Advanced Sentiment Analysis

Shan Guan
Statistics
Columbia University
sg3506@columbia.edu

Yuehan Kong
Statistics
Columbia University
yk2756@columbia.edu

Lin Jiang
Statistics
Columbia University
lj2438@columbia.edu

Abstract—**Sentiment analysis is very useful for us to know the emotional tone behind a series of words. It is essential for market research because companies can see not only customers' attitude about your own products and services, but also their attitude about your competitors. Firstly, we focus on classifying training dataset about Amazon reviews into the positive and negative part, then we make a prediction on test dataset and achieve high prediction accuracy. Three algorithms are used to train our models which are logistic regression, fast text, and universal sentence encoder. Secondly, we find the performance and functionality of universal sentence encoder are amazing. It can deal with the situation that we do not have enough labeled data. To find trump's emotion behind his twitter words, we manually label two hundred of his Twitter data and apply this algorithm as well as transfer learning to make a prediction on three thousand of his unlabeled twitter data. Unsurprisingly, the prediction result looks good.**

Keywords-Amazon reviews, Trump's twitter, sentiment analysis, TFIDF, logistic regression, FastText, universal sentence encoder, transfer learning, Python, Pyspark, Keras

I. Introduction

Nowadays, more and more people tend to do online shopping instead of shopping at physical stores, so previous customer reviews can significantly influence people's decisions of purchasing or giving up this product. Amazon is one of the biggest electric business platforms in the world. A significant number of customers leaves comments for the items they purchased and give the corresponding number of stars which indicates their attitude towards those products. By analyzing those reviews, sellers will know products flaws from customers' perspectives and then try to improve current commodities.

In our case, Amazon customer reviews are classified into two categories: positive and negative. If a customer gives one or two stars, it will be labeled negative. If he or she gives four or five stars, it will be labeled negative. Our training and testing dataset both have labels. We use different algorithms to build classifiers on the training dataset and make predictions on the test dataset. Algorithms we used are logistic regression, FastText and universal sentence encoder. The last one gives us an unexpectedly good outcome.

When we deplore more about the third algorithm, we found that it can solve the dilemma that most technical people have experienced. Manually labeling text data is time-consuming and inefficient, so in real working scenario, we do not always have enough labeled data that allows us to train an optimal classifier. We applied transfer learning with a combination of our pre-trained universal sentence encoder model and the two hundred labeled Trump's tweets to efficiently generate a new universal sentence encoder model with tuned parameters. We manually evaluated the predicted labels as well as the actual content of tweeter and proved that the algorithm works well on Trump's tweets. This proves that transfer learning and universal sentence encoder is a very promising method to perform sentiment analysis when we have limited labeled data for a specific dataset.

II. Related Works

Mandav's blog gives a great overview of Word2Vec, FastText and Universal Sentence Encoder and he applies them on Tweets sentiment analysis. Inspired by his comparison results, our team decided to focus on FastText, Universal Sentence Encoder, in addition to the logistic regression classifier we learned from classes [1].

Furthermore, FastText is developed by Facebook AI Research Lab, and Joulin et al., 2016. explains the algorithm in detail. It helps us understand its character level n-gram embeddings, which is an extension of word2vec, and how it can be averaged together to form good sentence representation [2].

For Universal Sentence Embeddings, we got inspired by Conneau at al., 2017. from Facebook AI Research. The paper explains this supervised learning algorithm can outperform many previous popular unsupervised algorithms that sentence encoders trained in natural language inference can learn sentence representations that capture universally useful features. It also introduces the idea of transfer learning and we decide to apply this transfer learning method on our dataset to verify [3].

III. System Overview

A. System Design

For this project, we use two datasets, Amazon Reviews with labels [8] and Trump's tweets from Jan 2017 to Aug 2018 without labels [9]. We perform overall sentiment analysis on the former: we train three models (Logistic Regression, FastText, Universal Sentence Encoder) on the training dataset and test them on the testing dataset. After the model evaluation, we applied the best-performed model, Universal Sentence Encoder, on the Trump's Tweets dataset for prediction. The Trump's Tweets dataset does not have labels originally. To make a better prediction on Trump's Tweets dataset, we applied Transfer Learning in addition to the Universal Sentence Encoder. We first manually make 200 labels of the Trump's Tweets, with roughly 50%, 50% positive and negative ones, and input these 200 labeled data to our pre-trained Universal Sentence Encoder model and generate a new model specifically for this new dataset. Then we use the new model to predict the rest data in Trump's Tweets (3320 rows). Lastly, we manually check the sentiment prediction score and the real content to evaluate our model.

Below is the visualization of our project system.

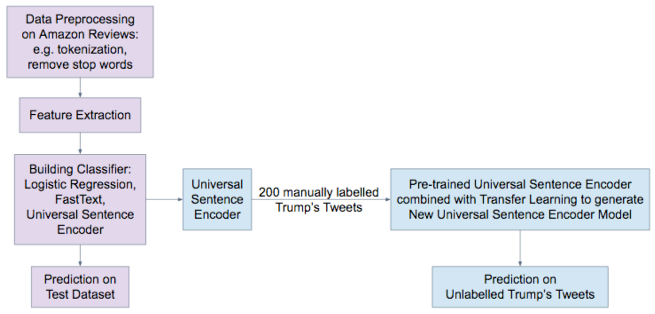

Data Preprocessing on Amazon Reviews: e.g. tokenization, remove stop words
Feature Extraction
Building Classifier: Logistic Regression, FastText, Universal Sentence Encoder
Prediction on Test Dataset
Universal Sentence Encoder
200 manually labelled Trump's Tweets
Pre-trained Universal Sentence Encoder combined with Transfer Learning to generate New Universal Sentence Encoder Model
Prediction on Unlabelled Trump's Tweets

Fig. 1 Project System

B. Dataset:

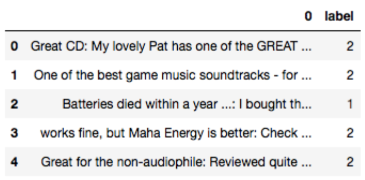
(i) Amazon Reviews for Sentiment Analysis [8]

Size: training: 3,600,000 rows, 1.6GB; testing: 400,000 rows, 185.3MB

Label: 1: negative, score 1 or 2; 2: positive, score 4 or 5

```
0     __label__2 Great CD: My lovely Pat has one of ...
1     __label__2 One of the best game music soundtra...
2     __label__1 Batteries died within a year ...: I...
3     __label__2 works fine, but Maha Energy is bett...
4     __label__2 Great for the non-audiophile: Revie...
```

Fig. 2 Original Unstructured Text File

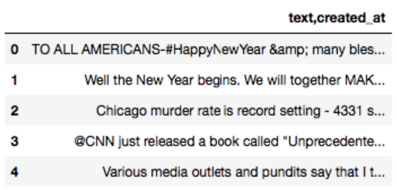


	0	label
0	Great CD: My lovely Pat has one of the GREAT ...	2
1	One of the best game music soundtracks - for ...	2
2	Batteries died within a year ...: I bought th...	1
3	works fine, but Maha Energy is better: Check ...	2
4	Great for the non-audiophile: Reviewed quite ...	2

Fig. 3 Reformatted Text File

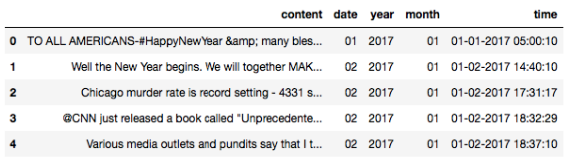
(ii) Trump Twitter Archive [9]

Size: 3520 rows



	text,created_at
0	TO ALL AMERICANS-#HappyNewYear &amp; many bles...
1	Well the New Year begins. We will together MAK...
2	Chicago murder rate is record setting - 4331 s...
3	@CNN just released a book called "Unprecedente...
4	Various media outlets and pundits say that I t...

Fig. 4 Original Unstructured CSV File



	content	date	year	month	time
0	TO ALL AMERICANS-#HappyNewYear &amp; many bles...	01	2017	01	01-01-2017 05:00:10
1	Well the New Year begins. We will together MAK...	02	2017	01	01-02-2017 14:40:10
2	Chicago murder rate is record setting - 4331 s...	02	2017	01	01-02-2017 17:31:17
3	@CNN just released a book called "Unprecedente...	02	2017	01	01-02-2017 18:32:29
4	Various media outlets and pundits say that I t...	02	2017	01	01-02-2017 18:37:10

Fig. 5 Reformatted CSV File

In the original datasets, texts contain lots of useless notations like commas, full mark and so on. These notations are simply replaced before feature extraction.

IV. ALGORITHM

A. Logistic Regression

(i) Data preprocessing

We use Python to read input CSV files about training and test dataset. Firstly, we delete several rows that do not contain any text information in the text column. Secondly, we found that reviews do not only include English words but some Spanish words as well, so we delete some rows that contain Spanish words in the text column. Thirdly, we use Pyspark Regex Tokenization to break the text into individual words based on specific regular expression. Then we remove stop words (e.g.: "the", "who", "which", "at", "on", "I") because those words appear frequently but not carry as much meaning.

(ii) Feature Extraction

In the case of the term frequency TF (t, d), it is used to count the number of times that term t appears in document d. The inverse document frequency (IDF) measures how much information words can provide and logarithmically scales the inverse fraction of documents that contain the word.

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

The term frequency-inverse document frequency (TF-IDF) is the product of two statistics.

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$

For each bag of words, we use HashingTF to hash them into feature vectors and use IDF to rescale them.

(iii) Classification

We apply logistic regression from Python to build our classifier for predicting binary classes, which is a special case of linear regression. Logistic regression predicts the probability of occurrence of a binary event by utilizing a logit function. Below is the general form of the logit function.

$$Ln\left(\frac{P}{1-P}\right)=\beta_0+\beta_1X_1+\beta_2X_2+...+\beta_kX_k$$

The sigmoid function is known as a logistic function that is an 'S' shaped curve which will map any real-valued number into a value between 0 and 1.

$$f(x)=\frac{1}{1+e^{-(x)}}$$

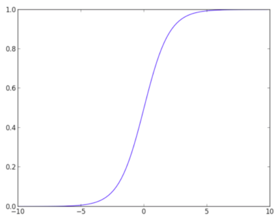


Fig. 6 Sigmoid Function Curve

If the output of the sigmoid function is more than 0.5, we classify the prediction result as 1. If the output of the sigmoid function is less than 0.5, we classify the prediction result as 0.

B. FastText

(i) Data Preprocessing

We removed all punctuations. For FastText, we did not need to remove stop words nor did we perform uniforming variant words (like, liked, likes) manually. After data cleaning, we tokenize the text to individual words and lemmatize words into letters since FastText is based on n-gram letters level.

(ii) Feature Extraction

Fast-text is an unsupervised learning algorithm of word embeddings and classification developed by Facebook AI in 2016. It breaks words into several n-grams (sub-grams). We create word embeddings vector for each word, and the embeddings vector is the sum for all the n-grams given the training dataset. For instance, the tri-grams for the word apple is an app, ppl, and ple (ignoring the starting and ending of boundaries of words). Rare words can now be properly represented since it is highly likely that some of their n-grams also appears in other words.

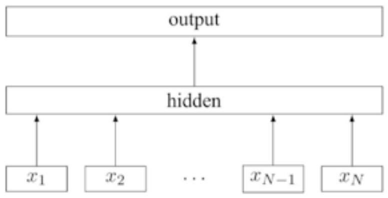


Fig. 7 Model Architecture of FastText for a Sentence with N-Grams Features X1, …, Xn.

We set the vector size to be 256, and we set the max number of tokens for each tweet to be 15. So, for each tweet, we have 15 tokens or lists in our data frame, and each token has 256 features. So, our feature vectors are 3,600,000 * 15 * 256.

(iii) Classification

FastText applies a combination of Convolution Neural Network and Bidirectional Long Short-Term Memory Network (CNN-LSTM). We set the batch_size to be 500 and after several trials, we set no_epochs to be 20, which is sufficient for this project. To prevent overfitting or over training of the network, EarlyStopping() is used in callbacks thus if the network does not improve or starts overfitting, the training comes to an end. The layers of neural network are as follow: Conv1D -> Conv1D -> Conv1D -> Max Pooling1D -> Bidirectional LSTM -> Dense -> Dropout -> Dense -> Dropout -> Dense -> Dropout -> Output.

C. Universal Sentence Encoder

(i) Data Preprocessing

Bag of Words are used for data preprocessing, which means the case, stop words, and punctuations are removed.

(ii) Feature Extraction

The DAN sentence encoding used in universal sentence encoder average together word and bi-gram level embeddings. Then the averaged representation is passed through a feedforward deep neural network. Finally, we have the sentence embeddings.

(iii) Classification

The Universal Sentence Encoder encodes text into high dimensional vectors, which can be used for text classification and other natural language tasks. This model is particularly useful for text such as sentences, phrases, and short paragraphs. The input must be English text and the output is a 512-dimensional vector. The universal-sentence-encoder model has trained with a deep averaging network(DAN) encoder. The DAN sentence encoding model is based on word and bi-gram level embeddings. It is trained on a feedforward deep neural network. The model works very well with transfer learning when using very little training data. Models are built in TensorFlow and available on TensorFlow Hub. There are two types of sentence encoding models: (i) transformer; (ii) deep averaging network. The first one achieves high accuracy while the second one performs more efficiently.

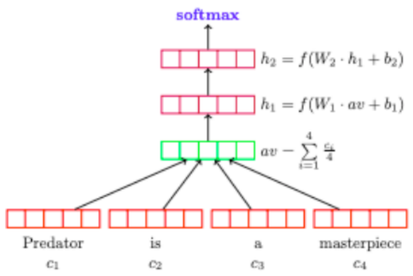

softmax
$h_2 = f(W_2 \cdot h_1 + b_2)$
$h_1 = f(W_1 \cdot av + b_1)$
$av = \sum_{i=1}^{4} \frac{c_i}{4}$
Predator
c_1
is
c_2
a
c_3
masterpiece
c_4

Fig. 8 DAN

This sentence encoding model computes the context-aware representation of words in a sentence that consider both the ordering and identity of other words. Then the context-aware word representations are transformed into a sentence-level embeddings by averaging. The softmax function of DAN is:

$$\mathrm{softmax}(\boldsymbol{q}) = \frac{\exp \boldsymbol{q}}{\sum_{j=1}^{k} \exp \boldsymbol{q}_j}$$

D. Transfer Learning

Transfer learning for text classification used linear text classification algorithms in which the parameters are pre-specified functions of training set statistics. It is a meta-learning technique that leverages data from many related classification tasks to obtain a good classifier for new tasks. This method automates the process of finding a good parameter function for text classifiers. To automatically learning a parameter function from example classification tasks, the statistics vectors are assumed fixed and then finding an optimal parameter function g.

There is two way to find the g function that achieves a small loss on test examples in the training collection. First softmax learning, we could optimize over the choice of:

$$\ell(\boldsymbol{\beta} : \mathscr{S}) := \sum_{j=1}^{m} \log p(y^{(j)} \mid \mathbf{x}^{(j)}; \boldsymbol{\beta}) - C\|\boldsymbol{\beta}\|^2$$
$$= \sum_{j=1}^{m} \log \left(\frac{\exp\left(\boldsymbol{\beta}^T \sum_i \mathbf{u}^{(j)}_{y^{(j)}i} x^{(j)}_i\right)}{\sum_k \exp\left(\boldsymbol{\beta}^T \sum_i \mathbf{u}^{(j)}_{ki} x^{(j)}_i\right)} \right) - C\|\boldsymbol{\beta}\|^2.$$

Second, using a Gaussian kernel to obtain a non-parametric representation of g:

$$g(\mathbf{u}_{ki}) = \boldsymbol{\beta}^T \Phi(\mathbf{u}_{ki}) = \sum_{j=1}^{m} \sum_k \sum_i \alpha_{jk} x^{(j)}_i \exp(-\gamma \|\mathbf{u}^{(j)}_{ki} - \mathbf{u}_{ki}\|^2).$$

The feature vector is a two-dimensional representation:

$$\mathbf{u}_{ki} = \begin{bmatrix} \log(\text{proportion of } w_i \text{ among words from documents of class } k) \\ \log(\text{proportion of documents containing } w_i) \end{bmatrix}.$$

V. Software Package Description

A. Logistic Regression

First, we use python packages (langdetect, pandas) and pyspark package (pyspark.ml.feature) to do data preprocessing.

```
from langdetect import detect
import pandas as pd
# determine whether dataframe contain text
def chooseText (df):
    series = pd.Series(df['0'])
    boolean = series.str.contains('[a-zA-Z]', regex=True)
    df['logic'] = boolean
    df = df[df['logic'] == True]
    return df.iloc[:,0:2]
```

Fig. 9 Selecting Text Information

```
# select only english reviews
idx1 = df_train['0'].apply(lambda x: detect(x) == 'en')
```

