CS226 Big Data Management

Project Report

Yunfan Kang |ykang040@ucr.edu

Longze Su |lsu018@ucr.edu

Zhijie Qiao |zqiao006@ucr.edu

Baixi Sun | bsun031@ucr.edu

Jiatai Chen | jchen446@ucr.edu

University of California, Riverside

Computer Science and Engineering

December 5, 2019

1. Introduction

Social networks have attracted huge attention due to their rich content and world-wide popularity. Four billion of microblogs are posted by more than one billion users every day on Facebook and Twitter [1][2] and around 1.7 billion visitors are recorded per month on Reddit [3]. Despite the rich content on those social platforms, users can be sometimes confused about how to choose a suitable platform that is most likely to have the content they want or other users they want to discuss with. Hence, we propose to make use of feature extraction and semantic analysis techniques to identify the difference between trending topics on different social platforms to explore the potential of helping users to find the communities they are interested to join. Semantic analysis tasks are to discover the meaning of a chunk of text automatically by discovering the semantic relationships [4], one of its major applications is topic modeling [5]. This would help us to analyze the trending topics on different platforms and how users discuss them. Based on the analysis result, we distinguish the difference of user behavior on different social platforms to cater the need of different users looking for a suitable community.

Due to the huge quantity of the contents on social networks, we find it infeasible to label a portion of the content that is enough for training a supervised machine learning model to classify the contents. Hence, we decided to make use of the unsupervised machine learning methods to cluster the contents first based on the textual features. The features encode the semantic meaning of the words and the context information. After the clustering, we sample from each cluster to infer and interpret the meaning of each cluster and identify the distribution of contents from each social platforms in each cluster to demonstrate the difference between social platforms. In this project, we did the experiment on Twitter and Reddit. We focus on the topic “Frozen” because the movie Frozen 2 was released during the quarter and we think it would be interesting to show what people are discussing on social platforms and the difference between the topics on different social platforms. The result of the comparison may help the users to choose a suitable community.

The report is organized as follows: In Section 2, we discuss the techniques and tools used to build this project. Then the experimental results and related interpretation and discussion are presented in Section 3. Finally we sum up the project with possible future works in Section 4.

2. Methodology

2.1 Data Collection

Data was collected from two major social platform, Twitter and Reddit using their respective api tweepy and PRAW. For this project, we carefully selected data from two categories, a more generic one containing all the movie-related tweets and subreddits, and a more specific one containing tweets and subreddits regarding the same movie. In this way, we hope that we can exploit and expose the difference between Twitter and Reddit contents.

When collecting data from Reddit, for the generic category, we fetched multiple movie-related subreddits from Reddit and collected the top 10000 submissions of all time for each movie-related subreddit. The list of subreddits is obtained from the recommendations online, such as snoopsnoo.com. For each subreddit, the crawler starts from the most popular submission for a

specified subreddit and works its way down to the 10000th most popular submission. For the specific category, the subreddit for one particular movie “Frozen 2” was collected.

As for Twitter, the data is less structured as on Reddit, so what we used hashtags and keywords to find what we are looking for. For the generic category, we used the keyword “movie” in the hope of finding movie related tweets, and for the specific category, in corresponding to Reddit, we used “Frozen” and “Frozen 2” as keyword. The one thing to mention when collecting data from Twitter is that a lot of tweets are retweets of the trailers or announcement from the film crew. We decided to exclude the retweets for two main reasons, the api constraints stops us from accessing the full text from the tweets being retweeted, and the retweets are essentially the exact copy of other tweets. We believe both behaviors of retweets would have negative impacts in our attempt of finding the difference between the two social platforms, as a result, retweets are discarded.

2.2 Doc2Vec

In order to analyse the difference between contents from different social media numerically, we have to transfer the text and content into their numeric representation. Many use the well known but simplistic method of bag of words (BOW), but the outcomes will be mostly mediocre, since BOW loses many subtleties of a possible good representation, e.g consideration of word ordering. Latent Dirichlet Allocation (LDA) is also a common technique for topic modeling (extracting topics/keywords out of texts) but it’s very hard to tune, and results are hard to evaluate.

Doc2vec method, a concept that was presented in 2014 by Mikilov and Le in [6], is a better way to compute numeric representation for paragraphs, which is also the method we used in this project. Design for doc2vec is heavily based on word2vec, which is a pre-trained model released by Mikolov et al. in 2013 [7]. To fully understand doc2vec, word2vec should be elaborated first.

In general, when you like to build some model using words, simply labeling/one-hot encoding them is a plausible way to go. However, when using such encoding, the words lose their meaning. e.g, if we encode Paris as id_4, France as id_6 and power as id_8, France will have the same relation to power as with Paris. Unlike BoW, it only represent 0 or 1 (Counting having a word or not approach) and it cannot represent whether two wordings have similar meanings or not. We would prefer a representation in which France and Paris will be closer than France and power.

Word2Vec intends to give you just that: a numeric representation for each word, that will be able to capture such relations as above. this is part of a wider concept in machine learning — the feature vectors. Such representations, encapsulate different relations between words, like synonyms, antonyms, or analogies.

At the very beginning, a word vector is of dimension n, and n "is an arbitrary size which defines the size of our embedding space." That is to say, this word vector doesn't mean anything concretely. It's just an abstract representation of certain qualities that this word might have, that we can use to distinguish words. In fact, the values of a vector embedding for a word is usually just randomized at initialization, and improved iteration-by-iteration with following training process. In word2vec, the input is one-hot encoded.

Before word2vec, a very popular model architecture for estimating neural network language model (NNLM) was proposed in [8], where a feedforward neural network with a linear projection layer and a non-li-ear hidden layer was used to learn jointly the word vector representation and a statistical language model. Mikolov et al proposed two new architectures [7] which reducing computation complexity and including additional context: Continuous Bag-of-Words model (CBOW) and the Skip-Gram model.

CBOW creates a sliding window around current word, to predict it from “context” — the surrounding words. Each word is represented as a feature vector. For instance, “the word vector encodes semantic relationships among words”. If the window (n) is 3, the subset of prediction list is:
Case 1, Before Words: {Empty}, After Words: (word, vector, encodes), Predict Word: “the”
Case 2, Before Words: (the), After Words: (vector, encodes semantic), Predict Word: “word”.

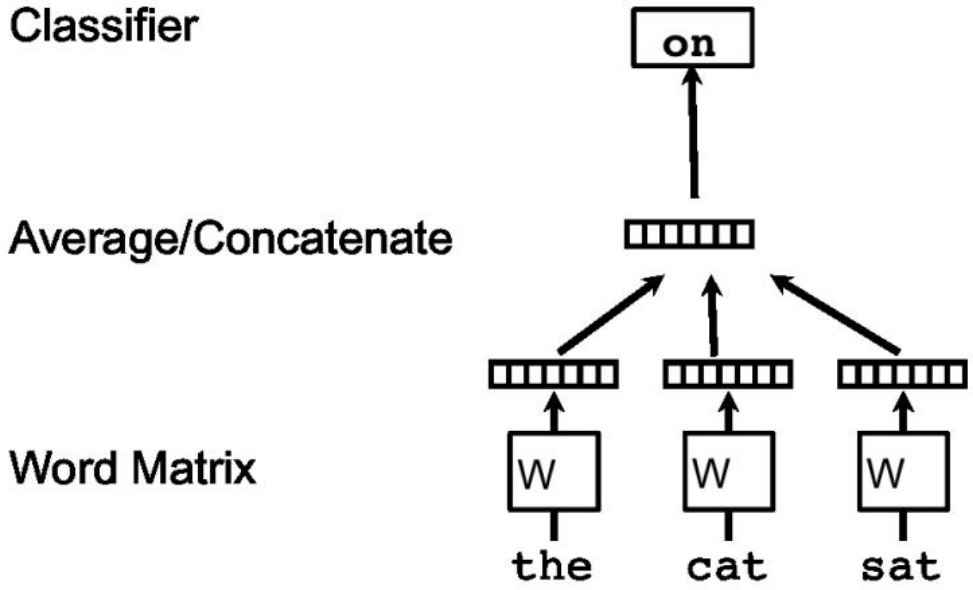


Figure 1. A framework for learning word vectors.

