C1_W2_Assignment

October 29, 2020

1 Assignment 2: Naive Bayes

Welcome to week two of this specialization. You will learn about Naive Bayes. Concretely, you will be using Naive Bayes for sentiment analysis on tweets. Given a tweet, you will decide if it has a positive sentiment or a negative one. Specifically you will:

- Train a naive bayes model on a sentiment analysis task
- Test using your model
- Compute ratios of positive words to negative words
- Do some error analysis
- Predict on your own tweet

You may already be familiar with Naive Bayes and its justification in terms of conditional probabilities and independence. * In this week's lectures and assignments we used the ratio of probabilities between positive and negative sentiments. * This approach gives us simpler formulas for these 2-way classification tasks.

Load the cell below to import some packages. You may want to browse the documentation of unfamiliar libraries and functions.

```
In [ ]: from utils import process_tweet, lookup
        import pdb
        from nltk.corpus import stopwords, twitter_samples
        import numpy as np
        import pandas as pd
        import nltk
        import string
        from nltk.tokenize import TweetTokenizer
        from os import getcwd
```

If you are running this notebook in your local computer, don't forget to download the twitter samples and stopwords from nltk.

```
nltk.download('stopwords')
nltk.download('twitter_samples')
```

```
In [ ]: # add folder, tmp2, from our local workspace containing pre-downloaded corpora files t
        filePath = f"{getcwd()}/../tmp2/"
        nltk.data.path.append(filePath)
```

```
In [ ]: # get the sets of positive and negative tweets
        all_positive_tweets = twitter_samples.strings('positive_tweets.json')
        all_negative_tweets = twitter_samples.strings('negative_tweets.json')

        # split the data into two pieces, one for training and one for testing (validation set
        test_pos = all_positive_tweets[4000:]
        train_pos = all_positive_tweets[:4000]
        test_neg = all_negative_tweets[4000:]
        train_neg = all_negative_tweets[:4000]

        train_x = train_pos + train_neg
        test_x = test_pos + test_neg

        # avoid assumptions about the length of all_positive_tweets
        train_y = np.append(np.ones(len(train_pos)), np.zeros(len(train_neg)))
        test_y = np.append(np.ones(len(test_pos)), np.zeros(len(test_neg)))
```

2 Part 1: Process the Data

For any machine learning project, once you've gathered the data, the first step is to process it to make useful inputs to your model. - **Remove noise**: You will first want to remove noise from your data – that is, remove words that don't tell you much about the content. These include all common words like 'I, you, are, is, etc...' that would not give us enough information on the sentiment. - We'll also remove stock market tickers, retweet symbols, hyperlinks, and hashtags because they can not tell you a lot of information on the sentiment. - You also want to remove all the punctuation from a tweet. The reason for doing this is because we want to treat words with or without the punctuation as the same word, instead of treating "happy", "happy?", "happy!", "happy," and "happy." as different words. - Finally you want to use stemming to only keep track of one variation of each word. In other words, we'll treat "motivation", "motivated", and "motivate" similarly by grouping them within the same stem of "motiv-".

We have given you the function `process_tweet()` that does this for you.

```
In [ ]: custom_tweet = "RT @Twitter @chapagain Hello There! Have a great day. :) #good #morning

        # print cleaned tweet
        print(process_tweet(custom_tweet))
```

2.1 Part 1.1 Implementing your helper functions

To help train your naive bayes model, you will need to build a dictionary where the keys are a (word, label) tuple and the values are the corresponding frequency. Note that the labels we'll use here are 1 for positive and 0 for negative.

You will also implement a `lookup()` helper 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.

For example: given a list of tweets `["i am rather excited", "you are rather happy"]` and the label 1, the function will return a dictionary that contains the following key-value pairs:

{ ("rather", 1): 2 ("happi", 1) : 1 ("excit", 1) : 1 }

- Notice how for each word in the given string, the same label 1 is assigned to each word.
- Notice how the words "i" and "am" are not saved, since it was removed by process_tweet because it is a stopword.
- Notice how the word "rather" appears twice in the list of tweets, and so its count value is 2.

Instructions Create a function `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).

