Twitter Disaster Classification: Natural Language Processing

Aayush Chhabra

Abstract

Using techniques from Natural Language processing like N-grams model and classical machine learning techniques like Logistic Regression, Naive Bayes, and others to classify a given tweet. The dataset used for the analysis consists of tweets that have language that hints towards a possibility of a disaster but a lot of them are fake disasters. The task at hand is to perform robust analysis of such tweets and use statistical techniques to make predictions about the authenticity of a tweet.

Introduction and Overview

As the world becomes increasing digital, a lot of us rely on digital sources for expressing our views, acquiring all sorts of information, and other kinds of communication with the rest of the world. One of the benefits of being in the digital age is human expression on social media contains a huge amount of information regarding the events happening in the world. In the analysis of interest for our work, we focused on using tweets (from twitter) as a source of information about the possibility of disasters happening in different parts of the world. A large number of organisations actively monitor tweets from users to learn about any emergencies or disasters happening, in real-time.

Due to the humongous volume of tweets shared every second, human tracking of these tweets becomes impractical and it becomes increasingly important to develop novel systems to track tweets in an autonomous manner to identify potential disasters in the world and take relief measures. However, a huge challenge of using autonomous systems for such a task is classifying a tweet as conveying a real disaster or fake. To make a proper classification, a true semantic understanding is required of the contents of the tweet.

We have used a dataset created by the company figure-eight that they shared on their website (downloaded from Kaggle) to perform our analysis. We will combine statistical methods from Natural Language Processing, data analysis, and machine learning to analyze these tweets in an attempt to classify them as real disaster tweets or fake. One abstract example where tweets use language conveying a disaster but is actually a fake disaster occurs during the use of hyperbole and metaphors in the English language.

Theoretical Background

Natural Language Processing

For performing analysis on natural language, we will need to represent the text of a tweet into a form that can be accepted by an algorithm. The standard choice for this is using some kind of word embedding where the embeddings are vectors in a very high dimensional space where the number of dimensions are determined by the total number of different words combined in all the tweets and the N-gram model being used.

Multinomial N-grams Model

In an N-grams model, instead of using just a single word as a token, each consecutive group of N words is also treated as a unique token and has a corresponding position in the word embedding where the value of

the coordinate represents the frequency of occurrence of the token.

Machine Learning

Naive-Bayes

Naive Bayes is a probabilistic classifier that makes classifications using Maximum A Posteriori rule in a Bayesian setting. It makes the assumption that the presence of a particular feature in a class is unrelated to any other feature.

Logistic Regression

Logistic Regression is another probabilistic classifier that defines probabilities by assuming a Sigmoid distribution. Specifically, $P(Y = 1|X; \Theta) = \frac{1}{1+e^{-z}}$

Support Vector Machines

Support Vector machine is an algorithm that finds an hyper-plane that separates the two classes. The objective function used by Support Vector Machine during optimization tries to maximize the margin. It also contains a regularization term to normalize the magnitude of the normal to the hyper-plane.

AdaBoost and XGBoost

AdaBoost and XGBoost are two different implementations of the general principle of Boosting which tries to combine many weak learners to produce a strong learner. It's an ensemble technique for improving the model predctions of any algorithm. AdaBoost specifically combines many decision trees to achieve this effect.

Evaluation Metrics

Accuracy

Accuracy is defined as the ratio of correct predictions to the total number of input samples.

Precision

Precision is defined as the ratio of correctly predicted positive observations to the total predicted positive observations.

Recall

Recall is defined as the ratio of correctly predicted positive observations to the sum of True positives and False negatives.

F1 score

F1 score is the weighted average of Precision and Recall.

Algorithm Implementation and Development

0.1 Train-Validation-Test Split

We took our dataset of 7613 labelled tweets and made a 60-20-20 split of the data into training, validation, and testing set.

0.2 Data preparation and Feature Selection

We took the raw data of tweets and used an object of the CountVectorizer class from the Sklearn package to perform text feature extraction. CountVectorizer provides arguments to specify the range of N for the N-grams model to prepare the word embeddings. We used the min value of 1 and experimented with different values of the max value for N to finally decide that N = 2 (bigram model) provided best results on the cross validation set. In this given dataset, there was a column for location of the tweet however most of the values for this column were either missing or corrupted (in the form of hashtags) and therefore, we decided to drop this out of the analysis.

