q8

February 23, 2024

```
[ ]: import sys
     !{sys.executable} -m pip install opencv-python
     !{sys.executable} -m pip install seaborn
     !{sys.executable} -m pip install sklearn
     !{sys.executable} -m pip install scipy
     !{sys.executable} -m pip install ipympl
```

```
Requirement already satisfied: opencv-python in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages
(4.8.0.76)
Requirement already satisfied: numpy>=1.21.2 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
opencv-python) (1.25.2)


[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: seaborn in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages
(0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
seaborn) (1.25.2)
Requirement already satisfied: pandas>=1.2 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
seaborn) (2.1.0)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
seaborn) (3.8.2)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.47.2)
```

```
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (23.1)
Requirement already satisfied: pillow>=8 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
pandas>=1.2->seaborn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)


[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'error'
```

```
  error: subprocess-exited-with-error

  × Getting requirements to build wheel did not run successfully.
   exit code: 1
  > [15 lines of output]
     The 'sklearn' PyPI package is deprecated, use 'scikit-learn'
     rather than 'sklearn' for pip commands.

     Here is how to fix this error in the main use cases:
     - use 'pip install scikit-learn' rather than 'pip install sklearn'
     - replace 'sklearn' by 'scikit-learn' in your pip requirements files
       (requirements.txt, setup.py, setup.cfg, Pipfile, etc …)
     - if the 'sklearn' package is used by one of your dependencies,
        it would be great if you take some time to track which package uses
        'sklearn' instead of 'scikit-learn' and report it to their issue tracker
      - as a last resort, set the environment variable
        SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True to avoid this
error

      More information is available at
      https://github.com/scikit-learn/sklearn-pypi-package
      [end of output]

  note: This error originates from a subprocess, and is likely not a problem
with pip.
error: subprocess-exited-with-error

× Getting requirements to build wheel did not run successfully.
  exit code: 1
 > See above for output.

note: This error originates from a subprocess, and is likely not a problem with
pip.

[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: scipy in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages
(1.12.0)
Requirement already satisfied: numpy<1.29.0,>=1.22.4 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
scipy) (1.25.2)


[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: ipympl in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (0.9.3)


[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip


Requirement already satisfied: ipython<9 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (8.14.0)
Requirement already satisfied: numpy in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (1.25.2)
Requirement already satisfied: ipython-genutils in
```

```
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (0.2.0)
Requirement already satisfied: pillow in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (10.2.0)
Requirement already satisfied: traitlets<6 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (5.9.0)
Requirement already satisfied: ipywidgets<9,>=7.6.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (8.1.0)
Requirement already satisfied: matplotlib<4,>=3.4.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (3.8.2)
Requirement already satisfied: backcall in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.2.0)
Requirement already satisfied: decorator in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (5.1.1)
Requirement already satisfied: jedi>=0.16 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.19.0)
Requirement already satisfied: matplotlib-inline in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.1.6)
Requirement already satisfied: pickleshare in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (3.0.39)
Requirement already satisfied: pygments>=2.4.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (2.16.1)
Requirement already satisfied: stack-data in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.6.2)
Requirement already satisfied: colorama in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.4.6)
Requirement already satisfied: comm>=0.1.3 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipywidgets<9,>=7.6.0->ipympl) (0.1.4)
Requirement already satisfied: widgetsnbextension~=4.0.7 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipywidgets<9,>=7.6.0->ipympl) (4.0.8)
Requirement already satisfied: jupyterlab-widgets~=3.0.7 in
```

```
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipywidgets<9,>=7.6.0->ipympl) (3.0.8)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (4.47.2)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (23.1)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (2.8.2)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
jedi>=0.16->ipython<9->ipympl) (0.8.3)
Requirement already satisfied: wcwidth in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30->ipython<9->ipympl) (0.2.6)
Requirement already satisfied: six>=1.5 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
python-dateutil>=2.7->matplotlib<4,>=3.4.0->ipympl) (1.16.0)
Requirement already satisfied: executing>=1.2.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
stack-data->ipython<9->ipympl) (1.2.0)
Requirement already satisfied: asttokens>=2.1.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
stack-data->ipython<9->ipympl) (2.2.1)
Requirement already satisfied: pure-eval in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
stack-data->ipython<9->ipympl) (0.2.2)
```

