Final Project Documentation

Project Contributors: Nadia Mohiuddin (nadiam2), Pallavi Narayanan (pallavi3), Thaniel Tong (tztong2)

Overall Use:
The purpose of this project is to classify tweets into categories `SARCASM` or `NOT_SARCASM`. The tweets that we categorize are written in response to other tweets, which are the context data. This is a form of sentiment analysis.

How to get the code:

```
git clone <repository url>
```

How to run the code:

```
cd data
python Sarcasm.py
```

Software Installation Details:

- To install each Python library to use in our project, we ran "pip install <library name>" in our terminal before using the library in our actual code.
- We then used regular import statements to use the libraries in our code.

Model used:

- We used the SVM model from the sklearn.svm library. All the training was done through SVM. We did some preprocessing of the data in order to improve the performance.
- First we tokenized all the words in the given tweets, then we removed stopwords, "`@USER`" and "`<URL>`" tags in the tweet.
- Then we lemmatized the data so we could get the root word.
- Finally to end the preprocessing, we converted emojis into strings which helped to increase the accuracy of classification, as emojis can often be a signal of sentiment.
- In order to train on context data we appended the context data to the response data to create a more robust text data to train on.
- We used TF and IDF weighting to convert the words to float probabilities, giving importance to frequent terms in a document, and infrequent terms across the entire collection.
- We fine tuned the hyperparameters for SVC using the RandomSearchCV algorithm from sklearn.
- We then used these hyperparameters to train our SVC model with the training TF-IDF scores, and then predicted the test data labels using the testing TF-IDF scores.
- Lastly, we output the test labels to a text file.

What we tried:

- We also tried to preprocess the responses and contexts by stemming each word and making each word lowercase. However, these methods did not give us the optimal F1 score.
- We attempted to use other sklearn classifiers such as GaussianNB, MultinomialNB, and

RandomForestClassifier, but none of these classifiers performed as well as SVC.

- We tried to use the isalpha() function to remove all non alphabetic characters from the responses and contexts, but this also did not give us the optimal F1 score.
- We tried using the default parameters for SVC, but those parameters did not perform as well as the hyperparameters we fine tuned using RandomSearchCV.
- We also tried to fine tune parameters with GaussianNB, MultinomialNB, and RandomForestClassifier. We used GridSearch to find tune hyper parameters in Naive Bayes and a large multi-nested for-loop (mimicking GridSearch) for the RandomForestClassifier. However these still did not perform as well as SVC did.

Final Data:

Precision: 0.6468010517090271
Recall: 0.82
F1 score: 0.7231749142577168

Contributions:

- The team started off working together, sharing ideas while one person typed with their screen shared to the other team members.
- Later on, the team members split up to try different classifiers and methods of classifying the dataset. This approach was taken to save time as three computers train and test in parallel rather than waiting on one computer to run.

Software Documentation/Tutorials:

- Sklearn SVC
    - https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- Sklearn RandomSearchCV Algorithm
    - https://intellipaat.com/community/18009/what-is-a-good-range-of-values-for-the-svm-svc-hyperparameters-to-be-explored-via-gridsearchcv
    - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
- Lemmatization
    - https://www.geeksforgeeks.org/python-lemmatization-with-nltk/
- Sklearn TfidfVectorizer
    - https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- Pypi emoji
    - https://pypi.org/project/emoji/