Qualitative Evaluation of Word2Vec for Recommendations

Siddha Ganju
School of Computer Science
Carnegie Mellon University
sganju1@cs.cmu.edu

Sravya Popuri
School of Computer Science
Carnegie Mellon University
spopuri@cs.cmu.edu

Srikant R. Avasarala
Information Networking Institute
Carnegie Mellon University
savasara@andrew.cmu.edu

ABSTRACT

There exists a certain amount of difference between which category a user would put a business into and what the user was suggested by a commercial recommender system. This research is oriented to understand users better and the reasons *'why'* one movie was given a *'5 star rating'* but other movies were not. This is done using not only the *'star rating'* but also learn user sentiments behind those *'star ratings'* by using models like Word2Vec. Hence, to build a system that improves recommendations provided to users through commercial recommending systems by taking into consideration additional metrics and inputs taken from Word2Vec. To do this we analyse user reviews of a movie and produce word vectors that embed the users' taste. We generate such vectors for all the movies rated by a user and generate a personalised user vector that qualifies as the users' taste. Now we can use these vectors in addition to the rating vectors to find similar users. To the best of our knowledge, this is the first implementation of using Word2Vec to generate recommendations for business using the text reviews.

Keywords

Word2Vec, Recommendation Systems, KNN, Logistic Regression, Spark, Pig, Natural Language Processing

1. INTRODUCTION

The difference between which category a user would put a business, such as a movie or a restaurant into and what the user was suggested by a commercial recommender system, like the ones used by Netflix, Amazon, or Yelp is because these systems learn a user's taste through similarity matching with another user. Initially, users give review about an item they used or a movie they watched or a restaurant they ate at. These reviews describes their experiences and how they liked or disliked that business. The processed text review is of the format,
"goldberg offers everything I look for in a general practitioner. he's nice and easy to talk to without being patronizing; he's always on time in seeing his patients"
is important to note that other users can read this recommendation and gain an outlook, as to what other people think about that business, but it is essential to understand that the recommender systems might not fully assimilate that review like a human user does. So, the algorithm is limited in its learning about why a user rated a commercially unsuccessful movie as *'5 stars'* or why a user gave a Thai restaurant *'2.5 stars'*, even though the user loves Thai food. This research project aims to understand *'why'* the numerical value of star rating is as such, and does this by interpreting the user's review text using Word2Vec. We focus on whether this understanding gives a better insight into the user's tastes resulting in better learning by the recommendation systems.

2. BACKGROUND

Many of the current recommendation systems use either Collaborative Filtering (CF) or content based algorithms to make predictions about movies or businesses that interest a user or predict rating given by a user to other movies or businesses the user has not seen. These methods relate two users based on the ratings they have given to movies. For example, consider two users $\{u1, u2\}$ and two movies $\{m1, m2\}$ and their corresponding ratings as $\{(u1,m1,4), (u1,m2,5)\}$, $\{(u2,m1,5), (u2,m2,4)\}$. Now if u1 rates a new movie say, *'m3'* as *'5'*, CF based recommendation system recommends it to u2 as they both have similar tastes with respect to movies *'m1'* and *'m2'* without considering the reasons why $\{u1, u2\}$ liked movies $\{m1, m2\}$.

2.1 Current Recommendation Models Overview

2.1.1 Content based recommendations

Content based filtering methods utilize the description of the item and compare it with the user's preference. There are certain keywords which are used to describe the items, which are present in the items description. Apart from these words, the user's profile has information about the type of items they like. Hence, content based recommendations recommend those items that are similar to the ones a user liked in the past or items that are similar to the item the user is examining in the present. Hence, content based algorithms recommend items similar to what users have already liked. For example, playing a *'Megadeth'* song after a *'Metallica'* song.

2.1.2 Collaborative Filtering

Collaborative filtering assumes that if *'user1'* has the same opinion as *'user2'* on an issue, then on a different issue, it is likely that *'user1'* and *'user2'* will have similar views, rather than *'user3'*, who is a random selection.
Using this underlying assumption, collaborative filtering can be used to make predictions about any item, it could be a movie, a television show, or a restaurant, when provided with a list of user's tastes, which includes both likes and dislikes. Hence, in collaborative filtering similar users are found based on some similarity metric like, cosine similarity or Pearson correlation coefficient.

