Tweet Classifier: Detecting Disasters

Brandon Arcilla

Problem

Can I build a machine learning model that can predict whether a tweet is referring to a disaster or not?

Why?
Disasters can strike at anytime and tweets provide real-time communication. It's important for local, state, and federal organizations that deal with disasters to be able to respond as quickly as possible. Being able to monitor social media for disasters would be one way to do this.

How?
I will apply various machine learning models on data that consists of over 10,000 tweets that have been classified as a relevant disaster tweet or not relevant.

Data

Data was downloaded from https://www.figure-eight.com/data-for-everyone/

This dataset consisted of 13 columns with 10,876. Out of 13, only 2 columns will be used.

```
tweets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10876 entries, 0 to 10875
Data columns (total 13 columns):
_unit_id                  10876 non-null int64
_golden                   10876 non-null bool
_unit_state               10876 non-null object
_trusted_judgments        10876 non-null int64
_last_judgment_at         10792 non-null object
choose_one                10876 non-null object
choose_one:confidence     10876 non-null float64
choose_one_gold           87 non-null object
keyword                   10789 non-null object
location                  7238 non-null object
text                      10876 non-null object
tweetid                   10876 non-null float64
userid                    10789 non-null float64
dtypes: bool(1), float64(3), int64(2), object(7)
memory usage: 1.0+ MB
```

```
tweets = tweets[['choose_one', 'text']]

tweets.head(5)
```

	choose_one	text
0	Relevant	Just happened a terrible car crash
1	Relevant	Our Deeds are the Reason of this #earthquake M...
2	Relevant	Heard about #earthquake is different cities, s...
3	Relevant	there is a forest fire at spot pond, geese are...
4	Relevant	Forest fire near La Ronge Sask. Canada

Data Cleaning Steps

1. Target column had 3 unique values. One of the values was not significant so it was removed.

2. New DataFrame was created with the 2 columns: target and text.

3. Mapped ‘Relevant’ and ‘Not Relevant’ to ‘Yes’ and ‘No’. This was a personal choice.

```
tweets.groupby('choose_one').count()
```

choose_one	text
Can't Decide	16
Not Relevant	6187
Relevant	4673

```
df = pd.DataFrame() #Create blank dataframe

#Add new columns with data while dropping rows that are marked as "Can't Decide"
df[['target', 'text']] = tweets[tweets['choose_one'] != "Can't Decide"]
```

```
df.groupby('target').count()
```

target	text
Not Relevant	6187
Relevant	4673

```
df['target'] = df['target'].map({'Relevant': 'Yes', 'Not Relevant': 'No'})
```

```
df.head()
```

	target	text
0	Yes	Just happened a terrible car crash
1	Yes	Our Deeds are the Reason of this #earthquake M...
2	Yes	Heard about #earthquake is different cities, s...
3	Yes	there is a forest fire at spot pond, geese are...
4	Yes	Forest fire near La Ronge Sask. Canada

Data Cleaning Steps cont.

4. Create text cleaning function

   4a. Remove URLs

   4b. Remove nametags (@)

   4c. Lowercase

   4d. Lemmatize

   4e. Remove symbols, punctuations and numbers

   4d. Remove stop words

```python
#Remove stopwords
stop_words = stopwords.words('english')

#New words to add to the stopwords list. This contains
newWords = ['afaik', 'cc', 'cx', 'dm', 'ff', 'ht', 'icymi',
            'mm', 'mt', 'nsfw', 'oh', 'prt', 'rlrt', 'rt',
            'smh', 'tftf', 'til', 'tl', 'dr', 'tmb', 'tqrt', 'tt', 'w']
stop_words.extend(newWords)

def text_cleaning(text):
    """If applying on DataFrame column, use within an apply method.
    INPUT:
        - text: text string"""

    text = re.sub(r'(www|http)\S+','',text) #Removes URLs
    text = re.sub(r'@\w+','', text) #Removes nametags
    text = text.lower() #lowercase all words

    def lemmatize(text):
        lemmatizer = WordNetLemmatizer()
        lemmatized_output = ' '.join(lemmatizer.lemmatize(x, 'v') for x in word_tokenize(text))
        return lemmatized_output

    text = lemmatize(text)

    text = re.sub(r'[^a-z\s]',' ', text) #remove random symbols and numbers in string

    def remove_stopwords(text):
        token = word_tokenize(text)

        remove_short = [x for x in token if len(x)>2 ] #remove words that are shorter than 2 letters

        #remove stopwords and put sentence back together
        remove_output = ' '.join(x for x in remove_short if x not in stop_words)

        return remove_output

    text = remove_stopwords(text)

    return text
```

