Homework 4, Problem 4 Classification on real data

ECE C143A/C243A, Spring Quarter 2022, Prof. J.C. Kao, TAs T. Monsoor, W. Yu

Background

Neural prosthetic systems can be built based on classifying neural activity related to planning. As described in class, this is analogous to mapping patterns of neural activity to keys on a keyboard. In this problem, we will apply the results of Problems 1 and 2 to real neural data. The neural data were recorded using a 100-electrode array in premotor cortex of a macaque monkey1. The dataset can be found on CCLE as `ps4_realdata.mat`.

The following describes the data format. The `.mat` file is loaded into Python as a dictionary with two keys: `train_trial` contains the training data and `test_trial` contains the test data. Each of these contains spike trains recorded simultaneously from 97 neurons while the monkey reached 91 times along each of 8 different reaching angles.

The spike train recorded from the i_{th} neuron on the n_{th} trial of the k_{th} reaching angle is accessed as

```
data['train_trial'][n,k][1][i,:]
```

where n = 0,...,90, k = 0,...,7, and i = 0, . . . , 96. The [1] in between [n,k] and [i,:] does not mean anything for this assignment and is simply an "artifact" of how the data is structured. A spike train is represented as a sequence of zeros and ones, where time is discretized in 1 ms steps. A zero indicates that the neuron did not spike in the 1 ms bin, whereas a one indicates that the neuron spiked once in the 1 ms bin. The structure test trial has the same format as train trial.

Each spike train is 700 ms long (and thus represented by an array of length 700). This comprises a 200ms baseline period (before the reach target turned on), a 500ms planning period (after the reach target turned on). Because it takes time for information about the reach target to arrive in premotor cortex (due to the time required for action potentials to propagate and for visual processing), we will ignore the first 150ms of the planning period. ***FOR THIS PROBLEM, we will take spike counts for each neuron within a single 200ms bin starting 150ms after the reach target turns on.***

In other words, to calculate firing rates, you will calculate it over the 200ms window:

```
data['train_trial'][n,k][1][i,350:550]
```

In [ ]:
```
import numpy as np
import numpy.matlib as npm
import matplotlib.pyplot as plt
import scipy.special
import scipy.io as sio
import math

data = sio.loadmat('ps4_realdata.mat') # load the .mat file.
NumTrainData = data['train_trial'].shape[0]
NumClass = data['train_trial'].shape[1]
NumTestData = data['test_trial'].shape[0]

# Reloading any code written in external .py files.
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

(a) (8 points)

Fit the ML parameters of model i) to the training data (91 × 8 observations of a length 97 array of neuron firing rates).

To calculate the firing rates, use a single 200ms bin starting from 150ms after the target turns on. This corresponds to using `data['train_trial'][n,k][1][i, 350:550]` to calculate all firing rates. This corresponds to a 200ms window that turns on 150ms after the reach turns on.

Then, use these parameters to classify the test data (91 × 8 data points) according to the decision rule (1). What is the percent of test data points correctly classified?

In [ ]:

```
##4a

# Calculate the firing rates.

trainDataArr =  np.zeros((NumClass,NumTrainData,97)) # contains the firing rat
es for all neurons on all 8 x 91 trials in the training set
testDataArr =  np.zeros((NumClass,NumTestData,97)) # for the testing set.

for classIX in range(NumClass):
    for trainDataIX in range(NumTrainData):
        trainDataArr[classIX,trainDataIX,:] = np.sum(data['train_trial'][train
DataIX,classIX][1][:,350:550],1)
    for testDataIX in range(NumTestData):
        testDataArr[classIX,testDataIX,:]=np.sum(data['test_trial'][testDataIX
,classIX][1][:,350:550],1)
#=====================================================#
# YOUR CODE HERE:
#   Fit the ML parameters of model i) to training data
#=====================================================#
modParam1, modParam2, modParam3 ={},{},{}
modParam1['pi']= np.array([1/NumClass]*NumClass)
modParam1['mean'] = np.mean(trainDataArr, axis=1)
modParam1['cov'] = np.cov(trainDataArr.reshape((NumClass*NumTrainData,97)).T)

#=====================================================#
# END YOUR CODE
#=====================================================#

#=====================================================#
# YOUR CODE HERE:
#   Classify the test data and print the accuracy
#=====================================================#
 #Classify Model
correct = 0.0
total = NumTestData*NumClass
for i in range(NumClass):
    for j in range(NumTestData):
        vals = np.zeros(NumClass)
        x = testDataArr[i,j]
        for classix in range(NumClass):
            dem = x-modParam1['mean'][classix]
            vals[classix] = np.log(modParam1['pi'][classix])- 0.5*dem@np.linal
g.pinv(modParam1['cov'])@dem - 0.5*np.log(np.linalg.det(modParam1['cov']))
        if np.argmax(vals)==i:
            correct += 1
print(float(correct)/total)
#=====================================================#
# END YOUR CODE
#=====================================================#
```

```
0.8351648351648352
```

Question:

What is the percent of test data points correctly classified?

Your answer:

83.5% of the test data points were correclty classified.

(b) (6 points)

Repeat part (a) for model ii). You `should encounter a Python error` when classifying the test data. What is this error? Why did the Python error occur? What would we need to do to correct this error?

To be concrete, the output of this cell should be a `Python error` and that's all fine. But we want you to understand what the error is so we can fix it later.

In [ ]:
```
##4b