0.3 Pipeline

To prevent data leakage and perform identical automated linear transformations on all the splits of the dataset, we used the Pipeline class from sklearn to develop an infrastructure to test multiple machine learning algorithms. The only transformation that we used was applying the CountVectorizer on the dataset. However, such a pipeline allows efficient modifications in any future work and allows easy reproduction of results.

0.4 Models

We used 4 different implementations from the sklearn package, namely the Naive-Bayes classifier, AdaBoost, Logistic Regression, and the Support Vector Classifier. We also used an implementation of the Gradient Boosting Algorithm from the widely known XGBoost open-source library.

0.5 Hyper-parameter Tuning

All the models came with different set of hyper-parameters and we used a telescopic search to manually tune these hyper-parameters.

0.6 Evaluation

Although we decided on using F1 score for making decisions at different stages on the analysis, we still observed 4 different metrics - Accuracy, Precision, Recall, and F1 score, for all the models evaluated on the cross validation set. These observations help get a deeper insight into the behavior of these models on the dataset.

0.7 Final

At the end of our analysis, we picked the model based on our F1 score (which we said would be used for decisions). Then we used the test set to publish our final results.

Computational Results

- After training on the different models, we looked at the results of the models and decided to proceed with the Naive-Bayes model based on its highest F1 score.

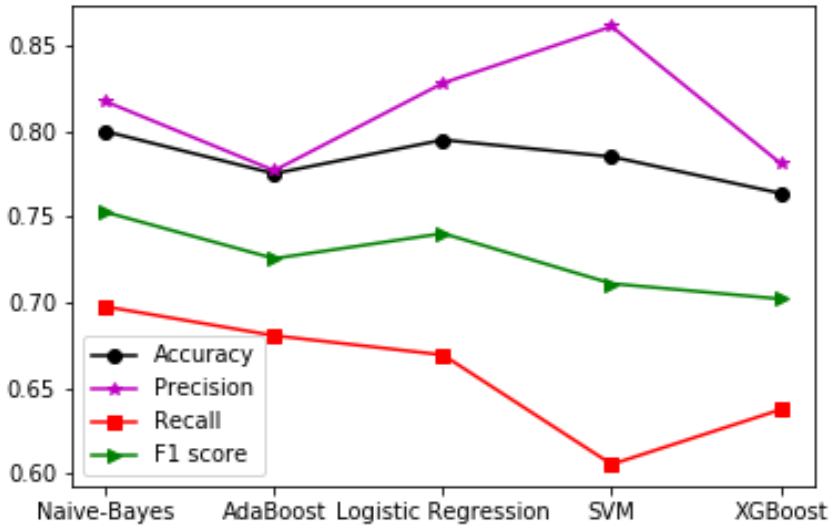


Figure 1: MODEL SELECTION- Comparison of different Models
as evaluated on the Cross Validation Set

- To get better insights into the workings of the model, we also looked at weights that Logistic Regression Model associated to different tokens.
(We are reporting the first few tokens however the code at the end can be used to query the weight for any token.)

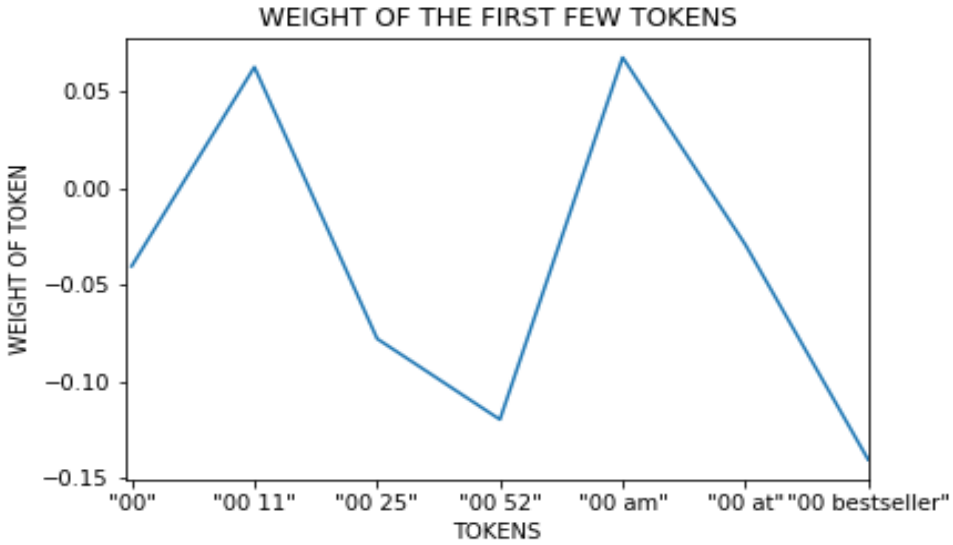


Figure 2: MODEL INSIGHT- WEIGHT OF THE FIRST FEW TOKENS

- Final Evaluation of Test Set
  Accuracy: 0.799080761654629
  Precision: 0.823943661971831
  Recall: 0.6943620178041543
  F1: 0.7536231884057971