Context of three words (“the,” “cat,” and “sat”) is used to predict the fourth word (“on”). The input words are mapped to columns of the matrix W to predict the output word.

In this framework(Fig 1), every word is mapped to a unique vector, represented by a column in a matrix W. The column is indexed by position of the word in the vocabulary. The concatenation or sum of the vectors is then used as features for prediction of the next word in a sentence. More formally, given a sequence of training words w1, w2, w3, ..., wT , the objective of the word vector model is to maximize the average log probability

$$\frac{1}{T}\sum_{t=k}^{T-k}\log p(w_t|w_{t-k},...,w_{t+k})$$

The prediction task is typically done via a multiclass classifier, such as softmax. There, we have

$$p(w_t|w_{t-k},...,w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

Each of yi is un-normalized log-probability for each output word i, computed as

$$y = b + Uh(w_{t-k},...,w_{t+k};W) \quad (1)$$

where U, b are the softmax parameters. h is constructed by a concatenation or average of word vectors extracted from W. The neural network based word vectors are usually trained using stochastic gradient descent where the gradient is obtained via backpropagation. After the training converges, words with similar meaning are mapped to a similar position in the vector space. The difference between word vectors also carry meaning. For example, the word vectors can be used to answer analogy questions using simple vector algebra: “King” - “man” + “woman” = “Queen”.

The second algorithm is actually the opposite of CBOW--Skip gram is much slower than CBOW, but considered more accurate with infrequent words.-- instead of predicting one word each time, we use 1 word to predict all surrounding words (“context”). For instance, “the word vector encodes semantic relationships among words”. If the window (n) is 3, the subset of prediction list is:
Case 1, Predict Word: “the”, Words: (word, vector, encodes).
Case 2, Predict Word: “word”, Words: (the, vector, encodes, semantic).

In detail, we’re going to train the neural network to do the following: given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random. The network is going to tell us the probability for every word in our vocabulary of being the “nearby word” that we chose. The output probabilities are going to relate to how likely it is find each vocabulary word nearby our input word.

We first build a vocabulary of words from our training documents.–let’s say we have a vocabulary of 10,000 unique words.Then represent an input word like “ants” as a one-hot vector. This vector will have 10,000 components (one for every word in our vocabulary). The output of the network is a single vector (also with 10,000 components) containing, for every word in our vocabulary, the probability that a randomly selected nearby word is that vocabulary word.

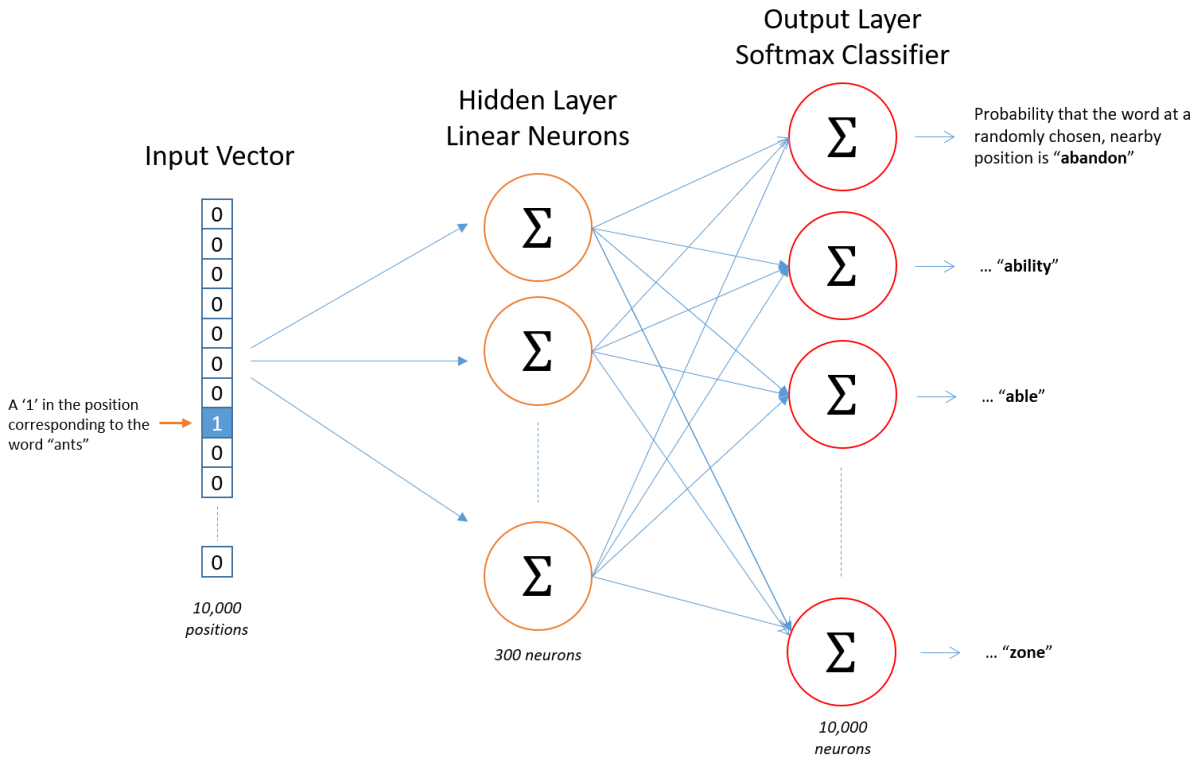


Figure 2. Network architecture

When training this network on word pairs, the input is a one-hot vector representing the input word and the training output is also a one-hot vector representing the output word. But when you evaluate the trained network on an input word, the output vector will actually be a probability distribution. If we're learning word vectors with 300 features. So the hidden layer is going to be represented by a weight matrix with 10,000 rows (one for every word in our vocabulary) and 300 columns (one for every hidden neuron). Now rows of this weight matrix are actually word vectors.

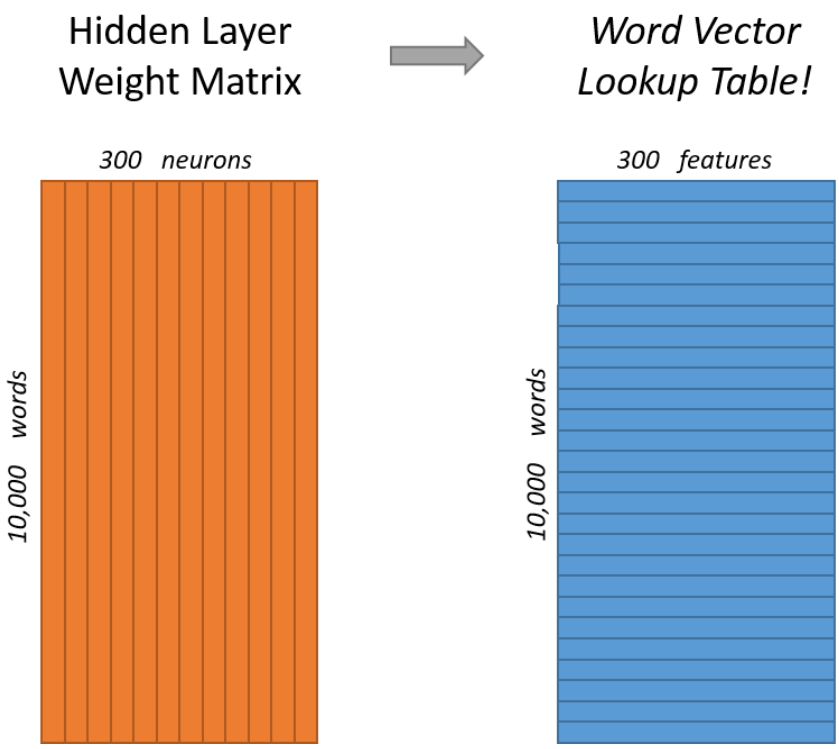


Figure 3. Hidden Layer weight is actually feature vector of words

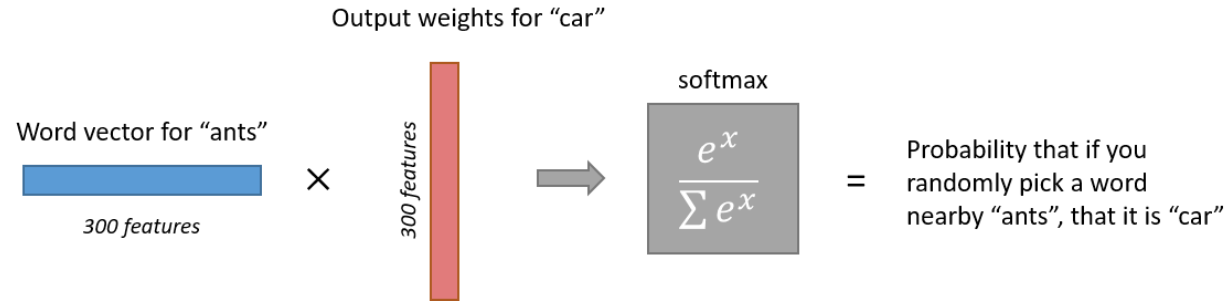


Figure 4. Output Layer

If two different words have very similar “contexts” (that is, what words are likely to appear around them), then model needs to output very similar results for these two words. And one way for the network to output similar context predictions for these two words is if the word vectors are similar. So, if two words have similar contexts, then the network is motivated to learn similar word vectors for these two words.

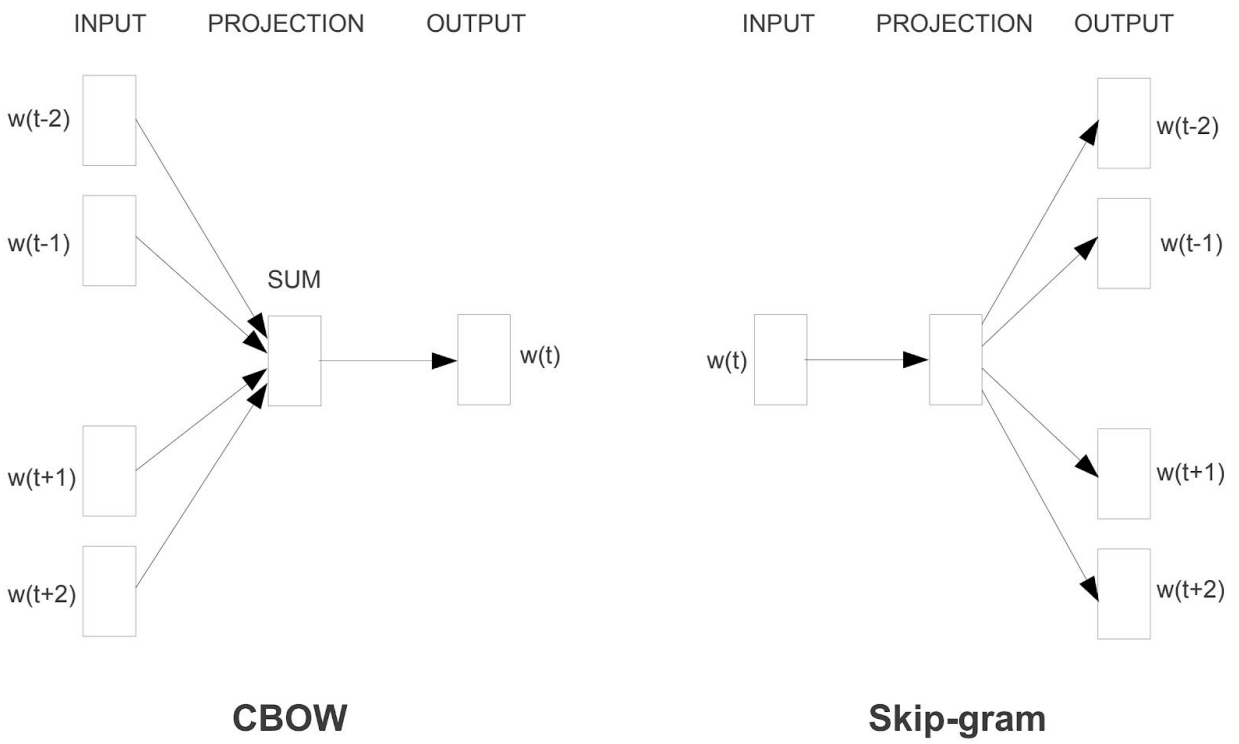


Figure 5. CBOW and Skip-gram

The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

So Word2Vec uses a trick you may have seen elsewhere in machine learning. It trained a simple neural network with a single hidden layer to perform a certain task, but then it is not actually going to use that neural network for the task we trained it on. Instead, the goal is actually just to learn the weights of the hidden layer– these weights are actually the “word vectors” that we’re trying to learn.

As for the goal of doc2vec, is to create a numeric representation of a document, regardless of its length. But unlike words, documents do not come in logical structures such as words, [6] used the word2vec model, and added another vector (Paragraph ID in Fig 6). It is a small extension to the CBOW model. But instead of using just words to predict the next word, it also added another feature vector, which is document-unique. Every paragraph is mapped to a unique vector, represented by a column in matrix D and every word is also mapped to a unique vector, represented by a column in matrix W. The paragraph vector and word vectors are averaged or concatenated to predict the next word in a context. More formally, the only change in this model compared to the word vector framework is in equation 1, where h is constructed from W and D.

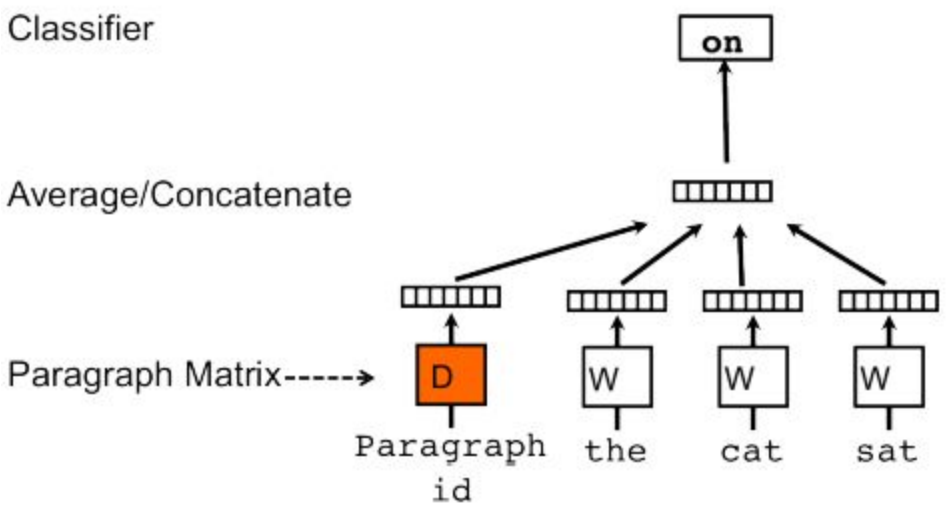


Figure 6. Distributed Memory version of Paragraph Vector

The paragraph vectors and word vectors are trained using stochastic gradient descent and the gradient is obtained via backpropagation. At every step of stochastic gradient descent, one can sample a fixed-length context from a random paragraph, compute the error gradient from the network in Fig 6 and use the gradient to update the parameters in our model.