Data Cleaning Steps cont.

5. Apply text cleaning function on text column

```
#Test out the text cleaning function

clean2 = df.text[7958]
print('Original:',clean2)
print()
print('Cleaned:', text_cleaning(clean2))
```

```
Original: Landslide caused by severe rainstorm kills 3 in ItalianåÊAlps https://t.co/8BhvxX2X19 http://t.co/4ou8s82Hx
J

Cleaned: landslide cause severe rainstorm kill italian alps
```

```
#Apply the text_cleaning function to all of the texts
df['clean_text'] = df['text'].apply(text_cleaning)
```

```
df.head(5)
```

	target	text	clean_text
0	Yes	Just happened a terrible car crash	happen terrible car crash
1	Yes	Our Deeds are the Reason of this #earthquake M...	deeds reason earthquake may allah forgive
2	Yes	Heard about #earthquake is different cities, s...	hear earthquake different cities stay safe eve...
3	Yes	there is a forest fire at spot pond, geese are...	forest fire spot pond geese flee across street...
4	Yes	Forest fire near La Ronge Sask. Canada	forest fire near ronge sask canada

Feature Engineering

Tweet Length

```
#tweet length feature
df['tweet_length'] = df['clean_text'].str.len()
```

```
df.head()
```

	target	text	clean_text	tweet_length
0	Yes	Just happened a terrible car crash	happen terrible car crash	25
1	Yes	Our Deeds are the Reason of this #earthquake M...	deeds reason earthquake may allah forgive	41
2	Yes	Heard about #earthquake is different cities, s...	hear earthquake different cities stay safe eve...	51
3	Yes	there is a forest fire at spot pond, geese are...	forest fire spot pond geese flee across street...	51
4	Yes	Forest fire near La Ronge Sask. Canada	forest fire near ronge sask canada	34

Number of Words

```
#number of words feature

def word_count(text):
    words = text.split()
    return len(words)

df['num_words'] = df['clean_text'].apply(word_count)

df.head()
```

	target	text	clean_text	tweet_length	num_words
0	Yes	Just happened a terrible car crash	happen terrible car crash	25	4
1	Yes	Our Deeds are the Reason of this #earthquake M...	deeds reason earthquake may allah forgive	41	6
2	Yes	Heard about #earthquake is different cities, s...	hear earthquake different cities stay safe eve...	51	7
3	Yes	there is a forest fire at spot pond, geese are...	forest fire spot pond geese flee across street...	51	9
4	Yes	Forest fire near La Ronge Sask. Canada	forest fire near ronge sask canada	34	6

Hashtag Count

```
#hashtag feature

def hashtag_count(text):
    words = text.split()
    hashtags = [word for word in words if word.startswith('#')]
    return len(hashtags)

df['hashtag_count'] = df['text'].apply(hashtag_count)

df.head()
```

	target	text	clean_text	tweet_length	num_words	hashtag_count
0	Yes	Just happened a terrible car crash	happen terrible car crash	25	4	0
1	Yes	Our Deeds are the Reason of this #earthquake M...	deeds reason earthquake may allah forgive	41	6	1
2	Yes	Heard about #earthquake is different cities, s...	hear earthquake different cities stay safe eve...	51	7	1
3	Yes	there is a forest fire at spot pond, geese are...	forest fire spot pond geese flee across street...	51	9	0
4	Yes	Forest fire near La Ronge Sask. Canada	forest fire near ronge sask canada	34	6	0

