similar('stout', topn=3)
[Out] -> (10, [(u'porter', 0.95), (u'ri', 0.93), (u'impi', 0.92)])
[Out] -> (50, [(u'porter', 0.85), (u'ri', 0.80), (u'wrassler', 0.68)])
[Out] -> (100, [(u'porter', 0.78), (u'ri', 0.75), (u'mackeson', 0.61)])
[Out] -> (300, [(u'porter', 0.71), (u'ri', 0.68), (u'mackeson', 0.53)])
[Out] -> (500, [(u'porter', 0.69), (u'ri', 0.65), (u'stouti', 0.51)])
[Out] -> (1000, [(u'porter', 0.69), (u'ri', 0.64), (u'stouti', 0.51)])
```

As you can see, I have tried different dimensions of Word2Vec models in my work to see and evaluate how this affects the performance of the models. The dimensions that we have used are: 10, 50, 100, 300, 500, 1000. The first numbers in each result are the model dimension and later the most similar entities. E.g. (10, [(u'porter', 0.98), (u'ri', 0.94), (u'oatmeal', 0.92)]) has size 10 and most similar entities are 'porter' (with similarity score 0.98), second most similar being 'ri' - preprocessed version of 'rye' (with similarity score 0.94) and the third most similar being 'oatmeal' (with similarity score 0.92). The model that was intersected with the pre-trained GoogleNews vectors was trained with dimension 300, as this is the dimension GoogleNews vectors were originally trained on and therefore it's required to have the same dimension to intersect the two models. When intersecting one model with another, there's no vocabulary added from the second model to the first model, first model's vocabulary remains unchanged and only the vector weights are adjusted according to the vectors in the second model for the intersecting vocabulary.

Another important point to add for GoogleNews trained corpus is that one cannot just load and continue training these vectors with other corpus as the loading function works with vectors-only format of Word2Vec C implementation and it's missing the necessary dictionaries and binary trees to be continued for model training [Mohr, 2015]

There are some reasonable conclusions to draw from the above list of similarities and spot checks in the Word2Vec models. Such as:

- The word vector Dimension ***is*** a factor affecting model quality
- It's hard to say or systematically decide which dimension is the best
    - Low dimension (20-) means that there's not enough resolution in the model output so things look similar although they are not, while high dimension (500+) means that there's too much resolution and things can look like far away from each other although in fact, they are similar and close to each other.
- Low dimensional models are more confident in similarity scores (most likely due to low resolution) and the confidence scores seem to decrease

with increasing dimension - for example, most similar word to ‘stout’ is ‘porter’ with 0.98 as similarity score when the dimension is 10 vs 0.79 when the dimension is 300.

- High similarity score doesn’t mean that it’s accurate. For example, we see that with the model with 10 dimensions, that has good first 2 guesses for most similar to words and consistent with higher dimensional models, but the third one is obviously worse off compared to higher dimensions which determined ‘stouti’ which is the processed version of ‘stouty’ as the 3rd most similar.
- Continuing on the previous example, we see that even the model with 50 dimensions picks up this similar word (‘stouti’) and increasing dimensions over 100 don’t affect the similarity scores very significantly - as much as the same increase in low dimensions.
- Doc2Vec with multiple tags can give similarities across tags. For example, the most similar entity to a user can be a beer, which is not very intuitive but makes sense. Beers that are similar to users could be recommended to users.
- GoogleNews intersected model is a lot less confident than non-intersected models despite the average dimensionality (300). We see that the results and confidence of non-intersected domain specific models are more accurate. Pre-trained vectors can add and indeed are (in our example) adding noise to the models.
- w2vmodels_r which is based on averaging review vectors performs better than w2vmodels_c which is based on concatenated reviews for each user, as the similarity scores are higher and more accurate for each corresponding dimension.

We continued our analysis to check the similarity of users and beers by Word2Vec and Doc2Vec models. We have trained Doc2Vec models with dm_concat parameter set to 0, so the documents are not obtained by concatenation but averaging review vectors. See below for some similarity measures for one of the users/beers and their similar items and the relative similarity scores.

```
[In] -> beer_models[s].docvecs.most_similar('246', topn=3)
[Out] -> (10, [(u'567', 0.992), (u'2435', 0.985), (u'266', 0.984)])
[Out] -> (50, [(u'449', 0.947), (u'1790', 0.928), (u'567', 0.927)])
[Out] -> (100, [(u'449', 0.882), (u'567', 0.876), (u'580', 0.856)])
[Out] -> (300, [(u'449', 0.773), (u'1426', 0.742), (u'436', 0.736)])
[Out] -> (500, [(u'449', 0.732), (u'436', 0.705), (u'1426', 0.693)])
[Out] -> (1000, [(u'449', 0.703), (u'436', 0.684), (u'2435', 0.670)])

[In] -> user_models[s].docvecs.most_similar('thyde606', topn=3)
[Out] -> (10, [(u'immacolata', 0.995), (u'lanfear', 0.994), (u'laddyboy', 0.993)])
```

```
[Out] -> (50, [(u'nkronma', 0.959), (u'vieille', 0.958), (u'rwalschlager',
0.958)])
[Out] -> (100, [(u'beerfoolish', 0.935), (u'reilly88', 0.933), (u'ketfunk',
0.930)])
[Out] -> (300, [(u'beerfoolish', 0.889), (u'shaman2788', 0.867), (u'brouwerijman',
0.861)])
[Out] -> (500, [(u'beerfoolish', 0.898), (u'rbartow', 0.860), (u'vaneson',
0.860)])
[Out] -> (1000, [(u'beerfoolish', 0.888), (u'darkhalf', 0.839), (u'nkronma',
0.838)])

[In] -> d2vmodels[s].docvecs.most_similar('beerfoolish', topn=3)
[Out] -> (10, [(u'25311', 0.996), (u'clr231', 0.993), (u'camzela', 0.992)])
[Out] -> (50, [(u'chugs13', 0.926), (u'ilovestouts', 0.924), (u'monkeefish',
0.921)])
[Out] -> (100, [(u'thyde606', 0.901), (u'bdeast1', 0.861), (u'ouroborus', 0.858)])
[Out] -> (300, [(u'thyde606', 0.854), (u'scottf2345', 0.757), (u'alodge', 0.757)])
[Out] -> (500, [(u'thyde606', 0.834), (u'nkronma', 0.755), (u'39672', 0.750)])
[Out] -> (1000, [(u'thyde606', 0.768), (u'ocpathfinder', 0.703),
(u'muchloveforhops3', 0.680)])

[In] -> d2vmodels[s].docvecs.most_similar('246', topn=3)
[Out] -> (10, [(u'580', 0.990), (u'567', 0.987), (u'2842', 0.985)])
[Out] -> (50, [(u'449', 0.934), (u'567', 0.927), (u'2563', 0.923)])
[Out] -> (100, [(u'449', 0.898), (u'567', 0.888), (u'580', 0.872)])
[Out] -> (300, [(u'449', 0.777), (u'1053', 0.751), (u'1426', 0.747)])
[Out] -> (500, [(u'449', 0.725), (u'436', 0.696), (u'1426', 0.687)])
[Out] -> (1000, [(u'449', 0.682), (u'2435', 0.657), (u'436', 0.648)])
```

Important beer_id's used in the above code snippet:
246: heineken lager beer (**category: euro pale lager**)
449: stella artois (**category: euro pale lager**)
567: rolling rock extra pale (category: american adjunct lager)
436: amstel light (category: light lager)
580: miller high life (category: american adjunct lager)

As we can see from beer_models similarities to beer_id 246 which is heineken lager in *euro pale lager category*, almost all models agree that 449 which is stella artois in *euro pale lager category* is the most similar beer, with the exception of the model with 10 dimensions. This indeed seems an accurate most similar beer to heineken lager as they are in the same category. However, the model with 10 dimensions identifies 567 which is rolling rock extra pale in *american adjunct lager category* as the most similar beer, and fails to pick up 449 in the second and third place as well. It seems like dimension as low as 10 can be too low to separate items correctly.

