```
###
### Code imports.
###

import numpy as np
```

Candace Edwards

ICS 635: Homework 3

Part 1: Notebook Link

Part 2: Q5: Notebook Link

```
class NaiveBayesClassifier:
  def __init__(self, smoothing_parameter):
    '''
    Input a smoothing parameter beta.
    '''
    self.smoothing_parameter = smoothing_parameter

  def fit(self, X_train, y_train):
    '''
    Each row in X_train respresents a sentence.
    X_train[i][0] is an indicator of the word "free".
    X_train[i][1] is an indicator of the word "dear".
    X_train[i][2] is an indicator of the word "sincerely".

    Each row in y_train represents the corresponding classification of each row
    in X_train. 0 = not spam, 1 = spam.
    '''

    ### Calculate prior probabilities (no smoothing).
    self.prob_spam = sum(y_train ==1) / len(y_train)
    self.prob_nospam = abs(1 - self.prob_spam)
    print(self.prob_spam) # expected 0.5
    print(self.prob_nospam) # expected 0.5

    ### Calculate conditional probabilities (use smoothing).
    classes = len(np.unique(y_train)) #for multinomial classification
    #print(classes) #expected 2

    '''
    SPAM
    '''
    self.prob_free_given_spam = (sum(X_train[:,0][y_train == 1])+ (self.smoothing_parameter))/ (sum(y_train == 1) + (classes * self.smoothing
    self.prob_dear_given_spam = (sum(X_train[:,1][y_train == 1])+ (self.smoothing_parameter))/ (sum(y_train == 1) + (classes * self.smoothing
    self.prob_sincerely_given_spam = (sum(X_train[:,2][y_train == 1])+ (self.smoothing_parameter))/ (sum(y_train == 1) + (classes * self.smoo


    #print(self.prob_free_given_spam) #Expected: 0.9997
    #print(self.prob_dear_given_spam) #Expected: 0.5
    #print(self.prob_sincerely_given_spam) #Expected: 0.749


    self.prob_nofree_given_spam = abs(1 - self.prob_free_given_spam)
    self.prob_nodear_given_spam = abs(1 - self.prob_dear_given_spam)
    self.prob_nosincerely_given_spam = abs(1 - self.prob_sincerely_given_spam)

    '''
    NOT SPAM
    '''

    self.prob_free_given_nospam =(sum(X_train[:,0][y_train == 0])+ (self.smoothing_parameter))/ (sum(y_train == 0) + (classes * self.smoothin
    self.prob_dear_given_nospam = (sum(X_train[:,1][y_train == 0])+ (self.smoothing_parameter))/ (sum(y_train == 0) + (classes * self.smoothi
    self.prob_sincerely_given_nospam = (sum(X_train[:,2][y_train == 0])+ (self.smoothing_parameter))/ (sum(y_train == 0) + (classes * self.sm

    self.prob_nofree_given_nospam = abs(1-self.prob_free_given_nospam)
    self.prob_nodear_given_nospam = abs(1 - self.prob_dear_given_nospam )
    self.prob_nosincerely_given_nospam = abs(1-self.prob_sincerely_given_nospam)

  def predict(self, x_test):
    '''
```

```
    Make a prediction of either spam (1) or not spam (0) for a single test data point x_test.

    Return a tuple of form
    (prediction of 0 or 1,
    numerator [of Bayes Rule] of probability of not spam from 0 to 1,
    numerator [of Bayes Rule] of probability of spam from 0 to 1)
    '''
    prob_spam = self.prob_spam * (self.prob_free_given_spam if x_test[0] == 1 else self.prob_nofree_given_spam ) \
                * (self.prob_dear_given_spam if x_test[1] else self.prob_nodear_given_spam ) \
                * (self.prob_sincerely_given_spam if x_test[2] else self.prob_nosincerely_given_spam )

    prob_nospam = self.prob_nospam * (self.prob_free_given_nospam if x_test[0] == 1 else self.prob_nofree_given_nospam ) \
                * (self.prob_dear_given_nospam if x_test[1] else self.prob_nodear_given_nospam ) \
                * (self.prob_sincerely_given_nospam if x_test[2] else self.prob_nosincerely_given_nospam )

    if prob_spam > prob_nospam:
      return (1, prob_nospam, prob_spam)
    else:
      return (0, prob_nospam, prob_spam)



def unit_test():
  '''
  Test your solution.
  '''
  X_train = np.array([[1,1,1],
                      [1,1,0],
                      [0,0,0],
                      [0,1,0],
                      [1,0,1],
                      [1,0,1],
                      [0,1,0],
                      [0,1,1]])
  y_train = np.array([ 1, 1, 0, 0, 1, 1, 0, 0 ])
  clf = NaiveBayesClassifier(0.001)
  clf.fit(X_train, y_train)

  X_test = np.array([[1,0,0],
                     [1,0,1],
                     [0,1,1],
                     [0,0,1]])

  print('Expected output: ')
  print_str = '''
(1, 2.3433589849604016e-05, 0.06251560938671095)
(1, 7.816402345702639e-06, 0.1874219218476719)
(0, 0.09375779298826566, 4.6843769519538096e-05)
(0, 0.03127342578515625, 4.6843769519538096e-05)
  '''
  print(print_str)

  print('Actual output: \n')
  for x_test in X_test:
    print(clf.predict(x_test))

unit_test()
```

```
    0.5
    0.5
    Expected output:

    (1, 2.3433589849604016e-05, 0.06251560938671095)
    (1, 7.816402345702639e-06, 0.1874219218476719)
    (0, 0.09375779298826566, 4.6843769519538096e-05)
    (0, 0.03127342578515625, 4.6843769519538096e-05)

    Actual output:

    (1, 2.3433589849604003e-05, 0.06251560938671094)
    (1, 7.816402345702636e-06, 0.1874219218476719)
    (0, 0.09375779298826564, 4.6843769519516764e-05)
    (0, 0.03127342578515624, 4.6843769519516764e-05)
```