hw4p4

May 14, 2022

0.1 Homework 4, Problem 4 Classification on real data

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

0.2 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]
```

[1]:
```
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
```

0.2.1 (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?

[19]:
```
##4a

# Calculate the firing rates.

trainDataArr =  np.zeros((NumClass,NumTrainData,97)) # contains the firing␣
 ↪rates 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'][trainDataIX,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
#=====================================================#
from scipy.stats import multivariate_normal

means=np.mean(trainDataArr,axis=1)
```

```
cov=np.zeros([97,97])
for i in range(8):
    cov+=np.cov(trainDataArr[i].T)*1/8

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

#=====================================================#
# YOUR CODE HERE:
#   Classify the test data and print the accuracy
#=====================================================#
n_correct=0
n_total=0
for i in range(8):
    predicted=np.empty((8,91))
    for j in range(means.shape[0]):
        predicted[j]=multivariate_normal(means[j],cov).pdf(testDataArr[i])

    n_correct+=np.sum(np.argmax(predicted,axis=0)==i)
    n_total+=91

#=====================================================#
# END YOUR CODE
#=====================================================#
print(f"percent correct={round(100*n_correct/n_total,4)}%")
```

```
percent correct=96.0165%
```

Question: What is the percent of test data points correctly classified?

Your answer: 96.0165%

0.2.2 (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.

[26]:
```
##4b

#=====================================================#
# YOUR CODE HERE:
# Fit the ML parameters of model ii) to training data
#=====================================================#
```

```
from scipy.stats import multivariate_normal

means=np.mean(trainDataArr,axis=1)
cov=np.zeros([8,97,97])
for i in range(8):
    cov[i]=np.cov(trainDataArr[i].T)*1/8

n_correct=0
n_total=0
for i in range(8):
    predicted=np.empty((8,91))
    for j in range(means.shape[0]):
        #print(cov[j])
        predicted[j]=multivariate_normal(means[j],cov[j]).pdf(testDataArr[i])

    n_correct+=np.sum(np.argmax(predicted,axis=0)==i)
    n_total+=91

print(f"percent correct={round(100*n_correct/n_total,4)}%")
#=====================================================#
# END YOUR CODE
#=====================================================#
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
/tmp/ipykernel_10194/312127973.py in <module>
     18     for j in range(means.shape[0]):
     19         #print(cov[j])
---> 20         predicted[j]=multivariate_normal(means[j],cov[j]).
 ↪pdf(testDataArr[i])
     21
     22     n_correct+=np.sum(np.argmax(predicted,axis=0)==i)

~/anaconda3/lib/python3.9/site-packages/scipy/stats/_multivariate.py in␣
 ↪__call__(self, mean, cov, allow_singular, seed)
    358         See `multivariate_normal_frozen` for more information.
    359         """
--> 360         return multivariate_normal_frozen(mean, cov,
    361                                           allow_singular=allow_singular
    362                                           seed=seed)

~/anaconda3/lib/python3.9/site-packages/scipy/stats/_multivariate.py in␣
 ↪__init__(self, mean, cov, allow_singular, seed, maxpts, abseps, releps)
    728         self.dim, self.mean, self.cov = self._dist._process_parameters(
    729                                                                 None, mean,
 ↪cov)
```

```
--> 730         self.cov_info = _PSD(self.cov, allow_singular=allow_singular)
    731         if not maxpts:
    732             maxpts = 1000000 * self.dim

~/anaconda3/lib/python3.9/site-packages/scipy/stats/_multivariate.py in␣
 ↪__init__(self, M, cond, rcond, lower, check_finite, allow_singular)
    163         d = s[s > eps]
    164         if len(d) < len(s) and not allow_singular:
--> 165             raise np.linalg.LinAlgError('singular matrix')
    166         s_pinv = _pinv_1d(s, eps)
    167         U = np.multiply(u, np.sqrt(s_pinv))

LinAlgError: singular matrix
```

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

Your answer: This error occurs because the covariance matrix is singular, which happens because certain neurons do not fire at all, which we can confirm running the command `plt.imshow(cov[j]==0)`

0.2.3 (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?

[69]:
```
##4c
neuronsToRemove = []
#=========================================================#
# YOUR CODE HERE:
#   Detect and then remove the offending neurons, so that
#   you no longer run into the bug in part (b).
#=========================================================#
neuronsToKeep=list(range(97))
for i in range(8):
    for neuron in range(97):
          if all(trainDataArr[i,:,neuron]==0) and neuron not in neuronsToRemove:
                neuronsToRemove.append(neuron)
                neuronsToKeep.remove(neuron)
#=========================================================#
# END YOUR CODE
#=========================================================#
##
#=========================================================#
# YOUR CODE HERE:
# Fit the ML parameters,classify the test data and print the accuracy
#=========================================================#
```

```
from scipy.stats import multivariate_normal

trainDataKept=trainDataArr[:,:,neuronsToKeep]
testDataKept=testDataArr[:,:,neuronsToKeep]

means=np.mean(trainDataKept,axis=1)
cov=np.zeros([8,len(neuronsToKeep),len(neuronsToKeep)])
for i in range(8):
    cov[i]=np.cov(trainDataKept[i].T,bias=True)

n_correct=0
n_total=0
for i in range(8):
    predicted=np.empty((8,91))
    for j in range(means.shape[0]):
        #print(cov[j])
        predicted[j]=multivariate_normal(means[j],cov[j]).
 ↪logpdf(testDataKept[i])

    n_correct+=np.sum(np.argmax(predicted,axis=0)==i)
    n_total+=predicted.shape[1]

print(f"percent correct={round(100*n_correct/n_total,4)}%")
#=====================================================#
# END YOUR CODE
#=====================================================#
```

```
percent correct=44.0934%
```

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?

Your answer: 44.0934%, which is lower than for part (a), this is could because the neurons removed still contains some information, such as that will not fire for certain orientations

0.2.4 (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.

[86]:
```
##4d
#=====================================================#
# YOUR CODE HERE:
# Fit the ML parameters,classify the test data and print the accuracy
#=====================================================#
from scipy.stats import poisson
from scipy.special import factorial
means=np.mean(trainDataKept,axis=1)
```

```
n_correct=0
n_total=0
for i in range(8):
    predicted=np.ones((8,91))
    for j in range(means.shape[0]):
        predicted[j]=np.sum(poisson.logpmf(testDataKept[i],means[j]),axis=1)
        #predicted[j]=np.sum(testDataKept[i]*np.log(means[j])-np.
 ↪log(factorial(testDataKept[i]))-means[j],axis=1)
    n_correct+=np.sum(np.argmax(predicted,axis=0)==i)
    n_total+=predicted.shape[1]

print(f"percent correct={round(100*n_correct/n_total,4)}%")
#=====================================================#
# END YOUR CODE
#=====================================================#
```

```
percent correct=92.033%
```

Question: what is the percent of test data points correctly classified?

Your answer: 92.033%