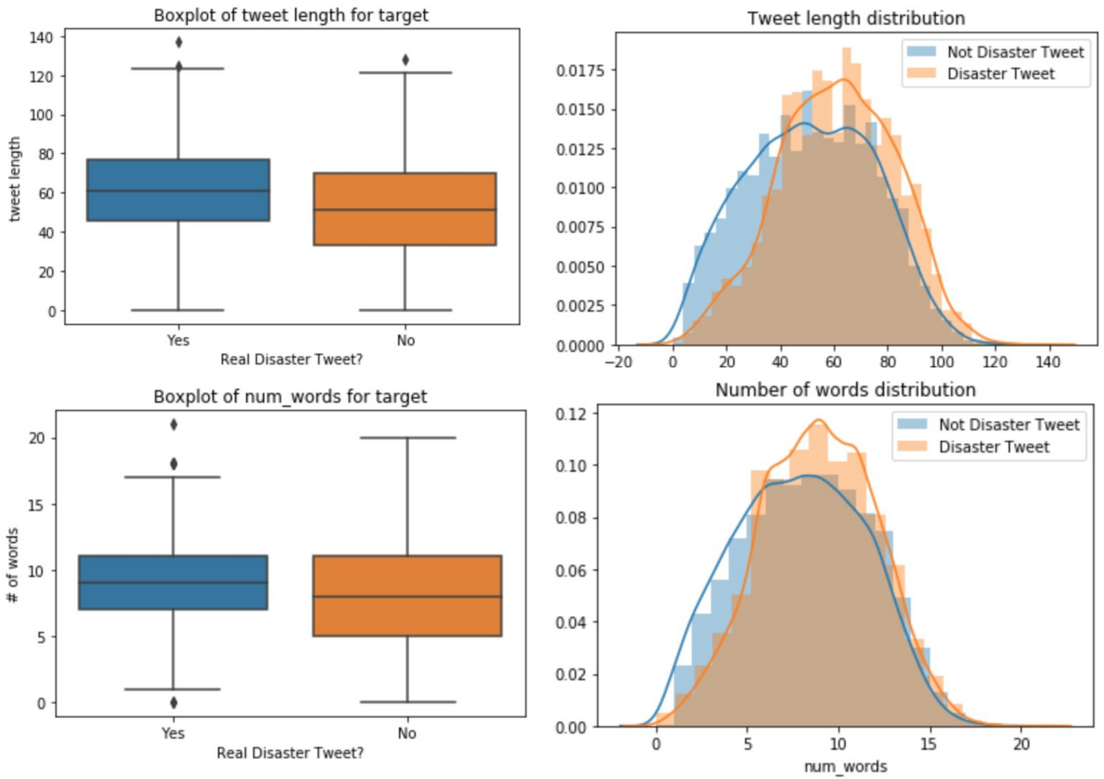
Mention (@) Count

```
#mention feature

def mention_count(text):
    words = text.split()
    hashtags = [word for word in words if word.startswith('@')]
    return len(hashtags)

df['mention_count'] = df['text'].apply(mention_count)

df.head()
```

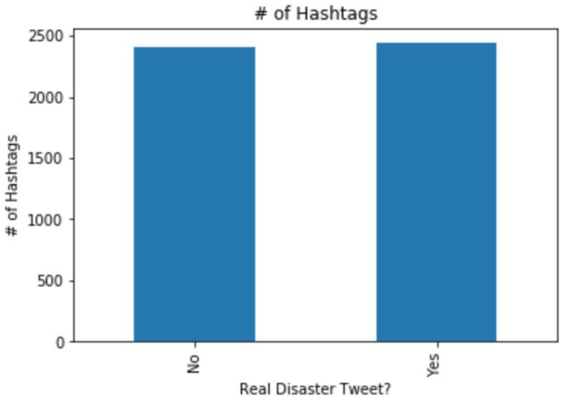
	target	text	clean_text	tweet_length	num_words	hashtag_count	mention_count
0	Yes	Just happened a terrible car crash	happen terrible car crash	25	4	0	0
1	Yes	Our Deeds are the Reason of this #earthquake M...	deeds reason earthquake may allah forgive	41	6	1	0
2	Yes	Heard about #earthquake is different cities, s...	hear earthquake different cities stay safe eve...	51	7	1	0
3	Yes	there is a forest fire at spot pond, geese are...	forest fire spot pond geese flee across street...	51	9	0	0
4	Yes	Forest fire near La Ronge Sask. Canada	forest fire near ronge sask canada	34	6	0	0

