index

March 18, 2022

1 Document Classification with Naive Bayes - Lab

1.1 Introduction

In this lesson, you'll practice implementing the Naive Bayes algorithm on your own.

1.2 Objectives

In this lab you will:

- Implement document classification using Naive Bayes

1.3 Import the dataset

To start, import the dataset stored in the text file `'SMSSpamCollection'`.

[1]:
```
# Your code here
import pandas as pd

df = pd.read_csv("SMSSpamCollection", sep = "\t", names = ["label", "text"])
df.head()
```

[1]:
```
  label                                               text
0   ham  Go until jurong point, crazy.. Available only …
1   ham                      Ok lar… Joking wif u oni…
2  spam  Free entry in 2 a wkly comp to win FA Cup fina…
3   ham  U dun say so early hor… U c already then say…
4   ham  Nah I don't think he goes to usf, he lives aro…
```

1.4 Account for class imbalance

To help your algorithm perform more accurately, subset the dataset so that the two classes are of equal size. To do this, keep all of the instances of the minority class (spam) and subset examples of the majority class (ham) to an equal number of examples.

[2]:
```
# Your code here
minority_spam = df[df["label"] == "spam"]
majority_ham = df[df["label"] == "ham"].sample(n = len(minority_spam))
df_equal = pd.concat([minority_spam, majority_ham], axis = 0)
classes_m = dict(df_equal["label"].value_counts(normalize = True))
```

```
classes_m
```

[2]: {'spam': 0.5, 'ham': 0.5}

1.5 Train-test split

Now implement a train-test split on the dataset:

[3]:
```
# Your code here
from sklearn.model_selection import train_test_split
X = df_equal["text"]
y = df_equal["label"]
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state = 17)
train_df = pd.concat([X_train, y_train], axis = 1)
test_df = pd.concat([X_test, y_test], axis = 1)
```

1.6 Create the word frequency dictionary for each class

Create a word frequency dictionary for each class:

[4]:
```
# Your code here
def wrod_frequency(data, corpus = False):

    if corpus == False:

        target = classes_m.keys()

        word_frequency = {}

        for label in target:

            line = data[data["label"] == label]

            word_text_frequency = {}

            for index in line["text"].index:

                text = line["text"][index]

                for word in text.split():

                    word_text_frequency[word]= word_text_frequency.get(word, 0)␣
 ↪+ 1

            word_frequency[label] = word_text_frequency

        return word_frequency
```

```
    else:

        vocabulary = set()
#         vocabulary = 0

        for line in data["text"]:

#             bag = {}

            for word in line.split():

                vocabulary.add(word)

#                 bag[word] = bag.get(word, 0) + 1

#             vocabulary += sum(bag.values())


#         return vocabulary
        return len(vocabulary)

train_frequency = wrod_frequency(train_df, corpus = False)
test_frequency = wrod_frequency(test_df,corpus = False)
vocabulary_train = wrod_frequency(train_df,corpus = True)
vocabulary_train
```

[4]: 5883

1.7 Count the total corpus words

Calculate V, the total number of words in the corpus:

[5]:
```
# Your code here
def wrod_corpus(data):

    vocabulary = set()

    for line in data["text"]:

        for word in line.split():

            vocabulary.add(word)

    return len(vocabulary)

train_frequency = wrod_frequency(train_df, corpus = False)
```

```
test_frequency = wrod_frequency(test_df,corpus = False)
V_m = wrod_corpus(train_df)
V_m
```

[5]: 5883

1.8 Create a bag of words function

Before implementing the entire Naive Bayes algorithm, create a helper function `bag_it()` to create a bag of words representation from a document's text.

[7]:
```
def bag_it_m(line):

    bag = {}

    for word in line.split():

        bag[word]= bag.get(word, 0) + 1

    return bag

# bag_it_m(X_train[312])
```

1.9 Implementing Naive Bayes

Now, implement a master function to build a naive Bayes classifier. Be sure to use the logarithmic probabilities to avoid underflow.