2.2 Limitations of Current Recommendation Models

The current recommendation models suffer from the following to a varying degree:
Similar Users: Users who give similar star ratings are considered as similar users, why a user likes a movie/item is not taken into consideration. As an example, aperson who likes Slumdog Millionaire and Avatar may not like Titanic, similarly a person may like Chinese and Korean food yet dislike spicy food.
Cold Start Problem: A new user makes an account for the first time on a commercial recommender system and it fails to suggest items. This can happen both for a new user or a user who rated very few other items, because there is no similarity that exists between this user and other users using the recommender system. In order to find similar users, there needs to be a lot of rated movies or businesses based on which similarity can be calculated and recommendations be given. We try to tackle the cold start problem by asking the first time user to give preferences regarding what kinds of restaurants or businesses the user likes. Based on this initial liking, other similar items can be recommended.
Shilling Attacks: People may give lots of positive ratings for their own items and negative ratings for their competitors. Example: A movie producer might want his movie to do well and so might pay people to give good reviews, even when essentially, these people might not agree with the review they are writing.
Gray Sheep: The algorithm fails to suggest items to people who disagree with the society, because they have less similar users. Example: A war veteran who is also a user might like movies which are based on the Vietnam War, so he might like movies such as, 'The Deer Hunter' or 'The Veteran'. The user rates such movies very high, but other war based movies not as high, so, he is dissimilar from users who generally rate all war movies high. The war veteran user does not have many similar users and hence, will not get recommendations.

3. LITERATURE REVIEW

Word2Vec uses (LSTMs) Long Short Term Memory networks [1]. They come within the category of Recurrent Neural Networks(RNN), and are capable of learning long-term dependencies. They were introduced by Hochreiter (1991)[1] and Bengio, et al. (1994)[2]. LSTMs are explicitly designed to avoid the long-term dependency problem, so their major advantage is that they are able to remember information for long periods of time. There is a lot of variation on the type of LSTM that can be used. Some types include, Depth Gated RNNs by Yao, et al. (2015)[3], and Koutnik, et al. (2014) [4].
Greff, et al. (2015)[5] reports work on comparison on some variants and reach the conclusion that the basic methodology followed is similar. Work by Jozefowicz, et al. (2015)[6] includes testing more than ten thousand RNN architectures, and report that some RNNs are better suited to some tasks, because of which, they give better results than LSTMs on certain tasks. Hence, the choice of which LSTM to use is largely based on the task at hand.
Mikolov[7] recommends using the skip-gram model with negative sampling (SGNS), as it outperformed the other variants on analogy tasks.
In [8] the authors discuss the usage of unsupervised vector approaches to model rich semantic meanings. They use Word2Vec as to model the semantic meanings and this paper serves as the main movtivation for this research.

4. DATA

This section covers the input training data and the preprocessing performed on it that converts it to a consumable form for the learning model. For this project, two different input training datasets, namely, a Yelp dataset and Amazon dataset were used:

4.1 Yelp Dataset

It contains JSON objects of the following format
{
    'type': 'review',
    'review_id': (encrypted review id),
    'business_id': (encrypted business id),
    'user_id': (encrypted user id),
    'stars': (star rating),
    'text': (review text),
    'date': (date, formatted like '2012-03-14'),
    'votes': {(vote type): (count)}
}

4.2 Amazon Dataset

It contains JSON objects of the following format
{
    'product/product_id': amazon.com/dp/B00006HAXW',
    'review_iduser_id': 'A1RSDE90N6RSZF',
    'business_id': (encrypted business id),
    'review/profileName': (name),
    'review/helpfulness': (number of users who found the review helpful),
    'review/score': (rating),
    'review/time': (unix time),
    'review/summary': (review summary),
    'review/text': (text of the review)
}

