Real-Time Sentiment Analysis and Hashtag Recommendation

Kuo-Wei Chung
kuowei.chung@rutgers.edu

Zhengjuan Fan
zhengjuan.fan@.rutgers.edu

Abstract

Sentiment Analysis

Sentiment analysis is one kind of text mining, identifying and extracting information in source text. There are different kinds of methods, and the most straightforward way is to tokenize a sentence, then use emotional adjective, like good and bad, to analyze sentiment. However, this kind of basic sentiment analysis is sometimes not good enough to analyze some complex sentences. because it will miss some high value messages that are needed to be discovered. Take I am not a good person as an example. After tokenizing, it will be I, am, not, a, good, person. In this sentence, if we only take care of good but not consider not, the sentiment is totally different with the original sentiment. Our goal is to find a reliable way to analyze sentence-level sentiment. In this project, we are going to train a model using machine learning, classifying one message is positive or negative. Then, we can use this model to monitor real-time Twitter messages for some keywords.

Hashtag Recommendation

Nowadays, short-text are widely used on the Internet. Most of social websites allow users to use hashtag categorize and express their sentiment. micro-blogging sites, like Twitter, continue to grow in popularity. Most of users may categorize their tweets through hashtags. Word, emoji and phrase might be used as hashtags. To solve this, we propose and implement hashtag recommendation to a user's tweet.

1 Introduction

Sentiment Analysis

Sentiment analysis is widely used these days. Business always cares about customer feedback, from social media, website, or any other source, which contains a lot of useful information. However, understanding what customers are talking about is not enough. We have to know how they feel. Therefore, we can use sentiment analysis to understand about customers feelings. That is to say, it can be applied to detect customers voice such as customer reviews and survey responses. If we have a good method to analyze sentiment, we can help a business to go deeper into the social sentiment of their product or service by monitor online reviews. Moreover, we can also monitor social media, such as Twitter and Facebook, to find out hidden issues on their brand. Through sentiment analysis, we can detect the trend of customers thinking, improving customer service and profits.

Hashtag Recommendation

For a user's Tweet, we can see it as three parts: Tweet's sentiment, Tweet's topic, and current trends. We can see it as a sentiment problem because social media is a way for people to express their feeling. We consider Tweet's sentiment as emoji prediction problem because most of people love to use emoji to express their feeling in general situation. Through pre-classified emoji dataset, we use word2vec + deep learning model to predict emoji's usage. About the Tweet's topic, some people see it as a multi-labels problem, but most of tweets do not contain hashtags which makes unbalanced labels affect accuracy. Therefore, it's better use a topic model which is trained from Tweets to provide some general hashtags for users. In this part, we use LDA topic modeling to generate word topics which is propose by "Using topic models for Twitter hashtag recommendation", *in Proceedings of the 22nd International Conference on World Wide Web (WWW '13 Companion)*[1] . LDA topic modeling is a unsupervised way to classify topic modeling based text

we gave. The last part is current trends. Hashtag recommendation can be seen as a trending problem because some hashtags are timeliness, like #WorldCup2018 and #NBAFinal. These tags are used only for special events. Therefore, the trending words can also be considered as good suggestions which can be counted by word frequencies in some time interval. In this project, we implement the first two part. The advantage of our approach is that we break the hashtag recommendation problem into three smaller pieces instead of predicting directly and define the problems more accurate.

2 Format of Data

There are two kind of data we are going to deal with. One is for offline ML training, and the data format is CSV. The other kind of data is streaming data, which is used for sentiment score application. Its format is JSON.

2.1 Data for Training

Positive/Negative Sentiment

We chose Sentiment140 as training dataset for our twitter sentiment classification model. Dataset can be downloaded from this link http://help.sentiment140.com/for-students/

Sentiment140 is a popular dataset for natural language processing. It contains 1,600,000 tweets with emoticons pre-removed. Data has 6 fields as below:
0 - the polarity of the tweet (0 = negative, 4 = positive)
1 - the id of the tweet
2 - the date of the tweet
3 - the query. If there is no query, then this value is NO QUERY.
4 - the user that tweeted
5 - the text of the tweet

Only column 0 and 5 are useful for detecting sentiment. So we dropped the other 4 columns in the data cleaning step. By exploring the dataset, we found that 50% of the data is with positive label and another 50% with negative label. Thus, there is no skewness on the class division. Since the dataset size is reasonably large and its content perfectly matches our twitter sentiment analysis purpose. Thus, we think Sentiment140 is a credible data for our project.

Hashtag Recommendation (Emoji Prediction)

We chose PsychExp as our training dataset for our Emoji prediction model. Dataset can be downloaded from this link: http://github.com/bfelbo/DeepMoji/

PsychExp contains 7480 text messages, and data has 2 fields as below:
text - contain the text message
label - seven classes represented by [0 - 6] = [joy, fear, anger, sadness, disgust, shame, guilt]

Here's an example with a disgust label: "The time I knocked a deer down - the sight of the animal's injuries and helplessness. The realization that the animal was so badly hurt that it had to be put down, and when the animal screamed at the moment of death."

Hashtag Recommendation (Topic Prediction)

We chose Sentiment140 as training dataset, which is also used on our sentiment analysis.

2.2 Clean the Training Data

P/N Sentiment

Before we could feed this dataset to our model, we found that the tweet text field contains some dirty data like: "@mention", url links "http://www" and numbers, this information does not contribute to our model. Lets remove it. HTML encode like "amp"; could not be converted to text, thus it needs to decode HTML to general text. We use the package BeautifulSoup to do this. UTF-8 Byte Order Marks like "\xef \xbf \xbd" in the text, the strange patterns of characters should be removed
The data cleaning method is borrowed from this blog: https://towardsdatascience.com/another-twitter-sentiment-analysis-bb5b01ebad90

Hashtag Recommendation (Emoji Prediction)

Before feeding the dataset into our model, we can do some procession for our data first. We only keep the alphabet and the special character ' in our data. This is because for most special characters, they are usually not considered as a valid vector in Googles pre-trained Word2Vec. However, for some words with ', like "don't" or "doesn't", they contain important information and they are also considered as valid vectors Googles pre-trained Word2Vec. Therefore, we will keep all alphabets and the special character '.

Hashtag Recommendation (Topic Prediction)

We follow the pre-processing way used for sentiment analysis, which is just mentioned. We tokenize the tweet and removed common stop words, word length smaller than 2 and do the lemmatiza-

tion for each tweet. This is good to refine information in each tweet.

3 Explore the Data

We explore and analyze the relation between labels and Tweets. About this part, please view the 543-Sentiment-Analysis and 543-Emoji-Analysis in the appendix section.

4 Sentiment Analysis Models

This project involves three sentiment analysis models :

1. Stanford CoreNLP, it's a pre-trained model and can be accessed by API. We use it for real-time sentiment analysis.
2. TF-IDF Logistic Regression on Pyspark.
3. Word2Vector Convolution Neural Network on Keras.

4.1 TF-IDF Logistic Regression on Pyspark

Apache Spark is an open-source powerful distributed computing platform. It allows the user to train and deploy sophisticated machine learning model for large data sets. Pyspark offers us to access Spark in Python. We design a TF-IDF Logistic Regression on Pyspark as the second sentimental analysis model in this project. Its flow chart is showed as below.

This post compared common machine learning models for sentiment analysis, TF-IDF Logistic Regression is quite a strong combination with high accuracy and average precision. We implement it on Pyspark. Pyspark provides two machine learning packages: SparkMLLib is used with RDD while SparkML supports Data Frame. Data Frame is much faster than RDD because it has meta-data associated with it. In this project, we use SparkML instead of SparkMLLib.

TF-IDF

TF-IDF is short for term frequency-inverse document frequency. It's a feature vectorization method that converts textual data to numeric form. TF-IDF reflects the importance of a term to a document in the corpus. Its formula [11] is given below.