We see that the same is happening for d2v_models similarities. All models with dimensions except the one with 10 dimensions accurately identify stella artois as the most similar beer. The model with 10 dimensions identify 580 and 567 which are both in *american adjunct lager category* as the most similar beers. We see that most other models also give 567 a high weight of similarity. We can guess from this that *american adjunct lager category* is talked about in a similar way as *euro pale lager category*.

Important user_ids and their reviews below:
thyde606 (reviewing beer with beer_id 23030 - portsmouth kate the great): "*the smell was innitially of darker fruits. plum and fig are coming through quick nicely with an aromatic blend of roasted malt and alcohol. also black coffee and slight bitter chocolate in the nose. the taste releases the fruit flavors quite nicely. the plum and fig taste almost chocolate covered with bitter-sweet chocolate. roasted (almost burnt) black coffee flavors and a slight but no too strong alcoholic warming. the beer is exceptionally smooth for a* 10.5% abv brew. *no aggressive bitterness only that which comes from the chocolate tastes. the mouthfeel is heavy but not syrupy. an even consistancy and balance to all the flavors. not an assertive brew that screams alcohol, but raher more mellow and subtlely warming.*"

beerfoolish (reviewing beer with beer_id 31548 - big black voodoo daddy): "first ever review... and damn what a way to start. *the smell was initially of darker fruits. plum and fig are coming through quick nicely with an aromatic blend of roasted malt and alcohol. also black coffee and slight bitter chocolate in the nose. the taste releases the fruit flavors quite nicely. the plum and fig taste almost chocolate covered with bitter-sweet chocolate. roasted (almost burnt) black coffee flavors and a slight but no too strong alcoholic warming. the beer is exceptionally smooth for a* 12% abv brew. *no aggressive bitterness only that which comes from the chocolate tastes. the mouthfeel is heavy but not syrupy. an even consistency and balance to all the flavors. not an assertive brew that screams alcohol, but rather more mellow and subtlety warming.* this was a great imperial stout"

For users on the other hand (thyde606 and beerfoolish being the most similar users identified by almost all of the models with all dimensions), we see that these users are indeed very very similar, with 90%+ of their sentences are overlapping - almost like a copy-paste. Models with dimensions 10 and 50 fail to pick these user similarities and pick other users that are not as similar upon inspection. For user similarities, dimension as low as 50 turns out to be too low to correctly represent users in the embedding space. However, dimension as high as 100 seems accurate enough for most applications and making obvious connections between similar users, without the necessity of increased complexity and more time and power consuming solutions which higher dimensions require.

From the above results it's also observed that users that have a very high similarity scores such as 0.85+ (depending on the dimension chosen) might be too similar as much as being duplicate. This could be used to detect bots in the system and remove duplicates.

If we compare user_models (based on only user tags) and d2vmodels (based on user and beer tags), all other things (like dimension, context size etc.) being equal, we find that d2vmodels are less certain (have lower similarity scores) for the same set of users (thyde606 vs beerfoolish). Therefore we conclude that multiple tags can harm the accuracy of the model and wouldn't behave the same as single tagged models.

Evaluation of word vectors based on analogies:
(full table test categories in appendix)

Researchers have created a list of analogies in different categories for general corpus which you can see below for a relevant list and appendix for the full list. They use these analogies to test their models against some standard syntactic or meaningful analogies and measure model accuracy based on these.

	# Correct results per dimension					
Category (volume)	**10**	**50**	**100**	**300**	**500**	**1000**
capital-common-countries (156)	0	16	18	27	29	34
capital-world (214)	0	15	25	48	48	50
currency (52)	0	0	1	3	2	1
city-in-state (514)	0	13	33	57	60	63
family (110)	0	7	14	20	19	17
gram3-comparative (1056)	15	274	369	420	413	424
gram4-superlative (812)	7	126	168	174	174	157
gram6-nationality-adjective (538)	2	15	35	64	70	73
gram8-plural (6)	0	1	3	5	6	6
total (3470)	24	468	666	818	821	825

Figure 18. The results of the analogy test based on each category, with the number of test clauses for each category, are given in brackets.