At prediction time, one needs to perform an inference step to compute the paragraph vector for a new paragraph. This is also obtained by gradient descent. In this step, the parameters for the rest of the model, the word vectors W and the softmax weights, are fixed.

Suppose that there are N paragraphs in the corpus, M words in the vocabulary, and we want to learn paragraph vectors such that each paragraph is mapped to p dimensions and each word is mapped to q dimensions, then the model has a total of $N \times p + M \times q$ parameters (excluding the softmax parameters).

In summary, the algorithm itself has two key stages: 1) training to get word vectors W, softmax weights U, b and paragraph vectors D on already seen paragraphs; and 2) “the inference stage” to get paragraph vectors D for new paragraphs (never seen before) by adding more columns in D and gradient descending on D while holding W, U, b fixed. It use D to make a prediction about some particular labels using a standard classifier, e.g., logistic regression.

It acts as a memory that remembers what is missing from the current context — or as the topic of the paragraph. While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document.

The implementation of Doc2Vec module is based on deeplearning4j. The sentiment analyze module was first used the opennlp APIs, which is from Apache to analyze the text. The document categorizer was used to classify the texts from twitter and reddit. However, because of its narrow application field and it is hard to implement with Spark. With the consideration of the performance, deeplearning4j was used. The API is used because it uses Apache Spark and Hadoop to accelerate training, and it is surprisingly efficient on the performance. It is said on the office webpage that on multi-GPUs, it is equal to Caffe in performance.

The processes to turn text to vector is shown as Figure 7. First, the text file from the twitter platform and reddit platform is parsed, stop words removed and all texts are stored into one array list, tags are stored in another. Here the tags are “reddit” and “Twitter”, which indicates the source of the text. And stop words are removed with the reference of the stop words’ dictionary. Second, the train data and test data from this list was got. Within the train data, the amount of reddit and twitter records should be equal to make sure the balance of the model. Third, the train data is used to train and generate a model. The model is a matrix that contains the information of a dictionary of all words in the text and their context information. Then we put the test data into the inferVector method that provided by the API. Then I can get a vector whose dimension amount is the out layer size of the CNN that was set on the training process. The vector is stored in the INDArray and the vector of each line of text can be gotten by iterating the INDArray.

Figure 7. Flow Chart of Doc2Vec Module

The implementation of the module is based on two packages: edu.cs.ucr.edu.cs226.bsun031.main and edu.cs.ucr.edu.cs226.bsun031.utils. The former package contains a Main method that calls the methods in the latter package. DataFileReader is used to read the twitter files. RedditParser is used to read the reddit files. The DataFileWrite is used to write the tags and the texts got from both platforms, where the texts are parsed and stop words in the text is removed. The stop words are defined as the insignificant words to the text such as “the”, “a”, “I”, so on and so forth. The stop words are also including the word that contains “#” which indicate that it is the topic, because here we are analyzing the same topic on different platforms to best compare the difference. And “@” means the retweet username and that is obvious insignificant. However, we should reserve those emojis because they express the emotions of users. Doc2Vec method basically used the deeplearning4j to train and generate the model and use the model to turn text into vector. The train data should contain more unique words as possible to ensure the dictionary size is enough for the model to recognize the texts. Figure 8 shows the architecture of the Doc2Vec Module. Form the dependency relation we can see that the implementation is based on DL4J and ND4J APIs.

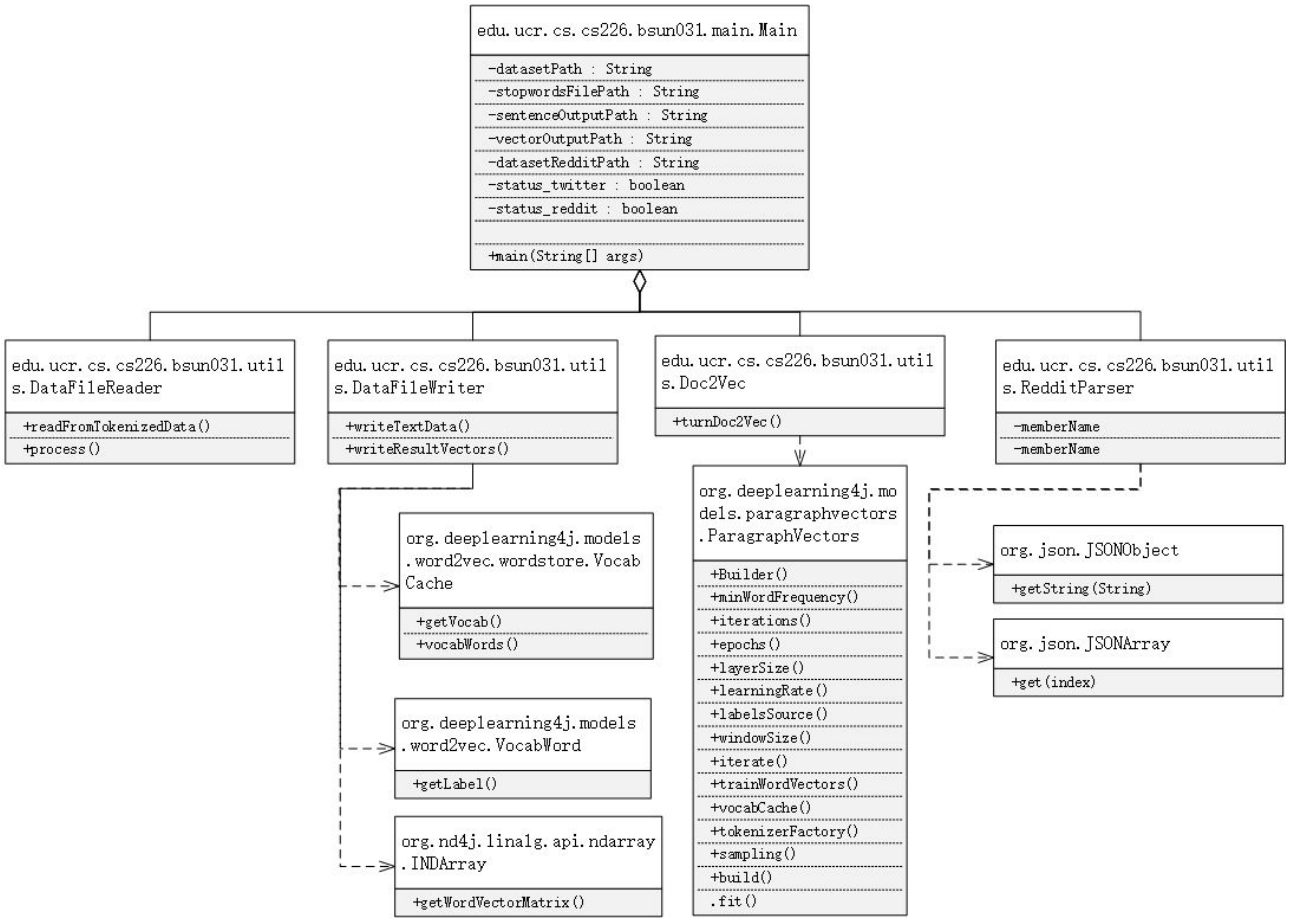


Figure 8. Doc2Vec Module UML

2.1 KMeans Clustering

Following the Doc2Vec, we made use of K-means clustering to cluster the 50-dimensional vectors. Each cluster is expected to be a group of twitter posts and reddit comments that share common features. The similarity can be interpreted by sampling from the cluster. The distribution of contents from different social platforms may demonstrate the difference of trends between different social platforms.