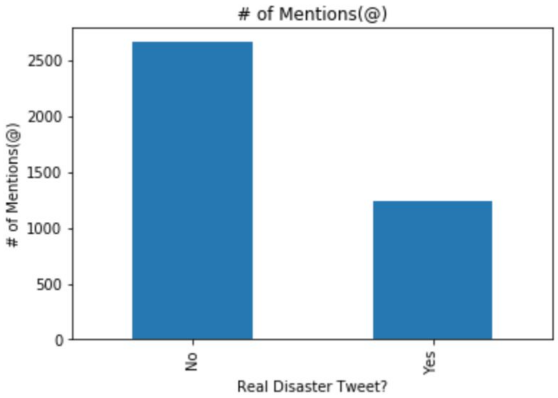
EDA

- Analysis
  - Tweet lengths that relate to disasters are usually longer.
  - Longer tweets = More words
  - Describing situations to better inform public could be a reason.



EDA cont.





- There are similar amounts of hashtags for each case.
- More mentions are prevalent in tweet that are not related to a disaster than those that are.

EDA cont.

Not Disaster

```
[('get', 455),
 ('like', 413),
 ('new', 238),
 ('one', 187),
 ('make', 181),
 ('would', 170),
 ('body', 159),
 ('say', 158),
 ('think', 158),
 ('love', 155),
 ('see', 154),
 ('time', 151),
 ('bag', 151),
 ('come', 150),
 ('via', 142),
 ('know', 140),
 ('video', 135),
 ('want', 132),
 ('people', 132),
 ('burn', 130)]
```

Unigram Analysis

Frequent non-disaster tweets are general. ‘Get’ and ‘Like’ are the top 2 most common words

Unigram might be too simple for model to be successful

‘California’ is a common word for disaster tweets. Why?

Disaster

```
[('fire', 400),
 ('bomb', 259),
 ('news', 226),
 ('kill', 222),
 ('get', 183),
 ('via', 183),
 ('flood', 170),
 ('disaster', 166),
 ('attack', 166),
 ('suicide', 166),
 ('crash', 152),
 ('people', 151),
 ('police', 151),
 ('california', 150),
 ('train', 147),
 ('hiroshima', 146),
 ('like', 146),
 ('home', 137),
 ('storm', 137),
 ('build', 134)]
```

EDA cont.

Bigram Analysis

Not Disaster

```
[(('body', 'bag'), 99),
 (('cross', 'body'), 53),
 (('like', 'video'), 51),
 (('look', 'like'), 47),
 (('full', 'read'), 42),
 (('feel', 'like'), 38),
 (('loud', 'bang'), 31),
 (('burn', 'build'), 29),
 (('first', 'responders'), 29),
 (('content', 'policy'), 28),
 (('china', 'stock'), 27),
 (('stock', 'market'), 27),
 (('market', 'crash'), 27),
 (('emergency', 'service'), 26),
 (('reddit', 'quarantine'), 26),
 (('quarantine', 'offensive'), 26),
 (('offensive', 'content'), 26),
 (('take', 'quiz'), 23),
 (('break', 'news'), 23),
 (('read', 'ebay'), 22)]
```

Bigram adds more context to the words.

‘Body’ and ‘bag’ in non-disaster tweets. Why?

California’ a lot with ‘wildfire’. This makes more sense than unigram. California has been stricken with deadly wildfires in the past few years.