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D) \quad (1)$$

where

- t denotes a term
- d denotes a document
- D denotes the corpus

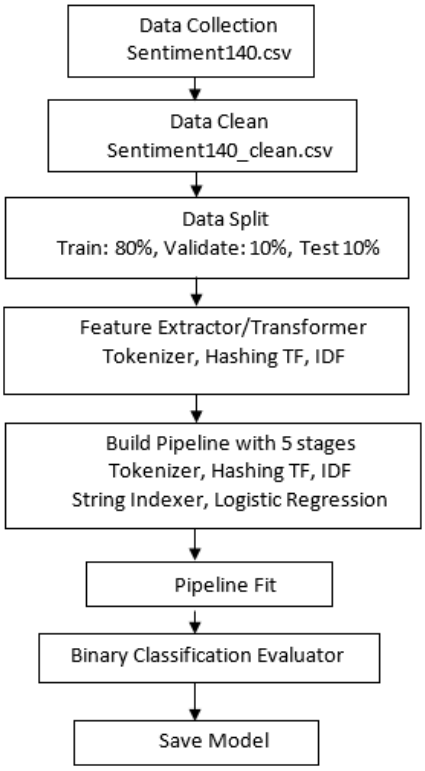


Figure 1: Flow Chart of Tf-Idf Logistic Regression Model

- TF(t, d) is the number of times that term t appears in document d

$$IDF(t, D) = \log \frac{|D| + 1}{DF(t, D) + 1} \quad (2)$$

where:

- $|D|$ is the total number of document in the corpus
- DF(t, D) is the number of documents that contains term t

Pyspark.ml package provides two methods (HashingTF and CountVectorizer) to generate the term frequency. We compare the two methods in the experiment section.

Implement Detail

1. **Transformers**: The transformer can transform data by appending a new column in DataFrame. Our TF-IDF Logistic Regression model involves 3 transformers:

- Transformer HashingTF takes a list of tokenized text and return a vector of with counts, parameter numFeatures is set as 2^{16}

- Transformer CountVectorizer has the same function as HashingTF to compute term frequecy for IDF calculation. However, they have two important differences [12]: 1) CountVectorizer is partially reversible while HashingTF is irreversible; 2)For how to filter top features: HashingTF uses dimensionality reduction with possible collisions while CountVectorizer discards infrequent features.
- Transformer IDF computes an Inverse Document Frequency for a list of documents, parameter minDocFreq is set as 5.

2. **Logistic Regression**: Since sentimental analysis is a binary classification problem, we chose Logistic Regression(LR) as our Estimator. LR uses a logit function to compute the probability of an observation belonging to negative or positive. We set maxIter as 100 and regParam as 0.01.
3. **Pipeline**: Pipeline is a end-to-end process with distinct stages. Our pipeline contains five stages: Tokenizer, HashingTF/CounterVectorizer, IDF, String Indexer and LR. The pipeline is saved for later use. For example, we will reload the pipeline to use it in our web application of real-time monitoring sentiment in the future work.

Experiments

Environment Setting

In this project, Spark actually works in local mode with my desktop computer. Environment setting is shown as below.

name	version	parameter	val
Ubuntu	16.04	spark.executor.memory	8g
Spark	2.2.2	spark.cores.max	3
PySpark	2.2.2	spark.driver.memory	8g
Python	3.6.6	spark.executor.cores	3

However, as Spark's local mode is fully compatible with the cluster mode, codes written locally can be run on a cluster with just a few additional steps.

Results and analysis

We output the result of feature transformer by the show() function of DataFrame. They are shown in the below two tables. Since we have only 1 document in our corpus, column tf and features have the same value. 65536 in column tf is the number of features that equals the value (2^{16}) we set before. For HashingTF, the elements in tf vector is in ascending order while they are in descending for CountVectorizer. This follows the reversibility of CoutVectorizer.

text	words	tf	features
not the whole crew	[not, the, whole,...	(65536,[2,25,427,...	(65536,[2,25,427,...
oh dear were you ...	[oh, dear, were, ...	(65536,[2,7,12,34...	(65536,[2,7,12,34...
i cry my asian ey...	[i, cry, my, asia...	(65536,[0,1,4,24,...	(65536,[0,1,4,24,...
too bad i won t b...	[too, bad, i, won...	(65536,[0,4,6,13,...	(65536,[0,4,6,13,...
not forever see y...	[not, forever, se...	(65536,[7,25,64,1...	(65536,[7,25,64,1...

Figure 2: Result of TF-IDF Logistic Regression by HashingTF

text	words	tf	features
not the whole crew	[not, the, whole,...	(65536,[8026,3398...	(65536,[8026,3398...
oh dear were you ...	[oh, dear, were, ...	(65536,[2830,9318...	(65536,[2830,9318...
i cry my asian ey...	[i, cry, my, asia...	(65536,[556,8436,...	(65536,[556,8436,...
too bad i won t b...	[too, bad, i, won...	(65536,[11430,127...	(65536,[11430,127...
not forever see y...	[not, forever, se...	(65536,[8026,1198...	(65536,[8026,1198...

Figure 3: Result of TF-IDF Logistic Regression by CounterVectorizer

We use three metrics to compare models using HashingTF and CountVectorizer. Metrics areaUnderROC and areaUnderPR are provided by BinaryClassificationEvaluator in Pyspark. For metric accuracy, we compute it based on column label and prediction.

Metric	HashingTF	CounterVectorizer
AreaUnderROC	0.8610	0.8668
AreaUnderPR	0.8521	0.8559
Accuracy	0.7903	0.7971
Train time	43.79	52.02

The above table shows that TF-IDF Logistic Regression by CounterVectorizer has a little bit better performance than by HashingTF.

4.2 Word2Vector CNN Model on Keras

As the fast development of deep learning, Keras, a Python Deep Learning library, provides an effective way to solve classification problem. For the third sentimental analysis model in this project, we design a Word2Vector Convolution Neural Network (CNN)[8] on Keras and use TensorFlow (an

open source deep learning framework) as its back-end engine.

Our Word2Vector CNN Model architecture consists of 1 embedding layer, 3 convolution layers, 3 global Max Pooling layers, 1 dense layer, 1 Dropout layer and 1 dense layer and the sigmoid output layer. It is shown as below.

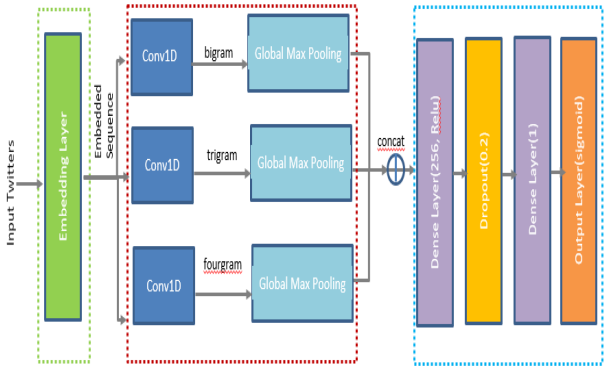


Figure 4: High Level of Network Architecture

Implement Detail

Before we can feed data to our Word2Vector CNN model, some preprocess works are needed:

1. **Train/test data split**: To avoid over-fitting, we use train_test_split function from sklearn to sample 70% data for training while holding out 30% of data for testing our model. The sample ratio of 70% has the best performance (the detail is explained in the experiment section) in our model.

2. **Tokenizer and pad sequence**: To format our text samples and labels into tensors that can be fed into CNN model, we rely on Keras Tokneizer and pad_sequences utilities. Tokenizer splits each word in a sentence; then we call texts_to_sequence function to get a sequential representation of each sentence. Each word in twitter now is represented as a number. Because each sentence has varying length, we use pad_sequences function to transform the data with the same length. We iterate through all training data to calculate the maximum (40) of sentence length of training data, thus set parameter maxlen of pad_sequences as 45.

3. **Embedding index**: Use "word2vec" technique to compute word embedding.Firstly, load pre-trained Continuous Bag Of Words and skip-gram models to provide the first initialization guideline for the embedding layer. Then, concatenate vectors of two word2vector models to build the embedding_index vector.

Embedding Layer

"Word embedding" [9] are a family of natural language processing techniques aiming at mapping semantic meaning into embedding space. This is done by associating a numeric vector to every word in a dictionary, such that the distance (e.g. L2 distance or more commonly cosine distance) between any two vectors would capture part of the semantic relationship between the two associated words. The following shows how we build the embedding layer:

First, prepare an "embedding matrix" which will contain at index i the embedding vector for the word of index i in our embedding index.

Then, load this embedding matrix into a Keras Embedding layer, set to be frozen (its weights, the embedding vectors, will not be updated during training).