Summary and Conclusions

Based on the analysis, we were able to classify tweets as conveying a real or fake disaster based on the textual content of the tweets alone. We were able to achieve an accuracy of 79.9080761654629% with the use of only 7613 tweets as our entire dataset. We had to drop the location and keyword columns of our dataset as most of the values were either corrupted or absent. However, in future work, given a large enough dataset with more information about the location of the tweets, we might be able to observe better results. One significant piece of information that might be a good place to explore might include the account from which tweet was made and the ip address. This can allow the algorithms to explore the correlation between different tweets of a single user and hold information regarding different ways that any specific user uses to convey his ideas. There are many such potential ideas still left to explore and we may explore them in our future works.

Appendix A

Python Reference:

sklearn: Scikit-learn is the most commonly used python package for classical machine learning tasks with frameworks for different algorithms.

pandas: Pandas is the standard package for representing datasets in python and performing operations on these datasets in a fast and efficient manner.

matplotlib : Matplotlib is the standard plotting package in python for data visualization.

Appendix B

March 19, 2020

1 Twitter Disaster Classification: Natural Language Processing

In [2]:
```
# Importing Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.feature_extraction.text import CountVectorizer
```

In [3]:
```
#Importing data
df = pd.read_csv('data/train.csv')
```

In [4]:
```
# Feature Selection
X = df['text']
y = df['target']

#vectorizer = CountVectorizer(ngram_range=(2,2))
#X = vectorizer.fit_transform(X)
#X = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
```

In [5]:
```
# Let's split the data for training, cross-validation, and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state = 72);
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
                                                  test_size = 0.2,
                                                  random_state = 72);
```

In [7]:
```
# Plotting:
acc = []
pre = []
rec=[]
fs=[]
```

```
        def evaluation(y_val, predictions):
            """ Prints out evaluation information.

                y_val: true value of the targets
                predictions: predicted value of the targets"""
            # Let's look at the evaluation metrics:
            ## 1. Accuracy
            accuracy = accuracy_score(y_val, predictions)
            acc.append(accuracy)

            ## 2. Precision
            precision = precision_score(y_val, predictions)
            pre.append(precision)

            ## 3. Recall
            recall = recall_score(y_val, predictions)
            rec.append(recall)

            ## 4. F1 score
            f1 = f1_score(y_val, predictions)
            fs.append(f1)
            print("Accuracy: ",accuracy, "\nPrecision:",
                  precision, "\nRecall:   ",recall, "\nF1:       ",f1)
            return f1
```

In [8]:
```
# Algorithm 1: Naive Bayes

from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
vectorizer = CountVectorizer(ngram_range=(1,2))

# Make a pipeline
stepsNB = [("count", vectorizer),("classifier", classifier)]
pipeNB = Pipeline(steps = stepsNB)

# Fit the pipe:
pipeNB.fit(X_train, y_train)
predictionsNB = pipeNB.predict(X_val)

# Evaluation
evaluation(y_val, predictionsNB)
```

```
Accuracy:  0.7996715927750411
Precision: 0.8171806167400881
Recall:    0.6973684210526315
F1:        0.7525354969574035
```

Out[8]:
```
0.7525354969574035
```

In [8]:
```
# Algorithm 2: AdaBoost
from sklearn.ensemble import AdaBoostClassifier
modelAB = AdaBoostClassifier(n_estimators = 150, random_state = 1)
stepsAB = [("count", vectorizer),("adaboost", modelAB)]
pipeAB = Pipeline(steps = stepsAB)
pipeAB.fit(X_train, y_train)
predictionsAB = pipeAB.predict(X_val)
evaluation(y_val, predictionsAB)
```

```
Accuracy:  0.7750410509031199
Precision: 0.776824034334764
Recall:    0.6804511278195489
F1:        0.7254509018036073
```

Out[8]: 0.7254509018036073

In [9]:
```
# Algorithm 3: Logistic Regression
from sklearn.linear_model import LogisticRegression
modelLR = LogisticRegression(random_state = 1, max_iter = 100000)
stepsLR = [("counts",vectorizer), ("LR", modelLR)]
pipeLR = Pipeline(steps = stepsLR)
pipeLR.fit(X_train, y_train)
predictionsLR = pipeLR.predict(X_val)
evaluation(y_val, predictionsLR)
```

```
Accuracy:  0.7947454844006568
Precision: 0.827906976744186
Recall:    0.6691729323308271
F1:        0.74012474012474
```

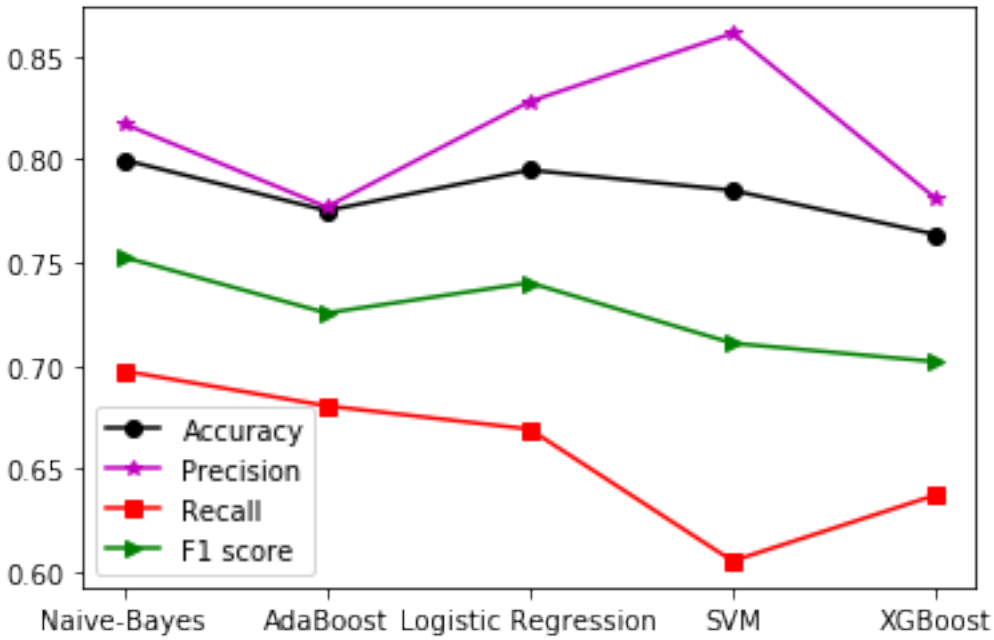
Out[9]: 0.74012474012474

In [10]:
```
# Algorithm 4: Support Vector Classifier
from sklearn.svm import SVC
modelSVC = SVC(C = 2, degree = 1)
stepsSVC = [("counts",vectorizer), ("SVC", modelSVC)]
pipeSVC = Pipeline(steps = stepsSVC)
pipeSVC.fit(X_train, y_train)
predictionsSVC = pipeSVC.predict(X_val)
evaluation(y_val, predictionsSVC)
```