Disaster

```
[(('suicide', 'bomber'), 91),
 (('atomic', 'bomb'), 67),
 (('northern', 'california'), 57),
 (('suicide', 'bomb'), 52),
 (('oil', 'spill'), 50),
 (('california', 'wildfire'), 46),
 (('bomber', 'detonate'), 46),
 (('burn', 'build'), 44),
 (('detonate', 'bomb'), 43),
 (('pkk', 'suicide'), 43),
 (('wild', 'fire'), 42),
 (('year', 'old'), 41),
 (('old', 'pkk'), 41),
 (('mass', 'murder'), 40),
 (('severe', 'thunderstorm'), 40),
 (('home', 'raze'), 40),
 (('latest', 'home'), 39),
 (('raze', 'northern'), 39),
 (('heat', 'wave'), 36),
 (('bomb', 'turkey'), 36)]
```

EDA cont.

Trigram Analysis

Not Disaster

```
[(('cross', 'body', 'bag'), 33),
 (('china', 'stock', 'market'), 27),
 (('stock', 'market', 'crash'), 27),
 (('reddit', 'quarantine', 'offensive'), 26),
 (('quarantine', 'offensive', 'content'), 25),
 (('full', 'read', 'ebay'), 22),
 (('break', 'news', 'unconfirmed'), 22),
 (('news', 'unconfirmed', 'hear'), 22),
 (('unconfirmed', 'hear', 'loud'), 22),
 (('hear', 'loud', 'bang'), 22),
 (('loud', 'bang', 'nearby'), 22),
 (('bang', 'nearby', 'appear'), 22),
 (('nearby', 'appear', 'blast'), 22),
 (('appear', 'blast', 'wind'), 22),
 (('blast', 'wind', 'neighbour'), 22),
 (('wind', 'neighbour', 'ass'), 22),
 (('reddit', 'new', 'content'), 22),
 (('new', 'content', 'policy'), 22),
 (('content', 'policy', 'effect'), 21),
 (('policy', 'effect', 'many'), 21)]
```

Offers a lot more context, which could lead to better results.

'Cross', 'body', 'bag': that's a bag worn on one side of the body

'California', 'wildfire' from bigram now includes 'northern', Giving a better idea of where in California wildfires are taking place.

Disaster

```
[(('suicide', 'bomber', 'detonate'), 46),
 (('pkk', 'suicide', 'bomber'), 42),
 (('bomber', 'detonate', 'bomb'), 42),
 (('northern', 'california', 'wildfire'), 41),
 (('old', 'pkk', 'suicide'), 41),
 (('latest', 'home', 'raze'), 39),
 (('home', 'raze', 'northern'), 39),
 (('raze', 'northern', 'california'), 38),
 (('severe', 'thunderstorm', 'warn'), 36),
 (('families', 'affect', 'fatal'), 34),
 (('affect', 'fatal', 'outbreak'), 34),
 (('watch', 'airport', 'get'), 34),
 (('airport', 'get', 'swallow'), 34),
 (('get', 'swallow', 'sandstorm'), 34),
 (('swallow', 'sandstorm', 'minute'), 34),
 (('detonate', 'bomb', 'turkey'), 34),
 (('bomb', 'turkey', 'army'), 34),
 (('turkey', 'army', 'trench'), 34),
 (('families', 'sue', 'legionnaires'), 33),
 (('wreckage', 'conclusively', 'confirm'), 33)]
```

Further Preprocessing

Before feeding data into models, need to convert texts to vectors!

```
features = df[['clean_text','tweet_length', 'num_words','hashtag_count', 'mention_count']]
target = df['target']

#Convert target to 1(yes) and 0(no)
Encoder = LabelEncoder()
target = Encoder.fit_transform(target)

#Split into train and test

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3, random_state=34)

#Use ColumnTransformer to separate the normalizer for numerical column from text vectors

counts = ['tweet_length', 'num_words','hashtag_count', 'mention_count']
texts = 'clean_text'

preprocessor = ColumnTransformer(
    transformers = [('cv', CountVectorizer(), texts)],
    remainder = MinMaxScaler() )

#Transformer for Decision Tree and Random Forest since numerical values do not need to be normalized.

tree_preprocessor = ColumnTransformer(
    transformers = [('cv', CountVectorizer(), texts)],
    remainder = 'passthrough' )
```