There are ~20000 analogies in this test, however, only 3470 of them have all 4 words in the analogies in the data set and therefore were considered as a valid test clause for our data set. As can be seen in fig. 18, the general corpus analogy evaluation set does not really apply well in a domain specific model.

If we have a closer look at the numbers above, in a lot of categories (and with the total set), we see a steady increase from dimension 10 to 1000. However, this increase is not at the same pace, there's not much increase in the number of correct results after dimension 300. We see that biggest jump in these numbers happen from 10 to 50 and We also observe that dimension 300 performs better at some categories such as currency, family, and gram4-superlative as an optimal point whereby the correct results decrease on either side of 300. A plot of the behaviors for each category can be useful which is plotted in fig. 19:

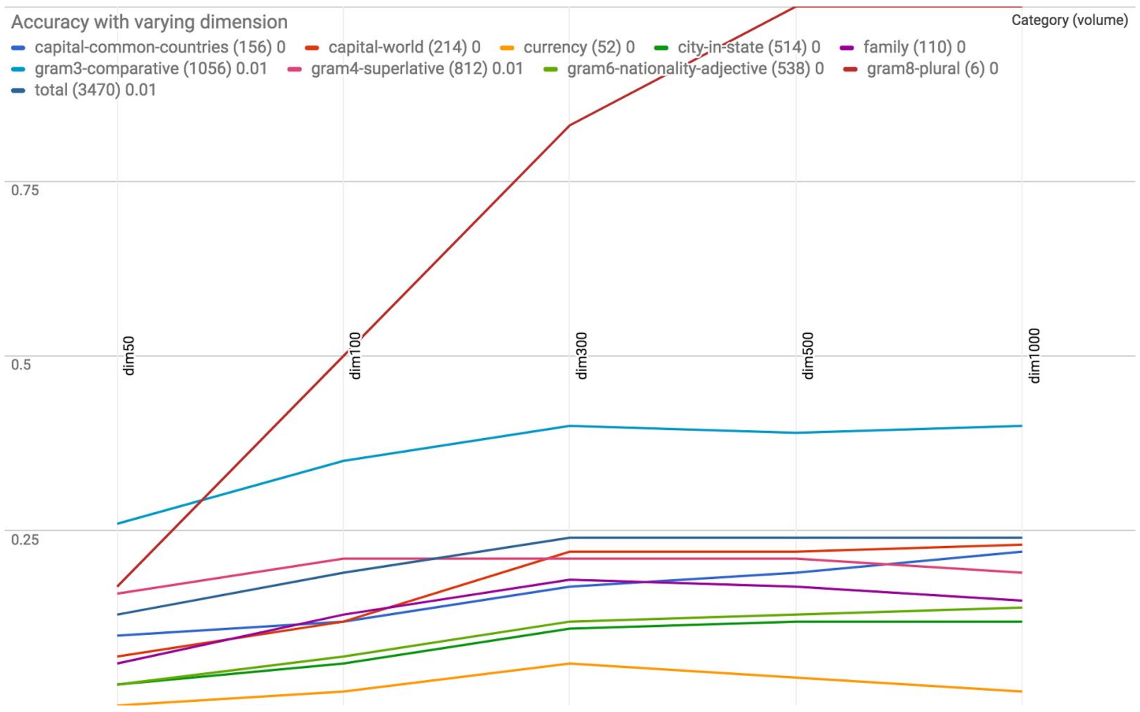
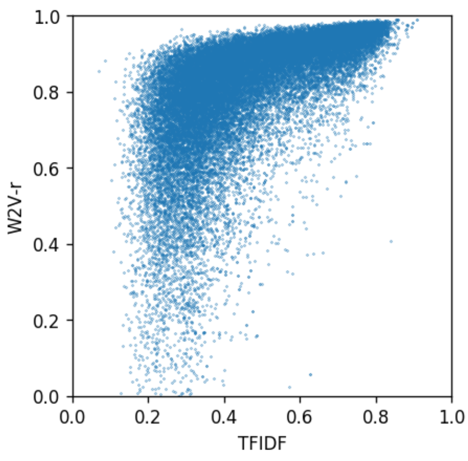


Figure 19. Accuracy rates per each category with changing dimension

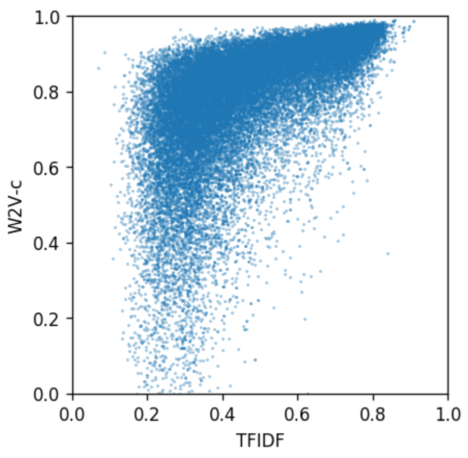
User Similarity: Word Embeddings vs. TF-IDF

The analogy tests whose results are given above may not give us the ground truth in accuracy of a model but give some idea on whether these embeddings actually do something and represent some relations in the data set, or do they fail representing at least some of the simple analogies. So these initial results are good to have, but they are not conclusive. To approach this more systematically, we had a prediction that Word2Vec embeddings will show a direct correlation with TF-IDF vectors. Fig. 20 shows the correlation between the TF-IDF and word vectors, however, it is clear that the correlation is not very strong on the whole.
To do this comparison, I have calculated vectors using both methodologies separately and calculated the most similar user for each user (based on cosine

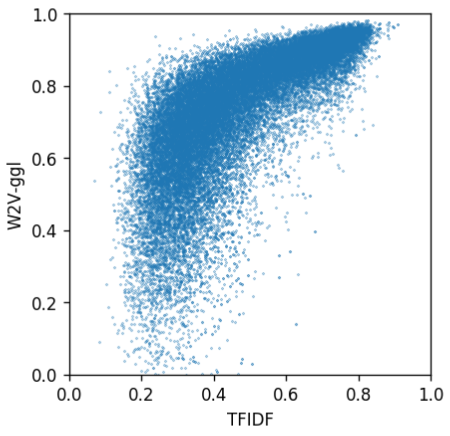
similarities of TF-IDF vectors and W2V vectors) and stored these scores in a dataframe. The scores ale plotted as below:



(a)



(b)



(c)

Figure 20. TF-IDF (x-axis) vs W2V (y-axis) most similar users' similarity scores for 3 different w2v models. First plot (a) shows W2V-r (based on each row in reviews table), second plot (b) shows W2V-c (based on concatenated reviews for each user) and the third plot (c) shows W2V-ggl (W2V-r model intersected with pre-trained GoogleNews vectors).

Before discussing the plots, it's worth highlighting some important points while training Word2Vec.

- gensim Word2Vec and Doc2Vec methods take a parameter to decide whether it should average the vectors or to concatenate and train the models to represent the tagged entity
- Intuitively averaging should work better as concatenation adds word contexts that are not naturally there. For example:
    - (U1, R1): _a_ _b_ _c_.
    - (U1, R2): _d_ _e_ _f_.

    - U1 Concat(R1,R2): _a_ _b_ _c_._d_ _e_ _f_.

  c now has a new context window with d, e and f and d now has a new context window with a, b and c while they should be in fact out of context of each other as this wasn’t in the original data set.

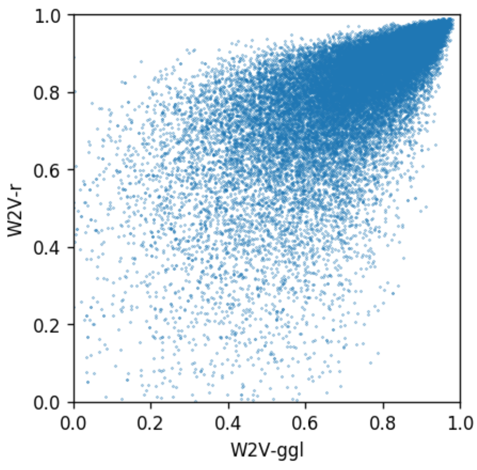
Evaluating the plots for W2V vs TF-IDF comparison and how this changes from one model to another (w2v_r, w2v_c and w2v_ggl), we can make the following points:

- Fig. 20 (a), which is the averaged word vectors on the reviews, show almost no linear correlation and looks like a power function when power is between 0 and 1.
- Fig. 20 (b) and (c) get closer to the straight line, but we still can’t really say that there is a linear correlation.
- This shows us that the original/ideal Word2Vec model does not have a linear correlation at all and that TF-IDF and Word2Vec somehow have different word embeddings, which encrypt different type of features within the data.
- Fig. 20 (b) and (c) look like more noisy and less characteristic Word2Vec model, therefore, we can come to the conclusion that intersecting with a pre-trained general corpus vectors (such as GoogleNews) often just adds noise and doesn’t improve domain specific accuracy of word vectors. This highlights the need and benefit of having domain-specific models for our data set for higher accuracy.

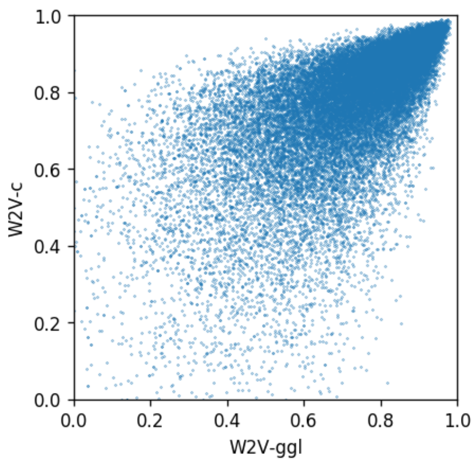
User Similarity by different Word Embeddings

In fig. 21 below, we see the comparison of different approaches to combining Word2Vec models to get document (user) models and how they compare to each other. We see that averaging and concatenation have almost a linear line showing a strong correlation, therefore we could say that for document (user or item) comparison purposes, the method we choose doesn’t affect the results too much, at least for our data set. However, as we’ve observed above, concatenation is generally less confident in its word embeddings.