Experiments

Environment Setting

we deploy our model on my desktop computer with the environment setting shown as below. And the batch size is 128, the number of epoch is 4, the dropout rate is 0.2.

name	version
Ubuntu	16.04
GPU	GeForce GTX 970
CUDA	toolkit 9.0, Cudnn 7.0
TensorFlow-gpu	1.5.0
Keras	2.2.4

Results and analysis

The summary of our model is shown as below.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 45)	0	
embedding_1 (Embedding)	(None, 45, 200)	10000000	input_1[0][0]
conv1d_1 (Conv1D)	(None, 44, 128)	51328	embedding_1[0][0]
conv1d_2 (Conv1D)	(None, 43, 128)	76928	embedding_1[0][0]
conv1d_3 (Conv1D)	(None, 42, 128)	102528	embedding_1[0][0]
global_max_pooling1d_1 (GlobalM	(None, 128)	0	conv1d_1[0][0]
global_max_pooling1d_2 (GlobalM	(None, 128)	0	conv1d_2[0][0]
global_max_pooling1d_3 (GlobalM	(None, 128)	0	conv1d_3[0][0]
concatenate_1 (Concatenate)	(None, 384)	0	global_max_pooling1d_1[0][0] global_max_pooling1d_2[0][0] global_max_pooling1d_3[0][0]
dense_1 (Dense)	(None, 256)	98560	concatenate_1[0][0]
dropout_1 (Dropout)	(None, 256)	0	dense_1[0][0]
dense_2 (Dense)	(None, 1)	257	dropout_1[0][0]
activation_1 (Activation)	(None, 1)	0	dense_2[0][0]

Total params: 10,329,601
Trainable params: 329,601
Non-trainable params: 10,000,000

Figure 5: Summary of word2vector CNN

To validate whether the input data are correctly

divided, we print out the result of data split showed as below:
1. Train set has total 1117727 entries with 49.98% negative, 50.02% positive
2. Validation set has total 239513 entries with 50.14% negative, 49.86% positive
3. Test set has total 239513 entries with 50.00% negative, 50.00% positive

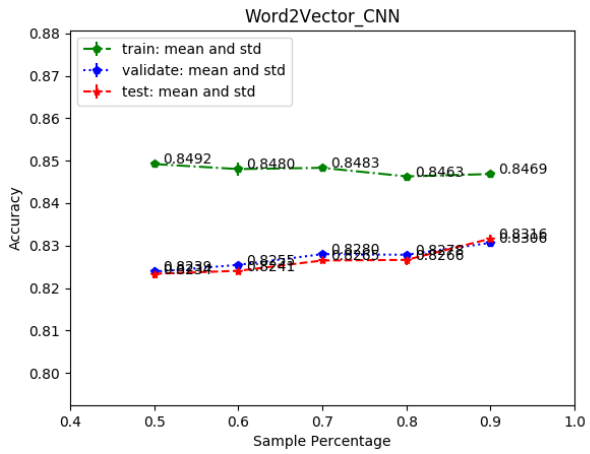


Figure 6: Accuracy of CNN with Different Train Size

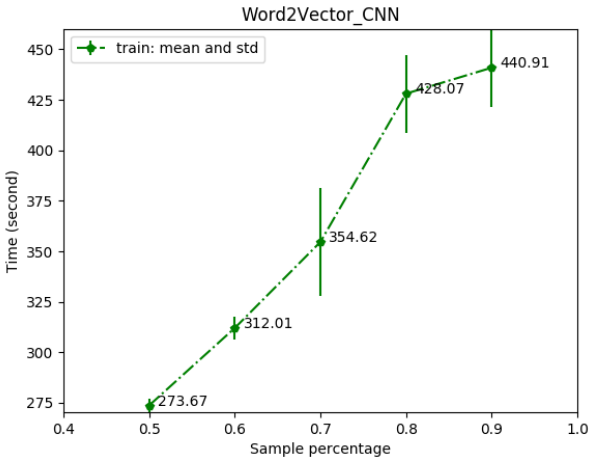


Figure 7: Time of CNN with Different Train Size

Though, there are some rule-of-thumb [7], for how to divide a data set into training and testing, such as 80/20 for the traditional machine learning method, 99.5/0.5 for deep learning with big data. We design an algorithm (keras_word2vec_cnn_compare.py) for finding the best data split ratio for our model.

First, use only 50% of the data point that are reserved for training, then 60%, 70%, 80%, 90%; Secondly, compare the performance of our model using different training data size Last, run our model 3 times for each sample rate. Figure 3-6 show accuracy/training time with different train/epoch size.

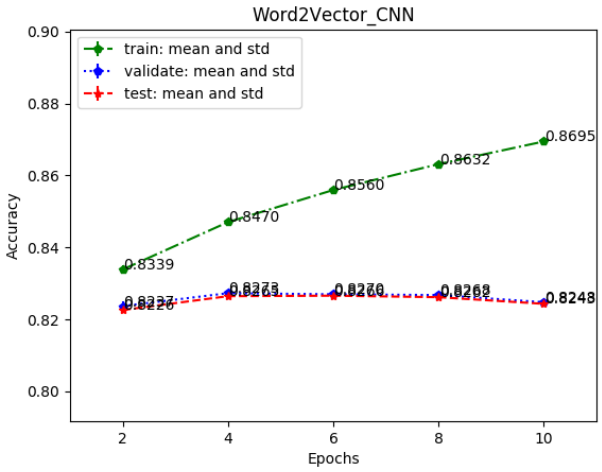


Figure 8: Accuracy of CNN with Different Epochs

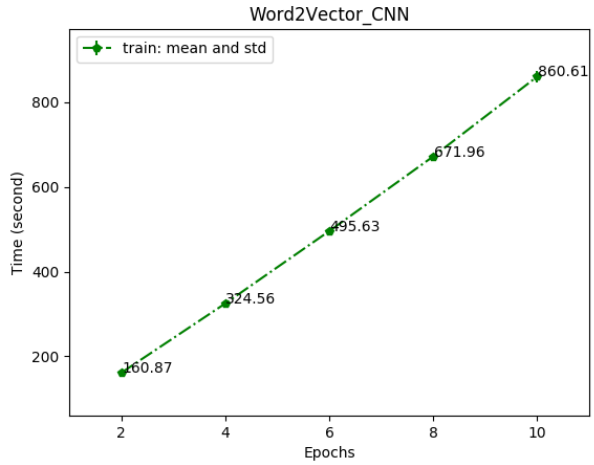


Figure 9: Time of CNN with Different Epochs

As the training size increases, the training time grows. The highest accuracy is around 84% using 90% of the training samples. However, 70% training samples can give a higher accuracy (around 83%) with relatively shorter training time. As the epoch size increases, the training time linearly increases. The highest accuracy is around 82% using 4 epochs.

5 Hashtag Recommendation (Emoji Prediction Model)

We treat and compare three models to predict emoji for text messages. The way we tokenize, pad sequence, and how to prepare data for embedding layer are mentioned in section 5.3. The difference is that the emoji prediction use Google pretrained word2vec because our Emoji data is not as big as Tweet data. To use more complete word2vec, we choose to use Google pretrained word2vec instead.

5.1 Word2Vector Neural Networks

To compare with LSTM and CNN models, we use a shallow neural network with one hidden layer as our baseline. We introduce our deep learning Word2Vector NN Model architecture that consists of 1 embedding layer, 1 flatten layer, 1 dense layer and the softmax output layer.

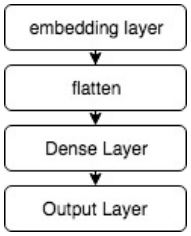


Figure 10: High Level of NN Architecture

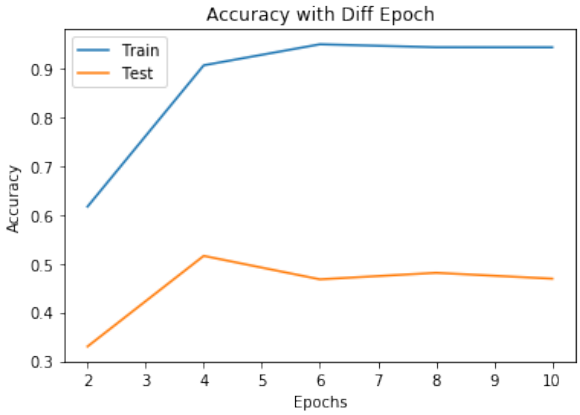


Figure 11: Accuracy of NNs

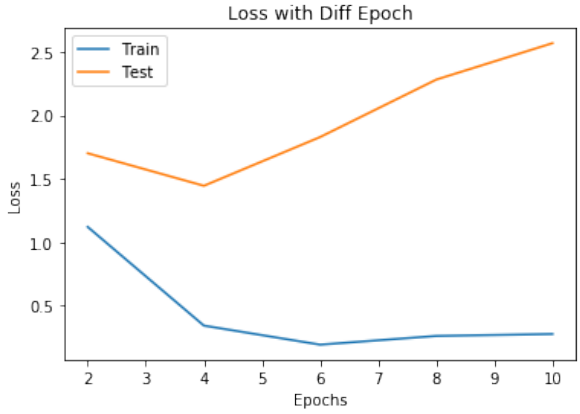


Figure 12: Loss of NNs

5.2 Word2Vector LSTM Model

Long Short Term Memory(LSTM) networks are a special kind of RNN. They are able to learn long term dependencies and work well on different problems. We introduce our deep learning Word2Vector LSTM Model architecture that consists of 1 embedding layer, 1 LSTM layer, 1 Dropout layer and the softmax output layer.

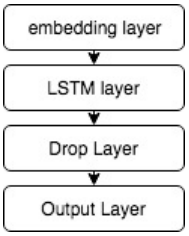


Figure 13: High Level of LSTM Architecture

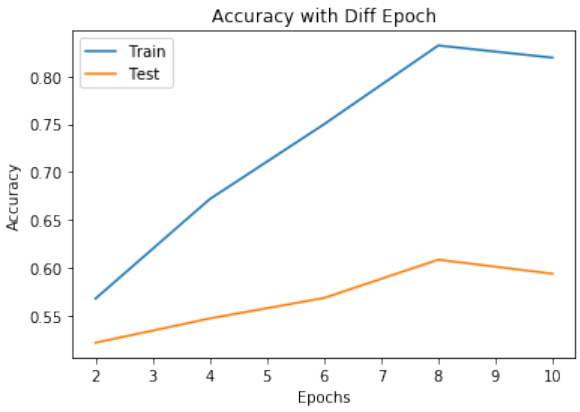


Figure 14: Accuracy of LSTM

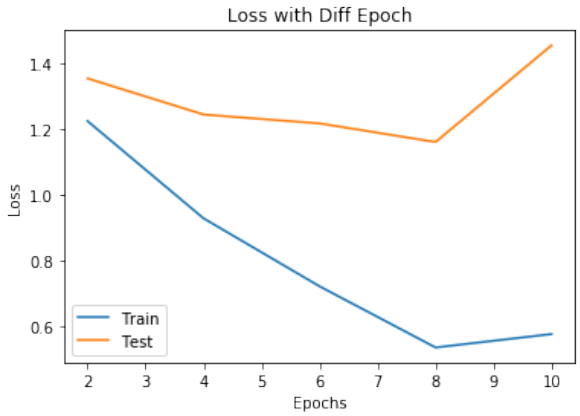


Figure 15: Loss of LSTM

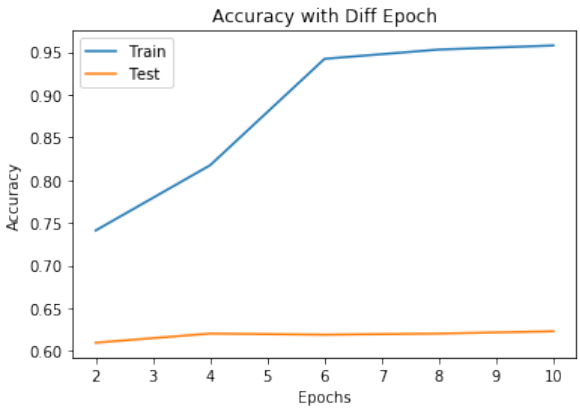


Figure 16: Accuracy of CNN

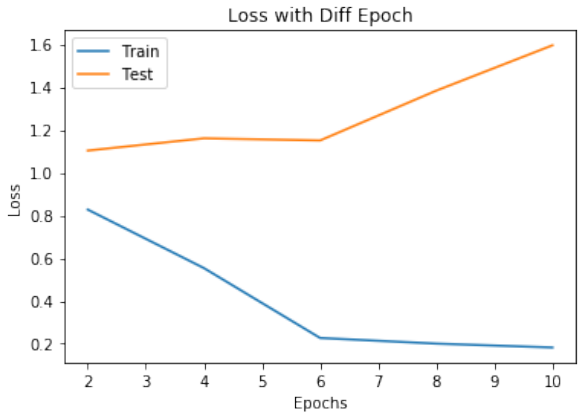


Figure 17: Loss of CNN

5.3 Word2Vector CNN Model

We use the similar model we just introduce into section 4.3. The only different is Emoji prediction is a multi-class problem, so we use softmax function instead of sigmoid function. Also, the emoji prediction use Google pre-trained word2vec because the data is not large enough. Therefore, we choose to use Google pre-trained word2vec instead. Experiments is in Figure 16 and 17.

5.4 Result and Analysis

We split the dataset into 90% for training and 10% for testing. We compare the three models and find that CNN model and LSTM model are much better than NNs model. Also, CNN model is a little better than LSTM model. In the last metric, we can see the training time of 6 epochs of the three models. CNN is much faster than LSM model. Even though NNs is a little faster than CNN, however, CNN is a better solution compared with others overall.

	NNs	CNN	LSTM
Accuracy	0.4892	0.6013	0.5762
Training Time(s)	29.77	40.69	75.69

6 Hashtag Recommendation (Topic Prediction Model)

Latent Dirichlet Allocation (LDA), Figure 19, is a generative statistical topic model. It is used to discover the topics in texts. Also, it can do the information retrieval efficiently. In LDA, each document may be viewed as a mixture of various topics where each document is considered to have a set of topics that are assigned to it via LDA. In our way, after creating a dictionary containing the number of times a word appears in the training set, we keep select the first 2000 common words as our new dictionary to train the LDA model. We train our lda model using gensim.models.LdaMulticore with num_topics is 75.

Tweet Text	Suggested Words	Suggested Emoji
when thieves broke into my house at night, and held my wife and me on gun point for at least ten minutes and took away a lot of property.	#worry #house #bad #night	😨😨😨
International sports events won by my favourite national team or player brings me joy when india won the world cup cricket match.	#win #game #luck #good #omg #ha #fan #believe	😁 😁 😁
I was sitting in a restaurant with friends. They asked me something which they thought I should know. Actually I know it but at that time I was not able to remember it.	#eat #god #remember #hopefully	😳😳😳

Figure 18: We highlight some suggested words of Tweet

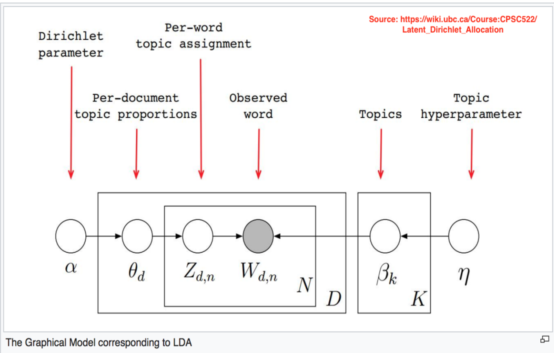


Figure 19: LDA Model

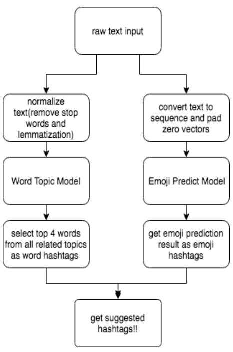


Figure 20: Input Output Flow

```
                                   text  \
0                     when thieves broke into my house at night, and held my wife and me on gun point for
at least ten minutes and took away a lot of property.
1                            International sports events won by my favourite national team or player bri
ngs me joy when india won the world cup cricket match.
2  I was sitting in a restaurant with friends. They asked me something which they thought I should know. Actually I k
now it but at that time I was not able to remember it.


                   suggested_words  \
0                                                   [take, rock, walk, ill, buy, stuff, busy, worry, a
way, person, soo, hug, house, sweet, pick, award, bad, welcome, hehe, dream, add, later, picture, plan, wait, minute,
yea, nap, morning, good, night, bed]
1  [saw, mom, tea, lil, win, hang, set, half, sorry, game, hear, kid, year, old, luck, good, week, book, class, have
n, man, omg, ha, fan, world, believe, blue, internet, cool, ur, post, update, twitter, finally, funny, real, lunch, b
ring, ah, site, miss, write, boy, far]
2                                                          [know, let, money, perfect, hey,
idea, course, reply, meet, ask, second, keep, friend, best, th, lovely, movie, eat, see, god, follow, remember, link,
english, way, sit, hopefully, room]

   suggested_emoji
0                1
1                0
2                5
```

Figure 21: The real output of our mix-model

7 Mix-model Experiment highlight

Our hashtag recommendation model combine unsupervised word topic model and supervised emoji predict model, Figure 20. Our result we would contain one emoji and a list of words which are considered as hashtag suggestion for a tweet. We give some real outputs of tweet example in Figure 21. The first column is input texts, the second column is suggested words, and the last column is suggested emoji. We highlight some meaningful words in Figure 18.