Since the k value of the K-means clustering needs to be specified before hand. We made use of the elbow method to select the value of k at which improvement in distortion declines the most. This is a widely used heuristic method to determine the k value. To visualize the clustering performance, we use the t-Distributed Stochastic Neighbor Embedding (t-SNE) method to reduce the

50-dimension vectors to 2-dimension vectors and visualize all the transformed vectors on a x-y coordinate.

The feature vectors generated from Doc2Vec is of high dimension. In this project, the dimension is set to be 50 to encode the desired features as recommended by the official documentation. However, due to the high dimensional property, it is not straightforward to visualize the data after clustering. Hence, we made use of dimension reduction algorithms to reduce the 50-dimension vectors to 2 dimensions and plot different clusters in different colors to visualize the result. To achieve this we tested the Principal Component Analysis (PCA) and the t-Distributed Stochastic Neighbor Embedding (t-SNE). We tried PCA first as it is a classic dimension reduction method in linear algebra. However, as shown in Figure 9 a, the result is not very desirable. The majority of data points is mapped to -0.4 to 0.4 in x and -0.25 to 0.25 in y. The resulting vectors tend to be overlapping and it is hard to tell whether the original vectors are in clusters or they are centralized. By comparison, the t-SNE is better in this aspect.

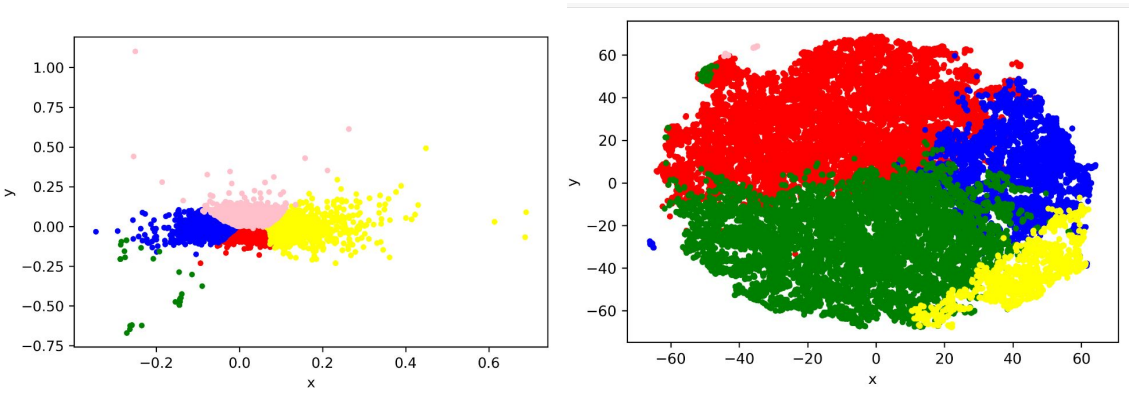


Figure 9. PCA and t-SNE

To verify the feasibility of k-means clustering, we first made a demo using 800 samples from both twitter and reddit data. The 800 vectors are clustered using the sklearn on python and visualized to see its performance. The intermediate result of the demo is shown in the in-class presentation.

3. Results and discussion

3.1 Preprocessing

```
df = spark.read().json(path).select(col);
des = df
        .select(functions.mean(col).alias("mean"), functions.min(col).alias("min"),
            functions.max(col).alias("max"), functions.stddev(col).alias("stddev"))
        .select( "mean", "stddev", "min", "max");
```

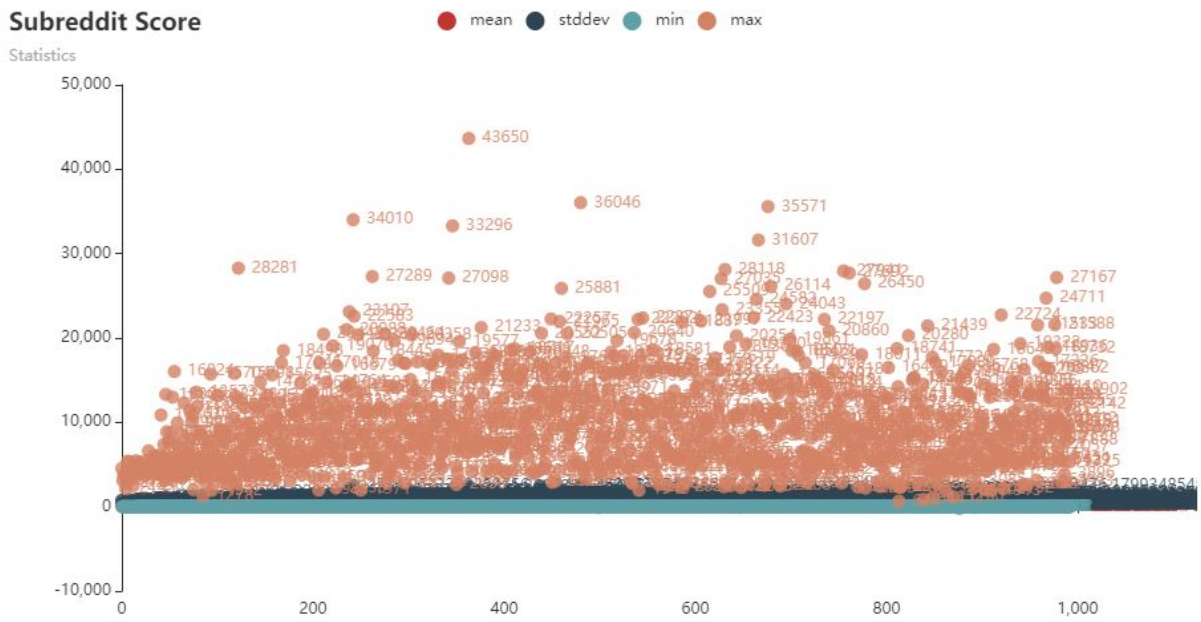


Figure 10.

The descriptions of mean, min, max and stddev on the scores of each subreddits, we could see that the scores of the subreddits vary from 0 to 50000, and according to the large value of mean and stddev, the hottest topic/subreddit attracted most of the attention, which means that we only need to be concerned about the content itself, and since all the content is chosen from top topic, the score and upvote can be negotiated.

3.2 Model Selection

```
JavaRDD<Vector> parsedData = data.map(s -> {
   String[] sarray = s.split(",");
   double[] values = new double[sarray.length-1];
   for (int i = 0; i < sarray.length-1; i++) {
      values[i] = Double.parseDouble(sarray[i+1]);
   }
   return Vectors.dense(values);
});
KMeansModel clusters = KMeans.train(parsedData.rdd(), numClusters, numIterations);
// Evaluate clustering by computing Within Set Sum of Squared Errors
double SE = clusters.computeCost(parsedData.rdd());
```