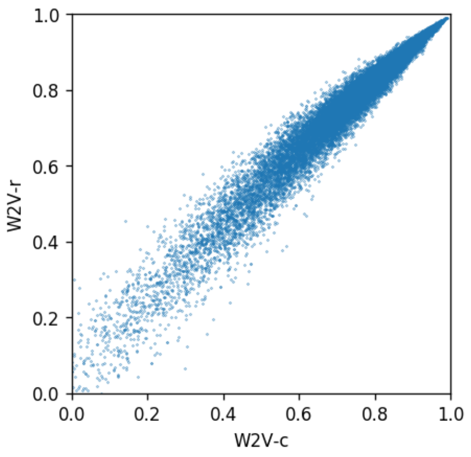
Concatenation method first concatenates the reviews and creates user or item documents, trains the model based on these documents and learn word vectors from it and then averages word vectors in each document. On the other hand, averaging method uses the documents without altering (no concatenation), trains the model based on review documents and then averages word vectors for each user or item document. Because averaging method doesn’t change the initial corpus structure, this method is a better way of generating document vectors.



(a)



(b)



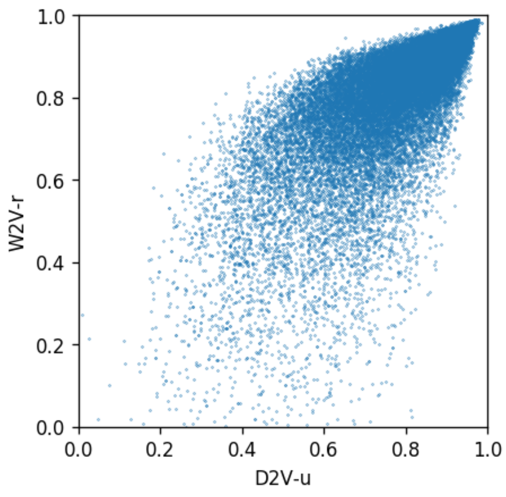
(c)

Figure 21. First plot (a) shows the word vectors' user similarities for W2V-ggl (x-axis) - W2V-r (y-axis) while (b) shows the same for W2V-ggl - W2V-c and (c) shows W2V-c - W2V-r

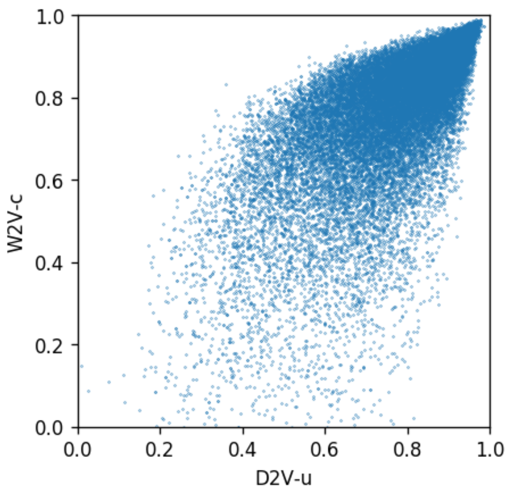
User Similarity by Doc2Vec:

When we check Fig. 22 below for the plots for the word vectors intersected with GoogleNews pre-trained vectors, it's as if some noise is added to the data on top of the domain specific model - it resembles a line that has a lot of noise on top of it. As a result, we can say that pre-trained vectors may not necessarily improve the document similarities compared to domain specific models, and in fact they can distort them.

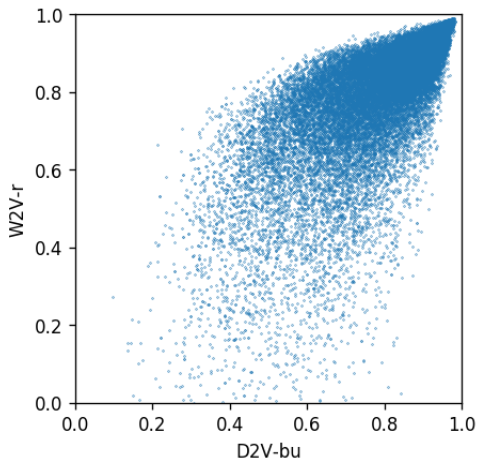
Fig. 22 (a), (b) and (c), Doc2Vec vectors based on user tags only or user tags+beer tags user similarities compared to W2V-r and W2V-c show that they somewhat correlate with W2V-r and W2V-c respectively. Plot (d) which represents D2V-u vs W2V-ggl looks like a slightly more dense version of the previous three comparisons and GoogleNews pre-trained vectors somehow make the model user similarities closer to what doc vectors generate. Plot (e) which shows D2V-bu vs D2V-u on the other hand shows a substantial correlation as it is nearly a perfectly straight line. It can be observed that they are doing things slightly different, for example, D2V-bu model makes use of both user and beer tags and can give out a beer as the most similar entity to a user, or a user can be the most similar entity to a beer, which changes the most similar item score for many users.



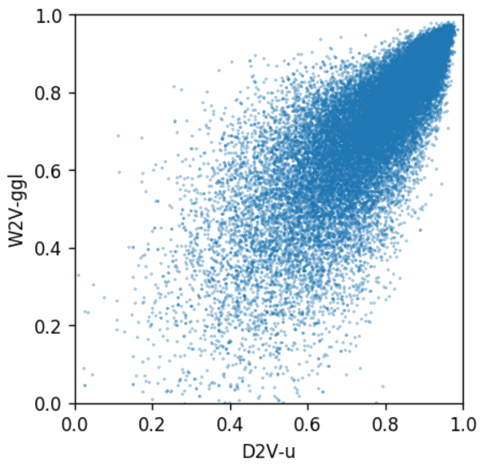
(a)



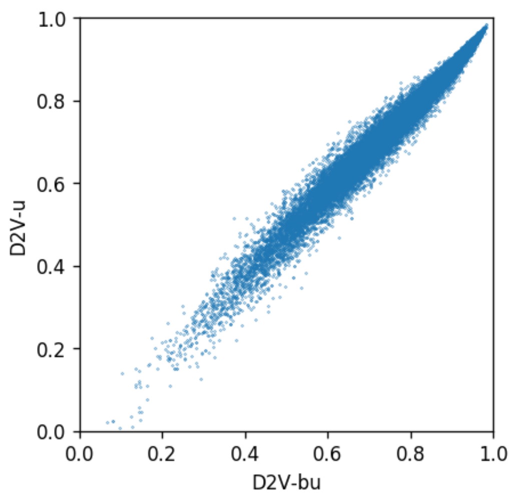
(b)



(c)



(d)



(e)

Figure 22. First plot (a) shows the doc vectors' (based on user tags) user similarities (D2V-u) with W2V-r while (b) shows the same for D2V-u - W2V-c, (c) shows the same for D2V-bu - W2V-r, (d) shows D2V-u - W2V-ggl and (e) shows D2V-bu - D2V-u

Evaluation of LSI:

Evaluation of LSI is also tricky as in the evaluation of word embeddings. You can see below the LSI similarity score distribution with varying topic size: 5-10-20 topics.

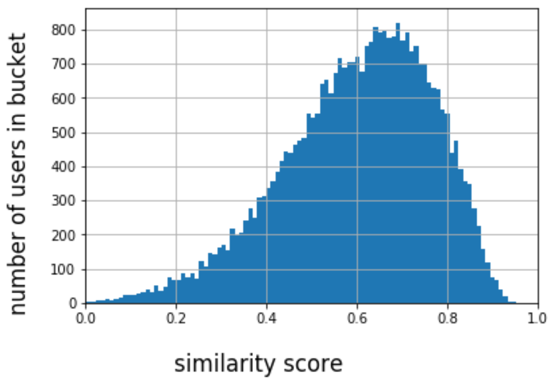
It's often difficult to say what each of these categories/topics mean with just a few words, but we can see the following themes in the topics:

e.g.
LSI-5: 0: malt beer flavor; 1: nice beer light pour;
2: nice malt taste note; 3: taste nice smell 4: nice ale beer pour color

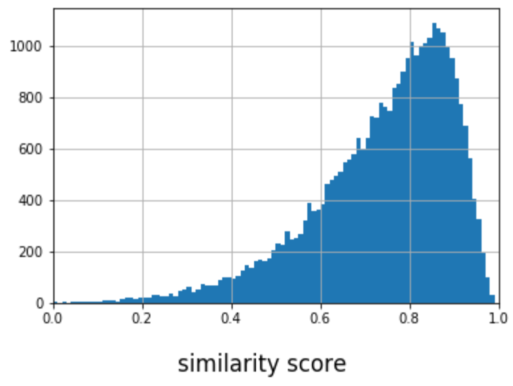
LSI topics-5:

```
[(0, u'0.31*"beer" + 0.22*"hop" + 0.22*"malt" + 0.21*"head" + 0.19*"flavor"'),
 (1, u'-0.78*"beer" + 0.23*"nice" + 0.16*"bodi" + 0.15*"light" + 0.15*"pour"'),
 (2, u'-0.30*"nice" + -0.26*"one" + 0.24*"malt" + -0.23*"tast" + 0.23*"note"'),
   (3,   u'0.36*"tast"   +   -0.31*"nice"   +   -0.28*"note"   +   0.23*"smell"   +
-0.23*"realli"'),
 (4, u'0.33*"beer" + 0.33*"nice" + 0.23*"pour" + 0.22*"color" + -0.21*"ale"')]
```

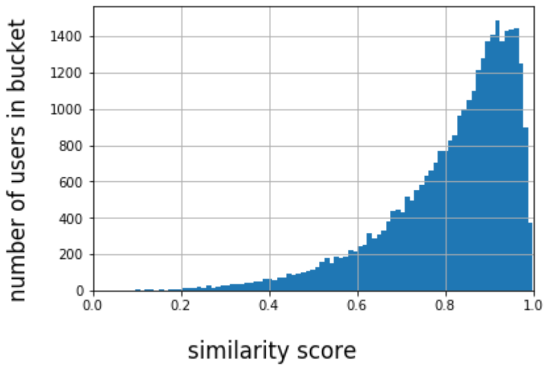
I've also graphed the distribution of user similarity scores (for the most similar user to each user) based on LSI projections of user reviews with varying topic size, which is presented in fig. 23 below.



(a)



(b)

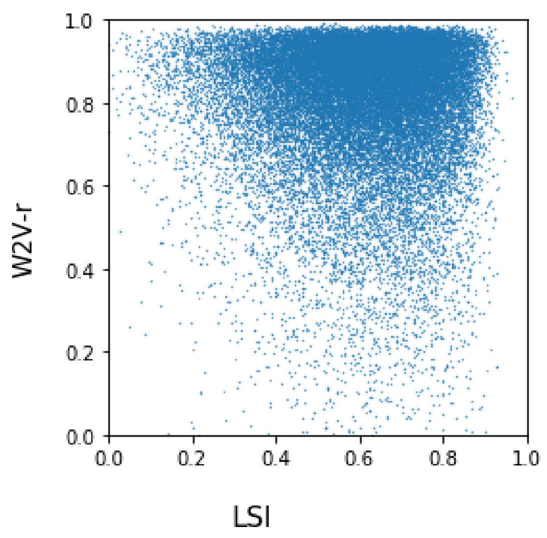


(c)

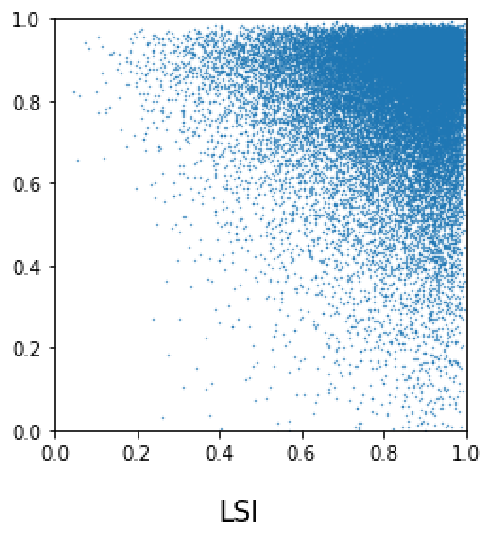
Figure 23. Plot (a) shows user similarity scores distribution based on LSI with 20 topics, (b) shows the same for 10 and (c) for 5 topics.

As we can see in fig. 24, decreasing number of topics shifts the similarity scores to the right meaning that users are more similar now due to lower resolution, all users are projected onto one of 5-10-20 number of topics, therefore, the lower the dimension (number of topics), the closer they get to each other.

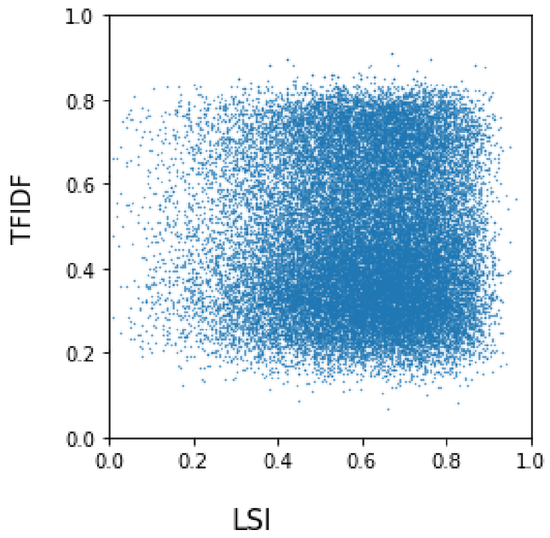
I've computed the user similarity by LSI topics and checked the correlation between this similarity metric and other similarity metrics previously introduced (TF-IDF, Word2Vec, Doc2Vec). The clear conclusion here is that there is no significant correlation between user-user similarity computed by LSI and any of the other methods. This is unexpected because Latent Semantic Indexing is designed to be sensitive to the semantic content of the documents and so it was expected that similarities computed by LSI would have a reasonable overlap with similarities computed by other distributional techniques, such as Word2Vec and Doc2Vec.



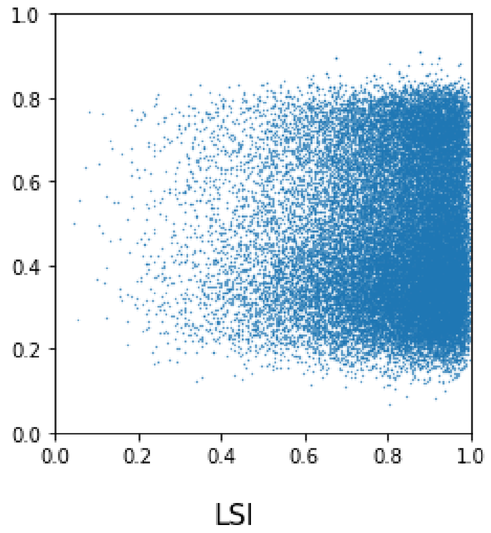
(a - LSI-40)



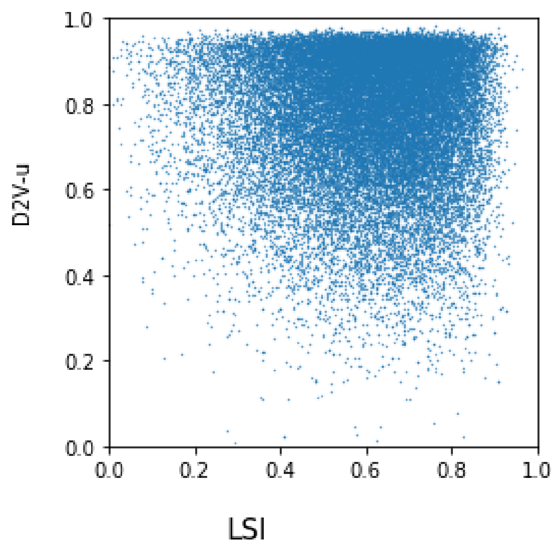
(b - LSI-5)



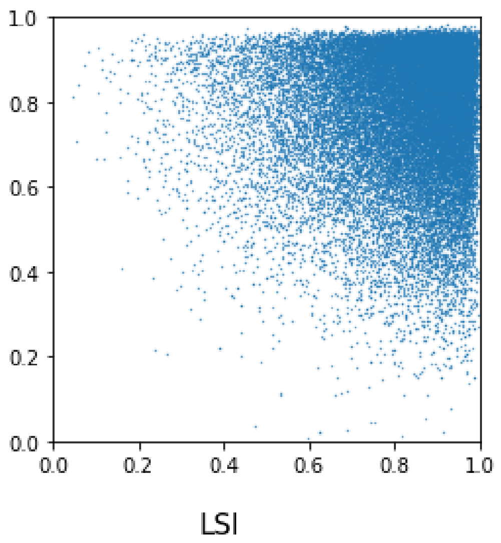
(c - LSI-40)



(d - LSI-5)



(e - LSI-40)



(f - LSI-5)

Figure 24. Comparison of distributions of user similarities for LSI vs some benchmark W2V and D2V models and TF-IDF in each of the above plot, (a) showing LSI-40 - W2V-r, (b) showing LSI-5 - W2V-r, (c) showing LSI-40 - TF-IDF, (d) showing LSI-5 - TF-IDF, (e) showing LSI-40 - D2V-u and (f) showing LSI-5 - D2V-u

Evaluation of document vectors:

t-SNE projection of Doc2Vec models beer style similarity

To analyze how Doc2Vec models reflect different beer categories and determine its accuracy based on how well it distinguishes different beer categories, I've attempted to visualise the Doc2Vec vectors. Since the vectors are high-dimensional, I used t-Distributed Stochastic Neighbor Embedding (t-SNE) for dimensionality reduction to project the multi-dimensional D2V vectors into 2-D space and picked up the top 5 beers (for ease of visualisation and observation of separation), t-SNE and bokeh plots produced the following result in fig. 25:

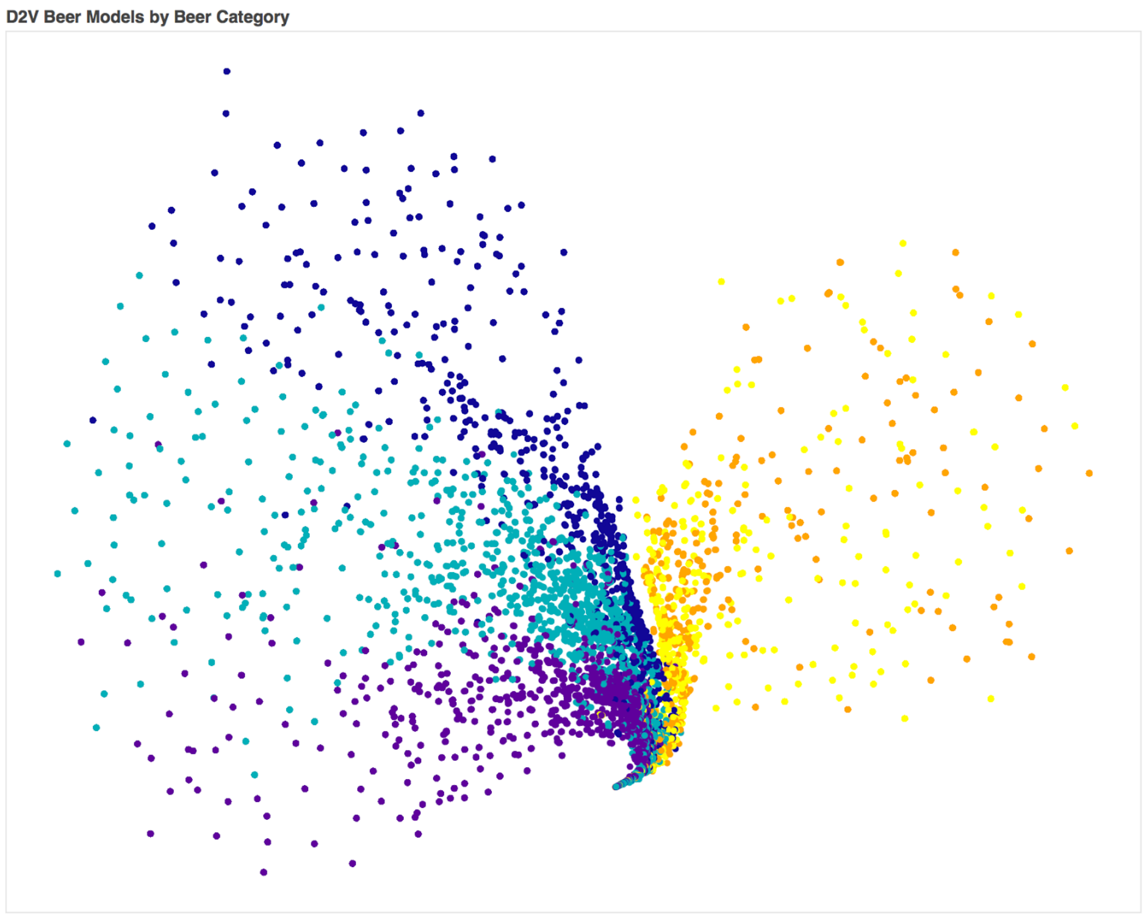


Figure 25. t-SNE bokeh plot based on D2V vectors (tagged with beer_ids) using the top 5 beer categories in the data set.

Each point in this plot represents a review, and the colors correspond to the top-5 beer categories.

