IOWA STATE UNIVERSITY

Department of Computer Science

Tweet Sentiment Analysis

Ryan Herren, Tanner Dunn, Mario Galvao-Wilson, Jayant Shah

12/5/21

Background

- What is the background of your task?

  This project aims to build a Machine Learning Model that allows you to evaluate the sentiment behind Tweets based off of the language, emojis, and other characters that they use.

- What kind of data you have?

  Tweets are obtained live from Twitter via the Twitter API and the Python package TweePy. Once handled, they are tokenized into words, emojis, mentions, and hashtags. Then, the tokenized Tweet is run through the pre-trained model and evaluated using SVMs to predict the sentiment (positive, negative, or neutral) of the Tweet.

- What problem you want to solve?

  Understanding sentiment when reading Tweets can sometimes be very difficult due to the lack of verbal inflection in typed/written language. Due to this, there is often misunderstands in the true meanings and attitudes behind different texts.

Datasets

- What is the dataset?

  The dataset is live tweets being brought into our ipynb file via the Twitter API. This means the data can be any series of Tweets by a user, any series of tweets that share a common hashtag, or a series of tweets derived from a users timeline based off of who they follow.

- What features you use?

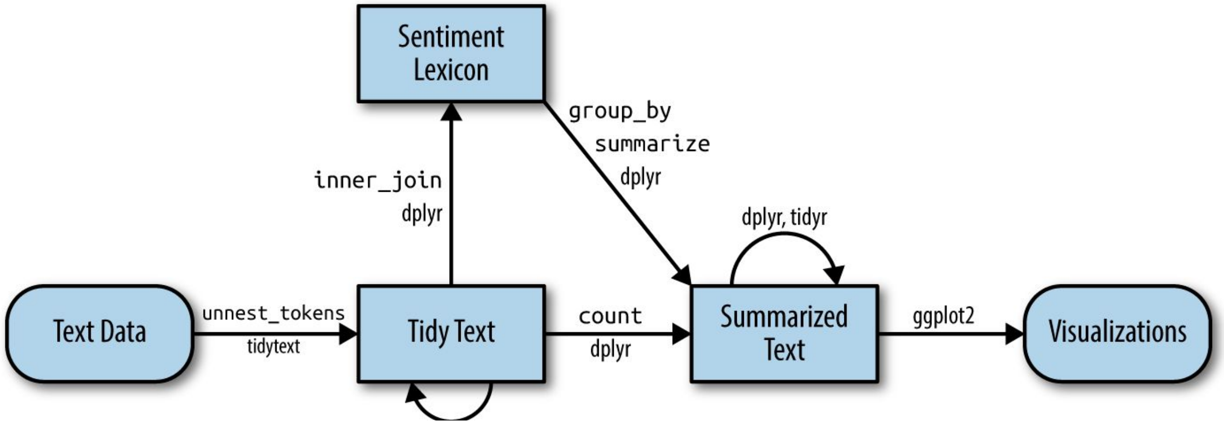
  The features are as follows: user, mentions, emojis, URLs, hashtags, and words. The labels will be in correspondence with the training set used with Bing Liu's Opinion Lexicon (Hu and Liu, KDD-2004).

- What is the expected output by your machine learning models.

  We expect to be able to submit Tweets to the model and get a predicted sentiment value of positive or negative

Related work

- The main related work to this project lies in the use of the Opinion Lexicon. The Opinion Lexicon was produced from a study, by Bing Liu, that began in 2004. The goal of the study was to identify words that are used primarily in a non-neutral setting in terms of sentiment. From the study, they were able to make an Opinion Lexicon, consisting of over 6800 positive or negative words that dictate true sentiment when used in a written manner.



