q1

October 14, 2020

0.0.1 Important Note for question1 !

- Please **do not** change the default variable names in this problem, as we will use them in different parts.
- The default variables are initially set to "None".
- You only need to modify code in the "TODO" part. We added a lot of "assertions" to check your code. **Do not** modify them.

[6]:
```
# load packages
import numpy as np
import pandas as pd
import time
from sklearn.naive_bayes import GaussianNB
```

0.1 P1. Load data and plot

0.1.1 TODO

- Load train and test data, and split them into inputs(trainX, testX) and labels(trainY, testY)

[8]:
```
# Use pandas to load q1_train.csv and q1_test.csv
# Each data point has 200 features(X), followed by 1 label(Y)
train = pd.read_csv("q1_train.csv")
test = pd.read_csv("q1_test.csv")

#### TODO ####
trainX = train.values[:, :200]
trainY = train.values[:, -1]
testX = test.values[:, :200]
testY = test.values[:, -1]
###############

assert(len(trainX.shape) == 2)
assert(len(trainY.shape) == 1)
assert(trainX.shape[1] == 200)
```

0.2 P2. Write your Gaussian NB solver

0.2.1 TODO

- Finish the myNBSolver() function.
  - Compute P(y == 0) and P(y == 1), saved in "py0" and "py1"
  - Compute mean/variance of trainX for both y = 0 and y = 1, saved in "mean0", "var0", "mean1" and "var1"
    - Each of them should have shape (N_train, M), where N_train is number of train samples and M is number of features.
  - Compute P(xi | y == 0) and P(xi | y == 1), compare and save **binary** prediction in "train_pred" and "test_pred"
  - Compute train accuracy and test accuracy, saved in "train_acc" and "test_acc".
  - Return train accuracy and test accuracy.

[17]:
```
def myNBSolver(trainX, trainY, testX, testY):

    N_train = trainX.shape[0]
    N_test = testX.shape[0]
    M = trainX.shape[1]

    m = np.ones((2, M))
    v = np.ones((2, M))
    trainY = trainY.astype(int)
    freq_0 = np.bincount(trainY)[np.nonzero(np.bincount(trainY))[0]][0]
    # print(freq_0) #frequency of y == 0


    #### TODO ####
    # Compute P(y == 0) and P(y == 1)

    py0 = freq_0/N_train
    py1 = (N_train-freq_0)/N_train

    ###############
    print("Total probablity is %.2f. Should be equal to 1." %(py0 + py1))

    #### TODO ####
    # Compute mean/var for each label

    trainX0 = np.ones((freq_0, M))
    trainX1 = np.ones((N_train-freq_0, M))

    #initialize index
    idx_0 = 0
    idx_1 = 0
    for i in range(0, N_train):
```

```
        if trainY[i] == 0:
            trainX0[idx_0] = trainX[i]
            idx_0 = idx_0 + 1
        else:
            trainX1[idx_1] = trainX[i]
            idx_1 = idx_1 + 1
    for j in range(0, M):
        m[0][j] = np.mean(trainX0.T[j])
        v[0][j] = np.var(trainX0.T[j])
        m[1][j] = np.mean(trainX1.T[j])
        v[1][j] = np.var(trainX1.T[j])

    mean0 = m[0, :]
    mean1 = m[1, :]
    var0 = v[0, :]
    var1 = v[1, :]

    ###############
    assert(mean0.shape[0] == M)

    #### TODO #### TRAIN
    # Compute P(xi|y == 0) and P(xi|y == 1), compare and make prediction
    # This part may spend 5 - 10 minutes or even more if you use for loop, so␣
↪feel free to
    # print something (like step number) to check the progress

    p_0 = np.exp(-np.square(trainX-mean0)/(2*var0))*(1/(np.sqrt(2*np.pi*var0)))
    p_1 = np.exp(-np.square(trainX-mean1)/(2*var1))*(1/(np.sqrt(2*np.pi*var1)))
    prod0 = np.prod(p_0, axis=1)
    prod0 = prod0 * py0

    prod1 = np.prod(p_0, axis=1)
    prod1 = prod1 * py1
    train_pred = 1*(prod1 > prod0)


    p_0 = np.exp(-np.square(testX-mean0)/(2*var0))*(1/(np.sqrt(2*np.pi*var0)))
    p_1 = np.exp(-np.square(testX-mean1)/(2*var1))*(1/(np.sqrt(2*np.pi*var1)))
    prod0 = np.prod(p_0, axis=1)
    prod0 = prod0 * py0

    prod1 = np.prod(p_0, axis=1)
    prod1 = prod1 * py1
    test_pred = 1*(prod1 > prod0)

    ### Compute test accuracy
    count_train = 0
```

```
    for i in range(train_pred.shape[0]):
        count_train += int(train_pred[i] == trainY[i])
    train_acc = count_train/train_pred.shape[0]

    count_test = 0
    for i in range(test_pred.shape[0]):
        count_test += int(test_pred[i] == testY[i])
    test_acc = count_test/test_pred.shape[0]

    ###############
    assert(train_pred[0] == 0 or train_pred[0] == 1)
    assert(test_pred[0] == 0 or test_pred[0] == 1)




    return train_acc, test_acc
```

[18]:
```
# driver to test your NB solver
train_acc, test_acc = myNBSolver(trainX, trainY, testX, testY)
print("Train accuracy is %.2f" %(train_acc * 100))
print("Test accuracy is %.2f" %(test_acc * 100))
```

```
Total probablity is 1.00. Should be equal to 1.
Train accuracy is 90.04
Test accuracy is 89.77
```

0.3 P3. Test your result using sklearn

0.3.1 TODO

- Finish the skNBSolver() function.
  - fit model, make prediction and return accuracy for train and test sets.

[15]:
```
def skNBSolver(trainX, trainY, testX, testY):

    #### TODO ####
    # fit model
    # make prediction
    # compute accuracy
    classifier = GaussianNB()
    cls_nb = classifier.fit(trainX, trainY)
    sk_train_acc = cls_nb.score(trainX, trainY)
    sk_test_acc = cls_nb.score(testX, testY)

    ###############
    return sk_train_acc, sk_test_acc
```

[16]:
```
# driver to test skNBSolver
sk_train_acc, sk_test_acc = skNBSolver(trainX, trainY, testX, testY)
print("Train accuracy is %.2f" %(sk_train_acc * 100))
print("Test accuracy is %.2f" %(sk_test_acc * 100))
```

```
Train accuracy is 90.04
Test accuracy is 89.77
```

[ ]: