# CAPSTONE PROPOSAL

# Tweet Sentiment Extraction

## 1. Domain Background

Microblogging websites like Twitter are platforms for the expression of opinions in real-time to millions of users. These tweets can potentially impact a person, brand, company, or country due to the freedom of expression, and the wide reach one can potentially obtain on the platform. A lot of effort and research [1] [2] [3] [4] has gone into understanding the sentiment of millions of tweets being made by the varied users of the platform. Running a sentiment analysis machine learning model allows brands/companies to monitor the sentiment of their campaign in real-time, thus allowing them to respond effectively to any negative sentiment on the rise.

I've built various sentiment analysis models in both professional and academic capability, most recently as part of Udacity's Machine Learning Nanodegree. This proposal is my attempt to go a step deeper into this domain.

## 2. Problem Statement

We have seen phenomenal growth in the accuracy of predictions by a sentiment analysis model in the past couple of years. However, a person must still go through the model predicted sentiments to capture patterns of words that reflect the sentiment. A machine learning model that captures the words in a tweet that support a positive, negative, or neutral sentiment can work in tandem with a regular sentiment analysis model.

Kaggle has created a competition for this task by offering the 'Sentiment Analysis: Emotion in Text' dataset from Figure Eight's Data for Everyone platform [5]. Thus, the objective is to look at the labeled sentiment for a given tweet and figure out what word or phrase best supports the sentiment.

## 3. Datasets and Inputs

As mentioned in the Problem Statement above, I'll be using the 'Sentiment Analysis: Emotion in Text' dataset [5]. This dataset is available for use under Creative Commons Attribution 4.0. International license. The dataset is split into training and test set. Being part of active competition, the test set does not have true labels.

The data is saved in a comma-separated file. Each row contains the 'text' of the tweet and a 'sentiment' label. In the training set, we have the 'selected_text'; that contains a word or phrase drawn from the tweet that encapsulates the provided sentiment. The 'selected_text' field is what the trained machine learning model will be predicting.

Here is a sample of the training datastet:

```
train.head()
```

| | textID | text | selected_text | sentiment |
|---|---|---|---|---|
| 0 | cb774db0d1 | I`d have responded, if I were going | I`d have responded, if I were going | neutral |
| 1 | 549e992a42 | Sooo SAD I will miss you here in San Diego!!! | Sooo SAD | negative |
| 2 | 088c60f138 | my boss is bullying me... | bullying me | negative |
| 3 | 9642c003ef | what interview! leave me alone | leave me alone | negative |
| 4 | 358bd9e861 | Sons of ****, why couldn`t they put them on t... | Sons of ****, | negative |

We won't require the column 'textID' for our model training here. There are 3 unique values of sentiment: positive, negative, and neutral.

```
train.isna().sum()
```

```
textID          0
text            1
selected_text   1
sentiment       0
dtype: int64
```

The dataset has only one 'textID' with NaN value for the columns 'text', and 'selected_text'. I will drop this entry in the data preprocessing stage.

The training dataset will have a total of 27480 data points after dropping the NaN entry. The test dataset has a total of 3534 data points with no NaN values in it.

## 4. Evaluation Metrics

The evaluation metric as defined in the Kaggle competition [5] is the word-level Jaccard score. The Jaccard score between two strings can be calculated using the following Python code snippet:

```
def jaccard(str1, str2):
    a = set(str1.lower().split())
    b = set(str2.lower().split())
    c = a.intersection(b)
    return float(len(c)) / (len(a) + len(b) - len(c))
```

The overall evaluation metric for all tweets in the test set is:

$$score = \frac{1}{n}\sum_{i=1}^{n} jaccard\left( gt_i, dt_i \right)$$

$$where,$$
$$n = number\ of\ \ documents$$
$$jaccard = the\ function\ provided\ above$$
$$gt_i = the\ ith\ ground\ truth$$
$$dt_i = the\ ith\ prediction$$

## 5. Benchmark Model

For a baseline model, I went over some of the submissions on Kaggle and adapted a simple solution [6] that achieved a score of 0.651 on the Kaggle leaderboard. This solution creates a dictionary of words for each sentiment group where the values are the proportion of tweets that contain those words. Afterward, this algorithm is applied to a tweet to extract words or phrases describing the tweet. My solution will try to beat this score, and I describe the approach I intend to take below.

# 6. Solution Statement

As part of preprocessing the data, I intend to tokenize the words using Bert Word Piece tokenizer from Huggingface [7]. I might also look at removing punctuations, stopwords as they are unlikely to have any effect on the predicted sentiment. For this analysis, I intend to do EDA on the 'selected_text' labels to see if they have any punctuations and stopwords among them.

For model training purposes, I intend to leverage transfer learning by stacking pre-trained weights from models like BERT, RoBERTa onto my custom layers. Keeping in mind the objective, it deals with a Question-Answering task, so I'll use the QuestionAnswering versions of the Huggingface BERT model.

# 7. Project Design

## A. Data Preprocessing and Transformation

Being part of a Kaggle competition, the dataset is relatively clean with only one data point with NaN values which can be dropped without any repercussions. I may, however, remove punctuations and stopwords after performing an analysis of their occurrence in 'selected_text'. Being an NLP task, I need to create a numerical mapping of the textual data in the form of numerical vectors. These numerical vectors must be able to preserve the semantic and syntactic relationships in the text. Some of the common approaches for this task that I have previously used are word embeddings like Word2Vec or FastText. One of the newer approaches that I want to experiment with is using the Bert Word Piece Tokenizer.

I will also need to decide on a *max_length* of tweet text to create a homogeneous shape of vectors for each tweet. Fortunately, this is easy for this problem due to the maximum tweet length imposed by Twitter, currently 280 characters. This means that the maximum number of possible worlds can be 140 (when every word is a single letter). I will pad the vectors of tweets that are not of this length.

## B. Data Modeling

I intend to use deep learning algorithms like Transformers (BERT/RoBERTa) for this task. I will also leverage the pre-trained model weights and then stack on some custom layers. I will start by using Adam optimizer in the model. The loss function of categorical cross-entropy is a good choice for this problem.

C. Data Inference

Once the model is trained and the weights are saved, I will apply the same preprocessing and transformation steps on the test dataset and then I'll pass the transformed data for model prediction. Once I get the results, I'll submit on Kaggle to get my score on the leaderboard for the test data (as we don't have true labels for test data).

# 8. Citations


[1] Agarwal, Apoorv, et al. "Sentiment analysis of twitter data." Proceedings of the Workshop on Language in Social Media (LSM 2011). 2011.

[2] Pak, Alexander, and Patrick Paroubek. "Twitter as a corpus for sentiment analysis and opinion mining." LREc. Vol. 10. No. 2010. 2010.

[3] Saif, Hassan, Yulan He, and Harith Alani. "Semantic sentiment analysis of twitter." International semantic web conference. Springer, Berlin, Heidelberg, 2012.

[4] Rosenthal, Sara, Noura Farra, and Preslav Nakov. "SemEval-2017 task 4: Sentiment analysis in Twitter." arXiv preprint arXiv:1912.00741 (2019).

[5] "Tweet Sentiment Extraction." Kaggle, www.kaggle.com/c/tweet-sentiment-extraction/overview/description.

[6] Nkoprowicz. "A Simple Solution Using Only Word Counts." Kaggle, Kaggle, 7 Apr. 2020, www.kaggle.com/nkoprowicz/a-simple-solution-using-only-word-counts.

[7] Huggingface. "Huggingface/Tokenizers." GitHub, 24 Apr. 2020, github.com/huggingface/tokenizers.