4.3 Preprocessing

The preprocessing done for the Yelp dataset is as under. The framework and steps for Amazon dataset largely remain the same differing only in the fields that are output. As mentioned before, the initial training data set is in form of JSON objects. The initial training data for each of the data sets is modeled as tuples of following form after some initial processing in Python:
⟨*type*⟩ ⟨*review_id*⟩ ⟨*business_id*⟩ ⟨*user_id*⟩ ⟨*stars*⟩ ⟨*review_text*⟩ ⟨*date*⟩ ⟨*votes*⟩

Table 1: yelp_table

User Id	Review Id	Business Id	Rating
qwdHuhJ	*kdSfDjFwGkn*	*sduAhShejnqm*	*3*
ajkdhHkjAhd	*akjdhSfUiAdfa*	*aDfIuaYdfuAufy*	*1*

Table 2: reviews_table

Review Id	Review Text
kdSfDjFwGkn	*this restaurant has a good ambience.*
akjdhSfUiAdfa	*Very dissatisfied with the service, dont go*

(1)

The final requirement is to transform the input dataset to the following form for each training instance:

$$\{ \langle rating \rangle \ \langle user\_id \rangle \ \langle business\_id \rangle \ \langle review\_vector \rangle \ \} \quad (2)$$

To perform this data transformation, dataflow programming paradigm was selected as it is capable of achieving parallelization of computation without introducing development complexity, resulting in an increased performance of applications built with it when using multi-core computers [9]. Pig [10] was chosen as the dataflow language for development. Pig allows relational operations on data sets and is especially suited for large datasets. Pig operations spawn map reduce tasks thus achieving parallelism and faster processing of data.
However, following points should be kept in mind as they affect performance and resource utilization of the nodes in Pig cluster. Firstly, training data including any other data resources being utilized must be modelled as proper relations. Secondly, keeping sizes of current training data set (1.2 Gb) and Word2Vec mappings set (5.2 Gb) in mind, any Pig operations involving joins between these two data sets will generate many temporary files depending on number of operations in the joins and after joins. All these files will be of comparable sizes (around 10 Gb in this case). In addition, the final views saved to the disk will also be of similar size. Thirdly, these operations will also take up a lot of input-output operations on the disk or will take up a lot of memory in case proper normalization of relations is not done. If these points are not rightly taken care of, Pig as a data flow language will be ineffective. In this case, basic algorithms too will require heavy compute enabled machinery which can otherwise be achieved with smaller instances. Furthermore, these algorithms will not be able to scale to larger datasets and break down.
Proper relational modelling of the input training data implies lossless decomposition of the relation as given in 1 and preserves functional dependencies too. To ensure these conditions, the input training data was modelled into three relations as below:
The idea is to read in the input data sets into these relations. Following this, a series of operations are performed on the input data set to transform the input data set into the format given in 2. These operations are detailed in the following sections. The data cleaning was performed in the following three different ways:

- Naive Manner
- Apache Spark
- Apache Pig, Guinea Pig[1]

We experimented with these approaches in order to compare the time, machine utilization and ease of working with the complete pipeline. The operations performed using Pig is detailed here. The algorithm and relations used in Spark largely remain the same except syntactical changes which are also described in the below sections. These operations remain same for both, the Yelp and Amazon dataset.

Table 3: word2vec_table

Words	Vector Representation
offers	*{[1.223, -.023,]}*
everything	*{[0.342, 1.343,]}*

4.4 Pig

The following algorithm outline the data preprocessing performed in Pig.
Step 1: Read the input Word2Vec library and then generated the *word2vec_table*.
Step 2: First, read the input training Yelp file, then extracted the following fields from the training set; *user_id, review_id, business_id, review_text* and *rating.* Then, tokenized the review_text field. Finally, remove all the punctuations and numbers in the tokens. Figure 1 explains the processing. The resulting fields are read into *raw_yelp_table* view. From this view, we generated *yelp_table* and *reviews_table* in the following steps.
Step 3: From the *raw_yelp_table*, perform map using custom lambda function to get tuples in following form of:
$\langle review\_id1 \rangle \ \langle review\_text1 \rangle$
$\langle review\_id2 \rangle \ \langle review\_text2 \rangle$
......
Step 4: From the *raw_yelp_table*, map using custom lambda function to get tuples in following form of *yelp_table* -
$\langle user\_id1 \rangle \ \langle review\_id1 \rangle \ \langle business\_id1 \rangle \ \langle rating1 \rangle$
$\langle user\_id2 \rangle \ \langle review\_id2 \rangle \ \langle business\_id2 \rangle \ \langle rating2 \rangle$
...
Step 5: Flatten *reviews_table* to obtain for each review the following tuples:
$\langle review\_id1 \rangle \ \langle review\_text\_word1 \rangle$
$\langle review\_id1 \rangle \ \langle review\_text\_word2 \rangle$
$\langle review\_id1 \rangle \ \langle review\_text\_word3 \rangle$
$\langle review\_id1 \rangle \ \langle review\_text\_word4 \rangle$
$\langle review\_id1 \rangle \ \langle review\_text\_word5 \rangle$
...
...
$\langle review\_id2 \rangle \ \langle review\_text\_word1 \rangle$
$\langle review\_id2 \rangle \ \langle review\_text\_word2 \rangle$
...
...
$\langle review\_idn \rangle \ \langle review\_text\_word1 \rangle$
$\langle review\_idn \rangle \ \langle review\_text\_word2 \rangle$

Step 6: Join *reviews_table* in its current form with *word2vec_table* and obtain:
$\langle review\_id1 \rangle \ \langle review\_text\_word1\_vector \rangle$
$\langle review\_id1 \rangle \ \langle review\_text\_word2\_vector \rangle$

[1]http://curtis.ml.cmu.edu/w/courses/index.php/Guinea_Pig

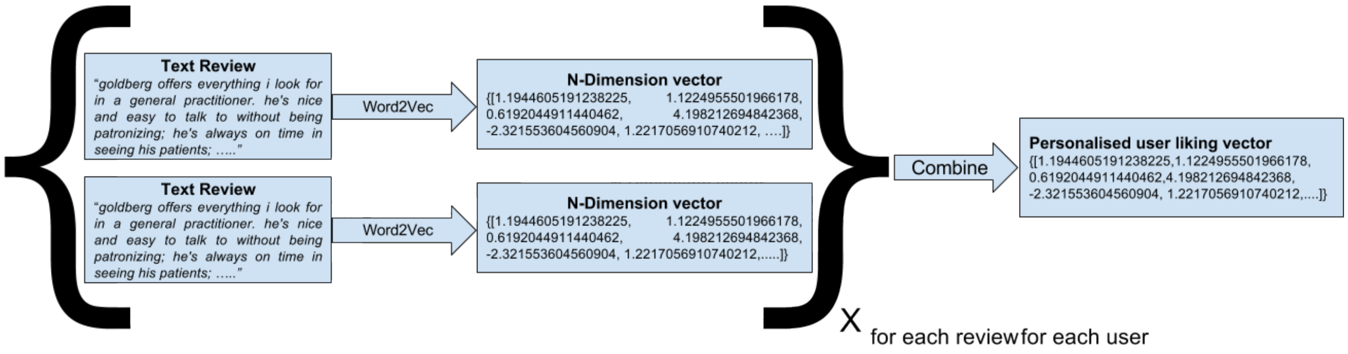


Figure 1: Word2Vec Processing

$\langle review\_id1\rangle$ $\langle review\_text\_word3\_vector\rangle$
$\langle review\_id1\rangle$ $\langle review\_text\_word4\_vector\rangle$
$\langle review\_id1\rangle$ $\langle review\_text\_word5\_vector\rangle$
...
...
$\langle review\_id2\rangle$ $\langle review\_text\_word1\_vector\rangle$
$\langle review\_id2\rangle$ $\langle review\_text\_word2\_vector\rangle$
...
...
$\langle review\_idn\rangle$ $\langle review\_text\_word1\_vector\rangle$
$\langle review\_idn\rangle$ $\langle review\_text\_word2\_vector\rangle$
...
...
Now, group by the *review_id* field and sum all of the vectors to obtain the vector representation of every review.
Step 7: Finally generate the final form by performing *join* on *reviews_table* in its current form with *yelp_table* to get the corresponding users for every review
$\langle rating1\rangle$ $\langle user\_id1\rangle$ $\langle business\_id1\rangle$ $\langle review\_vector1\rangle$
$\langle rating2\rangle$ $\langle user\_id1\rangle$ $\langle business\_id2\rangle$ $\langle review\_vector2\rangle$
$\langle rating3\rangle$ $\langle user\_id1\rangle$ $\langle business\_id3\rangle$ $\langle review\_vector3\rangle$
...
...
$\langle rating1\rangle$ $\langle user\_idn\rangle$ $\langle business\_id1\rangle$ $\langle review\_vector1\rangle$
$\langle rating2\rangle$ $\langle user\_idn\rangle$ $\langle business\_id2\rangle$ $\langle review\_vector2\rangle$
...
...