Baseline Model - DummyClassifier

```
dummy = make_pipeline(preprocessor, DummyClassifier(random_state=34))
dummy_acc = dummy.fit(X_train, y_train).score(X_test,y_test)
dummy_f1 = f1_score(y_test, dummy.fit(X_train,y_train).predict(X_test))

print ('Baseline Model Accuracy for CountVectorizer: %.3f' % dummy_acc)
print ('Baseline Model F1-score for CountVectorizer: %.3f' % dummy_f1)
print(confusion_matrix(y_test, dummy.fit(X_train,y_train).predict(X_test)))
```

```
Baseline Model Accuracy for CountVectorizer: 0.505
Baseline Model F1-score for CountVectorizer: 0.416
[[1071  777]
 [ 836  574]]
```

Models Tested

- Logistic Regression
- SVC
- KNN
- Decision Tree
- Random Forest
- AdaBoost
- Gradient Boost

Top 3 models will be analyzed more

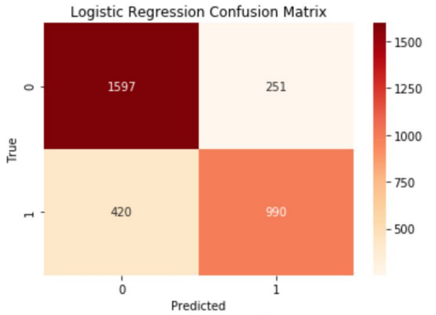
```
lr_pipe = make_pipeline(preprocessor, LogisticRegression(random_state=34))
lr_pipe.fit(X_train, y_train)
y_pred = lr_pipe.predict(X_test)
```

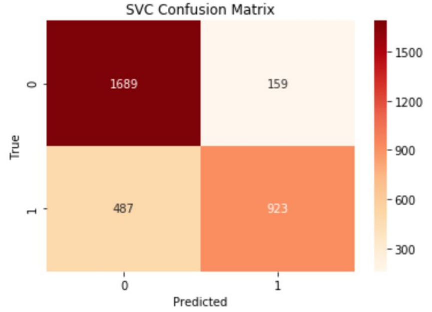
Model Evaluation

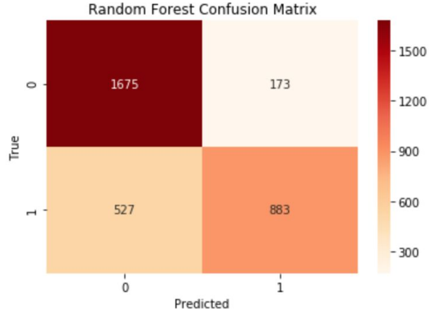
```
#Sorted F1 score

model_performance['f1_score'].sort_values(ascending = False)
```

```
CountVectorizer  LogReg           0.746888
                 SVC              0.740770
                 RandomForest     0.716139
                 DecisionTree     0.685457
                 GradientBoost    0.638070
                 KNN              0.627281
                 AdaBoost         0.624733
```

Logistic Regression, SVC, and Random Forest perform the best when looking at f1-score. The confusion matrix shows us the amount of False Negatives and False Positives are produced. Goal is to lower both when tuning.







Model Tuning

CountVectorizer or TFIDFVectorizer? Using default values with the top 3 performing models produce an even better f1-score.

```
#F1 Score
vect_performance.loc[idx[:, ('LogReg','SVC', 'RandomForest')], :].unstack()['f1_score']
```

	LogReg	SVC	RandomForest
CountVectorizer	0.746888	0.740770	0.716139
TFIDF	0.747569	0.742309	0.726911

Which Tfidf Parameters? These will be chosen during RandomizedSearchCV along with model parameters

- **ngram_range**: (1,1), (1,2), (1,3)
    - unigram only, unigram and bigram mix, unigram and trigram mix
- **min_df:** 1, 5, 10
    - Ignore words that show up less than X amount of times in the documents
- **max_df:** 0.5, 0.75, 0.95
    - Ignore words that show up in more that X% of the documents

Model Tuning cont.

