

Programming Assignment: Email Spam Naive Bayes

Overview/Task

The goal of this programming assignment is to build a naive bayes classifier from scratch that can determine whether email text should be labeled spam or not spam based on its contents

Review

Remember that a naive bayes classifier realizes the following probability:

$$P(Y|X_1, X_2, \dots, X_n) \propto P(Y) * P(Y|X_1) * P(Y|X_2) * \dots * P(Y|X_n)$$

Where Y is a binary class $\{0,1\}$

Where X_i is a feature of the input

The classifier will decide what class each input belongs to based on highest probability from the equation above

Reminders

Please remember that the classifier must be written from scratch; do NOT use any libraries that implement the classifier for you, such as but not limited to sklearn.

You CAN, however, use SKlearn to split up the dataset between testing and training.

Feel free to look up any tasks you are not familiar with, e.g. the function call to read a csv

Task list/Recommended Order

In order to provide some guidance, I am giving the following order/checklist to solve this task:

1. Compute the "prior": $P(Y)$ for $Y = 0$ and $Y = 1$

2. Compute the "likelihood": $P(Y|X_n)$
3. Write code that uses the two items above to make a decision on whether or not an email is spam or ham (aka not spam)
4. Write code to evaluate your model. Test model on training data to debug
5. Test model on testing data to debug

```
In [3]: #import cell
import numpy as np
import pandas as pd
import random
import csv
```

Function template

```
In [55]: def prior(df):
    ham_prior = 0
    spam_prior = 0
    '''YOUR CODE HERE'''
    num_ham = 0
    num_spam = 0
    for x in range(len(df.label)):
        if df.label[x] == "ham":
            num_ham += 1
        else:
            num_spam += 1

    ham_prior = num_ham/(num_ham+num_spam)
    spam_prior = num_spam/(num_ham+num_spam)
    '''END'''
    return ham_prior, spam_prior

def likelihood(df):
    ham_like_dict = {}
    spam_like_dict = {}

    '''YOUR CODE HERE'''
    lst_words_ham = []
    lst_words_spam = []

    for x in range(len(df.text)):
        text = df.text[x].split()
        for y in text:
            if y not in ' '!()-[]{};:'\,<>./?@#$$%^&*~'':
                if df.label[x] == "spam":
                    if y in spam_like_dict:
                        if y not in lst_words_spam:
                            spam_like_dict[y] += 1
                    else:
                        spam_like_dict[y] = 1
```

```

        lst_words_spam.append(y)
    else:
        if y in ham_like_dict:
            if y not in lst_words_ham:
                ham_like_dict[y] += 1
            else:
                ham_like_dict[y] = 1
        lst_words_ham.append(y)
    lst_words_spam = []
    lst_words_ham = []

'''END'''
return ham_like_dict, spam_like_dict

def predict(ham_prior, spam_prior, ham_like_dict, spam_like_dict, text):
    '''
    prediction function that uses prior and likelihood structure to compute
    '''
    #ham_spam_decision = 1 if classified as spam, 0 if classified as normal,
    ham_spam_decision = None

    '''YOUR CODE HERE'''

    theText = text.lower().split()

    prob_ham = 1
    prob_spam = 1

    numerator = 1
    denominator = 1
    word_check = []
    for x in theText:
        if x not in ' '!()-[]{};:'"\,<>./?@#$$%^&*~_':
            if x not in word_check: #check if the word already appeared in
                word_check.append(x)
            if x in ham_like_dict:
                numerator = ham_like_dict[x]* ham_prior
                denominator = numerator
            if x in spam_like_dict:
                denominator += spam_like_dict[x]*spam_prior
            prob_ham *= numerator/denominator #omit the word if not

    numerator = 1
    denominator = 1
    word_check = []
    for x in theText:
        if x not in ' '!()-[]{};:'"\,<>./?@#$$%^&*~_':
            if x not in word_check: #check if the word already appeared in
                word_check.append(x)
            if x in spam_like_dict:
                numerator = spam_like_dict[x]* spam_prior
                denominator = numerator
            if x in ham_like_dict:

```

```

        denominator += ham_like_dict[x]*ham_prior
        prob_spam *= numerator/denominator #omit the word if not

    if(prob_spam + prob_ham != 0):
        #ham_posterior = posterior probability that the email is normal/ham
        ham_posterior = prob_ham/ (prob_ham + prob_spam)

        #spam_posterior = posterior probability that the email is spam
        spam_posterior = prob_spam/(prob_ham + prob_spam)
    else:
        ham_posterior = prob_ham
        spam_posterior = prob_spam

    if(ham_posterior > spam_posterior):
        ham_spam_decision = 0
    else:
        ham_spam_decision = 1

    '''END'''
    return ham_spam_decision

def metrics(ham_prior, spam_prior, ham_dict, spam_dict, df):
    '''
    Calls "predict" function and report accuracy, precision, and recall of y
    '''

    '''YOUR CODE HERE'''

    ls = []
    true_positive = 0
    true_negative = 0
    false_positive = 0
    false_negative = 0

    for x in range(len(df.text)):
        text = df.text[x]
        predictedValue = predict(ham_prior, spam_prior, ham_dict, spam_dict,
                                ls.append(predictedValue))
    check_spam = 0
    for x in range(len(ls)):
        if df.label[x] == "spam":
            check_spam = 1
        else:
            check_spam = 0
        if check_spam == 1 and ls[x] == 1:
            true_positive += 1
        elif check_spam == 0 and ls[x] == 0:
            true_negative += 1
        elif check_spam == 1 and ls[x] == 0:
            false_positive += 1
        else:
            false_negative += 1

```

```

precision = true_positive / (true_positive + false_positive)
acc = (true_positive + true_negative)/(true_positive + true_negative + f

recall = true_positive/(true_positive + false_negative)

'''END'''
return acc, precision, recall

```

Generate answers with your functions

```

In [56]: #loading in the training data
train_df = pd.read_csv("./TRAIN_balanced_ham_spam.csv")
test_df = pd.read_csv("./TEST_balanced_ham_spam.csv")
df = train_df
df.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2398 entries, 0 to 2397
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Unnamed: 0.1    2398 non-null  int64
1   Unnamed: 0      2398 non-null  int64
2   label           2398 non-null  object
3   text            2398 non-null  object
4   label_num       2398 non-null  int64
dtypes: int64(3), object(2)
memory usage: 93.8+ KB

```

```

In [57]: #compute the prior

ham_prior, spam_prior = prior(df)

print(ham_prior, spam_prior)

0.5 0.5

```

```

In [58]: # compute likelihood

ham_like_dict, spam_like_dict = likelihood(df)

```

```

In [59]: # Test your predict function with some example TEXT

some_text_example = "I hope everyone is managing to stay warm to the best of
print(predict(ham_prior, spam_prior, ham_like_dict, spam_like_dict, some_text

```

1

In [60]: *# Predict on test_df and compute metrics*

```
df = test_df
acc, precision, recall = metrics(ham_prior, spam_prior, ham_like_dict, spam_
print(acc, precision, recall)
```

```
0.9616666666666667 0.9866666666666667 0.9396825396825397
```

In []: