Tweet classification with naive bayes

For this notebook we are going to implement a naive bayes classifier for classifying tweets about Trump or Obama based on the words in the tweet. Recall that for two events A and B the bayes theorem says

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where P(A) and P(B) is the ***class probabilities*** and P(B|A) is called ***conditional probabilities***. this gives us the probability of A happening, given that B has occurred. So as an example if we want to find the probability of "is this a tweet about Trump given that it contains the word "president" " we will obtain the following

$$P(\text{"Trump"}|\text{"president" in tweet}) = \frac{P(\text{""president" in tweet}|\text{"Trump"})P(\text{"Trump"})}{P(\text{""president" in tweet})}$$

This means that to find the probability of "is this a tweet about Trump given that it contains the word "president" " we need the probability of "president" being in a tweet about Trump, the probability of a tweet being about Trump and the probability of "president" being in a tweet.

Similarly if we want to obtain the opposite "is this a tweet about Obama given that it contains the word "president" " we get

$$P(\text{"Obama"}|\text{"president" in tweet}) = \frac{P(\text{"president" in tweet}|\text{"Obama"})P(\text{"Obama"})}{P(\text{"president" in tweet})}$$

where we need the probability of "president" being in a tweet about Obama, the probability of a tweet being about Obama and the probability of "president" being in a tweet.

We can now build a classifier where we compare those two probabilities and whichever is the larger one it's classified as

if P("Trump"|"president" in tweet) > P("Obama"|"president" in tweet)

Tweet is about Trump

else

Tweet is about Obama

Now let's expand this to handle multiple features and put the Naive assumption into bayes theroem. This means that if features are independent we have

$$P(A, B) = P(A)P(B)$$

This gives us:

$$P(A|b_1, b_2, ..., b_n) = \frac{P(b_1|A)P(b_2|A)...P(b_n|A)P(A)}{P(b_1)P(b_2)...P(b_n)}$$

or

$$P(A|b_1, b_2, ..., b_n) = \frac{\prod_i^n P(b_i|A)P(A)}{P(b_1)P(b_2)...P(b_n)}$$

So with our previous example expanded with more words "is this a tweet about Trump given that it contains the word "president" and "America" " gives us

$$P(\text{"Trump"}|\text{"president", "America" in tweet}) = \frac{P(\text{"president" in tweet}|\text{"Trump"})P(\text{"America" in tweet}|\text{"Trump"}}{P(\text{"president" in tweet})P(\text{"America" in tweet})}$$

As you can see the denominator remains constant which means we can remove it and the final classifier end up

$$y = argmax_A P(A) \prod_i^n P(b_i|A)$$

```
#stuff to import
import pandas as pd
import numpy as np
import random
import sklearn
from sklearn.model_selection import train_test_split
```

```
assert pd.__version__ == "1.2.1", "Looks like you don't have the same version of pandas as u
assert np.__version__ == "1.19.4", "Looks like you don't have the same version of numpy as u
assert sklearn.__version__ == "0.24.0", "Looks like you don't have the same version of sklea
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
/var/folders/qs/9s0640m51y5fhftgg8ysvryh0000gn/T/ipykernel_15186/3199633730.py in <module>
----> 1 assert pd.__version__ == "1.2.1", "Looks like you don't have the same version of par
      2 assert np.__version__ == "1.19.4", "Looks like you don't have the same version of nu
      3 assert sklearn.__version__ == "0.24.0", "Looks like you don't have the same version

AssertionError: Looks like you don't have the same version of pandas as us!
```

Load the data and explore

```
df_t = pd.read_csv('trump_20200530.csv')
trump_tweets = df_t['text']
df_t = pd.read_csv('Tweets-BarackObama.csv')
obama_tweets = df_t['Tweet-text']

tweet_data = trump_tweets.append(obama_tweets, ignore_index=True)
tweet_labels = np.array(['T' for _ in range(len(trump_tweets))] + ['O' for _ in range(len(ob
```

```
/var/folders/qs/9s0640m51y5fhftgg8ysvryh0000gn/T/ipykernel_15186/1785718735.py:6: FutureWarn
  tweet_data = trump_tweets.append(obama_tweets, ignore_index=True)
```

```
lab, counts = np.unique(tweet_labels, return_counts=True)
print('Number of tweets about ', lab[0], ': ', counts[0])
print('Number of tweets about ', lab[1], ': ', counts[1])
```

```
Number of tweets about  O :  6851
Number of tweets about  T :  18467
```

As you can see we have many more Trump than Obama Tweets so simlpy guessing that a tweet is a Trump tweet already gives you a classifier that is correct about 70% of the time, but we can do better than this.