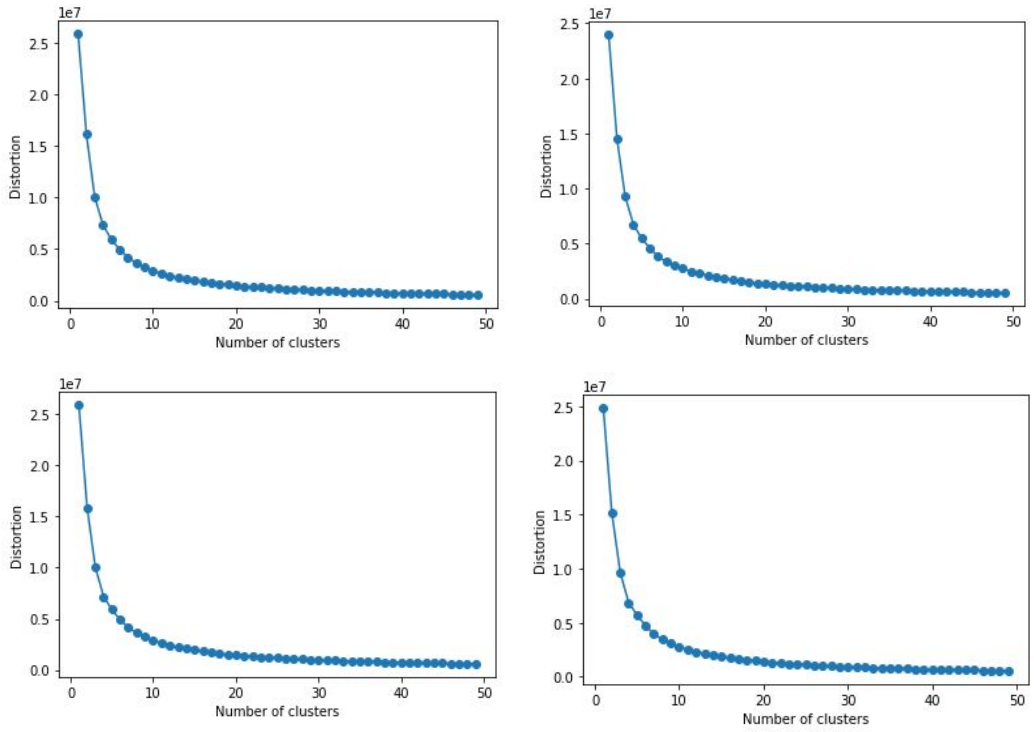


Figure 11 Distortion curves for kmeans (Top left to bottom right 10 - 40 dimensions)

The plots above are distortions-cluster number relationship, from top left to bottom right are separately derived from dimension of doc vectors 10, 20, 30, 40. We could easily get the best numbers of clusters are 4 or 5 because the curve reached stable after these values. We choose 4 as our cluster numbers.

3.3 KMeans and TSNE

```
parsedData = data.map(s -> {
        String[] sarray = s.split(",");
        double[] values = new double[sarray.length-1];
        for (int i = 0; i < sarray.length-1; i++) {
            values[i] = Double.parseDouble(sarray[i+1]);
        }
        return Vectors.dense(values);
    });
    parsedData.cache();
    KMeansModel clusters = KMeans.train(parsedData.rdd(), numClusters, numIterations);
    // Evaluate clustering by computing Within Set Sum of Squared Errors
    JavaRDD<Integer> res = clusters.predict(parsedData);

    JavaRDD<Line1> l1 = parsedData.map(s -> {
        double[] d = s.toArray();
        String value = Arrays.stream(d)
```

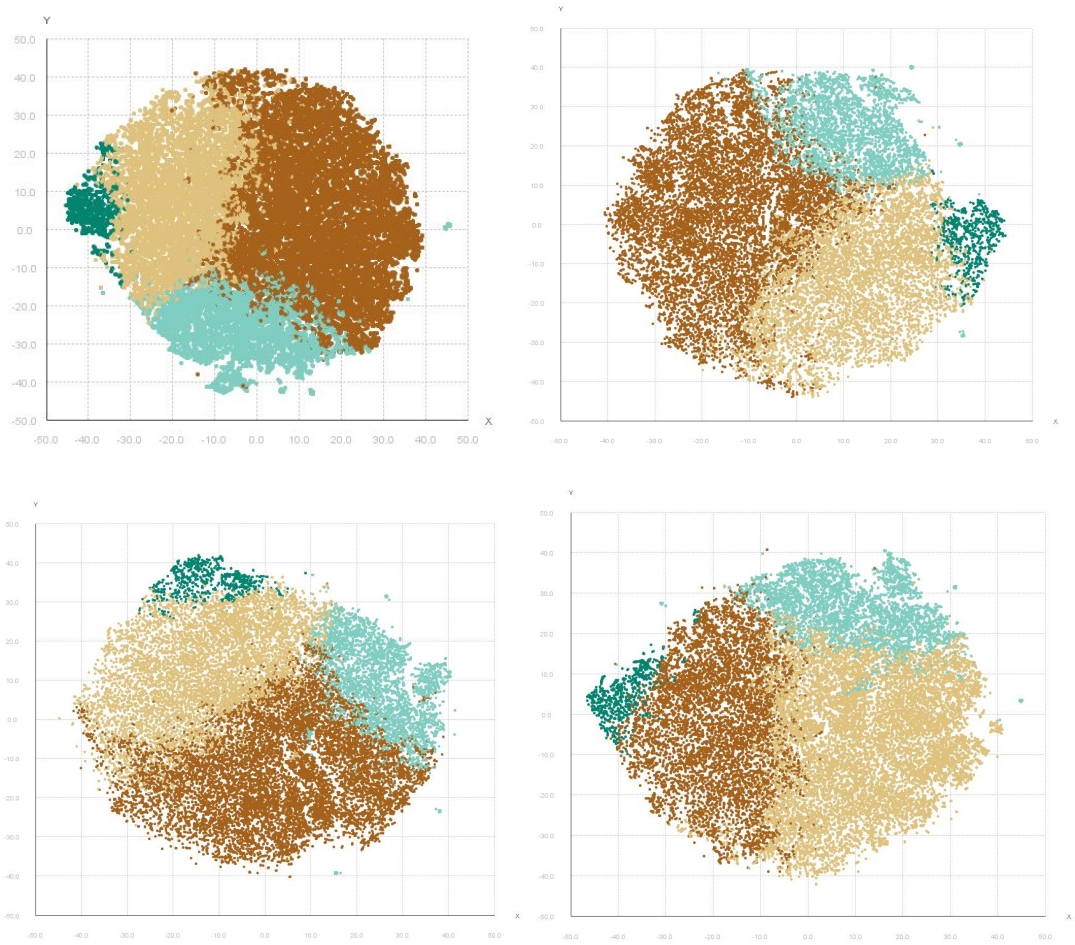
```
            .mapToObj(String::valueOf)
            .collect(Collectors.joining(","));;
    Line1 line = new Line1();
    line.setValue(value);
    return line;
});
JavaRDD<Line2> l2 = res.map(s -> {
    String label = s.toString();
    Line2 line = new Line2();
    line.setLabel(label);
    return line;
});
df1 = spark
        .createDataFrame(l1, Line1.class)
        .withColumn("id", functions.monotonically_increasing_id());
df2 = spark
        .createDataFrame(l2, Line2.class)
        .withColumn("id", functions.monotonically_increasing_id());
df = df1
        .join(df2,  df1.col("id").equalTo(df2.col("id")))
        .drop("id");
```

Parameters, we firstly applied KMeans cluster to the 10-40 dimension doc vectors. Then, use TSNE reducing dimension and do better visualization.
Select output dimension as 2 and 3, perplexity as 20, theta as 0.5