Fig. 10 Selecting English Information

```
from pyspark.ml.feature import RegexTokenizer, StopWordsRemover
# convert the input string to lowercase and then split it by regex pattern
regexTokenizer = RegexTokenizer(inputCol="text", outputCol="words", pattern="\\W")
words_data = regexTokenizer.transform(dataframe)
# remove stop words (e.g the, who, which, at, on, I)
stopWordsRemover = StopWordsRemover(inputCol="words", outputCol="words_removed")
words_removed_data = stopWordsRemover.transform(words_data)
```

Fig. 11 Tokenization and Removing Stop Words

Then, we use pyspark package pyspark.ml.feature to do feature extraction.

```
from pyspark.ml.feature import HashingTF, IDF
```

```
# transform list of words to words frequency vectors
hashingTF = HashingTF(inputCol="words_removed", outputCol="words_freq", numFeatures=nFeature)
words_freq_data = hashingTF.transform(words_removed_data)
# compute the IDF vector and scale words frequencies by IDF
idf = IDF(inputCol="words_freq", outputCol="features")
idf_model = idf.fit(words_freq_data)
feature_data = idf_model.transform(words_freq_data).select("features")
```

Fig. 12 Feature Extraction by TFIDF

Finally, we use python package sklearn.linear_model to build logistic regression classifier.

```
from sklearn.linear_model import LogisticRegression
# build logistic regression model
lr = LogisticRegression()
lrm = lr.fit(train_x, train_y)
```

Fig. 13 Logistic Regression Classifier

B. FastText

First, we import nltk to support the Tokenize and Lemmatize steps. Tsdm is used to visualize the processing status.

```
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer
from tqdm import tqdm
tokenizer = RegexpTokenizer('[a-zA-Z0-9]\w+')
```

Fig. 14 Tokenization

```
nltk.download('wordnet')
reviews = []
lemmatizer = WordNetLemmatizer()
with tqdm(total=len(review_unclean)) as pbar:
    for review in review_unclean:
        lemmatized = [lemmatizer.lemmatize(word) for word in review]
        reviews.append(lemmatized)
        pbar.update(1)
```

Fig. 15 Lemmatization

Then, we use Fast-Text package to generate the feature vector. FastText is from Gensim. Make sure there is a C compiler in your environment before installing Gensim.

```
from gensim.models import FastText
vector_size = 256
window = 5
fasttext_model = 'fasttext.model'
start = time.time()
model = FastText(size=vector_size)
model.build_vocab(review)
model.train(review, window=window, min_count=1, workers=4, total_examples=model.corpus_count,epochs=model.epochs)
```

Fig. 16 Feature Extraction for FastText Model

After that, we build the data pipeline in Keras backend to feed in a CNN model in Keras later. After this step, we'll fill in the actual values to x_train, etc.

```
import keras.backend as K
x_train = np.zeros((train_size, max_no_tokens, vector_size), dtype=K.floatx())
y_train = np.zeros((train_size, 2), dtype=np.int32)
x_test = np.zeros((test_size, max_no_tokens, vector_size), dtype=K.floatx())
y_test = np.zeros((test_size, 2), dtype=np.int32)
```

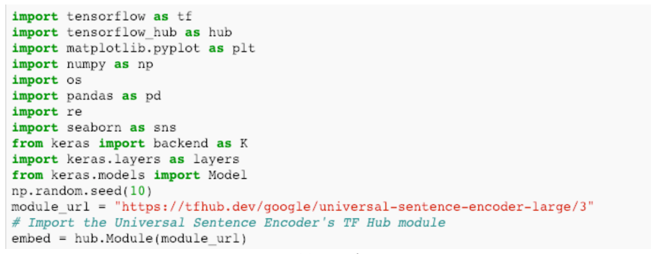
Fig. 17 Data Pipeline in Keras

```
from keras.models import Sequential
from keras.layers import Conv1D, Dropout, Dense, Flatten, LSTM, MaxPooling1D, Bidirectional
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, TensorBoard
model = Sequential()
model.add(Conv1D(32, kernel_size=3, activation='elu', padding='same',
                 input_shape=(max_no_tokens, vector_size)))
model.add(Conv1D(32, kernel_size=3, activation='elu', padding='same'))
model.add(Conv1D(32, kernel_size=3, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=3))
model.add(Bidirectional(LSTM(512, dropout=0.2, recurrent_dropout=0.3)))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.25))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.25))
model.add(Dense(2, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001, decay=1e-6), metrics=['accuracy'])
```