#=====================================================#
# YOUR CODE HERE:
# Fit the ML parameters of model ii) to training data
#=====================================================#
modParam2['pi']= np.array([1/NumClass]*NumClass)
modParam2['mean'] = np.mean(trainDataArr, axis=1)
modParam2['cov'] = np.zeros((NumClass, 97,97))
for classix in range(NumClass):
    modParam2['cov'][classix] = np.cov(trainDataArr[classix].T)
#Classify Model
correct = 0.0
total = NumTestData*NumClass
for i in range(NumClass):
    for j in range(NumTestData):
        vals = np.zeros(NumClass)
        x = testDataArr[i,j]
        for classix in range(NumClass):
            dem = x-modParam2['mean'][classix]
            vals[classix] = np.log(modParam2['pi'][classix])- 0.5*dem@np.linal
g.pinv(modParam2['cov'][classix])@dem - 0.5*np.log(np.linalg.det(modParam2['co
v'][classix]))
        if np.argmax(vals)==i:
            correct += 1
print(float(correct)/total)
#=====================================================#
# END YOUR CODE
#=====================================================#
```

```
/home/kost/miniconda3/envs/NSP_hw2/lib/python3.6/site-packages/ipykernel_laun
cher.py:21: RuntimeWarning: divide by zero encountered in log

0.125
```

Question:

Why did the python error occur? What would we need to do to correct this error?

Your answer:

This python error occured becuase there are some neruons in which a spike is never recoreded. These all zero spike trains make the determinant of our covariance matrix 0, and thus throw an error when computing the natural log of this value. To correct this error we can remove the neruon recordings in which no spikes are ever recore for a trial.

(c) (8 points)

Correct the problem from part (b) by detecting and then removing offending neurons that cause the error. Now, what is the percent of test data points correctly classified? Is it higher or lower than your answer to part (a)? Why might this be?

In [ ]:

```
##4c
neuronsToRemove = []
#========================================================#
# YOUR CODE HERE:
#   Detect and then remove the offending neurons, so that
#   you no longer run into the bug in part (b).
#========================================================#
for neuronix in range(97):
    for i in range(NumClass):
        bad_neruon = np.zeros(NumTrainData)
        if np.all(trainDataArr[i,:,neuronix] == bad_neruon):
            neuronsToRemove.append(neuronix)
neuronsToRemove = np.unique(neuronsToRemove)
trainDataArr_fix = np.delete(trainDataArr, neuronsToRemove, axis=2)
testDataArr_fix = np.delete(testDataArr, neuronsToRemove, axis=2)
print(neuronsToRemove)
#========================================================#
# END YOUR CODE
#========================================================#
##
#========================================================#
# YOUR CODE HERE:
# Fit the ML parameters,classify the test data and print the accuracy
#========================================================#
d_prime = 97-len(neuronsToRemove)
modParam2['pi']= np.array([1/NumClass]*NumClass)
modParam2['mean'] = np.mean(trainDataArr_fix, axis=1)
modParam2['cov'] = np.zeros((NumClass, d_prime,d_prime))
for classix in range(NumClass):
    modParam2['cov'][classix] = np.cov(trainDataArr_fix[classix].T)
#Classify Model
correct = 0.0
total = NumTestData*NumClass
for i in range(NumClass):
    for j in range(NumTestData):
        vals = np.zeros(NumClass)
        x = testDataArr_fix[i,j]
        for classix in range(NumClass):
            dem = x-modParam2['mean'][classix]
            vals[classix] = np.log(modParam2['pi'][classix])- 0.5*dem@np.linal
g.pinv(modParam2['cov'][classix])@dem - 0.5*np.log(np.linalg.det(modParam2['co
v'][classix]))
        if np.argmax(vals)==i:
            correct += 1
print(float(correct)/total)
#========================================================#
# END YOUR CODE
#========================================================#
```

```
[10 27 36 40 46 50 71 72 74 88]
0.4409340659340659
```

Question:

What is the percent of test data points correctly classified? Is it higher or lower than your answer to part (a)? Why might this be?

This classifier correctly identified 44% of the test points. This is much lower than our classifier in part a. This is potentially because the neurons may in actuality have very simiar covariances--meaning that when we partion the data to give each class a different covariance, we are artificially skewing our model in a way the ground truth is not. This is to say that the model in which our gaussians have different covariances may not be true. Additionally, in using smaller subsets of the data to compute each covariance matrix, we are less likely to converge to the true covariance matrix (law of large numbers). So since our estimate is also poorer, it is not surprising that our classifier performs more poorly.

Your answer:

(d) (8 points)

Now we classify using a naive Bayes model. Repeat part (a) for model iii). Keep the convention in part (c), where offending neurons were removed from the anal- ysis.

In [ ]:
```
##4d
#======================================================#
# YOUR CODE HERE:
# Fit the ML parameters,classify the test data and print the accuracy
#======================================================#

modParam3['pi']= np.array([1/NumClass]*NumClass)
modParam3['mean'] = np.zeros((NumClass, d_prime))
for classix in range(NumClass):
    modParam3['mean'][classix] = np.mean(trainDataArr_fix[classix], axis=0)
#Classify Model
correct = 0.0
total = NumTestData*NumClass
for i in range(NumClass):
    for j in range(NumTestData):
        vals = np.zeros(NumClass)
        x = testDataArr_fix[i,j]
        for classix in range(NumClass):
            vals[classix] = np.log(modParam3['pi'][classix]) +np.log(modParam3
['mean'][classix]).T@x - np.sum(modParam3['mean'][classix])- np.sum(np.log(sci
py.special.factorial(x)))
        if np.argmax(vals)==i:
            correct += 1
print(float(correct)/total)
#======================================================#
# END YOUR CODE
#======================================================#
```

```
0.9203296703296703
```

Question:

what is the percent of test data points correctly classified?

Your answer:

92.03% of the test data points were correctly classified.