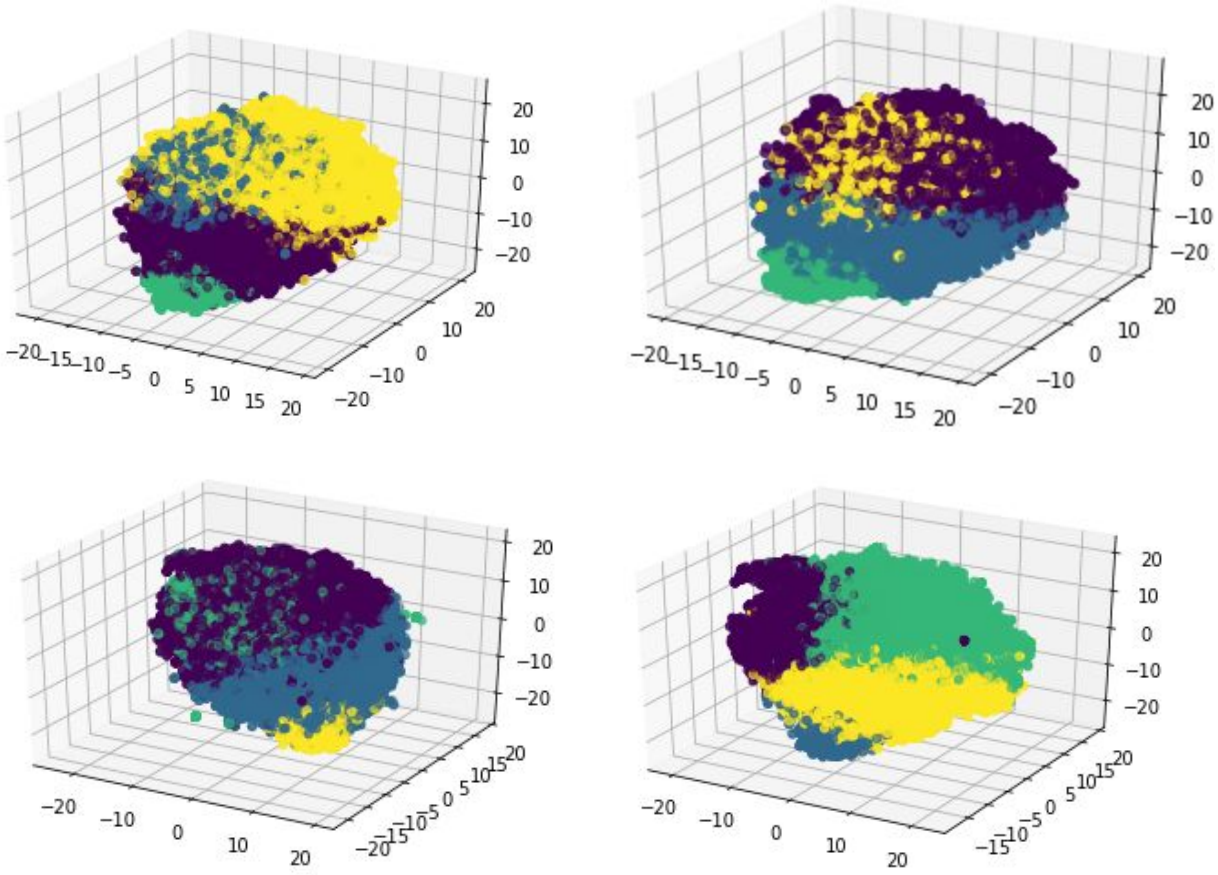
```
        .
5502: [0.0659, 0.0448, 0.0083, 0.0025, 0.0038, 0.0055, 0.0003, 0.0276, 0.0070, 0.0097, 0.0401, 0.0478, -0.0045, -0.0332
 -0.0058, -0.0090, 0.0130, -0.0186, -0.0030, 0.0002, 1.0000, ...]
5503: [0.0215, 0.0175, 0.0120, 0.0101, 0.0031, -0.0087, -0.0027, 0.0134, 0.0037, -0.0060, 0.0053, -0.0006, -0.0005, -0.
023, -0.0300, -0.0101, 0.0090, -0.0090, -0.0064, 0.0167, <0.0>, ...]
5504: [0.0513, 0.0185, 0.0094, -0.0209, 0.0128, 0.0113, 0.0065, 0.0367, 0.0066, -0.0239, 0.0558, 0.0188, -0.0285, -0.01
4, 0.0278, -0.0180, 0.0211, -0.0180, -0.0022, -0.0053, 1.0000, ...]
5505: [0.1292, 0.0351, 0.0038, -0.0540, 0.0058, 0.0396, 0.0038, 0.0429, -0.0021, -0.0786, 0.0912, 0.0078, -0.0679, -0.0
23, 0.0355, -0.0041, 0.0427, -0.0185, 0.0013, -0.0578, 1.0000, ...]
5506: [0.0360, 0.0119, 0.0112, -0.0086, -0.0047, -0.0061, -0.0107, 0.0008, -0.0023, -0.0151, 0.0070, 0.0013, -0.0036, -
.0114, -0.0154, -0.0058, 0.0105, -0.0057, 0.0021, 0.0121, <0.0>, ...]
:Shape is = 35507 x 30
sing no_dims = 2, perplexity = 20.000000, and theta = 0.500000
omputing input similarities...
one in 13.62 seconds (sparsity = 0.002733)!
earning embedding...
```

From the logs of dim=40 example, we can get the sparsity of the vector space, all of our examples approached 0.0027.

The following plots are the 2d plots of clustering results, the different inputs had similar output, which means the doc vectors with different dimensions have similar features. In the clustering, the yellow cluster in 2d have more than 80% twitter labels, the brown one in the 2d have more than 70% reddit labels, which means the two biggest cluster represented two communities. The two biggest clusters should represent reddit and twitter, while the smallest represented some unrelated topics with movies like music.



Plot 1. 2d plots, the yellow is twitter, the brown is reddit (Top left to bottom right 10 - 40 dimensions)

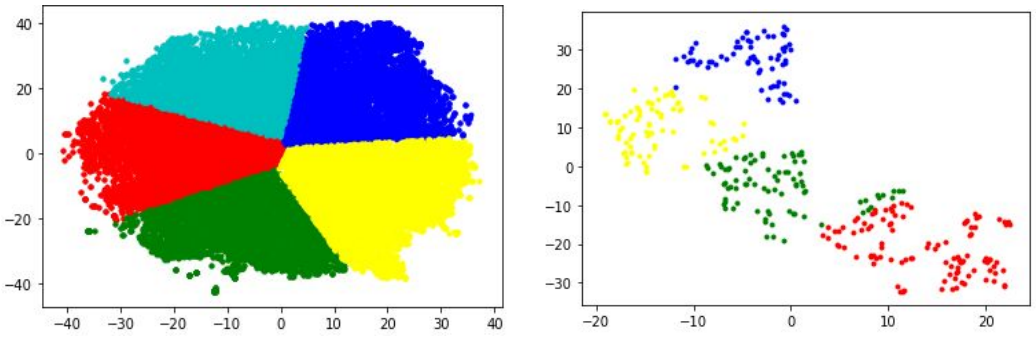


Plot 2. 3d plots, the purple is twitter, the yellow is reddit (Top left to bottom right 10 - 40 dimensions)

These are the 3d plots of clustering results, and they are similar with the 2d. And they have clearer boundaries than 2d plots.

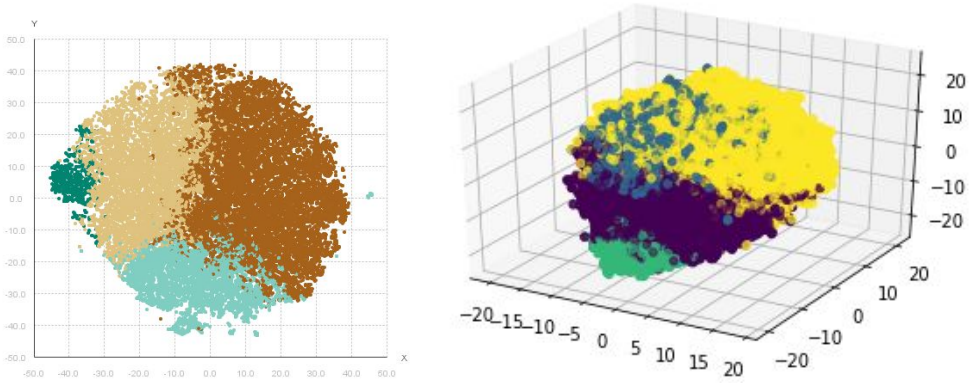
3.5 Discussion

The differences between the two communities are proven by k means clustering. The most important things in the clustering processes are the dimension reduction. Initially, we tested other methods like reducing dimension then clustering (bottom left), the results are really bad and meaningless. And we also tried k means on smaller datasets(400 senteces), the clustering effects looks good but they represented more detailed information like a typical movie, in our project is Frozen.(bottom right)



Plot 3. Dimension reduction then clustering (left), small datasets (right)

As a result, our final method is using total datasets (more than 20000 sentences) and applied the clustering first then reduce dimension. And to give more promising results, we tried on different doc vector dimensions and used different visualizations. The results looked better than before and the different doc vetor dimension and visualization dimensions actually look very similar, we actually can use a 2d plot to represent the main clusters in our project, which proved our hypothesis, the topics in these two communities are different. In the clustering, the yellow cluster in 2d have more than 80% twitter labels, the brown one in the 2d have more than 70% reddit labels, which means the two biggest cluster represented two communities.



