NLP EX3 Report

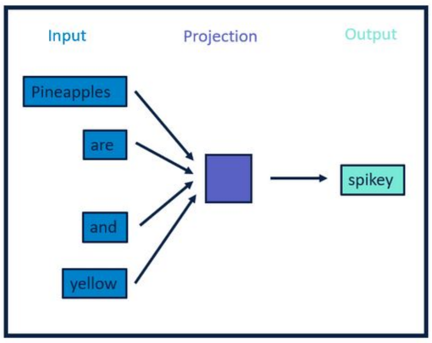
Preprocess:

To process the data we started with an assumption that all realDonaldTrump posted from Android are Trump's tweets. Otherwise, we assume it's his staff or others. We marked these tweets with label 0 (Trump's tweets) while all others got label 1(other). We also removed rows that had NA's. Later we removed the columns that were not necessary anymore, tweet_id and device, so that the test and train set will have the same columns.

Features & Data Representation:

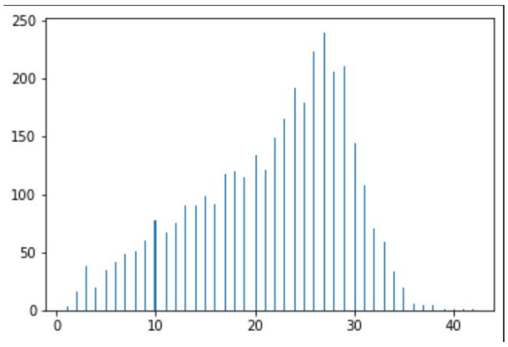
In order to be able to really compare the different models we used we decided to only use embeddings as our features. We trained an embedding model using our tweet_text. The process was the following:

- We first created a vocabulary of all words in the corpus. We did this using NLTK TweetTokenizer function.
- We also created a CBOW model that uses a sliding window of two words in each direction of the target word (2 words before and 2 after). We used the CBOW model to feed our embedding layer.



CBOW

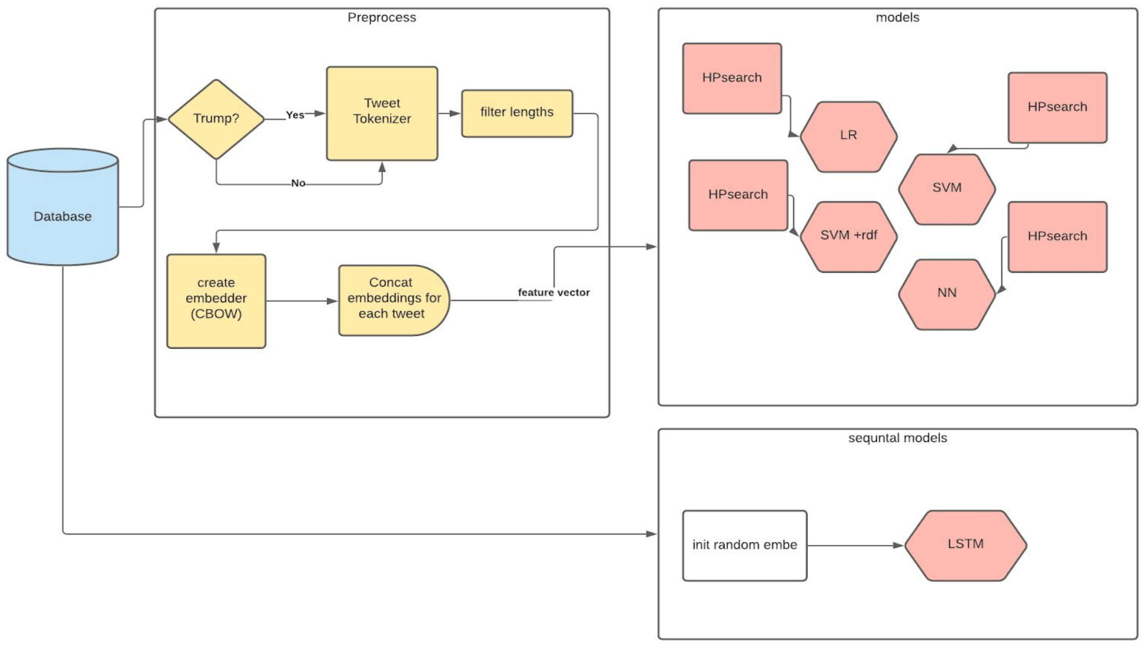
- Then, we used the predict function to get the embeddings for each tweet - we used the TweetTokenizer to split the tweet and then got the embedding values for each word in the split tweet. Before using tweetTokenizer we used a simple normalizing function that padded punctuations with space and then split the tweet by spaces. We decided to switch to tweetTokenizer since it deals better with tweets syntax. We will mention that while using the first methods we got substantially more tokens (most tweets were still around 50 tokens but we had outliers with hundreds of them), and after we switched to tweetTokenizer most tweets were smaller - as seen in the figure. Since most tweets had less than 35 tokens, we decided to limit all tweets to 35 tokens - everything beyond this was not used in our embedding model.



- For each tweet, we added the embeddings for its word in the features column, and those are the features we used in all the models but LSTM (will be explained later) and SVM. For SVM we used the embedding features after conducting a PCA dimensionality reduction. The reason for this is that SVM had a hard time converging and the dimensionality reduction helped the algorithms to converge.

To summarize, we only used our tweet_text column to create the embedding and we used the embeddings to train our models. Meaning, the input to our models was a vector of embedding values.