4.5 Naive Approach

For the naive approach, data framework provided by the Word2Vec library was used to perform the transformations. The following detail the steps taken:
Step 1: Read in the Word2Vec file into the data frame provided by the Word2Vec library.
Step 2: After reading the file, we process it to obtain the desired format. Figure 2 shows the pipeline for obtaining the text fields and finally processing the review. The dataset read as streaming input and the following fields are extracted, *user_id, review_id, business_id, review_text* and *rating.* Each review is tokenized which forms the *review_text* field. Then, text processing is done, following the conversion to lowercase, removal of punctuation and numbers, finally leading to the processed text review. For every word in the *review_text*, looked up the vector representation from the loaded data frame and accumulated it. This accumulated vector represents the *review_text.* Wrote the following fields to output
$\langle rating1\rangle$ $\langle user\_id1\rangle$ $\langle business\_id1\rangle$ $\langle review\_vector1\rangle$
$\langle rating2\rangle$ $\langle user\_id1\rangle$ $\langle business\_id2\rangle$ $\langle review\_vector2\rangle$
$\langle rating3\rangle$ $\langle user\_id1\rangle$ $\langle business\_id3\rangle$ $\langle review\_vector3\rangle$
...
...
$\langle rating1\rangle$ $\langle user\_idn\rangle$ $\langle business\_id1\rangle$ $\langle review\_vector1\rangle$
$\langle rating2\rangle$ $\langle user\_idn\rangle$ $\langle business\_id2\rangle$ $\langle review\_vector2\rangle$
...
...
Thus obtaining the same data in processed format for consumption by the learning model.

4.6 Word2Vec

In this section, a brief overview of Word2Vec [7] model is explained. Word2Vec is a class of neural network models that, given an unlabelled training corpus, produce a vector for each word in the corpus that encodes its semantic information. Word2Vec offers an advanced vector representations of a term, along with other distributed representations. These are based on the distributional hypothesis, which motivates that the meaning of a word can be gauged by its context. Hence, if two words occur in the same "position" in two sentences, they are very much related either in semantics or syntactics. We can measure the semantic similarity between two words by calculating the cosine similarity between their corresponding word vectors. It captures latent sentiments in word embeddings and gives a vector representation. Pennington, Socher, and Manning [11] shows that the Word2Vec model is equivalent to weighting a word co-occurrence matrix weighting based on window distance and lowering the dimension by matrix factorization. This is done by making a continuous bag of words (CBOW) and Skip-grams. In CBOW, the goal is to predict a word given the surrounding words while in Skip-grams, a window of the surrounding words is predicted given a single word. The word vectors thus generated can be fed now capture the context of the surrounding word.

4.6.1 *Word2Vec Resources Used*

In this section, the various Word2Vec resources used for this project are outlined. They are as below:
Stanford Glove: [11] It is a 5.7 Gb GloVe data set containing 2.2 million words, which are translated to 840 billion tokens and each is represented as a 300-dimensional vectors.
Google Word2Vec: This is a 3.7 Gb pre-trained vectors trained on part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3

million words and phrases.
Gensim: [12] Gensim is a library that realizes unsupervised semantic modelling from plain text.

4.6.2 *Word2Vec Usage In Project*

In this section, a detailed description of usage of Word2Vec model in the project is given. As shown in the Figure 1, the input to the Word2Vec module is the text review written by a user and the final output is a n-dimensional vector representing the user's taste.

5. APPROACH

In this section, the approach taken for this research project is outlined in form of steps.
Step I: Data Collection
The data is collected from the Yelp datasets and the Amazon datasets. All the work in this phase has already been detailed in the Pig section and are summarized in Figure 2.
Step II: Data Processing
Each review vector is processed using Word2Vec to generate a word embedding vector which inherently represents the users' opinion of that particular movie.
Step III: Personalized vectors
Each review written by a user for every movie that the user has watched has a semantic representation. In order to generalize this and to understand the users' tastes, a new vector which contains the average of all the vectors for that user is created.
Step IV: Similar users
All similar users to a given user are found using rating vectors and personalized vectors which were obtained through averaging all the movie vectors. Similar users are found by forming clusters using k-nearest-neighbours.
Step IV: Model Training
There are two approaches that we followed for model training. First, train one logistic regression model for each user and second, train one model for all users. For testing, there are two approaches. First, to test against the actual rating that the user has given to that movie, and second, to recommend movies and see if the user had actually seen them. Both the approaches are described as under.
We train a logistic regression model for each user so that each model is able to learn one user. However, we notice that there are some discrepancies with this model. Firstly, because the data about one user is limited, it is possible that the model learns the false values which are not the true representation of the users' tastes. Secondly, having information of just one user may lead to high error values, which were actually being observed in our experiments. The second approach was to train one model for all the users, which gave better results.
Step V: Recommending Movie
After finding all the similar users, we majority vote all the movies liked by some threshold number of the similar users. To calculate the threshold value, we majority vote for a range of values and then get the threshold value that gives the best cross validation accuracy. The similar users are selected using the k-nearest-neighbours. From the list of the similar users, generate the movies which are rated high (3 stars or higher) and recommend them to the user. It is important to note that because this user is similar to the other users, the given users rating and review vector will turn out to be similar. The similarity can be measured with cosine similarity or pearson correlation coefficient to compare if the user actually liked the movie or now.

Table 4: AWS Instances

Jobs	SPARK	Standalone	PIG
Cluster Type	*m3.xlarge*	*m4.4xlarge*	*m3.xlarge*
vCPU	*4*	*16*	*4*
RAM	15	64	15
SSD (GB)	2 x 40	150	2 x 40

5.1 Experimental Setup

We used AWS instances because of their computational power and the ability to store large datasets, in comparison to personal systems. Table 4 describes the various AWS instances that we used for different approaches.
The naive approach was to take an AWS instance with 64 GB RAM, 16 vCPUs. All the data was uploaded to S3 and data cleaning was performed successfully. Using Guinea Pig, we used model transformations to transform from one state to another the entire data. The transformations are from one representation to another, hence from one vector space to another. This helps to generate word vector for every review written by each user. Spark has pre-existing machine learning libraries, like Mllib which enables all the work proposed to be done on one framework. Hence, we can generate word vector for every review for each user. As a result we were able to successfully perform data cleaning.

6. RESULTS

This section discusses the results obtained for the research project, the reasons and inference derived from them. By comparing the values predicted by the logistic regression classifier for a particular user and those from the validation test set, a cross validation accuracy of 46.7% was obtained. In addition, experiments to find out the affect of number of clusters in K means and parameters of logistic regression on the final accuracy were performed. It was observed that as the number of clusters were increased the final accuracy improved. Also, it was observed that the accuracy follows a distorted U-curve peaking at a parameter value of 0.01. The plots are given in Figure 3.

7. FUTURE WORK

There are modules, like Doc2Vec[13] which are useful for paragraphs or sentences and the future work deals with using these modules. Doc2Vec does the vector averaging automatically whereas in Word2Vec, this was done manually. We would like to compare and contrast the results with Doc2Vec. It is also necessary to validate the current results using baseline methods, like collaborative filtering and content based approach. Another thing that is foreseeable is to use different models and try to compare and contrast the results given by each model.

