Samiksha Patil

Sentiment Analysis of Twitter

Pre-processing Tweets:

- Letter casing: Converting all letters to either upper case or lower case.
- Tokenizing: Turning the tweets into tokens. Tokens are words separated by spaces in a text.
- Noise removal: Eliminating unwanted characters, such as HTML tags, punctuation marks, special characters, white spaces etc.
- Stopword removal: Some words do not contribute much to the machine learning model, so it's good to remove them. A list of stopwords can be defined by the nltk library.
- Normalization: Normalization generally refers to a series of related tasks meant to put all text on the same level. Converting text to lower case, removing special characters, and removing stopwords will remove basic inconsistencies. Normalization improves text matching.
- Stemming: Eliminating affixes (circumfixes, suffixes, prefixes, infixes) from a word in order to obtain a word stem. Porter Stemmer is the most widely used technique because it is very fast. Generally, stemming chops off end of the word, and mostly it works fine.

Here we build a dictionary where the keys are a (word, label) tuple and the values are the corresponding frequency. here 1 is for positive and 0 is for negative.

lookup() is a function that takes in the **freqs** dictionary, a word, and a label (1 or 0) and returns the number of times that word and label tuple appears in the collection of tweets.

count_tweets() that takes a list of tweets as input, cleans all of them, and returns a dictionary.

- The key in the dictionary is a tuple containing the stemmed word and its class label, e.g. ("happi",1).
- The value the number of times this word appears in the given collection of tweets (an integer).

Sentiment Analysis using Naive Bayes

Naive bayes is an algorithm that could be used for sentiment analysis. It takes a short time to train and also has a short prediction time.

- The first part of training a naive bayes classifier is to identify the number of classes that you have.
- You will create a probability for each class. P($Dpos$) is the probability that the document is positive. P($Dneg$) is the probability that the document is negative.

$$\text{P}(Dpos)=Dpos/D$$

$$\text{P }(Dneg)=Dneg/D$$

Where D is the total number of documents, or tweets in this case, $Dpos$ is the total number of positive tweets and $Dneg$ is the total number of negative tweets.

Prior and Logprior

The prior probability represents the underlying probability in the target population that a tweet is positive versus negative. In other words, if we had no specific information and blindly picked a tweet out of the population set, what is the probability that it will be positive versus that it will be negative? That is the "prior".
The prior is the ratio of the probabilities P ($Dpos$)/P($Dneg$) We can take the log of the prior to rescale it, and we'll call this the logprior

$$\text{logprior}=log(P(Dpos)/P(Dneg))=log(Dpos/Dneg)$$

Positive and Negative Probability of a Word

To compute the positive probability and the negative probability for a specific word in the vocabulary, we'll use the following inputs:

- $freqpo$ and $freqneg$ are the frequencies of that specific word in the positive or negative class. In other words, the positive frequency of a word is the number of times the word is counted with the label of 1.
- $Npos$ and $Nneg$ are the total number of positive and negative words for all documents (for all tweets), respectively.
- V is the number of unique words in the entire set of documents, for all classes, whether positive or negative.

$$\text{P}(Wpos)=freqpos+1/Npos+V$$

$$\text{P}(Wneg)=freqneg+1/Nneg+V$$

we add the "+1" in the numerator for additive smoothing

Log likelihood

To compute the loglikelihood of that very same word, we can implement the following equations:

$$\text{loglikelihood}=\log(P(Wpos)/P(Wneg)$$

- In **freqs** dictionary the key is the tuple (word, label) and the value is the number of times it has appeared.It used to compute the positive and negative frequency of each word $freqpos$ and $freqneg$.

train_naive_bayes() calculates the logprior and loglikelihood of the training data set

Now that we have the logprior and loglikelihood, we can test the naive bayes function by making predicting on some tweets.

naive_bayes_predict() function is used to make predictions on tweets.

- The function takes in the **tweet, logprior, loglikelihood**.
- It returns the probability that the tweet belongs to the positive or negative class.
- For each tweet, sum up loglikelihoods of each word in the tweet.
- Also add the logprior to this sum to get the predicted sentiment of that tweet.

$$p=logprior+\sum(loglikelihoodi)$$

If p is >= 0 then the tweet is considered to be positive else negative