Logistic Regression Randomized Search

```
%%time
lr_param = {
    'columntransformer__tfidf__ngram_range': [(1,1),(1,2),(1,3)],
    'columntransformer__tfidf__min_df':[1, 5, 10],
    'columntransformer__tfidf__max_df':[0.5, 0.75, 0.95],
    'logisticregression__penalty': ['l1', 'l2', None],
    'logisticregression__C': [1, 25, 50, 100],
    'logisticregression__max_iter': [10, 100, 500, 1000],
}
random = RandomizedSearchCV(estimator = lr_pipe_tfidf,
                            param_distributions = lr_param,
                            cv=5,
                            scoring = 'f1',
                            random_state=34)
```

Results

```
Best Score:  0.7525072251167785

Best Params:  {'logisticregression__penalty': 'l2', 'logisticregression__max_iter': 10, 'logisticregression__C': 50,
'columntransformer__tfidf__ngram_range': (1, 1), 'columntransformer__tfidf__min_df': 5, 'columntransformer__tfidf__ma
x_df': 0.75}
CPU times: user 24.7 s, sys: 922 ms, total: 25.6 s
Wall time: 29 s
```

Model Tuning cont.

SVC Randomized Search

```
%%time
svc_param = {
    'columntransformer__tfidf__ngram_range': [(1,1),(1,2),(1,3)],
    'columntransformer__tfidf__min_df':[1, 5, 10],
    'columntransformer__tfidf__max_df':[0.5, 0.75, 0.95],
    'svc__C': [1, 25, 50, 100],
    'svc__kernel': ['linear', 'rbf'],
    'svc__gamma': [0.01, 0.1, 1, 10, 100]
}
random = RandomizedSearchCV(estimator = svc_pipe_tfidf,
                            param_distributions = svc_param,
                            cv=5,
                            scoring = 'f1',
                            random_state=34)
```

Results

```
Best Score:  0.7505131014008629

Best Params:  {'svc__kernel': 'rbf', 'svc__gamma': 0.1, 'svc__C': 25, 'columntransformer__tfidf__ngram_range': (1,
3), 'columntransformer__tfidf__min_df': 1, 'columntransformer__tfidf__max_df': 0.75}
CPU times: user 6min 50s, sys: 7.2 s, total: 6min 58s
Wall time: 8min 49s
```

Model Tuning cont.

Random Forest Randomized Search

```
%%time
rf_param = {
    'columntransformer__tfidf__ngram_range': [(1,1),(1,2),(1,3)],
    'columntransformer__tfidf__min_df':[1, 5, 10],
    'columntransformer__tfidf__max_df':[0.5, 0.75, 0.95],
    'randomforestclassifier__n_estimators': [50, 100, 250, 500],
    'randomforestclassifier__max_features': ['sqrt', 'log2', None],
    'randomforestclassifier__max_depth': [50, 100, 250, 500, None],
}
random = RandomizedSearchCV(estimator = rf_pipe_tfidf,
                            param_distributions = rf_param,
                            cv=5,
                            scoring = 'f1',
                            random_state=34)
```