[ ]:
```
import cv2
import seaborn
import sys
if sys.version_info[0] < 3:
        raise Exception("Python 3 not detected.")
```

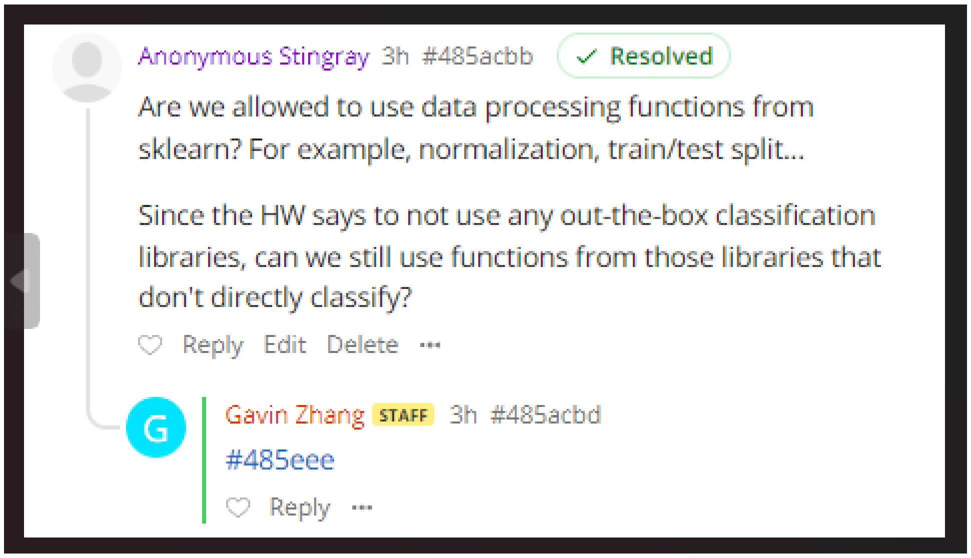
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import io
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
import scipy.stats as stats
from PIL import Image
from collections import defaultdict
```

[ ]:
```python
mnistData = np.load(f"../data/mnist-data-hw3.npz")
print("\nloaded %s data!" % "minst")
fields = "test_data", "training_data", "training_labels"
for field in fields:
    print(field, mnistData[field].shape)
mnistTest = mnistData["test_data"]
mnistAllData = mnistData["training_data"]
mnistAllLabels = mnistData["training_labels"]

spamData = np.load(f"../data/spam-data-hw3.npz")
print("\nloaded %s data!" % "spam")
fields = "test_data", "training_data", "training_labels"
for field in fields:
    print(field, spamData[field].shape)
spamTest = spamData["test_data"]
spamAllData = spamData["training_data"]
spamAllLabels = spamData["training_labels"]
```

```
loaded minst data!
test_data (10000, 1, 28, 28)
training_data (60000, 1, 28, 28)
training_labels (60000,)

loaded spam data!
test_data (1000, 32)
training_data (4171, 32)
training_labels (4171,)
```

[ ]:
```python
SEED = 0
```

[ ]:
```python
mnistXReshaped = np.reshape(mnistAllData, (60000, 28*28))
mnistTestReshaped = np.reshape(mnistTest, (10000, 28*28))
# print(mnistAllData[0][0][20])
# print(mnistXReshaped[0])
```


Anonymous Stingray 3h #485acbb ✓ Resolved

Are we allowed to use data processing functions from sklearn? For example, normalization, train/test split...

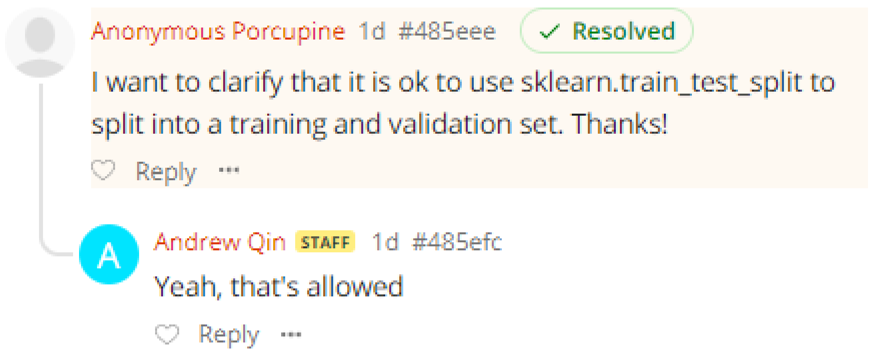
Since the HW says to not use any out-the-box classification libraries, can we still use functions from those libraries that don't directly classify?

Reply Edit Delete ···

Gavin Zhang STAFF 3h #485acbd

#485eee

Reply ···


Anonymous Porcupine 1d #485eee ✓ Resolved

I want to clarify that it is ok to use sklearn.train_test_split to split into a training and validation set. Thanks!

Reply ···

Andrew Qin STAFF 1d #485efc

Yeah, that's allowed

Reply ···

