Twitter Sentiment Analysis

Jason Singh, Siva Senthil, Titus Scott, Paul Merica

Motivation

With the massive volume of tweets being generated by Twitter every second, sentiment analysis can help determine the overall opinion of a particular topic or issue. It is a valuable technique for analyzing the opinions, emotions, and attitudes expressed by individuals over text since companies and other entities have interest in tracking public support behind their products.

Data Description

- 1.6 million tweets tagged with sentiment associated with them.
    - 0 = Negative Sentiment
    - 2 = Neutral sentiment
    - 4 = Positive sentiment

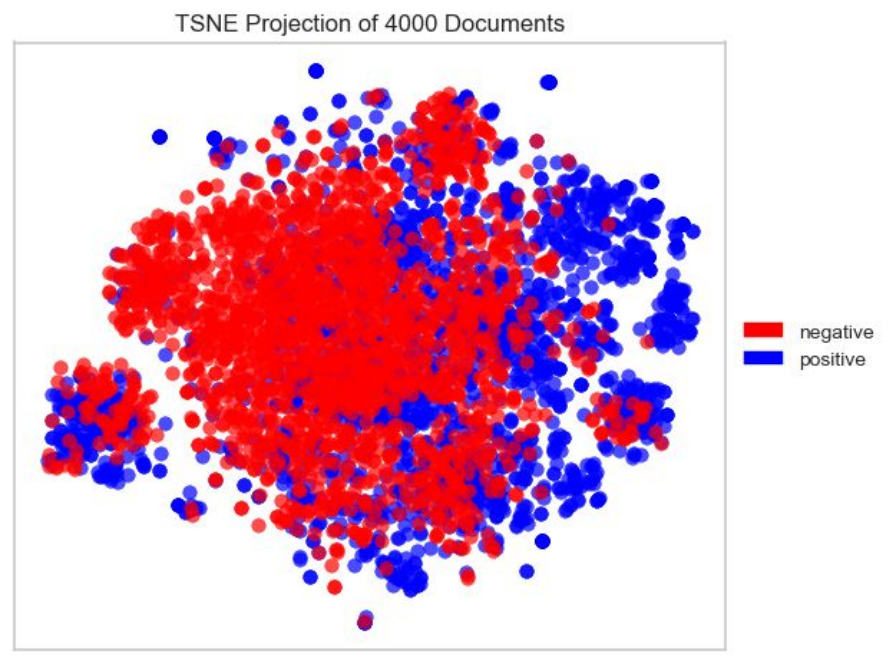
- preprocessing steps for data

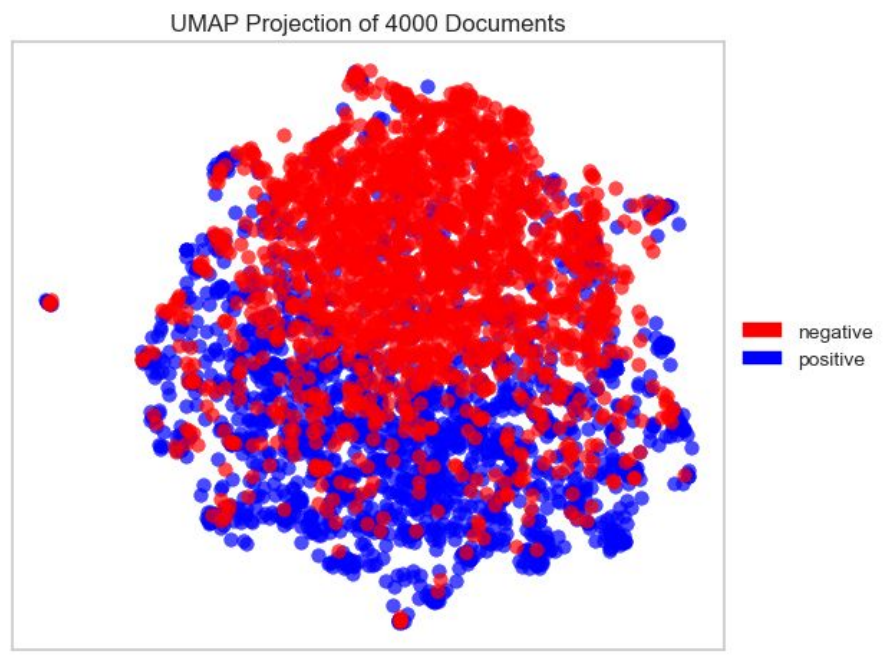
1. remove links remove @'s, punctuation
2. lower-case all text
3. remove stopwords
4. tokenize words per row
5. stem words per row

	target	ids	date	flag	user	raw_text	process_text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...	[httptwitpiccom2y1zl, awww, thats, bummer, sho...

Modeling

UMAP/tSNE


TSNE Projection of 4000 Documents
negative
positive


UMAP Projection of 4000 Documents
negative
positive

Logistic Regression

- Our basic Logistic Regression model returned an accuracy score of approximately 78% along with an F1 score of approximately 79%
- A confusion matrix seemed to indicate a very slight lean for the model predicting more positive results

```
F1 Score:  0.7873475438327454
Accuracy:  0.7833270833333333

              precision    recall  f1-score   support

           0       0.80      0.76      0.78    240539
           1       0.77      0.80      0.79    239461

    accuracy                           0.78    480000
   macro avg       0.78      0.78      0.78    480000
weighted avg       0.78      0.78      0.78    480000

Confusion Matrix:
 [[183461  57078]
 [ 46925 192536]]
```

SVM

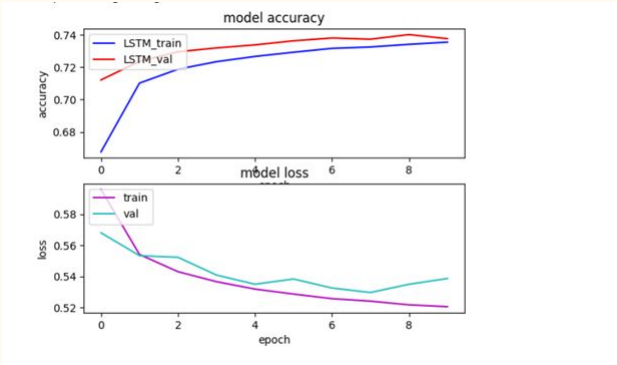
- We built a base SVC model to predict whether a tweet was positive or negative
- We were able to achieve approximately 77% Accuracy with this base model
- Ultimately, we ended up needing to scale our overall dataset to 10% due to multiple issues with memory and computing unit shortages
- We defined a custom tokenizer that also functioned as a tool to make sure the strings were in lower case and converted the text data using TF-IDF
- To ensure the best classification model, we also have built out multiple SVM models utilizing different kernels
- We have also built out multiple Hyper-Parameter Tuners
    - Grid
    - Random
    - Bayes

*Screenshot of Hyperparameters and models in appendix

LSTM Predictions

- We built an LSTM to predict whether a certain string (Tweet) was positive or negative.
- We were able to reach around 73% accuracy with 10 epochs.
- Used GloVe (Global Vectors for Word Representation) Embedding From Stanford.
    - Helps the LTSM understand the information we are feeding it.