Your method

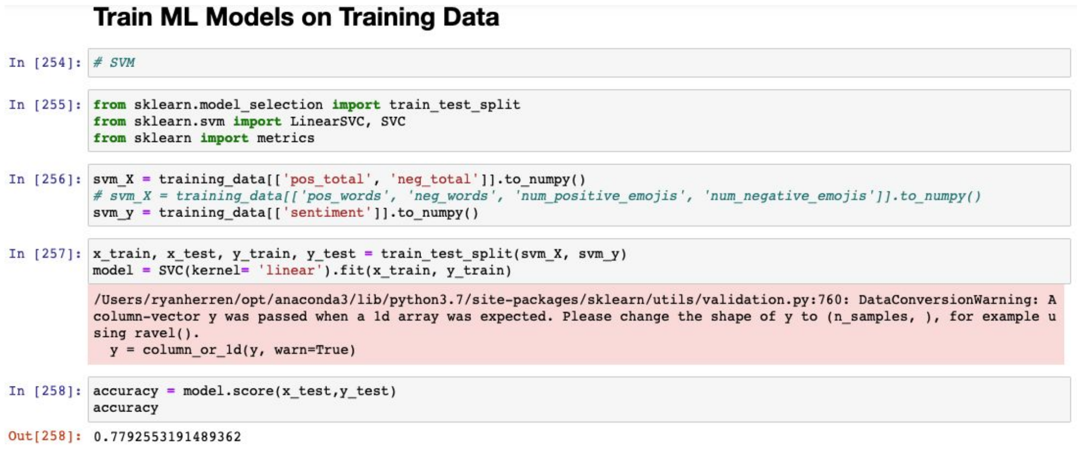
- **Intuition**
  - First we imported the Opinion Lexicon full of the positive and negative words
  - Load in 1.6 million tweets and their sentiment (Training Data)
- **Data processing**
  - Configure Twitter API connection to pull in live tweets of choice (Testing Data)
  - Connect to twitter API, get tweets as a json file and convert from json to a dataframe.
  - Use Tokenizing functions that break tweets into words, emojis, mentions, and hashtags
  - Run functions on tokenized dataset to obtain a number of negative and positive words and add those values to the dataframe
- **Algorithm details**
  - Loop through tokenized tweet text word by word, compare it to opinion lexicon and update positive and negative counts
  - Train SVM and Decision Tree models on training data and use to predict live Tweets from the API

Experimental setups

What features to use?

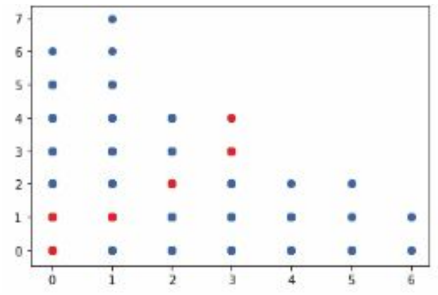
- Tweets
  - Tweet_Id, Created_at, Tokenized, Tweet_text
  - Sentiment
  - Number of positive words, number of positive emojis
  - Number of negative words, number of negative emojis
  - Number of hashtags, mentions
- **What are the choices of hyper-parameters?**
  - Number of positive, negative words
  - Number of positive, negative emojis
  - Sentiment
- **How about the training process?**
  - Support Vector Machines - ~77% accuracy
  - Decision Tree Classification without Neutrals - ~78% accuracy
  - Decision Tree Classification with Neutrals - ~99% accuracy

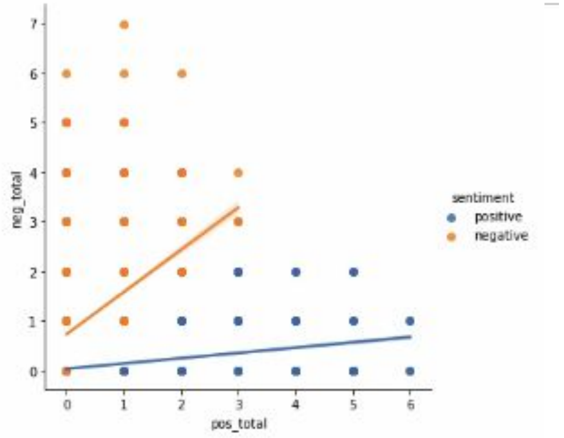
Experimental results



Train ML Models on Training Data

```
In [254]: # SVM

In [255]: from sklearn.model_selection import train_test_split
          from sklearn.svm import LinearSVC, SVC
          from sklearn import metrics

In [256]: svm_X = training_data[['pos_total', 'neg_total']].to_numpy()
          # svm_X = training_data[['pos_words', 'neg_words', 'num_positive_emojis', 'num_negative_emojis']].to_numpy()
          svm_y = training_data[['sentiment']].to_numpy()

In [257]: x_train, x_test, y_train, y_test = train_test_split(svm_X, svm_y)
          model = SVC(kernel= 'linear').fit(x_train, y_train)

          /Users/ryanherren/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:760: DataConversionWarning: A
          column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example u
          sing ravel().
            y = column_or_1d(y, warn=True)

In [258]: accuracy = model.score(x_test,y_test)
          accuracy

Out[258]: 0.7792553191489362
```





Using SVM, we achieved ~78% accuracy when testing on our training data.

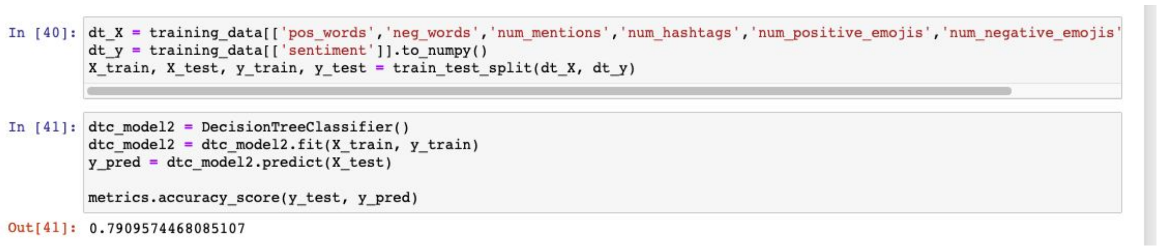
We found that it is difficult for the model to decipher between tweets with similar amounts of negative/positive words. If we had strengths of negative and positive words(ex. 'distraught' is more negative than 'confused') it would be easier to make those decisions

Similarly, the predicted sentiment of tweets with low amounts of both positive and negative words is easily swayed by one or two words.

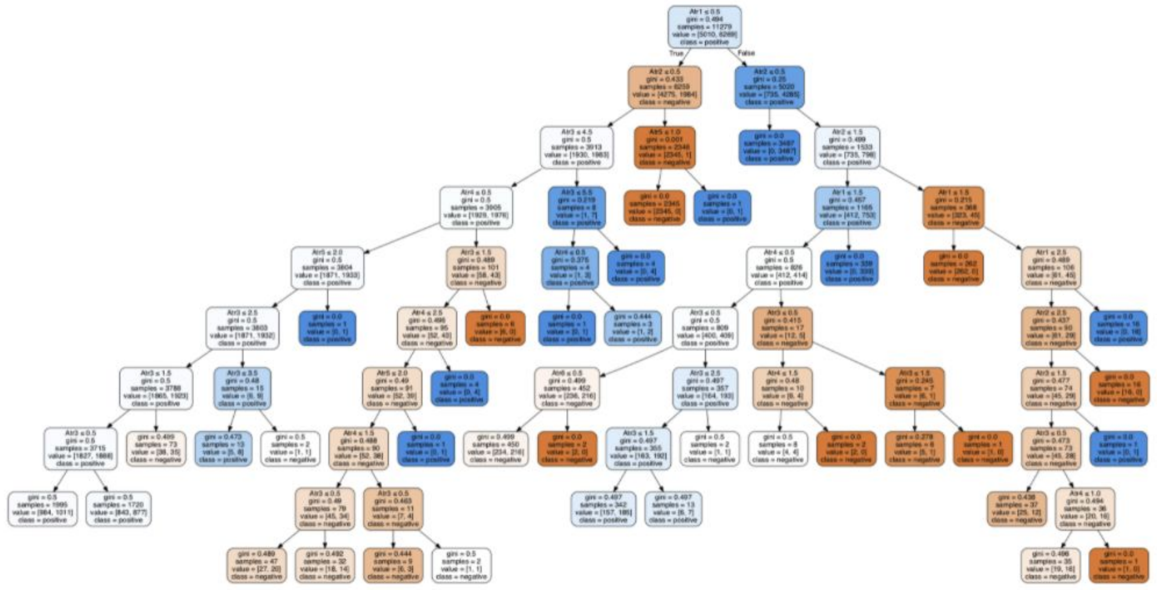
The largest issue is the model's ability to understand sarcasm/figurative language. On top of this, many people will use crying emojis as 'tears of joy' and as 'tears of sadness'. Our best guess at using emojis was to find the emojis most used in positive and negative settings and count the uses of them based off of a static list

Experimental results cont.