```
[ ]: #Data processing
     mnistXReshaped = normalize(mnistXReshaped, norm='l2')
     mnistTestReshaped = normalize(mnistTestReshaped, norm='l2')
```

0.1 Question 1

```
[ ]: #Fit Gaussian to Data
     mnistTrainCount = np.zeros(10)
     mnistTrainRunningSum = np.zeros((10, 784))
     mnistTrainMean = np.zeros((10, 784))

     for (data, label) in zip(mnistXReshaped, mnistAllLabels):
         mnistTrainCount[label] += 1
         mnistTrainRunningSum[label] += data

     for i in range(0,10):
         mnistTrainMean[i] = mnistTrainRunningSum[i] / mnistTrainCount[i]
```

```
[ ]: print(mnistTrainRunningSum.shape)
     print(mnistTrainMean.shape)
     print(mnistTrainCount[0])
```

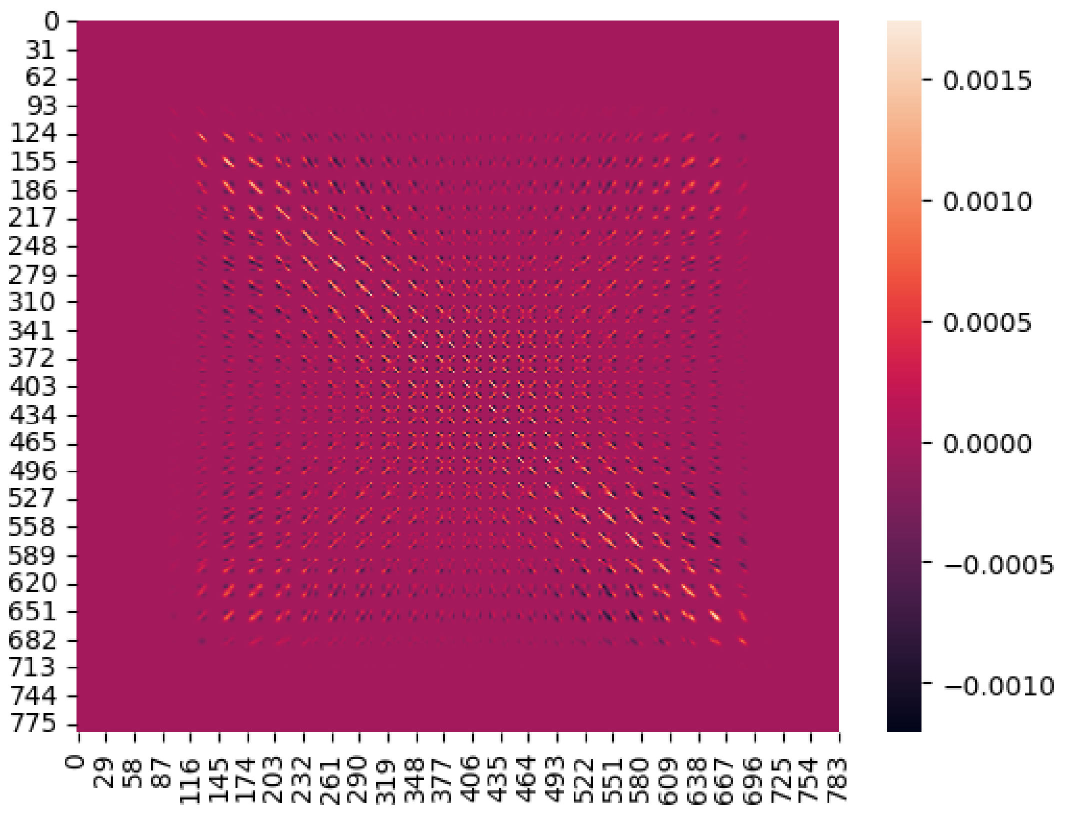
```
(10, 784)
(10, 784)
5902.0
```

[ ]:
```
#QDA Estimation of Varience
mnistTrainVar = np.zeros((10, 784, 784))
mnistTrainQDAVar = np.zeros((10, 784, 784))

for (data, label) in zip(mnistXReshaped, mnistAllLabels):
    meanDiff = data - mnistTrainMean[label]
    mnistTrainVar[label] += np.outer(meanDiff, meanDiff)

for i in range(0,10):
    mnistTrainQDAVar[i] = mnistTrainVar[i] / mnistTrainCount[i]
```

0.2 Question 2

[ ]:
```
#Heatmap
ax = seaborn.heatmap(mnistTrainQDAVar[0])
print(mnistTrainQDAVar[1].shape)
```

```
(784, 784)
```



0.3 Question 3

0.3.1 Part a

```
[ ]: mnistTrainX, mnistTestX, mnistTrainY, mnistTestY =␣
      ↪train_test_split(mnistXReshaped, mnistAllLabels, test_size=10000,␣
      ↪random_state=SEED, shuffle=True)
```

```
[ ]: print("mnist trainData shape: ", mnistTrainX.shape)
     print("mnist testData shape: ", mnistTestX.shape)
     print("mnist trainLabel shape: ", mnistTrainY.shape)
     print("mnist testLabel shape: ", mnistTestY.shape)
```

```
mnist trainData shape:  (50000, 784)
mnist testData shape:  (10000, 784)
mnist trainLabel shape:  (50000,)
mnist testLabel shape:  (10000,)
```

```
[ ]: def computeMean(inputData, inputLabels):
         #Fit Gaussian to Data
         mnistTrainCount = np.zeros(10)
         mnistTrainRunningSum = np.zeros((10, 784))
         mnistTrainMean = np.zeros((10, 784))

         for (data, label) in zip(inputData, inputLabels):
             mnistTrainCount[label] += 1
             mnistTrainRunningSum[label] += data

         for i in range(0,10):
             mnistTrainMean[i] = mnistTrainRunningSum[i] / mnistTrainCount[i]

         # print(mnistTrainRunningSum.shape)
         # print(mnistTrainMean.shape)
         # print(mnistTrainCount[0])

         return mnistTrainCount, mnistTrainMean
```

```
[ ]: def computeClassVarience(inputData, inputLabels, mean):
         #QDA Estimation of Varience
         classVar = np.zeros((10, 784, 784))

         for (data, label) in zip(inputData, inputLabels):
             meanDiff = data - mean[label]
             classVar[label] += np.outer(meanDiff, meanDiff)
         return classVar
```

```
[ ]: def computeLDAVar(classVar, totalPoints):
         #Find Pooled Varience
         pooledVarience = np.zeros((784, 784))
```

```
    for i in range(0, 10):
        pooledVarience += classVar[i]

