Homework_H3.2-Bayes_Learning_for_Text_Classification

October 13, 2020

1 Naive Bayes Text Classification

We made a simple Algorithm to try and classify senteces into either Sports or Not Sports sentences. We start with a couple sentences either classed "Sports" or "Not Sports" and try to classify new sentences based on that. At the end we make a comparison, which class ("Sports" or "Not Sports") the new sentence is more likely to end up in.

1.1 What happens here:

1. import everything we need
2. Provide training data and do transformations.
3. Create dictionaries and count the words in each class.
4. Calculate probabilities of the words.

To evaluate a new sentence...

5. Vectorize and transform all sentences
6. Count all words
7. Transform new sentence
8. Perform Laplace Smoothing, so we dont multiply with 0
9. Calculate probability of the new sentence for each class
10. Output whats more likely

[1]:
```
# This notebook was created by Alireza Gholami and Jannik Schwarz


# Importing everything we need
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import word_tokenize

# to check the time of execution, import function time
import time
```

[2]:
```
# Naming the columns
columns = ['sentence', 'class']

# Our training data
rows = [['A great game', 'Sports'],
```

```
        ['The election was over', 'Not Sports'],
        ['Very clean match', 'Sports'],
        ['A clean but forgettable game', 'Sports'],
        ['It was a close election', 'Not Sports'],
        ['A very close game', 'Sports']]

# the data inside a dataframe
training_data = pd.DataFrame(rows, columns=columns)
print(f'The training data:\n{training_data}\n')
```

```
The training data:
                       sentence       class
0                  A great game      Sports
1          The election was over  Not Sports
2               Very clean match      Sports
3  A clean but forgettable game      Sports
4       It was a close election  Not Sports
5             A very close game      Sports
```

[3]:
```
# Turns the data into vectors
def vectorisation(my_class):

    # my_docs contains the sentences for a class (sports or not sports)
    my_docs = [row['sentence'] for index, row in training_data.iterrows() if␣
↪row['class'] == my_class]

    # creates a vector that counts the occurence of words in a sentence
    my_vector = CountVectorizer(token_pattern=r"(?u)\b\w+\b") # Token-Pattern␣
↪damit einstellige Wörter wie 'a' gelesen werden

    # transform the sentences
    my_x = my_vector.fit_transform(my_docs)

    # tdm = term_document_matrix_sport | create the matrix with the vectors for␣
↪a class
    tdm = pd.DataFrame(my_x.toarray(), columns=my_vector.get_feature_names())
    return tdm, my_vector, my_x
```

[4]:
```
# Here we are actually creating the matrix for sport and not sport sentences
tdm_sport, vector_sport, X_sport = vectorisation('Sports')
tdm_not_sport, vector_not_sport, X_not_sport = vectorisation('Not Sports')

print(f'Sport sentence matrix: \n{tdm_sport}\n')
print(f'Not sport sentence matrix: \n{tdm_not_sport}\n')
print(f'Amount of sport sentences: {len(tdm_sport)}')
print(f'Amount of not sport senteces: {len(tdm_not_sport)}')
```

```
print(f'Total amount of sentences: {len(rows)}')
```

```
Sport sentence matrix:
   a  but  clean  close  forgettable  game  great  match  very
0  1    0      0      0            0     1      1      0     0
1  0    0      1      0            0     0      0      1     1
2  1    1      1      0            1     1      0      0     0
3  1    0      0      1            0     1      0      0     1

Not sport sentence matrix:
   a  close  election  it  over  the  was
0  0      0         1   0     1    1    1
1  1      1         1   1     0    0    1

Amount of sport sentences: 4
Amount of not sport senteces: 2
Total amount of sentences: 6
```

[5]:
```
# creates a dictionary for each class
def make_list(my_vector, my_x):
    my_word_list = my_vector.get_feature_names()
    my_count_list = my_x.toarray().sum(axis=0)
    my_freq = dict(zip(my_word_list, my_count_list))
    return my_word_list, my_count_list, my_freq
```

[6]:
```
# create lists

# word_list_sport = word list ['a', 'but', 'clean', 'forgettable', 'game',␣
 ↪'great', 'match', 'very']
# count_list_sport = occurence of words [2 1 2 1 2 1 1 1]
# freq_sport = combining the two to create a dictionary
word_list_sport, count_list_sport, freq_sport = make_list(vector_sport, X_sport)
word_list_not_sport, count_list_not_sport, freq_not_sport =␣
 ↪make_list(vector_not_sport, X_not_sport)

print(f'sport dictionary: \n{freq_sport}\n')
print(f'not sport dictionary: \n{freq_not_sport}\n')
```

```
sport dictionary:
{'a': 3, 'but': 1, 'clean': 2, 'close': 1, 'forgettable': 1, 'game': 3, 'great':
1, 'match': 1, 'very': 2}

not sport dictionary:
{'a': 1, 'close': 1, 'election': 2, 'it': 1, 'over': 1, 'the': 1, 'was': 2}
```

[7]:
```
# calculate the probabilty of a word in a sentence of a class
def calculate_prob(my_word_list, my_count_list):
    my_prob = []
    for my_word, my_count in zip(my_word_list, my_count_list):
        my_prob.append(my_count / len(my_word_list))
    prob_dict = dict(zip(my_word_list, my_prob))
    return prob_dict
```

[8]:
```
# probabilities of the words in a class
prob_sport_dict = calculate_prob(word_list_sport, count_list_sport)
prob_not_sport_dict = calculate_prob(word_list_not_sport, count_list_not_sport)
print(f'probabilites of words in sport sentences: \n{prob_sport_dict}\n')
print(f'probabilites of words in not sport sentences: \n{prob_not_sport_dict}')
```

```
probabilites of words in sport sentences:
{'a': 0.3333333333333333, 'but': 0.1111111111111111, 'clean':
0.2222222222222222, 'close': 0.1111111111111111, 'forgettable':
0.1111111111111111, 'game': 0.3333333333333333, 'great': 0.1111111111111111,
'match': 0.1111111111111111, 'very': 0.2222222222222222}

probabilites of words in not sport sentences:
{'a': 0.14285714285714285, 'close': 0.14285714285714285, 'election':
0.2857142857142857, 'it': 0.14285714285714285, 'over': 0.14285714285714285,
'the': 0.14285714285714285, 'was': 0.2857142857142857}
```

[9]:
```
# all sentences again
docs = [row['sentence'] for index, row in training_data.iterrows()]

# vectorizer
vector = CountVectorizer(token_pattern=r"(?u)\b\w+\b")

# transform the sentences
X = vector.fit_transform(docs)

# counting the words
total_features = len(vector.get_feature_names())
total_counts_features_sport = count_list_sport.sum(axis=0)
total_counts_features_not_sport = count_list_not_sport.sum(axis=0)

print(f'Amount of distinct words: {total_features}')
print(f'Amount of distinct words in sport sentences:␣
 ↪{total_counts_features_sport}')
print(f'Amount of distinct words in not sport sentences:␣
 ↪{total_counts_features_not_sport}')
```

```
Amount of distinct words: 14
Amount of distinct words in sport sentences: 15
```

```
Amount of distinct words in not sport sentences: 9
```

[10]:
```
# a new sentence
new_sentence = 'Hermann plays a TT match'

# gets tokenized
new_word_list = word_tokenize(new_sentence)
```

[11]:
```
# We're using laplace smoothing
# if a new word occurs the probability would be 0
# So every word counter gets incremented by one
def laplace(freq, total_count, total_feat):
    prob_sport_or_not = []
    for my_word in new_word_list:
        if my_word in freq.keys():
            counter = freq[my_word]
        else:
            counter = 0
        # total_count is the amount of words in sport sentences and total_feat␣
 ↪the total amount of words
        prob_sport_or_not.append((counter + 1) / (total_count + total_feat))
    return prob_sport_or_not
```

[12]:
```
# probability for the new words
prob_new_sport = laplace(freq_sport, total_counts_features_sport,␣
 ↪total_features)
prob_new_not_sport = laplace(freq_not_sport, total_counts_features_not_sport,␣
 ↪total_features)

print(f'probability that the word is in a sport sentece: {prob_new_sport}')
print(f'probability that the word is in a not sport sentece:␣
 ↪{prob_new_not_sport}')
```

```
probability that the word is in a sport sentece: [0.034482758620689655,
0.034482758620689655, 0.13793103448275862, 0.034482758620689655,
0.06896551724137931]
probability that the word is in a not sport sentece: [0.043478260869565216,
0.043478260869565216, 0.08695652173913043, 0.043478260869565216,
0.043478260869565216]
```

[13]:
```
# multiplying the probabilities of each word
new_sport = list(prob_new_sport)
sport_multiply_result = 1
for i in range(0, len(new_sport)):
    sport_multiply_result *= new_sport[i]
```

```
# multiplying the result with the ratio of sports senteces to the total amount␣
 ↪of sentences (here its 4/6)
sport_multiply_result *= ( len(tdm_sport) / len(rows) )

# multiplying the probabilities of each word
new_not_sport = list(prob_new_not_sport)
not_sport_multiply_result = 1
for i in range(0, len(new_not_sport)):
    not_sport_multiply_result *= new_not_sport[i]

# multiplying the result with the ratio of sports senteces to the total amount␣
 ↪of sentences (here its 2/6)
not_sport_multiply_result *= ( len(tdm_not_sport) / len(rows) )
```

[14]:
```
# comparing whats more likely

print(f'The probability of the sentence "{new_sentence}":\nSport vs not␣
 ↪sport\n{sport_multiply_result} vs {not_sport_multiply_result}\n\n')

if not_sport_multiply_result < sport_multiply_result:
    print('Verdict: It\'s probably a sports sentence!')
else:
    print('Verdict: It\'s probably not a sport sentence!')
```

```
The probability of the sentence "Hermann plays a TT match":
Sport vs not sport
2.6002118815154297e-07 vs 1.0357848652047699e-07


Verdict: It's probably a sports sentence!
```

[15]:
```
# print current date and time
print("date",time.strftime("%d.%m.%Y %H:%M:%S"))
print ("*** End of Homework H3.2-Bayes Learning for Text Classification ***")
```

```
date 13.10.2020 14:52:26
*** End of Homework H3.2-Bayes Learning for Text Classification ***
```