theoretical1_sentiment_analysis_version

February 6, 2024

1 Tweet classification with naive bayes

For this notebook we are going to implement a naive bayes classifier for classifying positive or negative 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 positive tweet given that it contains the word"good" " we will obtain the following

$$P(\text{"positive"}|\text{"good" in tweet}) = \frac{P(\texttt{"}\text{"good" in tweet}|\text{"positive"})P(\text{"positive"})}{P(\texttt{"}\text{"good" in tweet})}$$

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

Similarly, if we want to obtain the opposite "is this a negative tweet given that it contains the word"boring" " we get

$$P(\text{"negative"}|\text{"boring" in tweet}) = \frac{P(\text{"boring" in tweet}|\text{"negative"})P(\text{"negative"})}{P(\text{"boring" in tweet})}$$

where we need the probability of "boring" being in a negative tweet, the probability of a tweet negative being and the probability of "boring" 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("positive"|"good" in tweet) > P("negative"|"boring" in tweet)

Tweet is positive

else

Tweet is negative

Now let's expand this to handle multiple features and put the Naive assumption into bayes theorem. 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 positive tweet given that it contains the word"good" and "interesting" " gives us

$$P(\text{"positive"}|\text{"good", "interesting" in tweet}) = \frac{P(\text{"good" in tweet}|\text{"positive"})P(\text{"interesting" in tweet}|\text{"positive"})P}{P(\text{"good" in tweet})P(\text{"interesting" in tweet})}$$

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

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

[ ]:
```python
#stuff to import
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

Load the data

[ ]:
```python
tweets=pd.read_csv('data_for_theoretical_notebook_1.csv',encoding='latin')
tweets
```

[ ]:
```
        Unnamed: 0  sentiment  \
0                0          0
1                1          0
2                2          1
3                3          0
4                4          0
…              …          …
199374      199995          1
199375      199996          1
```

```
199376      199997          0
199377      199998          1
199378      199999          1

                                                     tweet  \
0       Hanging out with my friend waiting for a rain …
1                        yesdir… layin not feelin good
2                                       Hae a nice night
3                   Tay…where are you? miss you SO bad
4                                   @NeilMcDaid shizer!
…                                                    …
199374            Eating cobb salad w my Patricia at CF.
199375  @hallowed_ground ever been to fremont? You cou…
199376  @StacyLynn1985 as far as anything to scrape th…
199377  ksh scripting today to produce html code to di…
199378               @frelle  Ohhh I've love a Chai too!

                                          processed_tweets
0       anging friend waiting rain band looking huge s…
1                                 esdir layin feelin good
2                                              nice night
3                                            aywhere miss
4                                        neilmcdaid shizer
…                                                      …
199374                       ating cobb salad patricia compare
199375                   hallowedground ever fremont could help
199376                 stacylynn1985 anything scrape teeth clue
199377  scripting today produce html code display pgra…
199378                                   frelle ohhh love chai

[199379 rows x 4 columns]
```

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

[ ]:
```
tweets_data = tweets["processed_tweets"]
tweets_labels = tweets["sentiment"]

train_tweets, test_tweets, train_labels, test_labels =␣
 ↪train_test_split(tweets_data, tweets_labels)


#Split data into train_tweets, test_tweets, train_labels and test_labels
```

What we need to build our classifier is "probability of positive tweet" P(pos) , "probability of negative tweet" P(neg), "probability of word in tweet given tweet is positive" P(w|pos) and "probability of word in tweet given tweet is negative" P(w|neg). Start by calculating the probability that a tweet is positive and negative respectively

[ ]:
```python
neg_tweets, pos_tweets = tweets_labels.value_counts()
tot_tweets = neg_tweets + pos_tweets
P_pos = pos_tweets/tot_tweets
P_neg = 1 - P_pos
```

For P(w|pos), P(w|neg) 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 {'good': 'Pos':150, 'Neg': 10} meaning good occurs in 150 positive tweets and 10 negative tweets. Be aware that we are not interested in calculating multiple occurrences of the same word in the same tweet. Also, we change the labels from 0 for "Negative" and 1 for "Positive" to "Neg" and "Pos" respectively.For each word convert it to lower case. You can use Python's lower. Another handy Python string method is split.

[ ]:
```python
new_train_labels = train_labels.replace(0, "Neg", regex=True)
final_train_labels = new_train_labels.replace(1, "Pos", regex=True)
word_counter = {}
for (tweet, label) in zip(train_tweets, final_train_labels):
    words = tweet.split()
    words = set(words)
    for word in words:
        word = word.lower()

        if word not in word_counter:
            word_counter[word] = {"Neg": 0, "Pos": 0}

        word_counter[word][label] += 1
```

Let's work with a smaller subset of words just to save up some time. Find the 1500 most occuring words in tweet data.

[ ]:
```python
nr_of_words_to_use = 1500
popular_words = sorted(word_counter.items(), key=lambda x: x[1]['Pos'] +␣
 ↪x[1]['Neg'], reverse=True)
popular_words = [x[0] for x in popular_words[:nr_of_words_to_use]]
```

Now let's compute P(w|pos), P(w|neg) for the popular words

[ ]:
```python
P_w_given_pos = {}
P_w_given_neg = {}
for word in popular_words:
    n_pos = word_counter[word]["Pos"]
    n_neg = word_counter[word]["Neg"]

    P_w_given_pos[word] = n_pos / neg_tweets
    P_w_given_neg[word] = n_neg / pos_tweets
```

[ ]:
```python
classifier = {
    'basis'  : popular_words,
    'P(pos)'   : P_pos,
    'P(neg)'   : P_neg,
    'P(w|pos)' : P_w_given_pos,
    'P(w|neg)' : P_w_given_neg
    }
```

Train and predict Write a tweet_classifier function that takes your trained classifier and a tweet and returns wether it's about Positive or Negative using 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 Positive or Negative. i.e 'Pos' or 'Neg'.

[ ]:
```python
def tweet_classifier(tweet, classifier_dict):
    """ param tweet: string containing tweet message
        param classifier: dict containing 'basis' - training words
                                          'P(pos)' - class probabilities
                                          'P(neg)' - class probabilities
                                          'P(w|pos)' - conditional probabilities
                                          'P(w|neg)' - conditional probabilities

        return: either 'Pos' or 'Neg'
    """
    prob_pos = classifier_dict['P(pos)']
    prob_neg = classifier_dict['P(neg)']
    words = list(map(str.lower, tweet.split()))

    for basis_word in classifier_dict["basis"]:
        if basis_word in words:
            prob_pos *= classifier_dict["P(w|pos)"][basis_word]
            prob_neg *= classifier_dict["P(w|neg)"][basis_word]
        else:
            prob_pos *= 1 - classifier_dict["P(w|pos)"][basis_word]
            prob_neg *= 1 - classifier_dict["P(w|neg)"][basis_word]

    if prob_pos >= prob_neg:
        return "Pos"
    else:
        return "Neg"
```

[ ]:
```python
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)
```

[ ]:
```
new_test_labels = test_labels.replace(0, "Neg", regex=True)
final_test_labels = new_test_labels.replace(1, "Pos", regex=True)
```

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

```
Accuracy: 0.7209
```

Optional work In basic sentiment analysis classifications we have 3 classes “Positive”, “Negative” and “Neutral”. Although because it is challenging to create the “Neutral” class. Try to improve the accuracy by filtering the dataset from the perspective of removing words that indicate neutrality.

[ ]:
```

```