8 Visualization and Application

For visualization and application part, we implement sentiment analysis through designing a web application for real-time monitoring sentiment on Twitter for keyword queries. The web page allows user to enter some keywords they want to monitor. After entering keywords, the system will scratch Tweets containing these keywords on Twitter and show the corresponding sentiment score in the chart.

8.1 System Architecture

There are two part in our system. First part is offline training model. In this part, we are going to use preprocessed dataset to train a model for sentiment analysis.

In the second part, we are going to collect real-time data using Kafka and then classify sentiment of Tweets through Spark Streaming based on the training model. Our web server will run on Flask and interact with Angular-based frontend to visualize streaming data. For now, our system connected with Stanford Sentiment Analysis local

server for demo instead of our own models temporarily.

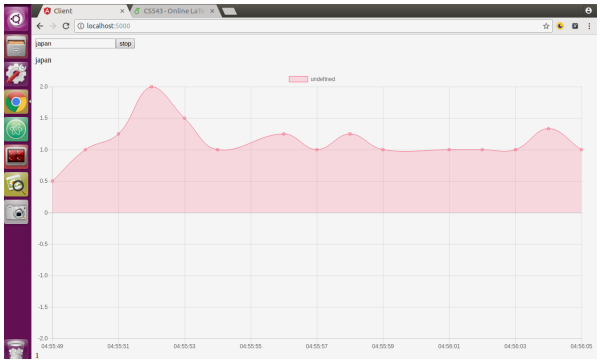


Figure 22: x axis is time, y axis is sentiment scores

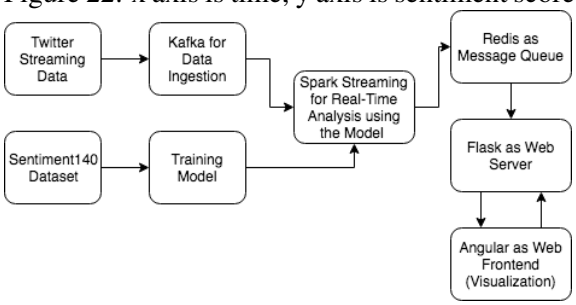


Figure 23: Tweet Sentiment Score System Architecture

9 Conclusion and Future Work

Sentimental analysis model

TF-IDF Logistic Regression has around 80% accuracy. The accuracy is slightly lower. In the next stage, we will try bigram and trigram with more features to boost the model performance. The accuracy of Word2vector CNN model is around 83%. According to Yin[14], Gated Recurrent Unit(GRN) can achieve better performance for sentimental analysis. We will try GRN in the next stage.

Hashtag Recommendation

In this project, we propose a new approach to recommend hashtags. The method we proposed considered more aspects of Tweets. The contribution of our work is that we break the hashtag recommendation problem into three part, making texts easier to analyze and predict.The advantage of our approach is that it can recommend hashtags considered three parts: emoji sentiment, text topic, and current trends. For future work:

1. Emoji Sentiment Models: We can add more recommending emojis through collecting more Tweets not just the current seven emojis we predict.

2. Topic Models: We can use more high quality Tweets and personalize topic model for word topic recommendation, improving the reliability of recommendation.

3. Trending Word Hashtags: Trending words recommendation can also be considered as a short-term label problem. For example, we can collect the top 50 hashtags people uses the most in the recent three days. Then, we can train a supervised model to predict the hashtags for a new tweet.

10 Acknowledge

The libraries we use: pySpark, Gensim, Keras, Pandas, nltk, numpy, matplotlib, Tweepy, socketio, redis. The third frameworks we use: Angular and Flask. We use StanforNLP as a part of module in our web application, We followed Susan Li and the Rickest Ricky on towardsdatascience.com to learn how to through Keras and Gensim. we followed the paper's idea to build a LDA model for word hashtag prediction: "Using topic models for Twitter hashtag recommendation", *in Proceedings of the 22nd International Conference on World Wide Web (WWW '13 Companion)*[1] We thank for their contribution of knowledge.

References

[1] Frderic Godin, Viktor Slavkovikj, Wesley De Neve, Benjamin Schrauwen, and Rik Van de Walle, "Using topic models for Twitter hashtag recommendation", *in Proceedings of the 22nd International Conference on World Wide Web (WWW '13 Companion)*, 2013

[2] Working with Streaming Twitter Data Using Kafka, *https://www.bmc.com/blogs/working-streaming-twitter-data-using-kafka/*

[3] minrk/findspark, GitHub, *https://github.com/minrk/findspark*

[4] sherlockjjj/T-Watch, GitHub, *https://github.com/sherlockjjj/T-Watch*

[5] StanfordNLP, Stanford University, *http://www-cs-faculty.stanford.edu/~uno/abcde.html*

[6] tthustla/setiment-analysis-pyspark, GitHub, *https://github.com/tthustla/setiment_analysis_pyspark*

[7] data split rule, StackOverFlow, *https://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validatio*

[8] cnnSentimentcnnSentimentStructure, *https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-11-cnn-word2vec-41f5e28eda74*

[9] word embedding, *https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html*

[10] The Rickest Ricky, Another Twitter sentiment analysis with Python *https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-11-cnn-word2vec-41f5e28eda74*

[11] tfidf, *https://spark.apache.org/docs/latest/ml-features.html#tf-idf*

[12] hashtfCountervector, *https://stackoverflow.com/questions/35205865/what-is-the-difference-between-hashingtf-and-countvectorizer-in-spark*

[13] Susan Lee, Topic Modeling and Latent Dirichlet Allocation (LDA) in Python *https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24*

[14] Yin, Wenpeng and Kann, Katharina and Yu, Mo. Comparative study of cnn and rnn for natural language processing. *arXiv preprint* :1702.01923 .

A Appendix

543-sentiment-analysis

FINISHED

```
import pyspark as ps
from pyspark.sql import SQLContext
from pyspark.sql.functions import *
from pyspark.sql import SparkSession
import pandas
import matplotlib.pyplot as plt
import re
```

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:40:18 PM.

FINISHED

```
sc = ps.SparkContext.getOrCreate()
sqlContext = SQLContext(sc)

# load data sentiment140_clean.csv
data = sc.textFile('hdfs:///user/kc1026/training.1600000.processed.noemoticon.csv')
df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('hdfs:///user/kc1026/training.160(
    .processed.noemoticon.csv')
```

```
... 18/12/17 12:40:23 WARN FileStreamSink: Error while looking for metadata directory.
18/12/17 12:40:23 WARN FileStreamSink: Error while looking for metadata directory.
[Stage 257:>                                                        (0 + 1) / 2]
[Stage 257:============================>                          (1 + 1) / 2]
```

Took 9 sec. Last updated by kc1026 at December 17 2018, 12:40:27 PM.

FINISHED

```
# rename column
df = df.withColumnRenamed("0", "target").withColumnRenamed("1467810369", "id") .withColumnRenamed("Mon Apr 06 22:19:45 PDT 2009", "date'
    ).withColumnRenamed("NO_QUERY", "flag").withColumnRenamed("_TheSpecialOne_", "user").withColumnRenamed("@switchfoot http://twitpic.c
    Awww, that\'s a bummer.  You shoulda got David Carr of Third Day to do it. ;D","text")
```

Took 8 sec. Last updated by kc1026 at December 17 2018, 12:40:27 PM.

