

**BUDT737: Big Data and AI Final Project Report**

Sofia Marie Vinas

Jacob Mackoff

Liheng Wang

Karthik Nuvvula

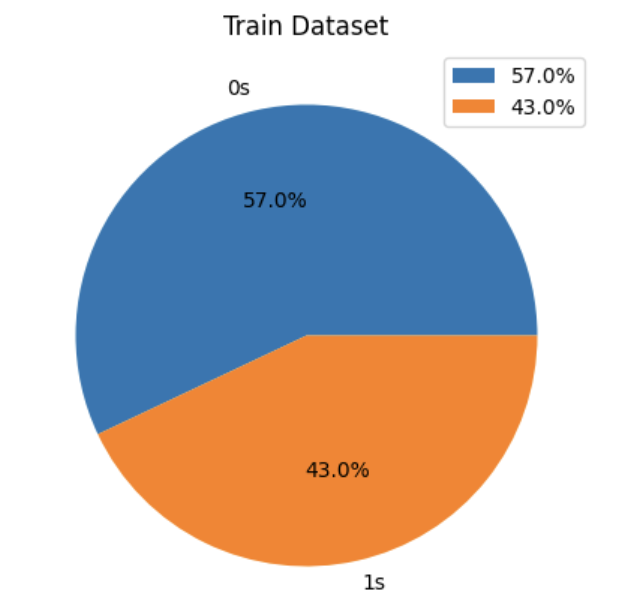
Jie Gao

**Kaggle Team Name:** Sofia Marie Vinas

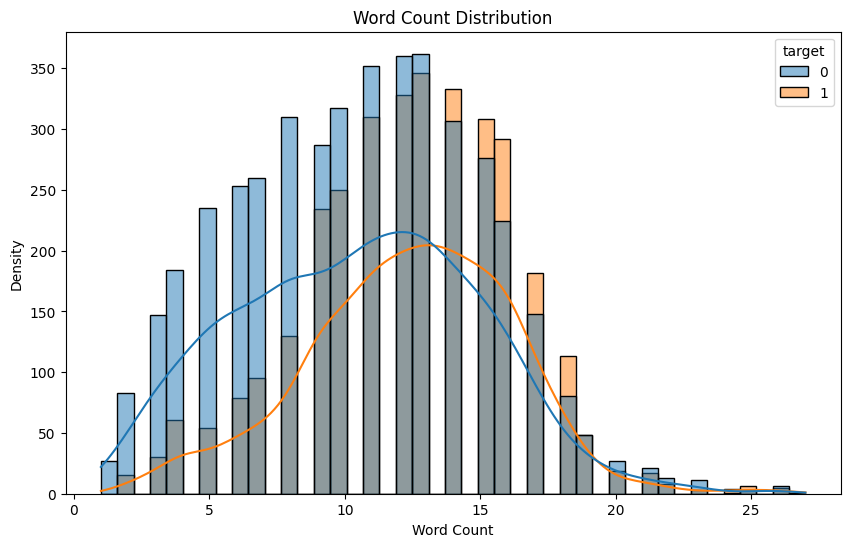
**Original Code:**

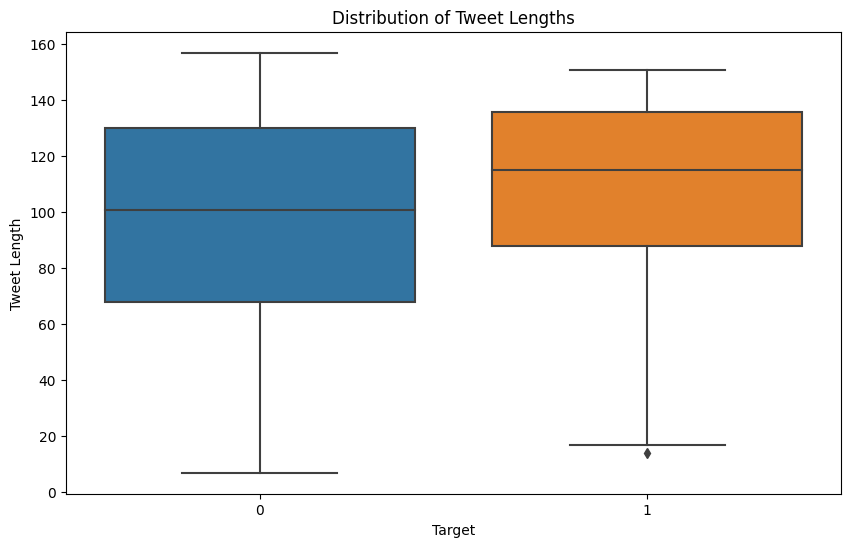
* [Beginners NLP DisasterTweetss](https://www.kaggle.com/code/priteshpatel25/beginners-nlp-disastertweetss) (Chosen model used this code)
* [Introduction to TensorFlow and NLP](https://www.kaggle.com/code/olemagnushiback/introduction-to-tensorflow-and-nlp?scriptVersionId=126979046)
* [Nlp tweets NN](https://www.kaggle.com/code/bennaserabdelmomen/nlp-tweets-nn)

**Exploratory Data Analysis**

****

When examining the above pie chart, we can conclude that the training dataset is close to evenly split but predominantly consists of 0s, meaning most of the observations are false alarms regarding disasters. 57% of all tweets in the dataset are false alarms while 43% are tweets that are truly referencing disasters.