Fig. 18 CNN-LSTM Model in Keras

C. Universal Sentence Encoder with Transfer Learning

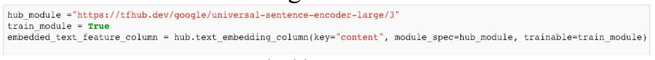
First, we used Tensorflow, Keras, and Tensorflow_hub to import universal sentence encoder and pre-trained model.



```
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import re
import seaborn as sns
from keras import backend as K
import keras.layers as layers
from keras.models import Model
np.random.seed(10)
module_url = "https://tfhub.dev/google/universal-sentence-encoder-large/3"
# Import the Universal Sentence Encoder's TF Hub module
embed = hub.Module(module_url)
```

Fig. 19 Packages

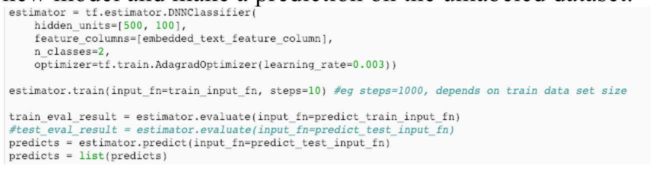
Then we embed the training dataset.



```
hub_module ="https://tfhub.dev/google/universal-sentence-encoder-large/3"
train_module = True
embedded_text_feature_column = hub.text_embedding_column(key="content", module_spec=hub_module, trainable=train_module)
```

Fig. 20 Embedding Training Dataset

After embedding, we use pre-trained parameters to train a new model and make a prediction on the unlabeled dataset.



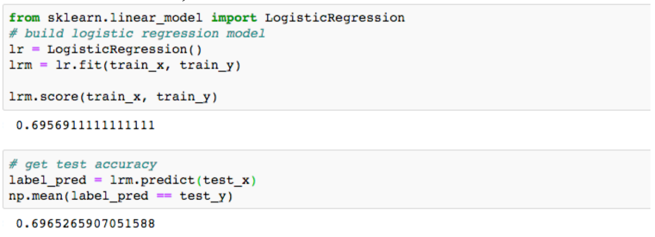
```
estimator = tf.estimator.DNNClassifier(
    hidden_units=[500, 100],
    feature_columns=[embedded_text_feature_column],
    n_classes=2,
    optimizer=tf.train.AdagradOptimizer(learning_rate=0.003))

estimator.train(input_fn=train_input_fn, steps=10) #eg steps=1000, depends on train data set size

train_eval_result = estimator.evaluate(input_fn=predict_train_input_fn)
#test_eval_result = estimator.evaluate(input_fn=predict_test_input_fn)
predicts = estimator.predict(input_fn=predict_test_input_fn)
predicts = list(predicts)
```

Fig. 21 Prediction on New Model

VI. EXPERIMENT RESULTS

A. Logistic Regression

We train the logistic regression classifier on the training dataset and evaluate the classifier performance based on the prediction accuracy of test dataset. Below is the training result on the 3,600,000-training data. Second is the testing result on the 400,000-test data.



```
from sklearn.linear_model import LogisticRegression
# build logistic regression model
lr = LogisticRegression()
lrm = lr.fit(train_x, train_y)

lrm.score(train_x, train_y)
```

```
 0.6956911111111111
```

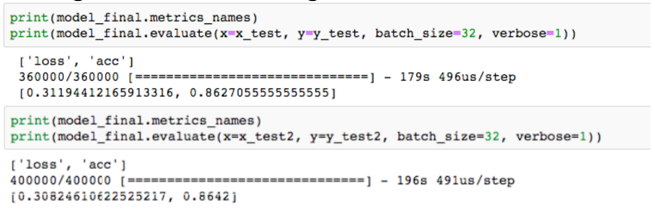
```
# get test accuracy
label_pred = lrm.predict(test_x)
np.mean(label_pred == test_y)
```

```
 0.6965265907051588
```

Fig. 22 Logistic Regression Outcome

B. FastText

We train the CNN-LSTM on the training dataset and evaluate the prediction accuracy of test dataset. Below is the training result in the 10% validation data from 3,600,000 training. Second is the testing result on the 400,000-test data.