u'american ipa': Turquoise
u'american double/imperial ipa': Blue(navy)
u'american pale ale (apa)': Purple
u'american double/imperial stout': Brown
u'russian imperial stout': Yellow (light)

having the following number of reviews for each, correspondingly: 113124, 85091, 58059, 53406, 50130.

The t-SNE plot shows that the word vectors for different categories are well separated into distinguishable clusters. An interesting case is the categories (american double/imperial stout: Yellow) and (russian imperial stout: Brown) which show a strong degree of overlap, indicating similar language used in the reviews of beers in those categories. They are also both stouts, therefore they are most likely talked about in a similar way, therefore the overlap that we see is reasonable.

5 Conclusion

Various word embedding techniques were examined for the purposes of building enhanced user and item profiles. The user and item profiles constructed from word vectors learned on reviews should represent the semantic content of the reviews in a more meaningful way. As an initial evaluation, the similarities of user and item profiles constructed using traditional information retrieval and NLP techniques (TF-IDF, LSI) were tested against combined word vector models (Word2Vec, Doc2Vec).

The results indicate that different word vector models capture different semantic and syntactic properties (ie. there isn't a very strong correlation between the word2vec and doc2vec models). Also, the parameters of the model are important too and difficult to evaluate independently. In addition, there isn't a strong correlation between TF-IDF and LSI similarity techniques and word vector techniques. This is somewhat surprising, since LSI is designed to capture semantic content in the reviews, and while we would expect it to pick up on different semantic content to the word vector models, we expect there should be a better correlation between the two. In short, users and items which are found to be similar under TF-IDF or LSI, are not necessarily similar by word vector models.

6 References

Barkan, O., & Koenigstein, N. (2016, September). Item2Vec: neural item embedding for collaborative filtering. In Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on (pp. 1-6). IEEE.

Baroni, M., Dinu, G., & Kruszewski, G. (2014, June). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In ACL (1) (pp. 238-247).

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning research, 3(Feb), 1137-1155.

Bhingardive, S., Singh, D., Murthy, R., Redkar, H., & Bhattacharyya, P. (2015). Unsupervised most frequent sense detection using word embeddings. In DENVER.

De Boom, C., Van Canneyt, S., Bohez, S., Demeester, T., & Dhoedt, B. (2015, November). Learning semantic similarity for very short texts. In Data Mining Workshop (ICDMW), 2015 IEEE International Conference on (pp. 1229-1234). IEEE.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. Journal of the American society for information science, 41(6), 391.

Diaz, F., Mitra, B., & Craswell, N. (2016). Query expansion with locally-trained word embeddings. arXiv preprint arXiv:1605.07891.

Dickinson, B., & Hu, W. (2015). Sentiment analysis of investor opinions on twitter. Social Networking, 4(03), 62.

Harris, Z. S. (1954). Distributional structure. Word, 10(2-3), 146-162.

Heidarian, A., & Dinneen, M. J. (2016, March). A Hybrid Geometric Approach for Measuring Similarity Level Among Documents and Document Clustering. In Big Data Computing Service and Applications (BigDataService), 2016 IEEE Second International Conference on (pp. 142-151). IEEE.

Hinton, G. E. (1986, August). Learning distributed representations of concepts. In Proceedings of the eighth annual conference of the cognitive science society (Vol. 1, p. 12).

Jiang, R., Liu, Y., & Xu, K. A General Framework For Text Semantic Analysis And Clustering On Yelp Reviews.

Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

Kusner, M., Sun, Y., Kolkin, N., & Weinberger, K. (2015, June). From word embeddings to document distances. In International Conference on Machine Learning (pp. 957-966).

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural architectures for named entity recognition. arXiv preprint arXiv:1603.01360.

Lau, J. H., & Baldwin, T. (2016). An empirical evaluation of Doc2Vec with practical insights into document embedding generation. arXiv preprint arXiv:1607.05368.

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In Proceedings of the 31st International Conference on Machine Learning (ICML-14) (pp. 1188-1196).

Levy, O., Goldberg, Y., & Ramat-Gan, I. (2014, June). Linguistic Regularities in Sparse and Explicit Word Representations. In CoNLL (pp. 171-180).

Ma, L., & Zhang, Y. (2015, October). Using Word2Vec to process big text data. In Big Data (Big Data), 2015 IEEE International Conference on (pp. 2895-2897). IEEE.

Ma, X., & Hovy, E. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. arXiv preprint arXiv:1603.01354.

Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of Machine Learning Research, 9(Nov), 2579-2605.

McAuley, J., & Leskovec, J. (2013, October). Hidden factors and hidden topics: understanding rating dimensions with review text. In Proceedings of the 7th ACM conference on Recommender systems (pp. 165-172). ACM.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting similarities among languages for machine translation. arXiv preprint arXiv:1309.4168.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

Minarro-Giménez, J. A., Marín-Alonso, O., & Samwald, M. (2015). Applying deep learning techniques on medical corpora from the world wide web: a prototypical system and evaluation. arXiv preprint arXiv:1502.03682.

Mohr G. (2015, October 16th), wor2vec.train errors: 'Word2Vec' object has no attributes 1) 'corpus_count' 2) 'syn0_lockf', Retrieved from https://groups.google.com/forum/#!msg/gensim/icSyNtCJsw8/vW8rf52bCgAJ

Musto, C., Semeraro, G., De Gemmis, M., & Lops, P. (2015). Word Embedding Techniques for Content-based Recommender Systems: An Empirical Evaluation. In RecSys Posters.

Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In EMNLP (Vol. 14, pp. 1532-1543).

Phi, V. T., Chen, L., & Hirate, Y. (2016). Distributed representation based recommender systems in e-commerce. In DEIM Forum.

Pouransari, H., & Ghili, S. (2014). Deep learning for sentiment analysis of movie reviews. Technical report, Stanford University.

Ramos, J. (2003, December). Using TF-IDF to determine word relevance in document queries. In Proceedings of the first instructional conference on machine learning (Vol. 242, pp. 133-142).

Rong, X. (2014). Word2Vec parameter learning explained. arXiv preprint arXiv:1411.2738.

Schnabel, T., Labutov, I., Mimno, D. M., & Joachims, T. (2015, September). Evaluation methods for unsupervised word embeddings. In EMNLP (pp. 298-307).

Shi, T., & Liu, Z. (2014). Linking GloVe with Word2Vec. arXiv preprint arXiv:1411.5595.

Shirani-Mehr, H. (2014). Applications of deep learning to sentiment analysis of movie reviews. Technical report, Stanford University.

Sienčnik, S. K. (2015, May). Adapting Word2Vec to named entity recognition. In Proceedings of the 20th Nordic Conference of Computational Linguistics, NODALIDA 2015, May 11-13, 2015, Vilnius, Lithuania (No. 109, pp. 239-243). Linköping University Electronic Press.

Tang, D., Wei, F., Yang, N., Zhou, M., Liu, T., & Qin, B. (2014, June). Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. In ACL (1) (pp. 1555-1565).

Trask, A., Michalak, P., & Liu, J. (2015). sense2vec-A fast and accurate method for word sense disambiguation in neural word embeddings. arXiv preprint arXiv:1511.06388.