A flowchart that demonstrated the process:



The algorithms

1. Logistic regression
    a. max_iter - 800
    b. C - 2.7
2. SVM linear
    a. gamma - auto
    b. C - 1.8
3. SVM non-linear
    a. RBF kernel for non-linear
    b. C - 1.8
4. Neural Network using PyTorch -
    a. 2 hidden layers
    b. ReLU activation function
    c. Dropout of 0.1
    d. 200 neurons in the first layer and 200 in the second layer
    e. Learning rate of 0.001

5. RNN using PyTorch -
    a. Embedding layer with 10 dimensions
    b. 124 hidden layers of RNN
    c. Last vector from each batch
    d. Output layer using sigmoid
6. Sklearn Gradient Boosting Classifier
    a. n_estimators = 200
    b. max_depth = 7
    c. learning_rate = 0.09

For Logistic Regression, SVM and GradientBoostingClassifier we used 5 k-folds cross-validations to examine the algorithm's performance. For the NN and RNN, we split the data into train and validation sets. All algorithms but the RNN received the embedding values as their features. For the RNN, we trained a new embedding layer within the network. The reason for using a new embedding layer is that we wanted the embeddings to be time-sequenced. In our embedding, we used CBOW with a sliding window of two, but the context for the tweets can be longer than that. We don't know how far behind the context impacts so we thought to create a "language model" in a dynamic way - according to the label, it will know how much information it needs before the current word in the tweet. Moreover, the RNN model takes into consideration batches of tweets and not a single tweet at a time and it can impact the results.

Performance

Algorithm	F1-Score	Accuracy	Recall
Logistic Regression	0.704	0.791	0.687
SVM - linear	0.732	0.824	0.666
SVM - non-linear	0.751	0.839	0.671
NN	0.722	0.742	0.742
RNN	0.54	0.65	0.65
Gradient Boosting	0.756	0.844	0.683

For logistic regression, SVM (both linear and non-linear), and Gradient Boosting we used RandomizedSearchCV to tune the hyperparameters and create the cross-validation. For each algorithm, we used the best estimator to check the performance and since we used cross-validation, it enhances the differences between the 3 algorithms. For NN and RNN we split the data to train and test without using cross-validation and the results were low regardless of the different hyperparameters used. For NN we also tried tuning the number of neurons in each layer and took the values of the best network.

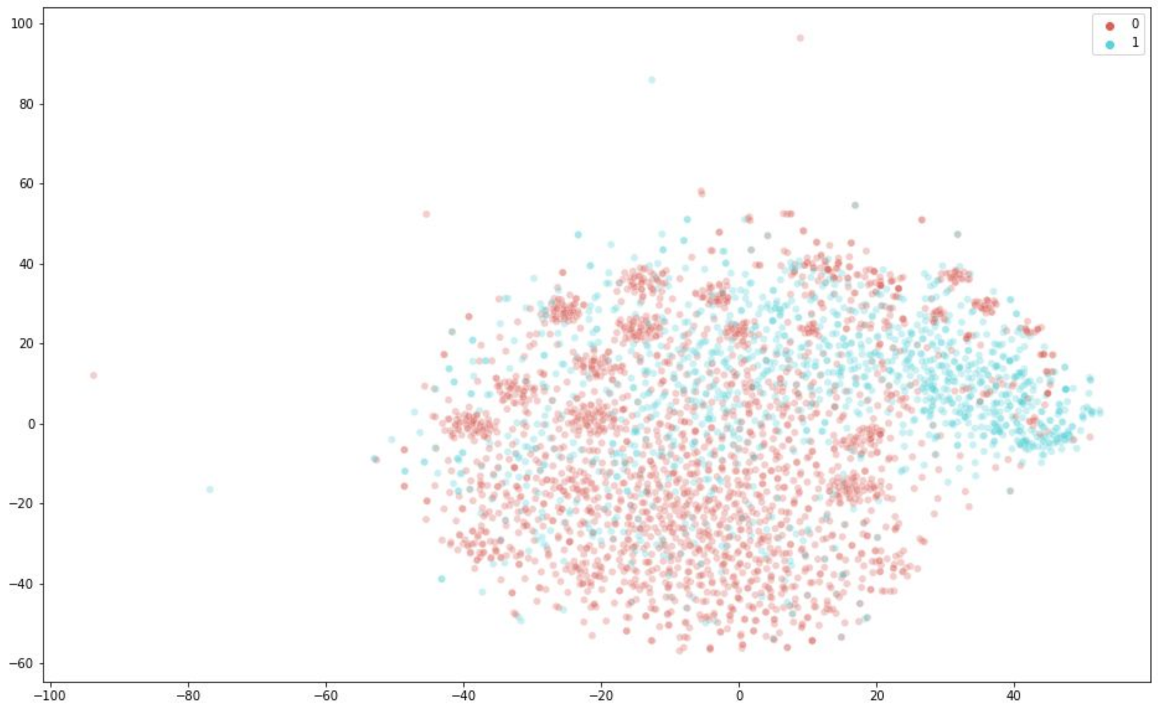
Best Model

The best model from the ones we used was the Gradient Boosting Classifier of sklearn.
The parameters used to get the best estimator are:

- n_estimators = 200
- max_depth = 7
- learning_rate = 0.09

Random Thoughts

There is no certain point in the questions that address this, but an interesting plot we encountered while using dimensionality reduction is the following:



All the red dots are labeled as Trump's tweets. What is interesting is that we can see various clusters of tweets in the data. We are not sure if that implies that indeed different people used Trump's Twitter account, or is it just Trump talking on different events, but the embeddings we get is different.