FINISHED

Top 20 Users

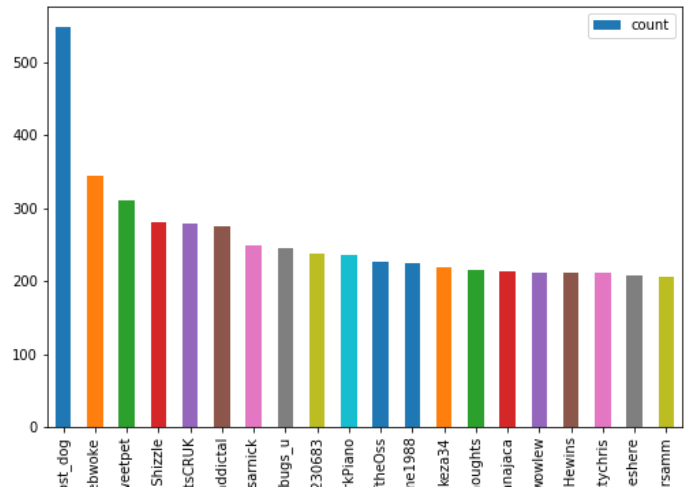
We hightlight the top20 users who tweets the most in the dataset

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:41:18 PM.

FINISHED

```
top20Users = df.groupby('user').count().orderBy('count', ascending=False).limit(20)
top20U = top20Users.toPandas()
top20U.plot(kind='bar',x="user", y= "count")
```

```
[Stage 258:>                                                        (0 + 1) / 2]
[Stage 258:>                                                        (0 + 2) / 2]
[Stage 258:============================>                          (1 + 1) / 2]
[Stage 259:==================>                                   (70 + 2) / 200]
[Stage 259:============================>                         (103 + 2) / 200]
[Stage 259:====================================>                 (143 + 2) / 200]
[Stage 259:============================================>         (184 + 2) / 200]

<matplotlib.axes._subplots.AxesSubplot object at 0x7f35b2705780>
```



Took 18 sec. Last updated by kc1026 at December 17 2018, 12:40:45 PM.

Top 20 Common Words FINISHED

Based on the dataset, we remove special characters, stop words and numbers

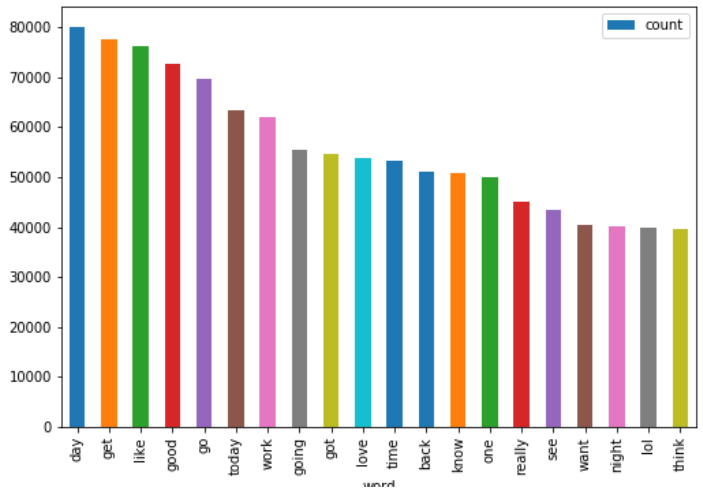
Then do the word count, and sort by frequency

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:40:19 PM.

FINISHED

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))
stop_words.add('I')
stop_words.add('http')
stop_words.add('com')
stop_words.add('amp')
stop_words.add('u')
stop_words.add('im')
stop_words.add('quot')

top20Words = data.map(lambda line: line.split(",", 5)) \
.map(lambda x: x[5]) \
.flatMap(lambda x: re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x)) \
.filter(lambda word: word != '') \
.map(lambda word: (word, 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] not in stop_words and not x[0].isnumeric()) \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)

top20W = top20Words.toPandas()
top20W.plot(kind='bar',x="word", y= "count")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /ilab/users/kc1026/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
... ... ... ... ... ... ... ... ...
[Stage 260:>                                                        (0 + 1) / 2]
[Stage 260:>                                                        (0 + 2) / 2]
[Stage 260:============================>                           (1 + 1) / 2]
[Stage 261:>                                                        (0 + 2) / 2]

[Stage 263:>                                                        (0 + 2) / 2]

[Stage 265:>                                                        (0 + 2) / 2]

<matplotlib.axes._subplots.AxesSubplot object at 0x7f35b2c61f28>
```



Took 40 sec. Last updated by kc1026 at December 17 2018, 12:41:08 PM.

top 20 common postive word FINISHED

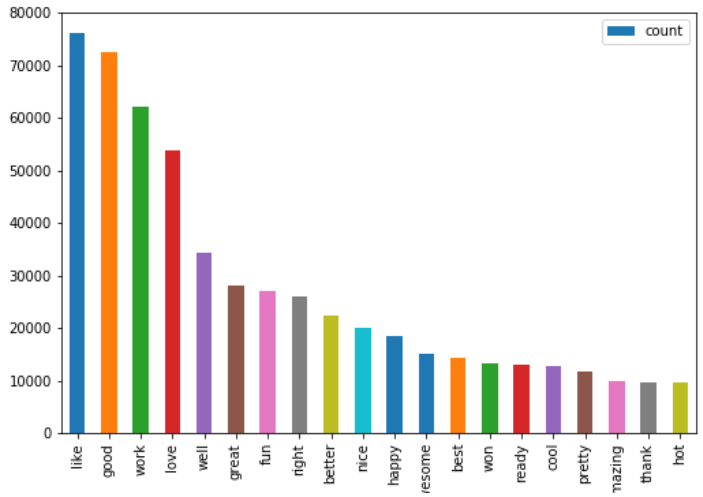
We use the positive sentiment dataset: http://ptrckprry.com/course/ssd/data/positive-words.txt (http://ptrckprry.com/course/ssd/data/positive-words.txt)
Based on the dataset, we filter the postive words and count them
The chart as below is the top20 postive words

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:40:19 PM.

FINISHED

```
p_set = set()
f = open('/ilab/users/kc1026/Documents/cs543/p_words_list.txt', 'r')
word = f.readline()
while word:
    p_set.add(word.strip())
    word = f.readline()


top20PosWords = data.map(lambda line: line.split(",", 5)) \
.map(lambda x: x[5]) \
.flatMap(lambda x: re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x)) \
.filter(lambda word: word != '') \
.map(lambda word: (word, 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] in p_set) \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)

top20Pos = top20PosWords.toPandas()
top20Pos.plot(kind='bar',x="word", y= "count")
```

```
... ... ... ... ... ... ... ... ... ... ... ... ...
[Stage 270:>                                                        (0 + 1) / 2]
[Stage 270:>                                                        (0 + 2) / 2]
[Stage 270:============================>                            (1 + 1) / 2]

<matplotlib.axes._subplots.AxesSubplot object at 0x7f35b2329320>
```



Took 44 sec. Last updated by kc1026 at December 17 2018, 12:41:30 PM.

Top 20 Common Negative Words FINISHED

We use the positive sentiment dataset: http://ptrckprry.com/course/ssd/data/negative-words.txt (http://ptrckprry.com/course/ssd/data/negative-words.txt)

Based on the dataset, we filter the negaitve words and count them

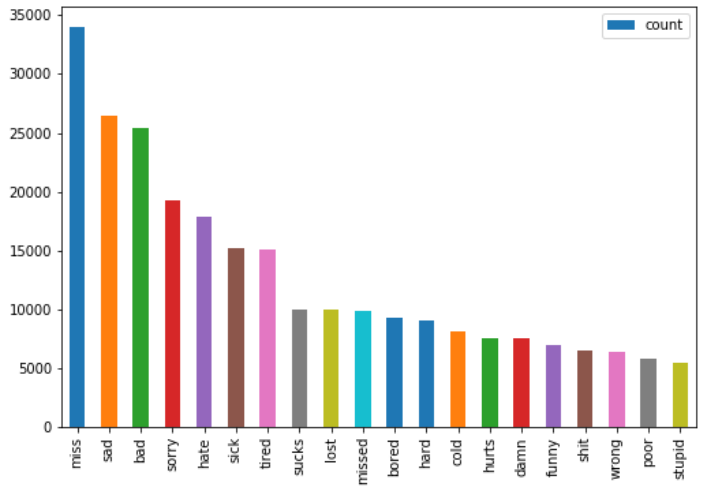
The chart as below is the top20 negative words

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:40:19 PM.

FINISHED

```
n_set = set()
f = open('/ilab/users/kc1026/Documents/cs543/n_words_list.txt', 'r')
word = f.readline()
while word:
    n_set.add(word.strip())
    word = f.readline()


top20NegWords = data.map(lambda line: line.split(",", 5)) \
.map(lambda x: x[5]) \
.flatMap(lambda x: re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x)) \
.filter(lambda word: word != '') \
.map(lambda word: (word, 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] in n_set) \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)

top20Neg = top20NegWords.toPandas()
top20Neg.plot(kind='bar',x="word", y= "count")
```

```
18/12/17 12:45:25 WARN TaskSetManager: Lost task 1.3 in stage 283.0 (TID 753, data3.cs.rutgers.edu, executor 2): TaskKilled (stage cancell
ed)
... ... ... ... ... ... ... ... ... ... ... ... ...
[Stage 284:>                                                        (0 + 1) / 2]
[Stage 284:>                                                        (0 + 2) / 2]
[Stage 284:============================>                           (1 + 1) / 2]

[Stage 289:>                                                        (0 + 2) / 2]

<matplotlib.axes._subplots.AxesSubplot object at 0x7f35b26e0ba8>
```



Took 22 sec. Last updated by kc1026 at December 17 2018, 12:46:56 PM.

READY

543-emoji-analysis

FINISHED

```
import pyspark as ps
from pyspark.sql import SQLContext
from pyspark.sql.functions import *
from pyspark.sql import SparkSession
import pandas
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:19 PM.

FINISHED

Type of Labels

there are seven kind of emojis in our dataset:

happy, fear, anger, digust, sadness, guilt, shame

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:19 PM.

FINISHED

```
# load file
sc = ps.SparkContext.getOrCreate()
data = sc.textFile('hdfs:///user/kc1026/emoji.csv')
```

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:19 PM.

FINISHED

Happy Emoji

Here's the way we use to count the words:

1. We split the text, and then remove stop words and special characters.
2. Count each word and pick up the top 20 common word which is relating to the happy emoji.

In the text, we highlight some strong emotional words relating to the happy emoji:

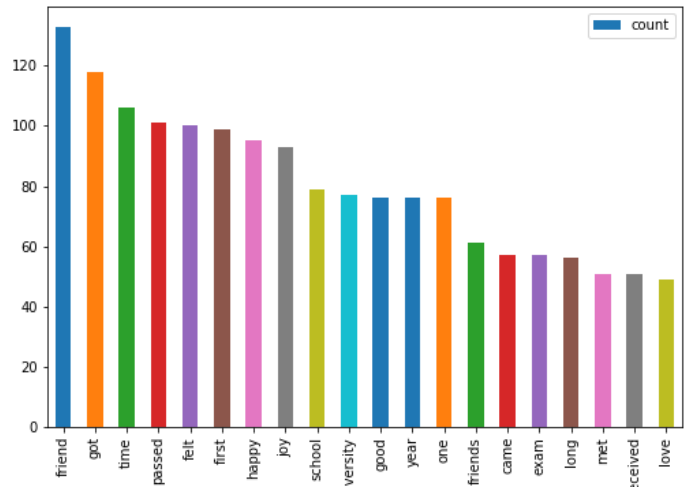
joy, good, love

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:19 PM.

FINISHED

```
import re
top20_happy_words = data.map(lambda line: line.rsplit(",", 1)) \
.map(lambda x: (x[1], re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x[0]))) \
.flatMapValues(lambda x: x) \
.filter(lambda pair: pair[0] == '0') \
.map(lambda x: x[1]) \
.map(lambda word: (word.lower(), 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] not in stop_words and x[0] != '') \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)


top20_happy = top20_happy_words.toPandas()
top20_happy.plot(kind='bar',x="word", y= "count")
```

```
... ... ... ... ... ... ... ... ... ... <matplotlib.axes._subplots.AxesSubplot object at 0x7f35b27301d0>
```



Took 4 sec. Last updated by kc1026 at December 17 2018, 12:29:23 PM.

Fear Emoji FINISHED

We highlight some strong emotional words relating to the anger emoji:

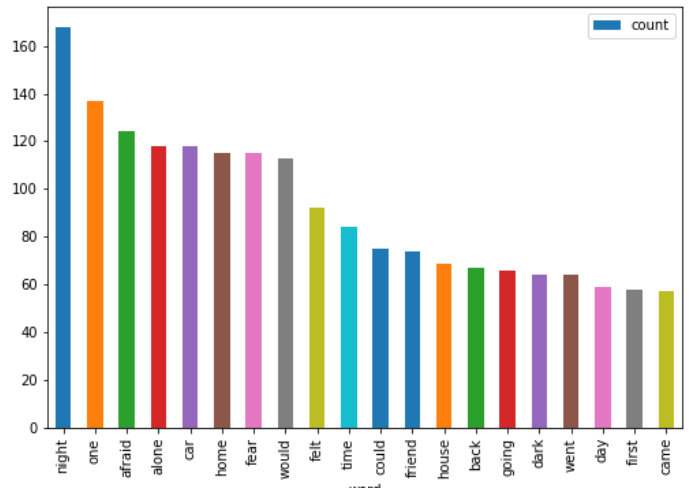
night, afraid, fear

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:19 PM.

FINISHED

```
import re
top20_fear_words = data.map(lambda line: line.rsplit(",", 1)) \
.map(lambda x: (x[1], re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x[0]))) \
.flatMapValues(lambda x: x) \
.filter(lambda pair: pair[0] == '1') \
.map(lambda x: x[1]) \
.map(lambda word: (word.lower(), 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] not in stop_words and x[0] != '') \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)


top20_fear = top20_fear_words.toPandas()
top20_fear.plot(kind='bar',x="word", y= "count")
```

```
... ... ... ... ... ... ... ... ... ... <matplotlib.axes._subplots.AxesSubplot object at 0x7f35b23eaa58>
```



Took 7 sec. Last updated by kc1026 at December 17 2018, 12:29:27 PM.

FINISHED

Anger Emoji

We highlight some strong emotional words relating to the anger emoji:

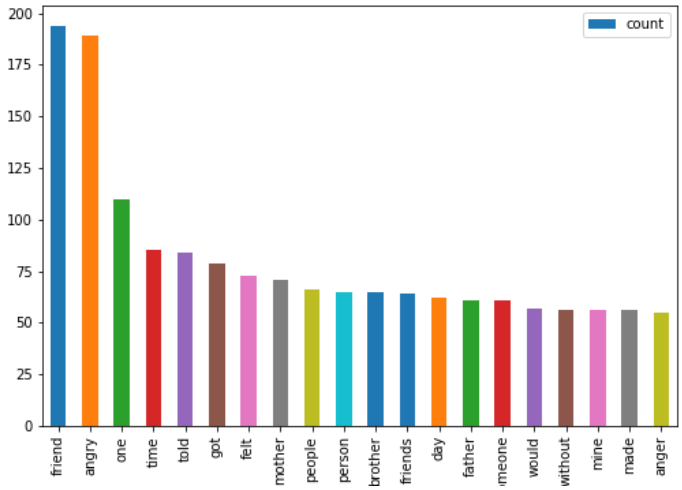
angry, anger

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:20 PM.

FINISHED

```
import re
top20_anger_words = data.map(lambda line: line.rsplit(",", 1)) \
.map(lambda x: (x[1], re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x[0]))) \
.flatMapValues(lambda x: x) \
.filter(lambda pair: pair[0] == '2') \
.map(lambda x: x[1]) \
.map(lambda word: (word.lower(), 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] not in stop_words and x[0] != '') \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)


top20_anger = top20_anger_words.toPandas()
top20_anger.plot(kind='bar',x="word", y= "count")
```

```
... ... ... ... ... ... ... ... ... ... <matplotlib.axes._subplots.AxesSubplot object at 0x7f35b242bcf8>
```



Took 8 sec. Last updated by kc1026 at December 17 2018, 12:29:32 PM.