4. Conclusion and Future work

This project aims to distinguish the difference between mainstream topics on different platforms. We sampling the corresponding documents in our raw datasets, and found that the two main clusters are mainly composed of twitter and reddit, which gave strong proven of our hypothesis. There is a chance that if we extracted the keywords from different clusters, the meaning of different clusters could be more clear and meaningful, and which could be our following improvement and modification of this project. Another prospect of optimization is that if we could apply more clustering algorithms and reducing methods, which is a time-consuming and hard process, but the result might be promising. At last, in our project, we only survey the twitter and reddit and crawled their data in a short time range. If we could get more data sources from more communities or a longer time range, the results might be very different and interesting results could be fetched. Finally, better word embedded method could also help a lot.

References

[1] Facebook Statistics. https://newsroom.fb.com/company-info/
[2] Twitter Statistics https://about.twitter.com/en_us/company.html
[3] Total global visitor traffic to Reddit.com 2019
https://www.statista.com/statistics/443332/reddit-monthly-visitors/
[4] Magdy A, Abdelhafeez L, Kang Y, Ong E, Mokbel MF. Microblogs data management: a survey. The VLDB Journal. 2019:1-40.
[5] Vosecky J, Jiang D, Leung KW, Xing K, Ng W. Integrating social and auxiliary semantics for multifaceted topic modeling in twitter. ACM Transactions on Internet Technology (TOIT). 2014 Dec 17;14(4):27.
[6] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." International conference on machine learning. 2014.
[7] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).
[8] Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137-1155, 2003.
[9] DL4J. https://deeplearning4j.org/
[10] TSNE for Java https://github.com/lejon/T-SNE-Java
[11] spark SQL https://spark.apache.org/docs/2.2.0/

Appendix A: Source Code Instruction

Source code can be downloaded from https://github.com/srewac/CS-226

crawler/reddit_crawler.py:

The reddit_crawler.py makes use of the Python Reddit API Wrapper (PRAW). It reads the subreddit.txt file which contains the list of subreddits to be crawled. The data collected will be stored to ./data/[subreddit_name].

crawler/twitter_crawler.py:

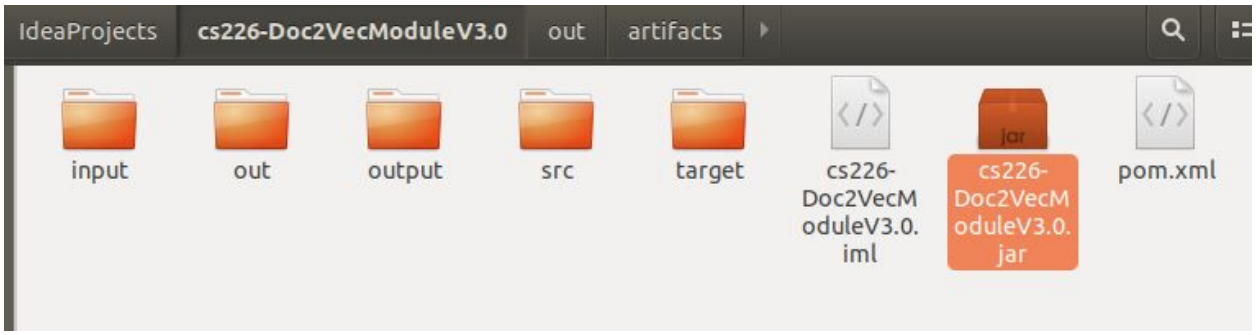
The twitter_crawler.py makes use of the Twitter API tweepy. It searches the Twitter using keywords provided and stores the distinct tweets to ./data/[twitter_data].

RedditParser.java:

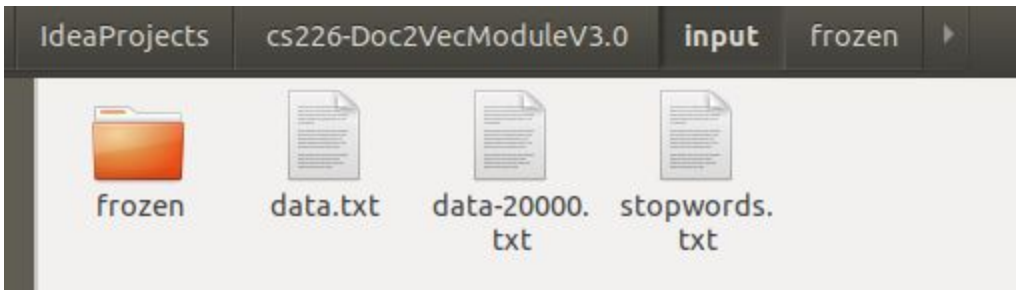
This java program parses the JSON files collected by the parser to generate a txt file named “reddit.txt” whose lines are in the format “reddit;[content]”. To run the code, change the path in the main function t the root directory of the reddit data.

Doc2Vec Module:

This part is basically implemented by the Java language and the executable is built as a jar file: cs226-Doc2VecModuleV3.0.jar. To successfully run the jar file, make sure that the input folder and output folder is in the same directory. The output results of this module will be in the output folder. And in the input folder, you should put data.txt(or data-20000.txt) and stopwords.txt and reddit folder such as frozen, as shown in the following screenshot:



Appendix A : Image 1 Screenshot of the folder structure 1



Appendix A : Image 2 Screenshot of the folder structure 2

To run the jar, you should first make sure that the Spark is installed and the environment is set up. Then you can run the jar file with java -classpath <jar file> edu.ucr.cs.cs226.bsun031.main.Main <twitter data file directory> <stopwords.txt directory>.

TFIDF/main:

This part is for preprocessing. Spark and maven required

all.java

Implemented the method of read data and transform the data into TFIDF form.

multi.java

For the keywords extraction from TFIDF
spark-submit --class main.multi target/lsu018-1.0-SNAPSHOT.jar [input_reddit] [output_keywords]

Single.java

For input twitter and reddit crawler data and using TFIDF to get similar matrix
spark-submit --class main.single target/lsu018-1.0-SNAPSHOT.jar [input_reddit] [input_twitter] [output_similar_matrix]

Score.java

For statistics of score in reddit data.
spark-submit --class main.score target/lsu018-1.0-SNAPSHOT.jar [input_reddit] [output_stat]

TFIDF/kmeans:

This part is for clustering and visualization. Spark and maven required

kmeans.java

Implemented kmeans labels get method and kmeans result method

spark-submit --class kmeans.result target/lsu018-1.0-SNAPSHOT.jar [input_vector] [max_cluster_num] [output_cluser] [output_best]

tsne.java

Dimension reduction, run after k means.

java -jar tsne-demos-2.4.0.jar --parallel --nohdr --nolbls --lblcolno [label_column_index, start from 1] [kmeans_result_input]