*LSTM details in appendix



VADER

-Valence Aware Dictionary and sEntiment Reasoner
-Rule-based sentiment analysis
-Trained on social media data

```
Index: 5314
Text: @SouthwestAir My husband a first time flyer with you has had the worst experience on board a aircraft just a few minutes ago.
Compound Score: -0.62
Positive Score: 0.00
Negative Score: 0.16
Neutral Score: 0.84
```

```
Index: 2342
Text: @united I JUST ASKED MY BOYFRIEND TO PROM OVER THE LOUDSPEAKER ON FLIGHT 494 HE SAID YES!!!! BEST DAY EVER!!! THANK U SO MUCH!!!!!!
Compound Score: 0.95
Positive Score: 0.44
Negative Score: 0.00
Neutral Score: 0.56
```

Takeaways

Conclusion

- SVM and Logistic Regression outperformed our LSTM model.
- UMAP and tSNE struggled with differentiation, showing that this is a rather complex problem
- Large data is hard to parse through and used a lot of computational resources that limited our ability to fully train out some models.

Appendix

SVM

```
# Scale overall dataset to 10% to try to resolve computing issues
df_sampled = pre_processed_data.sample(frac=0.1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(df_sampled["process_text"], df_sampled["target"], test_size=0.3, random_state=39)

# Define a custom tokenizer function
def my_tokenizer(text):
    tokens = text.lower().split()  # Split text into words
    return tokens

# Convert text data into numerical features using TF-IDF representation
vectorizer = TfidfVectorizer(tokenizer=my_tokenizer)

X_train_list = X_train.tolist()
X_test_list = X_test.tolist()

X_train_tfidf = vectorizer.fit_transform(X_train_list)
X_test_tfidf = vectorizer.transform(X_test_list)

svc_model = SVC()
svc_model.fit(X_train_tfidf, y_train)
y_pred = svc_model.predict(X_test_tfidf)

test_acc = accuracy_score(y_test, y_pred)
print(test_acc)

svc_cfm = confusion_matrix(y_test, y_pred)
print(svc_cfm)
```

```
[ ]  #GRID SEARCH
     param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}

     grid_search = GridSearchCV(INSERT_MODEL_HERE_FOR_SVM, param_grid, cv=5)
     grid_search.fit(X, y)

     best_params = grid_search.best_params_
```

```
[ ]  #RANDOM SEARCH
     param_distributions = {'C': loguniform(0.1, 1, 10), 'kernel': ['linear', 'rbf']}
     random_search = RandomizedSearchCV(INSERT_MODEL_HERE_FOR_SVM, param_distributions, n_iter=10, cv=5)
     random_search.fit(X, y)

     best_params = random_search.best_params
```

```
[ ]  #BAYES SEARCH
     param_space = {'C': (0.1, 10.0, 'log-uniform'),
                    'kernel': ['linear', 'rbf'],
                    'gamma': (1e-6, 1e+1, 'log-uniform'),
                    'degree': (1, 8, 'uniform')}

     opt = BayesSearchCV(INSERT_MODEL_HERE_FOR_SVM, param_space, n_iter=50, cv=5, n_jobs=-1)
     opt.fit(X_train, y_train)

     best_params = opt.best_params_

     best_svm = SVC(**best_params)
     best_svm.fit(X_train, y_train)

     accuracy = best_svm.score(X_test, y_test)
     print("Best SVM :", best_params)
     print("Accuracy :", accuracy)
```

LTSM Details

- Input layer with maximum string length of 30 allowed
- embedding layer mapping each integer index to a dense vector of fixed size.
- 1D spatial dropout to the embedding sequences, randomly dropping out entire 1D feature maps in the embedding tensor, to reduce overfitting.
- 1D convolutional layer to the output of the spatial dropout layer, with 64 filters of size 5 and a ReLU activation function, to extract local features from the sequence.
- bidirectional LSTM layer to the output of the convolutional layer, with 64/512 hidden units, and dropout and recurrent dropout rates of 0.2, to capture long-term dependencies in the sequence.
- Dense layer to the output of the LSTM layer, with 64/512 hidden units and a ReLU activation function, to learn a higher-level representation of the sequence.
- Dropout layer dropping 50% of hidden units
- Another dense layer to the output of the dropout layer, with 64 hidden units and a ReLU activation function.
- Final dense layer to the output of the second dense layer, with a single output and sigmoid activation function.