8. CONCLUSIONS

When the approach of one logistic regression model for each user so that each model is able to learn one user. There are noticeable discrepancies with this model owing to limited data of a single one user, because of which it is possible that the model learns the false values which are not the true

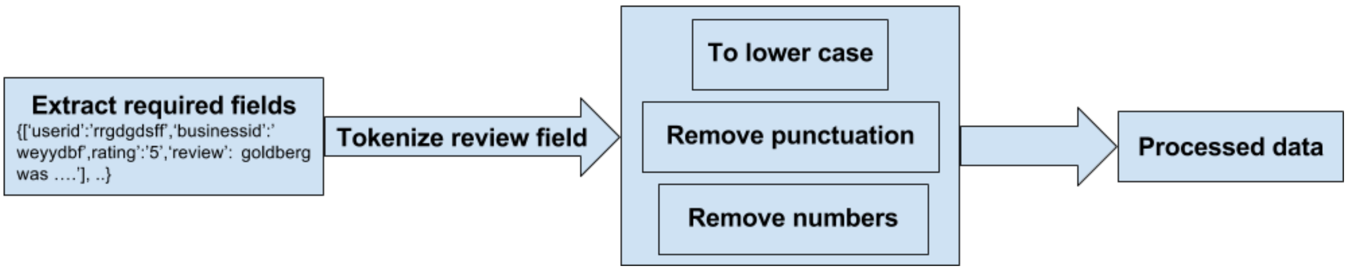
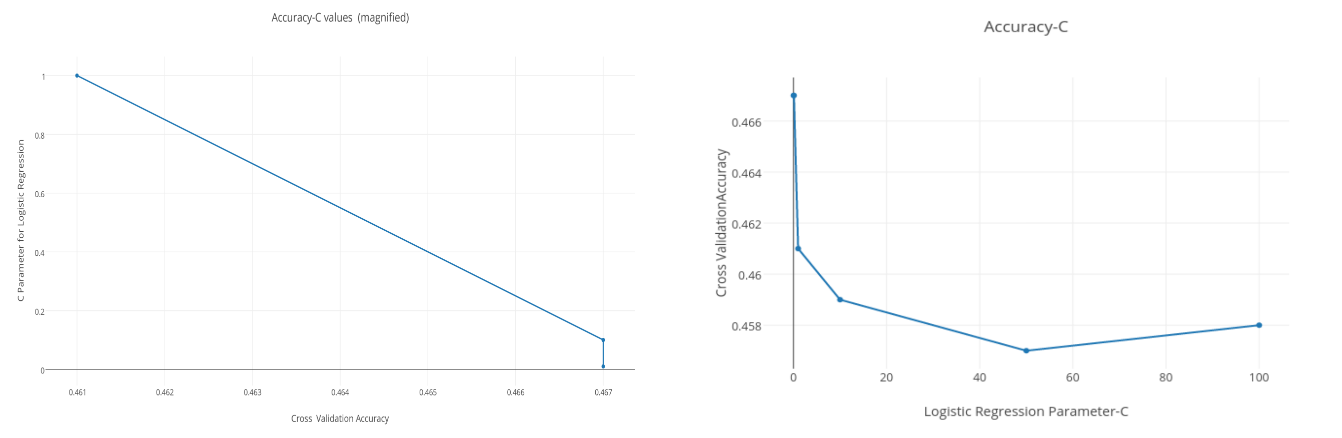


Figure 2: Text Pre-Processing

Figure 3 Accuracy vs C



representation of the users' tastes. Also, having information of just one user may lead to high error values, which were actually being observed in our experiments.

9. ACKNOWLEDGMENTS

We would like to thank Prof. William Cohen for giving us this opportunity and for guiding and mentoring us throughout this project.

10. REFERENCES

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

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. 1994.

[3] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated LSTM. *CoRR*, abs/1508.03790, 2015.

[4] Jan Koutník, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber. A clockwork RNN. *CoRR*, abs/1402.3511, 2014.

[5] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

[6] Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. pages 2342–2350, 2015.

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. pages 3111–3119, 2013.

[8] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis.

[9] Tiago Boldt Sousa. Dataflow programming concept, languages and applications. 2012.

[10] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: A not-so-foreign language for data processing. pages 1099–1110, 2008.

[11] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. pages 1532–1543, October 2014.

[12] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. pages 45–50, May 2010. http://is.muni.cz/publication/884893/en.

[13] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.