The following box plots display that both false alarms and tweets actually talking about disasters possess similar distributions regarding tweet lengths. This means that both types of tweets may possess similar lengths. However, false alarm tweets seem to have a slightly wider distribution meaning there is more variability in the length of these types of tweets in the dataset. True disaster tweets have less variability and on average, tend to be slightly longer.

**Procedure, Methods, and Functions Used**

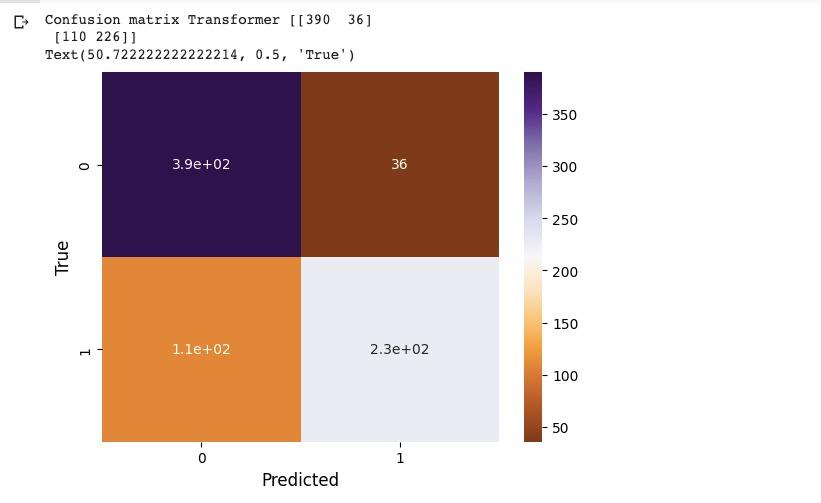
* We used numpy for the linear algebra, and pandas for data framing as well as the data processing for the csv files. We used many different packages and plotting tools to help us compute and visualize the data. We included those imports at the top of our colab notebook so that we could implement them into our multiple models. After importing, we used the process\_input function that was provided to process and clean the data. In terms of the data format, we were given three csv files- a training file, a testing file, and a sample submission file (all in the same format). Each sample in the training and testing files contained a text of the tweet, a keyword from the tweet that “could” be blank, and the location the tweet was sent from that could also be blank. We used the train\_test\_split function that we have used in previous homeworks and activities to split the data. We also use the CountVectorizer function from sklearn to convert a collection of text to a matrix of token counts. The shape functions are used to double check that the training and validation splits are of the correct size.
* Once we decide what classifier we want to use, we can call the fit and predict functions with the training data. To find the accuracy score between the actual and predicted models we use the accuracy\_score, f1\_score, and recall\_score functions. This is how we get the total accuracy classification score that we can use to compare multiple models.
* Taking only the word count in documents may have certain repercussions. The presence of frequent word vectors will not be sparse, including stopwords, though the word itself may not necessarily be important. Rare words look very sparse and occur infrequently, hence they are less important. So we can use TF-IDF vectorization to address and work around these problems. We use the train\_test\_split function differently than before, this time utilizing the text column values of our training set, and the target column values of the training set. We also slightly adjusted the test size parameter to fit the size of the submission- 3263 values. With all of our models we periodically print the length of the prediction using the len function to make sure it fits the submission format, or else Kaggle will not accept it. We call the TfidfVectorizer constructor, and then transform the training data and validation data into vectors. We used logistic regression as the classifier for the first model.
* For the decision tree, we imported the tree library from sklearn to create a decision tree classifier with a maximum depth of 60. Similarly as we did with our models above, we fit the tree with training vectors. We continuously print out the length of our y\_pred variables to make sure our output matches the length of the intended submission csv file.
* For the SVC model, we had to install a sentence transformer, which we referenced in the code. To create the model itself, we simply call the SentenceTransformer function with all-MiniLM-L12-v2 as the parameter () and assign it to our model variable. Printing the model itself allows us to see its structure. Next we encode the training, validation and test data, call the SVC function with the regularization parameter and fit the model with the encoding training/validation/test data sets.