Using Decision Trees



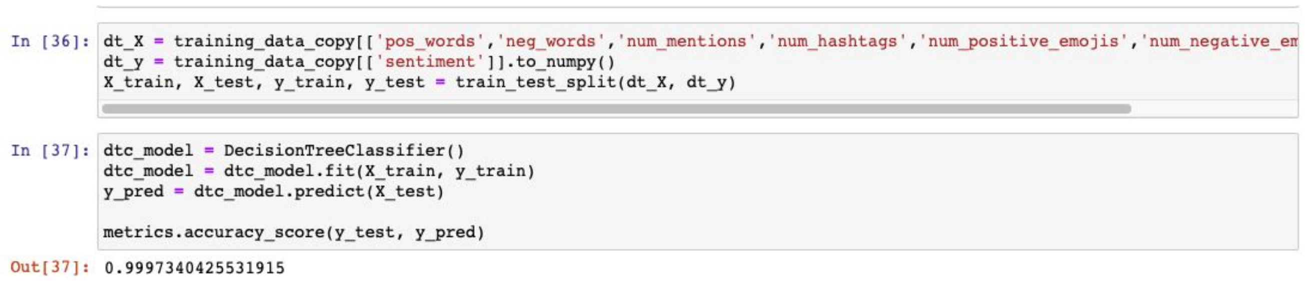
```
In [40]: dt_X = training_data[['pos_words','neg_words','num_mentions','num_hashtags','num_positive_emojis','num_negative_emojis'
         dt_y = training_data[['sentiment']].to_numpy()
         X_train, X_test, y_train, y_test = train_test_split(dt_X, dt_y)

In [41]: dtc_model2 = DecisionTreeClassifier()
         dtc_model2 = dtc_model2.fit(X_train, y_train)
         y_pred = dtc_model2.predict(X_test)

         metrics.accuracy_score(y_test, y_pred)

Out[41]: 0.7909574468085107
```



This model performed very similarly to the SVM model, but did so in a much more complex way. There is probably some overfitting here in terms of the tree, but in general, it struggled with determining neutrally-toned tweets and assigning positive/negative sentiment ratings to them

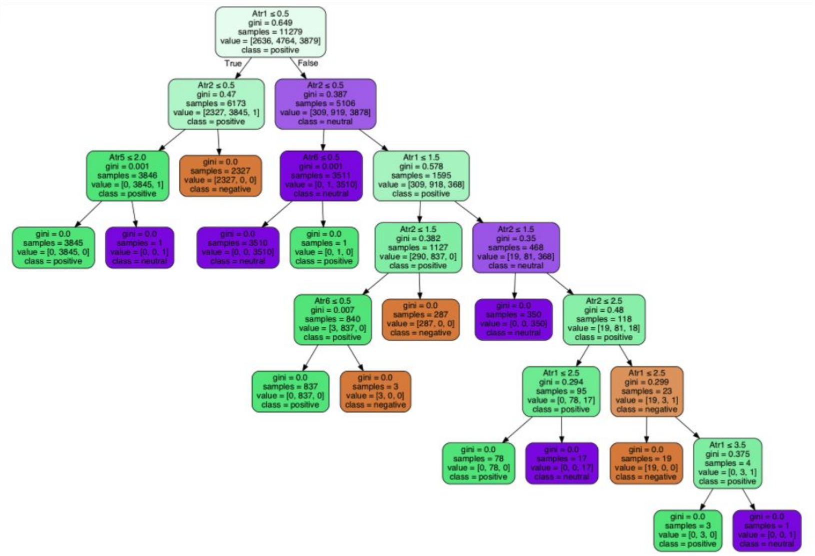
Experimental results cont.

Using Decision Trees with a neutral sentiment rating possible



```
In [36]: dt_X = training_data_copy[['pos_words','neg_words','num_mentions','num_hashtags','num_positive_emojis','num_negative_em
         dt_y = training_data_copy[['sentiment']].to_numpy()
         X_train, X_test, y_train, y_test = train_test_split(dt_X, dt_y)

In [37]: dtc_model = DecisionTreeClassifier()
         dtc_model = dtc_model.fit(X_train, y_train)
         y_pred = dtc_model.predict(X_test)

         metrics.accuracy_score(y_test, y_pred)

Out[37]: 0.9997340425531915
```

Out[38]:


Atr1 ≤ 0.5 gini = 0.649 samples = 11279 value = [2636, 4764, 3879] class = positive
True
False
Atr2 ≤ 0.5 gini = 0.47 samples = 6173 value = [2327, 3845, 1] class = positive
Atr2 ≤ 0.5 gini = 0.387 samples = 5106 value = [309, 919, 3878] class = neutral
Atr5 ≤ 2.0 gini = 0.001 samples = 3846 value = [0, 3845, 1] class = positive
gini = 0.0 samples = 2327 value = [2327, 0, 0] class = negative
Atr6 ≤ 0.5 gini = 0.001 samples = 3511 value = [0, 1, 3510] class = neutral
Atr1 ≤ 1.5 gini = 0.578 samples = 1595 value = [309, 918, 368] class = positive
gini = 0.0 samples = 3845 value = [0, 3845, 0] class = positive
gini = 0.0 samples = 1 value = [0, 0, 1] class = neutral
gini = 0.0 samples = 3510 value = [0, 0, 3510] class = neutral
gini = 0.0 samples = 1 value = [0, 1, 0] class = positive
Atr2 ≤ 1.5 gini = 0.382 samples = 1127 value = [290, 837, 0] class = positive
Atr2 ≤ 1.5 gini = 0.35 samples = 468 value = [19, 81, 368] class = neutral
Atr6 ≤ 0.5 gini = 0.007 samples = 840 value = [3, 837, 0] class = positive
gini = 0.0 samples = 287 value = [287, 0, 0] class = negative
gini = 0.0 samples = 350 value = [0, 0, 350] class = neutral
Atr2 ≤ 2.5 gini = 0.48 samples = 118 value = [19, 81, 18] class = positive
gini = 0.0 samples = 837 value = [0, 837, 0] class = positive
gini = 0.0 samples = 3 value = [3, 0, 0] class = negative
Atr1 ≤ 2.5 gini = 0.294 samples = 95 value = [0, 78, 17] class = positive
Atr1 ≤ 2.5 gini = 0.299 samples = 23 value = [19, 3, 1] class = negative
gini = 0.0 samples = 78 value = [0, 78, 0] class = positive
gini = 0.0 samples = 17 value = [0, 0, 17] class = neutral
gini = 0.0 samples = 19 value = [19, 0, 0] class = negative
Atr1 ≤ 3.5 gini = 0.375 samples = 4 value = [0, 3, 1] class = positive
gini = 0.0 samples = 3 value = [0, 3, 0] class = positive
gini = 0.0 samples = 1 value = [0, 0, 1] class = neutral

This essentially turns any ambiguity between positive and negative and turns it into neutral. This resolves the major issues of accuracy because it allows for a middle ground between positive and negative, thus allowing the model to classify tweets easily when there are low numbers of positive/negative words/emojis or similar amounts of both

Conclusion and Future Work

In the future, we would like to use the newest version of the tweepy package. For now we are using the most up to date supported version. The new version allows for more metadata with the Tweets.

Additionally, having a spectrum of words from ‘most positive’ to ‘least positive’ and ‘most negative’ to ‘least negative’ would allow for better training of models. This would make words like ‘distraught’ have more effect on sentiment in a negative way than ‘confused’ would.

The last thing to add would be the comparative lexicon that stemmed from Bing Liu’s same study, which evaluates sentences with terms like “can’t” followed by positive and negative words