[26]:
```
# Your code here
# def classify_doc(doc, class_word_freq, classes, V, return_posteriors=False):

def classify_doc_m(doc, class_freq, classes_m, V_m, return_posteriors=False):

    bag  = bag_it_m(doc)

    posteriors = []

    clss = []


    for item in class_freq.keys():


        p = np.log(classes_m[item])


        for word in bag.keys():
```

```
            num = bag[word] + 1

            denum = class_freq[item].get(word, 0) + V_m

            p += np.log(num / denum)

#           print(p)

        clss.append(item)

        posteriors.append(p)

    if return_posteriors:

        print(posteriors)

    return clss[np.argmax(posteriors)]
```

1.10 Test your classifier

Finally, test your classifier and measure its accuracy. Don't be perturbed if your results are sub-par; industry use cases would require substantial additional preprocessing before implementing the algorithm in practice.

[27]:
```
# Your code here
classify_doc_m(X_train[312], train_frequency, classes_m, V_m,␣
 ↪return_posteriors=True)
```

```
[-200.4960104602396, -200.41192443826716]
```

[27]:
```
'ham'
```

[28]:
```
y_train_hat = X_train.apply(lambda x: classify_doc_m(x, train_frequency,␣
 ↪classes_m, V_m, return_posteriors=False))
```

[30]:
```
diff = y_train == y_train_hat
diff.value_counts(normalize = True)
```

[30]:
```
False    0.757143
True     0.242857
dtype: float64
```

2 READ ME:

check the solution on github provided below. The solution here is not that good.

[31]:
```
# Import the data
import pandas as pd
df = pd.read_csv('SMSSpamCollection', sep='\t', names=['label', 'text'])
df.head()

# Account for class imbalance
minority = df[df['label'] == 'spam']
undersampled_majority = df[df['label'] == 'ham'].sample(n=len(minority))
df2 = pd.concat([minority, undersampled_majority])
df2.label.value_counts()


# p-classes
p_classes = dict(df2['label'].value_counts(normalize=True))
p_classes

# Train-test split
# from sklearn.model_selection import train_test_split
# X = df2['text']
# y = df2['label']
# X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=17)
train_df = pd.concat([X_train, y_train], axis=1)
test_df = pd.concat([X_test, y_test], axis=1)


# Create the word frequency dictionary for each class
# Will be a nested dictionary of class_i : {word1:freq, word2:freq..., wordn:
↪freq},.... class_m : {}
class_word_freq = {}
classes = train_df['label'].unique()
for class_ in classes:
    temp_df = train_df[train_df['label'] == class_]
    bag = {}
    for row in temp_df.index:
        doc = temp_df['text'][row]
        for word in doc.split():
            bag[word] = bag.get(word, 0) + 1
    class_word_freq[class_] = bag


# Count the total corpus words
vocabulary = set()
for text in train_df['text']:
    for word in text.split():
        vocabulary.add(word)
V = len(vocabulary)
V
```

```python
# Create a bag of words function
def bag_it(doc):
    bag = {}
    for word in doc.split():
        bag[word] = bag.get(word, 0) + 1
    return bag

# Implementing Naive Bayes
def classify_doc(doc, class_word_freq, p_classes, V, return_posteriors=False):
    bag = bag_it(doc)
    classes = []
    posteriors = []
    for class_ in class_word_freq.keys():
        p = np.log(p_classes[class_])
        for word in bag.keys():
            num = bag[word]+1
            denom = class_word_freq[class_].get(word, 0) + V
            p += np.log(num/denom)
        classes.append(class_)
        posteriors.append(p)
    if return_posteriors:
        print(posteriors)
    return classes[np.argmax(posteriors)]

#Test your classifier
import numpy as np

y_hat_train = X_train.map(lambda x: classify_doc(x, class_word_freq, p_classes,␣
 ↪V))
residuals = y_train == y_hat_train
residuals.value_counts(normalize=True)
```

[31]:
```
False    0.757143
True     0.242857
dtype: float64
```

2.1 Level up (Optional)

Rework your code into an appropriate class structure so that you could easily implement the algorithm on any given dataset.

2.2 Summary

Well done! In this lab, you practiced implementing Naive Bayes for document classification!