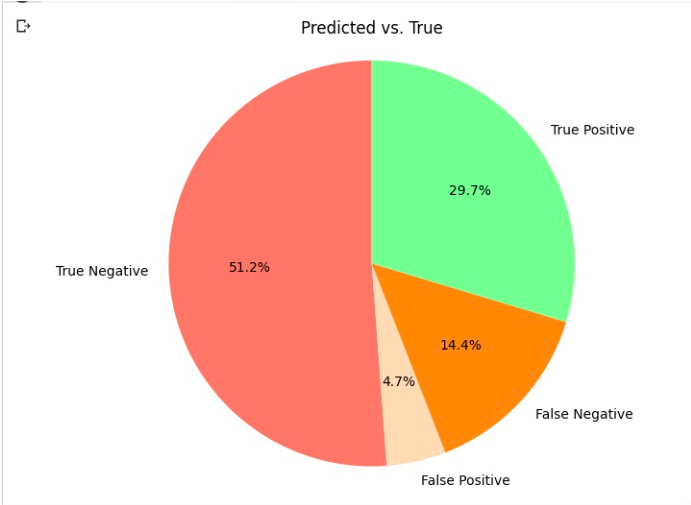
**Score from Kaggle.com**

* 81.152% using the transformers with SVC model

**Results and Reports**

* The first model we implemented used logistic regression as the classifier. Using this function we were able to achieve an accuracy score of around 80%. We called this classifier clf.
* The second model we generated used naive bayes as the classifier. Using this function we were able to achieve an accuracy score of 79%. We called this classifier clf2. It was much easier to compare the two models because we kept the training and testing data the same- the only difference was the classifier we used. And so all we had to do was place the variable we wanted to test in the code (either clf for logistic regression or clf2 for the naive bayes classification).
* For the TF-IDF vectorizer, we were able to generate an accuracy score of about 80% using logistic regression as the classifier.
* For the decision tree, we were able to generate an accuracy score of about 74% using the imported DecisionTreeClassifier.
* Finally we implemented our transformer model. Using this model we were able to generate an accuracy score of a little over 81%. This model was built using a support vector classifier (SVC) with the regularization parameter set to 0.3, knowing that this parameter must be positive.
* For the other models, I ran the same few lines of code to run the models on test data. We created a pandas dataframe called df, and used the vectorizer.transform method to transform the text column of the test dataset. Then we called the predict function with the specified classifier and the vectors\_test variable to generate the given submission.csv file that we turned in. For the models that did not perform as well, we comment out the code that generates the csv file- until we find the best model.
* For the last transformer model with the support vector classifier, we again fit the model with the training data. This model uses cross entropy loss and AdamW as the learning rate. We print the transformer accuracy using the accompanying function and call the predict function with the modelTransformer variable. The answer then gets placed in another pandas dataframe to generate the final submission.csv file that will be turned in.
* For all models, we tried to generate an accompanying confusion matrix with the corresponding validation and predicted sets.
* On Kaggle, we submitted quite a few models. At the beginning we were averaging around 75% accuracy using naive bayes, logistic regression, decision trees and random forest.





**What did each of you do**

* Sofi Vinas: Implemented the svc model and commented/modified existing code. Created a confusion matrix for each model to compare efficiency and accuracy of classifiers with accompanying models.
* Jacob Mackoff: I adjusted three different models using three of the five provided code to try to enhance the accuracy and the kaggle scores of the existing model. First, I adjusted the Beginners NLP DisasterTweets to try to get random forest to run as it is a very powerful classifier and would not run with the original code. Once I got it to run, it ended up having poor performance. I then updated the neural network in NLP tweets NN and the neural network using TensorFlow in the Introduction to TensorFlow and NLP code. Both experienced increases in accuracy from Random Forests however they were not as good of a classifier as the svc model Sofi made.
* Liheng Wang: Assisted in improving the model on creating ROC Curve and helped improve the documentation.
* Jie Gao: I conducted exploratory data analysis by data visualization to investigate the characteristics of the data. I tried to leverage the Random Forest model to improve the accuracy, but compared to other models, it didn’t yield the desired results.
* [Karthik Nuvvula](mailto:knuvvula@umd.edu): I tried the Bert model with neural networks and got an accuracy of 0.7937 and I also tried to increase the accuracy of the existing model and helped in the EDA of prediction part.

**What have each of you learned**

* Sofi Vinas: I learned how to fit the data with specific models and how those models perform against each other. I also learned about support vector classifiers and transformers. Any new information I had to implement in the code I referenced through markdown cells and comments.
* Jacob Mackoff: This project improved my ability to try to edit code for a variety of different models with the goal of increasing accuracy. Throughout the project, I had to try to use different types of models and change them constantly to achieve better performance. This gave me a better understanding on how the models I chose works (neural networks and random forest) and how different layers of these models can impact how efficient of a classifier it is. In addition, this project gave me a better understanding on Natural Language Processing models,
* Liheng Wang: I learned how to fit data into a more complex model with different classifiers. I also learned how to communicate as a team in a complex coding situation.
* Jie Gao: I learned how to investigate the characteristics of data through exploratory data analysis and how important it is to select the right classifier which impacts the accuracy. I also learned the significance of comparing and evaluating different models.
* [Karthik Nuvvula](mailto:knuvvula@umd.edu): I learnt how to implement different types of ensemble method algorithms in Machine learning and also learnt how to improve the performance metrics of the models by trying different batch sizes, using different optimizers, different activations and splits.