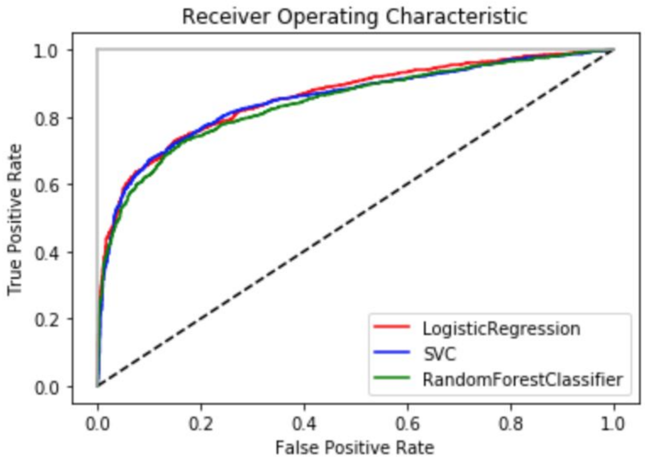
Results

```
Best Score:  0.7221987051533116

Best Params:  {'randomforestclassifier__n_estimators': 50, 'randomforestclassifier__max_features': 'sqrt', 'randomfor
estclassifier__max_depth': 250, 'columntransformer__tfidf__ngram_range': (1, 3), 'columntransformer__tfidf__min_df':
5, 'columntransformer__tfidf__max_df': 0.5}
CPU times: user 28min 20s, sys: 4.3 s, total: 28min 24s
Wall time: 53min 31s
```

Model Evaluation

Best Accuracy Score	
LogReg	0.796808
SVC	0.791283
RandomForest	0.782382

Best F1 Score	
LogReg	0.757153
SVC	0.751280
RandomForest	0.717867



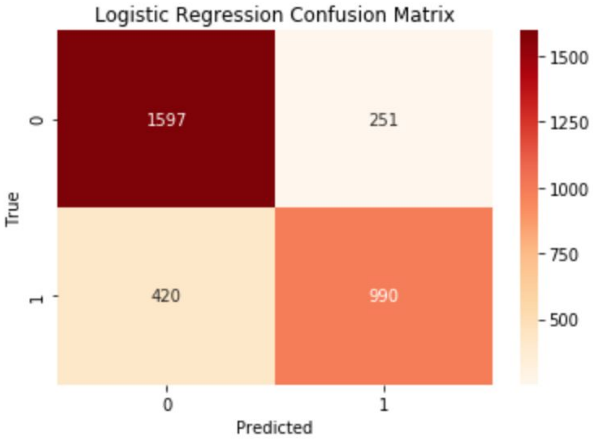

```
ROC AUC Score - LogReg:  0.8587767876331706
ROC AUC Score - SVC:  0.8507548893187191
ROC AUC Score - RandomForest:  0.8417010147063337
```

Logistic Regression performed the best in terms of accuracy score, f1-score, and AUC Score. It was also the fastest running model.

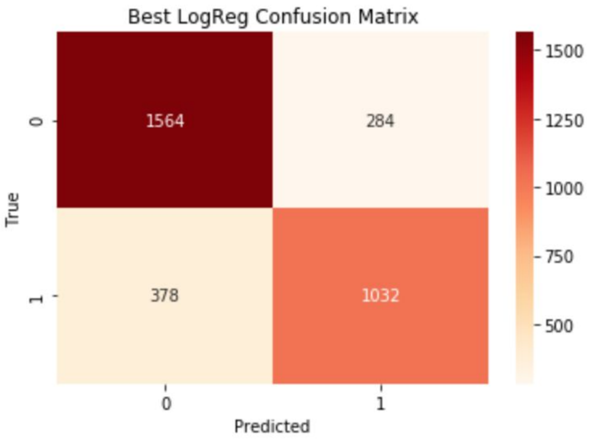
Conclusion

Looking at the performance from the original confusion matrix to the optimized model confusion matrix, there's an improvement in the amount of False Negatives. Logistic Regression provides the best performing in terms of metrics and speed!

Original



Tuned



Limitations and Future Work

- Being that this project was time-sensitive, I would've liked to use Grid Search for a more exhaustive search means to tune the parameters.
- More time could've been taken by cleaning the text.
    - There were some words that had multiple letters in it, for example 'aaaand' and 'and' are the same word but lemmatizing could not solve this.
        - Acronyms could've been handled better and could've provided more insight. For example, 'LA' was cleaned to display 'la' but I would've rather seen it as 'los angeles'.
- I would've liked to use more features like time and location of tweet to see how that can affect the model.