Sadness Emoji

FINISHED

We highlight some strong emotional words relating to the sadness emoji:

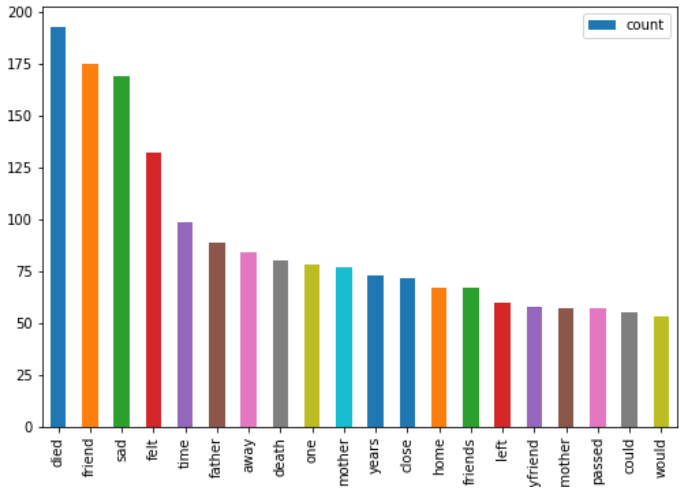
died, sad

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:20 PM.

FINISHED

```
import re
top20_sadness_words = data.map(lambda line: line.rsplit(",", 1)) \
.map(lambda x: (x[1], re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x[0]))) \
.flatMapValues(lambda x: x) \
.filter(lambda pair: pair[0] == '3') \
.map(lambda x: x[1]) \
.map(lambda word: (word.lower(), 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] not in stop_words and x[0] != '') \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)


top20_sadness = top20_sadness_words.toPandas()
top20_sadness.plot(kind='bar',x="word", y= "count")
```

```
... ... ... ... ... ... ... ... ... ... <matplotlib.axes._subplots.AxesSubplot object at 0x7f35b24e9588>
```



Took 9 sec. Last updated by kc1026 at December 17 2018, 12:29:36 PM.

Disgust Emoji FINISHED

We highlight some strong emotional words relating to the disgusted emoji:

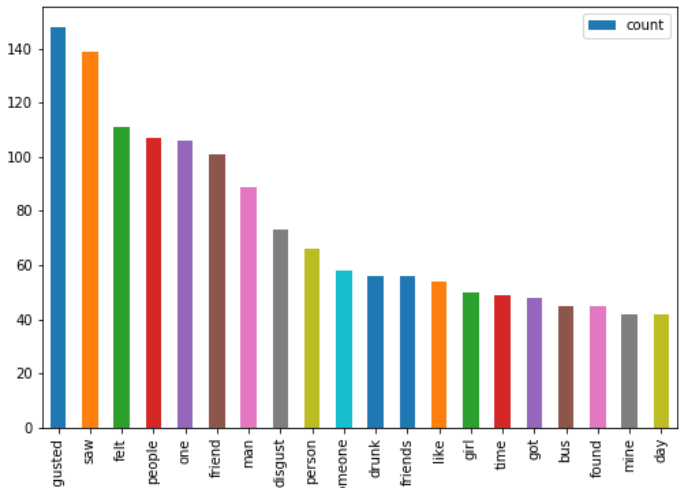
disgusted, disgust, drunk

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:20 PM.

FINISHED

```
import re
top20_digust_words = data.map(lambda line: line.rsplit(",", 1)) \
.map(lambda x: (x[1], re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x[0]))) \
.flatMapValues(lambda x: x) \
.filter(lambda pair: pair[0] == '4') \
.map(lambda x: x[1]) \
.map(lambda word: (word.lower(), 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] not in stop_words and x[0] != '') \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)


top20_digust = top20_digust_words.toPandas()
top20_digust.plot(kind='bar',x="word", y= "count")
```

```
... ... ... ... ... ... ... ... ... ... <matplotlib.axes._subplots.AxesSubplot object at 0x7f35b26bc710>
```



Took 8 sec. Last updated by kc1026 at December 17 2018, 12:29:40 PM.

FINISHED

Shame Emoji

We highlight some strong emotional words relating to the guilt emoji:

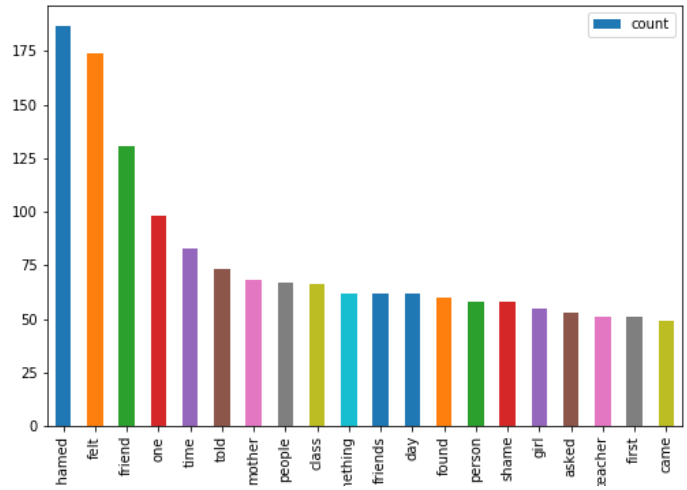
shamed, shame

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:20 PM.

FINISHED

```
import re
top20_shame_words = data.map(lambda line: line.rsplit(",", 1)) \
.map(lambda x: (x[1], re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x[0]))) \
.flatMapValues(lambda x: x) \
.filter(lambda pair: pair[0] == '5') \
.map(lambda x: x[1]) \
.map(lambda word: (word.lower(), 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] not in stop_words and x[0] != '') \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)


top20_shame = top20_shame_words.toPandas()
top20_shame.plot(kind='bar',x="word", y= "count")
```

```
... ... ... ... ... ... ... ... ... ... <matplotlib.axes._subplots.AxesSubplot object at 0x7f35b2bbd048>
```



Took 8 sec. Last updated by kc1026 at December 17 2018, 12:29:44 PM.

Guilt Emoji

FINISHED

We highlight some strong emotional words relating to the guilt emoji:

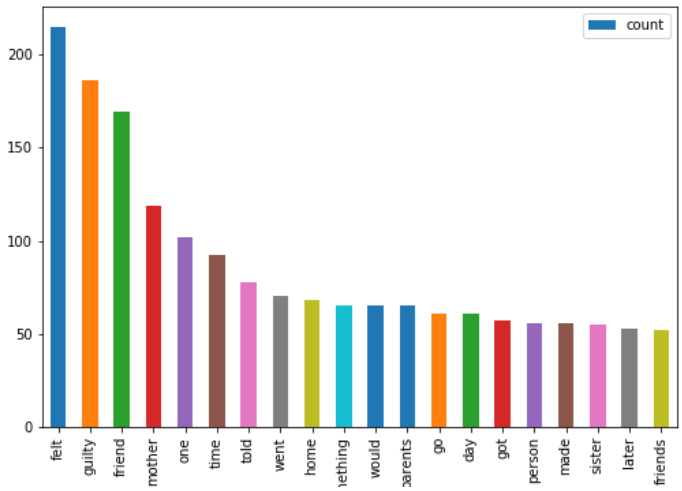
guilty

Took 0 sec. Last updated by kc1026 at December 17 2018, 12:29:20 PM.

FINISHED

```
import re
top20_guilt_words = data.map(lambda line: line.rsplit(",", 1)) \
.map(lambda x: (x[1], re.split(r'[`\-=~!@#$%^&*()_+\[\]{};\'\\:"|<,./<>? ]', x[0]))) \
.flatMapValues(lambda x: x) \
.filter(lambda pair: pair[0] == '6') \
.map(lambda x: x[1]) \
.map(lambda word: (word.lower(), 1)) \
.reduceByKey(lambda a, b: a + b) \
.filter(lambda x: x[0] not in stop_words and x[0] != '') \
.sortBy(lambda x: x[1], ascending=False) \
.toDF(['word', 'count']) \
.limit(20)


top20_guilt = top20_guilt_words.toPandas()
top20_guilt.plot(kind='bar',x="word", y= "count")
```

```
... ... ... ... ... ... ... ... ... ... <matplotlib.axes._subplots.AxesSubplot object at 0x7f35b24d7b38>
```



Took 7 sec. Last updated by kc1026 at December 17 2018, 12:29:47 PM.

FINISHED