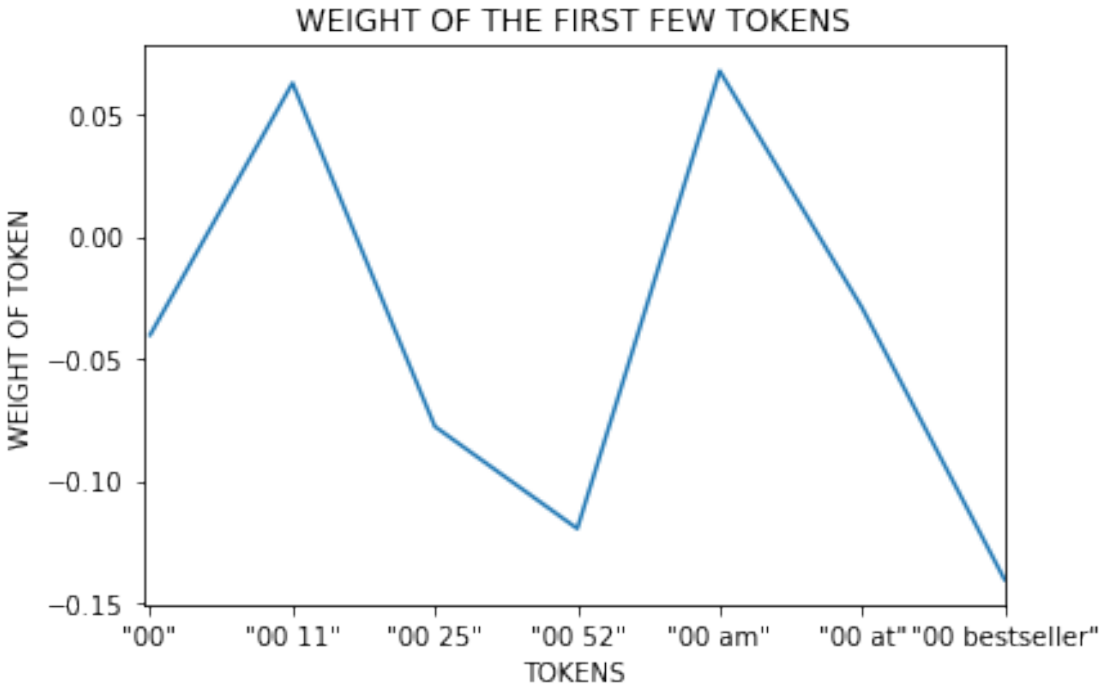
```
Accuracy:  0.7848932676518884
Precision: 0.8609625668449198
Recall:    0.6052631578947368
F1:        0.7108167770419427
```

Out[10]: 0.7108167770419427

```
In [11]: # Algorithm 5: XGBoost
         from xgboost import XGBClassifier
         modelXGB = XGBClassifier(max_depth = 100, learning_rate = .1)
         stepsXGB = [("counts",vectorizer), ("XGB", modelXGB)]
         pipeXGB = Pipeline(steps = stepsXGB)
         pipeXGB.fit(X_train, y_train)
         predictionsXGB = pipeXGB.predict(X_val)
         evaluation(y_val, predictionsXGB)

Accuracy:  0.7635467980295566
Precision: 0.7811059907834101
Recall:    0.6372180451127819
F1:        0.7018633540372671


Out[11]: 0.7018633540372671

In [65]: #Plot 1

         fig = plt.figure();
         plt.plot([0, 10, 20, 30, 40],acc,'ko-')
         plt.xticks([0, 10, 20, 30, 40], ["Naive-Bayes", "AdaBoost",
                                          "Logistic Regression", "SVM", "XGBoost"])
         plt.plot([0, 10, 20, 30, 40],pre,'m*-')
         plt.plot([0, 10, 20, 30, 40],rec,'rs-')
         plt.plot([0, 10, 20, 30, 40],fs, 'g>-')
         plt.legend(["Accuracy","Precision","Recall","F1 score"])
         plt.savefig(fname = "comparison.png")
```



In [63]:
```
#Plot 2

# We can take a look at the importance of tokens with
# the help of Logistic Regression Model
coeffs = modelLR.coef_
df = pd.read_csv('data/train.csv')
X = df['text']
y = df['target']
vectorizer = CountVectorizer(ngram_range=(1,2))
Xa = vectorizer.fit_transform(X)
X = pd.DataFrame(Xa.toarray(), columns=vectorizer.get_feature_names())
# Since this is a very high dimensional space, we will look at the first few tokens

x_axis = np.linspace(2,500, 7)
plt.plot(x_axis, coeffs[0][0:7])
plt.xticks(x_axis, ["\"00\"", "\"00 11\"", "\"00 25\"", "\"00 52\"",
                    "\"00 am\"", "\"00 at\"", "\"00 bestseller\""])
plt.xlabel("TOKENS")
plt.xlim([-2, 500])
plt.ylabel("WEIGHT OF TOKEN")
plt.title("WEIGHT OF THE FIRST FEW TOKENS")
plt.savefig(fname = "weight.png")
```



```
In [10]: # Final Results
         # Test Set evaluation

         testPredictions = pipeNB.predict(X_test)
         evaluation(y_test, testPredictions)
```

```
Accuracy:  0.799080761654629
Precision: 0.823943661971831
Recall:    0.6943620178041543
F1:        0.7536231884057971
```

```
Out[10]: 0.7536231884057971
```