Hints

Please use the `process_tweet` function that was imported above, and then store the words in their respective dictionaries and sets.

You may find it useful to use the `zip` function to match each element in `tweets` with each element in `ys`.

Remember to check if the key in the dictionary exists before adding that key to the dictionary, or incrementing its value.

Assume that the `result` dictionary that is input will contain clean key-value pairs (you can assume that the values will be integers that can be incremented). It is good practice to check the datatype before incrementing the value, but it's not required here.

```
In [ ]: # UNQ_C1 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
        def count_tweets(result, tweets, ys):
            '''
            Input:
                result: a dictionary that will be used to map each pair to its frequency
                tweets: a list of tweets
                ys: a list corresponding to the sentiment of each tweet (either 0 or 1)
            Output:
                result: a dictionary mapping each pair to its frequency
            '''

            ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###
            for y, tweet in zip(ys, tweets):
                for word in process_tweet(tweet):
                    # define the key, which is the word and label tuple
                    pair = (word,y)

                    # if the key exists in the dictionary, increment the count
                    if pair in result:
                        result[pair] += 1

                    # else, if the key is new, add it to the dictionary and set the count to 1
                    else:
                        result[pair] = 1
            ### END CODE HERE ###

            return result
```

```
In [ ]: # Testing your function


        result = {}
        tweets = ['i am happy', 'i am tricked', 'i am sad', 'i am tired', 'i am tired']
        ys = [1, 0, 0, 0, 0]
        count_tweets(result, tweets, ys)
```

Expected Output: {('happi', 1): 1, ('trick', 0): 1, ('sad', 0): 1, ('tire', 0): 2}

3 Part 2: Train your model 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.

So how do you train a Naive Bayes classifier?

- 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. $P(D_{pos})$ is the probability that the document is positive. $P(D_{neg})$ is the probability that the document is negative. Use the formulas as follows and store the values in a dictionary:

$$P(D_{pos}) = \frac{D_{pos}}{D} \tag{1}$$

$$P(D_{neg}) = \frac{D_{neg}}{D} \tag{2}$$

Where D is the total number of documents, or tweets in this case, D_{pos} is the total number of positive tweets and D_{neg} 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 $\frac{P(D_{pos})}{P(D_{neg})}$. We can take the log of the prior to rescale it, and we'll call this the logprior

$$\text{logprior} = log \left(\frac{P(D_{pos})}{P(D_{neg})} \right) = log \left(\frac{D_{pos}}{D_{neg}} \right)$$

.

Note that $log(\frac{A}{B})$ is the same as $log(A) - log(B)$. So the logprior can also be calculated as the difference between two logs:

$$\text{logprior} = \log(P(D_{pos})) - \log(P(D_{neg})) = \log(D_{pos}) - \log(D_{neg}) \tag{3}$$

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:

- $freq_{pos}$ and $freq_{neg}$ 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.
- N_{pos} and N_{neg} 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.

We'll use these to compute the positive and negative probability for a specific word using this formula:

$$P(W_{pos}) = \frac{freq_{pos} + 1}{N_{pos} + V} \tag{4}$$

$$P(W_{neg}) = \frac{freq_{neg} + 1}{N_{neg} + V} \tag{5}$$

Notice that we add the "+1" in the numerator for additive smoothing. This wiki article explains more about additive smoothing.

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

$$\text{loglikelihood} = \log\left(\frac{P(W_{pos})}{P(W_{neg})}\right) \tag{6}$$

Create `freqs` dictionary

- Given your `count_tweets()` function, you can compute a dictionary called `freqs` that contains all the frequencies.
- In this `freqs` dictionary, the key is the tuple (word, label)
- The value is the number of times it has appeared.

We will use this dictionary in several parts of this assignment.

```
In [ ]: # Build the freqs dictionary for later uses

        freqs = count_tweets({}, train_x, train_y)
```

Instructions Given a freqs dictionary, `train_x` (a list of tweets) and a `train_y` (a list of labels for each tweet), implement a naive bayes classifier.

Calculate V

- You can then compute the number of unique words that appear in the `freqs` dictionary to get your V (you can use the `set` function).