Now lets split the data into a training set and a test set using scikit-learns train_test_split function https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
#Split data into train_tweets, test_tweets, train_labels and test_labels
train_tweets, test_tweets, train_labels, test_labels = train_test_split(tweet_data, tweet_la
```

What we need to build our classifier is "probability of tweet about Obama" P(O) , "probability of tweet about Trump" P(T), "probability of word in tweet given tweet about Obama" P(w|O) and "probability of word in tweet given tweet about Trump" P(w|T). Start by calculating the probability that a tweet is about Obama and trump respectively

```
train_counts = np.unique(train_labels, return_counts=True)[1]
P_O = train_counts[0]/(train_counts[0]+train_counts[1])
P_T = train_counts[1]/(train_counts[0]+train_counts[1])
```

For P(w|O), P(w|T) we need to count how many tweets each word occur in. Count the number of tweets each word occurs in and store in the word counter. An entry in the word counter is for instance {'president': 'O':87, 'T': 100} meaning president occurs in 87 words about Obaman and 100 tweets about Trump. Be aware that we are not interested in calculating multiple occurances of the same word in the same tweet. For each word convert it to lower case. You can use Python's lower. Another handy Python string method is split.

```
word_counter = {}

for (tweet, label) in zip(train_tweets, train_labels):
# ... Count number of tweets each word occurs in and store in word_counter where an entry lo
```

```
            words = list(set(tweet.lower().split()))
            for word in words:
                if word in word_counter:
                    word_counter[word][label] += 1
                else:
                    word_counter[word] = {'O': 0, 'T': 0}
```

Lets work with a smaller subset of words. Find the 100 most occuring words in tweet data.

```
nr_of_words_to_use = 100
popular_words = sorted(word_counter.items(), key=lambda x: x[1]['O'] + x[1]['T'], reverse=Tr
popular_words = [x[0] for x in popular_words[:nr_of_words_to_use]]
```

Now lets compute P(w|O), P(w|T) for the popular words

```
P_w_given_t = {}
P_w_given_o = {}

T_words, O_words = 0, 0
for word in word_counter:
    T_words += word_counter[word]['T']
    O_words += word_counter[word]['O']

for word in popular_words:
    P_w_given_t[word] = word_counter[word]['T']/T_words
    P_w_given_o[word] = word_counter[word]['O']/O_words
```

```
classifier = {
    'basis'  : popular_words,
    'P(T)'   : P_O,
    'P(O)'   : P_T,
    'P(w|O)' : P_w_given_o,
    'P(w|T)' : P_w_given_t
    }
```

Write a tweet_classifier function that takes your trained classifier and a tweet and returns wether it's about Trump or Obama unsing the popular words selected. Note that if there are words in the basis words in our classifier that are not in the tweet we have the opposite probabilities i.e P(w_1 occurs)* P(w_2 does not occur) * .... if w_1 occurs and w_2 does not occur. The function should return wether the tweet is about Obama or Trump i.e 'T' or 'O'

```
def tweet_classifier(tweet, classifier_dict):
    """ param tweet: string containing tweet message
        param classifier: dict containing 'basis' - training words
                                          'P(T)' - class probabilities
                                          'P(O)' - class probabilities
                                          'P(w|O)' - conditional probabilities
```

```
                                        'P(w|T)' - conditional probabilities

        return: either 'T' or 'O'
    """
    words_in_tweet = np.unique([x.lower() for x in tweet.split()])
    # ... Code for classifying tweets using the naive bayes classifier
    yT, yO = classifier_dict['P(T)'], classifier_dict['P(O)']
    for word in classifier_dict['basis']:
        if(word in words_in_tweet):
            yT *= classifier_dict['P(w|T)'][word]
            yO *= classifier_dict['P(w|O)'][word]
        else:
            yT *= 1-classifier_dict['P(w|T)'][word]
            yO *= 1-classifier_dict['P(w|O)'][word]
    if(yT > yO):
        return 'T'
    else:
        return 'O'

def test_classifier(classifier, test_tweets, test_labels):
    total = len(test_tweets)
    correct = 0
    for (tweet,label) in zip(test_tweets, test_labels):
        predicted = tweet_classifier(tweet,classifier)
        if predicted == label:
            correct = correct + 1
    return(correct/total)

acc = test_classifier(classifier, test_tweets, test_labels)
print(f"Accuracy: {acc:.4f}")
```

```
Accuracy: 0.8429
```