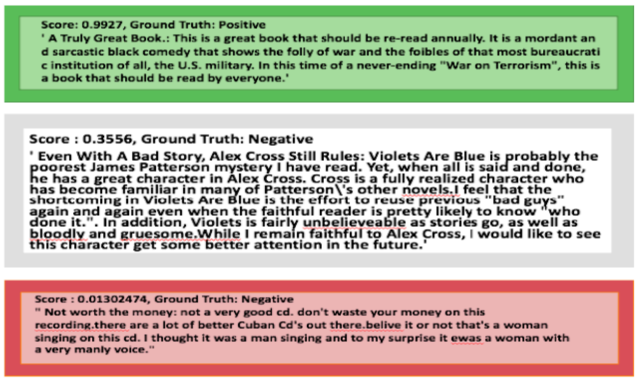
```
print(model_final.metrics_names)
print(model_final.evaluate(x=x_test, y=y_test, batch_size=32, verbose=1))
```

```
 ['loss', 'acc']
 360000/360000 [==============================] - 179s 496us/step
 [0.31194412165913316, 0.8627055555555555]
```

```
print(model_final.metrics_names)
print(model_final.evaluate(x=x_test2, y=y_test2, batch_size=32, verbose=1))
```

```
['loss', 'acc']
400000/400000 [==============================] - 196s 491us/step
[0.30824610622525217, 0.8642]
```

Fig. 23 FastText Outcome



Score: 0.9927, Ground Truth: Positive
' A Truly Great Book.: This is a great book that should be re-read annually. It is a mordant an
d sarcastic black comedy that shows the folly of war and the foibles of that most bureaucrati
c institution of all, the U.S. military. In this time of a never-ending "War on Terrorism", this is
a book that should be read by everyone.'

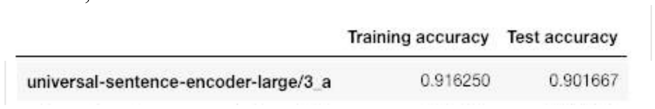
Score : 0.3556, Ground Truth: Negative
' Even With A Bad Story, Alex Cross Still Rules: Violets Are Blue is probably the poorest James Patterson mystery I have read. Yet, when all is said and done, he has a great character in Alex Cross. Cross is a fully realized character who has become familiar in many of Patterson\'s other novels.I feel that the shortcoming in Violets Are Blue is the effort to reuse previous "bad guys" again and again even when the faithful reader is pretty likely to know "who done it.". In addition, Violets is fairly unbelieveable as stories go, as well as bloodly and gruesome.While I remain faithful to Alex Cross, I would like to see this character get some better attention in the future.'

Score : 0.01302474, Ground Truth: Negative
" Not worth the money: not a very good cd. don't waste your money on this recording.there are a lot of better Cuban Cd's out there.belive it or not that's a woman singing on this cd. I thought it was a man singing and to my surprise it ewas a woman with a very manly voice."

Fig. 24 FastText Sentiment Analysis Results

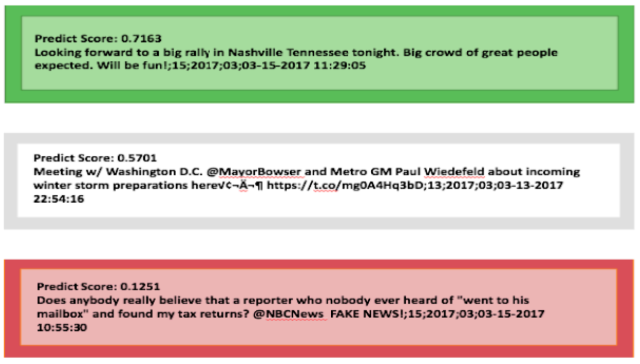
C. Universal Sentence Encoder

We train the DNN classifier in Universal Sentence Encoder on the training dataset and evaluate the classifier performance with pre-trained parameters based on the prediction accuracy of test dataset. Below is the training result on the 3,600,000-training data and testing result on the 400,000-test data.



	Training accuracy	Test accuracy
universal-sentence-encoder-large/3_a	0.916250	0.901667

Fig. 25 Universal Sentence Encoder Outcome



Predict Score: 0.7163
Looking forward to a big rally in Nashville Tennessee tonight. Big crowd of great people expected. Will be fun!;15;2017;03;03-15-2017 11:29:05

Predict Score: 0.5701
Meeting w/ Washington D.C. @MayorBowser and Metro GM Paul Wiedefeld about incoming winter storm preparations here√¢-Ä¬¶ https://t.co/mg0A4Hq3bD;13;2017;03;03-13-2017 22:54:16