Tsvetkov, Y., Faruqui, M., Ling, W., Lample, G., & Dyer, C. (2015). Evaluation of word vector representations by subspace alignment

Turney, P. D., & Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. Journal of artificial intelligence research, 37, 141-188.

Vector Representations of Words. (2017, June 19), Retrieved from https://www.tensorflow.org/tutorials/Word2Vec

Wiemer-Hastings, P., Wiemer-Hastings, K., & Graesser, A. (2004, November). Latent semantic analysis. In Proceedings of the 16th international joint conference on Artificial intelligence (pp. 1-14).

Word2Vec. (2013, July 30), Retrieved from https://code.google.com/archive/p/Word2Vec/

Wu, Y., Xu, J., Zhang, Y., & Xu, H. (2015, July). Clinical abbreviation disambiguation using neural word embeddings. In Proceedings of the 2015 Workshop on Biomedical Natural Language Processing (BioNLP) (pp. 171-176).

Xing Margaret, F. U., & Xiaocheng, L. I. (2015). From Movie Reviews to Restaurants Recommendation.

7 Appendices

1. Word2Vec analogies evaluation full table

Category / Dimension	Dimension ->	10	50	100	300	500	1000
capital-common-countries	Correct	0	16	18	27	29	34
	Incorrect	156	140	138	129	127	122
capital-world	Correct	0	15	25	48	48	50
	Incorrect	214	199	189	166	166	164
currency	Correct	0	0	1	3	2	1
	Incorrect	52	52	51	49	50	51
city-in-state	Correct	0	13	33	57	60	63
	Incorrect	514	501	481	457	454	451
family	Correct	0	7	14	20	19	17
	Incorrect	110	103	96	90	91	93
gram1-adjective-to-adverb	Correct	0	0	0	0	0	0
	Incorrect	0	0	0	0	0	0
gram2-opposite	Correct	0	1	0	0	0	0
	Incorrect	12	11	12	12	12	12
gram3-comparative	Correct	15	274	369	420	413	424
	Incorrect	1041	782	687	636	643	632
gram4-superlative	Correct	7	126	168	174	174	157
	Incorrect	805	686	644	638	638	655
gram5-present-participle	Correct	0	0	0	0	0	0
	Incorrect	0	0	0	0	0	0
gram6-nationality-adjective	Correct	2	15	35	64	70	73
	Incorrect	536	523	503	474	468	465
gram7-past-tense	Correct	0	0	0	0	0	0
	Incorrect	0	0	0	0	0	0
gram8-plural	Correct	0	1	3	5	6	6
	Incorrect	6	5	3	1	0	0
gram9-plural-verbs	Correct	0	0	0	0	0	0
	Incorrect	0	0	0	0	0	0
total	Correct	24	468	666	818	821	825
	Incorrect	3446	3002	2804	2652	2649	2645

2. LSI with 10 and 20 topics respectively

LSI topics-10:

```
[(0,u'0.31*"beer" + 0.22*"hop" + 0.22*"malt" + 0.21*"head" + 0.19*"flavor"'),
 (1,u'-0.78*"beer" + 0.23*"nice" + 0.16*"bodi" + 0.15*"light" + 0.15*"pour"'),
 (2,u'-0.30*"nice" + -0.26*"one" + 0.24*"malt" + -0.23*"tast" + 0.22*"note"'),
 (3,u'0.36*"tast" + -0.31*"nice" + -0.28*"note" + 0.23*"smell" + -0.23*"realli"'),
 (4,u'0.33*"beer" + 0.33*"nice" + 0.23*"pour" + 0.22*"color" + -0.21*"ale"'),
 (5,u'-0.61*"flavor" + 0.34*"note" + 0.24*"tast" + 0.19*"quit" + -0.18*"aroma"'),
 (6,u'0.34*"flavor" + 0.34*"smell" + 0.33*"tast" + -0.22*"bottl" + -0.22*"lace"'),
 (7,u'0.35*"aroma" + -0.32*"flavor" + 0.27*"malt" + -0.26*"smell" + -0.24*"bit"'),
 (8,u'-0.38*"aroma" + 0.28*"malt" + -0.27*"bottl" + -0.26*"glass" +
-0.23*"mouthfeel"'),
 (9,u'-0.50*"nice" + -0.27*"bodi" + 0.25*"light" + 0.25*"pour" + 0.22*"good"')]
```

LSI topics-20:

```
[(0, u'0.31*"beer" + 0.22*"hop" + 0.22*"malt" + 0.21*"head" + 0.19*"flavor"'),
(1, u'-0.78*"beer" + 0.23*"nice" + 0.16*"bodi" + 0.15*"light" + 0.15*"pour"'),
(2, u'-0.30*"nice" + -0.26*"one" + 0.24*"m1alt" + -0.23*"tast" + 0.22*"note"'),
(3, u'0.36*"tast" + -0.31*"nice" + -0.28*"note" + 0.23*"smell" + -0.23*"realli"'),
(4, u'0.33*"beer" + 0.33*"nice" + 0.23*"pour" + 0.22*"color" + -0.21*"ale"'),
(5, u'-0.61*"flavor" + 0.34*"note" + 0.24*"tast" + 0.19*"quit" + -0.18*"aroma"'),
(6, u'0.34*"flavor" + 0.34*"smell" + 0.33*"tast" + -0.22*"bottl" + -0.22*"lace"'),
(7, u'0.35*"aroma" + -0.32*"flavor" + 0.27*"malt" + -0.26*"smell" + -0.24*"bit"'),
(8, u'-0.38*"aroma" + 0.28*"malt" + -0.27*"bottl" + -0.26*"glass" +
-0.23*"mouthfeel"'),
(9, u'-0.50*"nice" + -0.27*"bodi" + 0.25*"light" + 0.25*"pour" + 0.22*"good"'),
(10, u'0.35*"light" + -0.28*"bit" + 0.27*"glass" + -0.21*"nose" + -0.19*"good"'),
(11, u'-0.40*"aroma" + -0.27*"good" + 0.23*"color" + 0.23*"pour" + 0.23*"sweet"'),
(12, u'0.30*"color" + -0.28*"bit" + 0.26*"note" + 0.25*"nose" + -0.22*"sweet"'),
(13, u'0.41*"good" + 0.21*"lace" + -0.20*"aroma" + -0.19*"littl" + -0.18*"like"'),
(14, u'0.35*"light" + 0.29*"nose" + 0.24*"pour" + 0.20*"mouthfeel" +
-0.19*"note"'),
(15, u'0.36*"light" + 0.25*"tast" + -0.25*"hop" + -0.24*"malt" + 0.21*"nose"'),
(16, u'-0.43*"bit" + -0.30*"malt" + 0.23*"head" + -0.22*"mouthfeel" +
0.18*"lace"'),
(17, u'-0.26*"brew" + -0.25*"bottl" + -0.25*"flavor" + 0.24*"mouthfeel" +
-0.21*"carbon"'),
(18, u'0.29*"note" + 0.27*"good" + -0.270*"color" + 0.20*"bottl" + -0.18*"head"'),
(19, u'-0.26*"carbon" + 0.24*"brew" + 0.24*"light" + -0.22*"pour" +
-0.19*"flavour"')]
```

3. iPython notebook and source code (PDF attached)

The iPython notebook used in this work is available at this github repo: https://github.com/cagatayoz/cagatay-thesis/blob/master/cagatay-thesis-notebook.ipynb
and the iPython source code is also attached at the end of this report.