Predict Score: 0.1251
Does anybody really believe that a reporter who nobody ever heard of "went to his mailbox" and found my tax returns? @NBCNews FAKE NEWS!;15;2017;03;03-15-2017 10:55:30

Fig. 26 Transfer Learning Sentiment Result

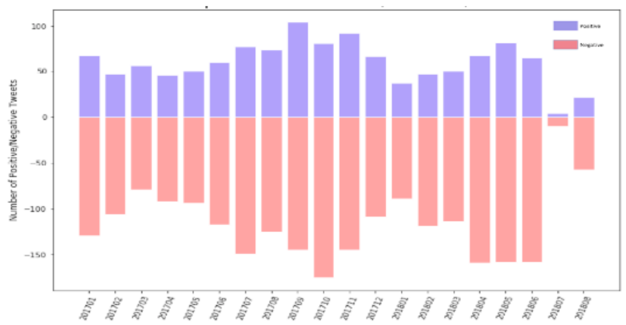
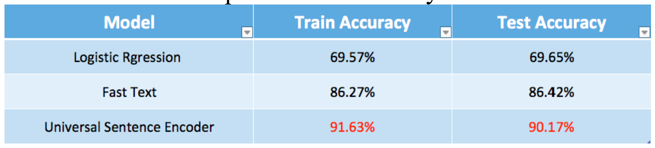


Fig. 27 Trump's Twitter Sentiment from 01/2017 to 08/2018

Below is the overall prediction summary.



Model	Train Accuracy	Test Accuracy
Logistic Rgression	69.57%	69.65%
Fast Text	86.27%	86.42%
Universal Sentence Encoder	91.63%	90.17%

Fig. 28 Prediction Accuracy by Three Models

VII. Conclusion

We use training dataset of Amazon reviews to train our classifiers and test those classifiers using test dataset. After evaluating prediction accuracy by three different algorithms (Logistic Regression, FastText, Universal Sentence Encoder), the universal sentence encoder wins by achieving the highest prediction accuracy (96%). Apply 200 manually labeled Trump's Tweets data to the combination of out best pre-trained model and transfer learning, which will produce a new universal sentence encoder model that can make a satisfying prediction on unlabeled Trump's Tweets. The best model in our project inspires us that transfer learning with universal sentence embedding could be very useful and powerful when we do not have enough labeled data but want to achieve high precision simultaneously. For some future works, we want to applied this method to different sentiment topics, including how does sentiment of China-US trade war in news change over time. Besides, how does people's sentiment about Gene modified organism in social media change over time? And so on.

Peer Evaluation: every team member contributes equally to the project.
Shan Guan (sg3506): built FastText model and wrote the report together.
Yuehan Kong (yk2756): built Logistic Regression classifier and wrote the report together.
Lin Jiang (lj2438): built Universal Sentence Encoder model and wrote the report together.

Acknowledgment

We would like to express our greatest appreciation to Prof. Ching-Yung Lin and all the Teaching Assistants for their help and support on this project.

References

[1] Mandav, J. (2018). *Sentiment Analysis Using Word2Vec, FastText and Universal Sentence Encoder in Keras*. [online] Medium. Available at: https://medium.com/@jatinmandav3/opinion-mining-sometimes-known-as-sentiment-analysis-or-emotion-ai-refers-to-the-use-of-natural-874f369194c0.

[2] Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T. (2016). *Bag of Tricks for Efficient Text Classification.*

[3] Conneau, A., Kiela, D., Schwenk, H., Barrault, L. and Bordes, A. (2018). *Supervised Learning of Universal Sentence Representations from Natural Language Inference Data.*

[4] Do, C. and Ng, A. *Transfer learning for text classification.*

[5] Cer, D. (2018). Universal Sentence Encoder for English.

[6] Iyyer, M., Manjunatha, V., Boyd-Graber, J. and Daume III, H. (2015). *Deep Unordered Composition Rivals Syntactic Methods for Text Classification*.

[7] Sarkar, D. (2018). *Deep Transfer Learning for Natural Language Processing—Text Classification with Universal…*. [online] Towards Data Science. Available at: https://towardsdatascience.com/deep-transfer-learning-for-natural-language-processing-text-classification-with-universal-1a2c69e5baa9.

[8] https://www.kaggle.com/bittlingmayer/amazonreviews

[9] http://www.trumptwitterarchive.com/archive

[10] https://tfhub.dev/google/universal-sentence-encoder/1