    mnistTrainLDAVar = pooledVarience / totalPoints

    return mnistTrainLDAVar
```

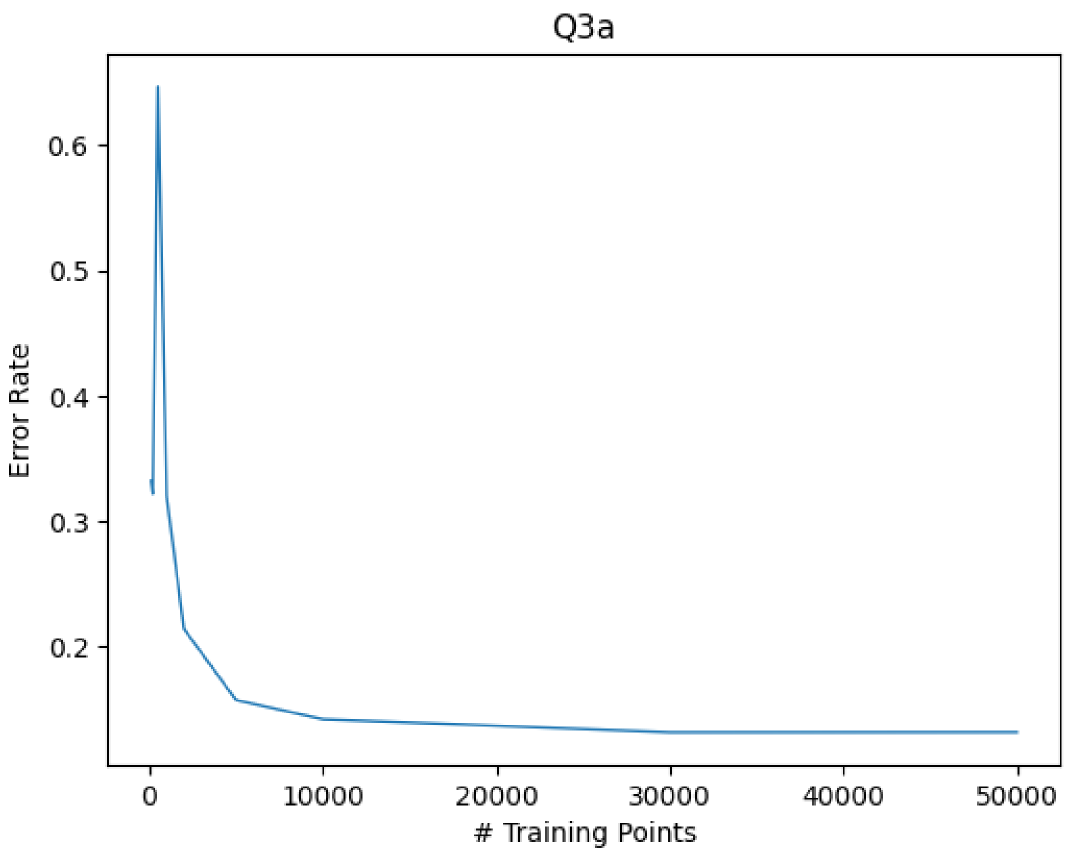
```
[ ]: def linDiscrim(classIndex, classMean, point, classPreCalc, priorProb):
         result = np.dot(classPreCalc[classIndex], point)
         result = result - (np.dot(classPreCalc[classIndex], classMean[classIndex]))/
      ↪2
         result = result + np.log(priorProb)
         return result

     errorLin = defaultdict(lambda:np.zeros((2, 10)))
     for numTestPoints in [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]: #,␣
      ↪500, 1000, 2000, 5000, 10000, 30000, 30000]:
         #Train on i points
         trainSetX = mnistTrainX[:numTestPoints]
         trainSetY = mnistTrainY[:numTestPoints]

         classCount, classMean = computeMean(trainSetX, trainSetY)
         classVar = computeClassVarience(trainSetX, trainSetY, classMean)
         LDAVar = computeLDAVar(classVar, numTestPoints)
         LDAVarPseudoInv = np.linalg.pinv(LDAVar)

         classPreCalc = {}
         #Precalculate some values
         for i in range(0, 10):
             classPreCalc[i] = np.dot(classMean[i].T, LDAVarPseudoInv)

         #Using linear Disc
         #Test
         errorCount = 0
         for testPointIndex in range(0, len(mnistTestX)):
             linDisc = np.empty((10))
             for i in range(0, 10):
                 pi = classCount[i]/numTestPoints
                 linDisc[i] = linDiscrim(i, classMean, mnistTestX[testPointIndex],␣
      ↪classPreCalc, pi)
             pred = np.argmax(linDisc)
             if (pred != mnistTestY[testPointIndex]):
                 errorLin[numTestPoints][0][mnistTestY[testPointIndex]] += 1
                 errorCount+=1
             errorLin[numTestPoints][1][mnistTestY[testPointIndex]] += 1

         print(errorCount/(len(mnistTestX)))
```

```
    # errorLin.append((numTestPoints, errorCount/(errorCount+correctCount)))
```

```
0.3325
0.3218
0.6465
0.3203
0.2143
0.1576
0.1423
0.1319
0.132
```

[ ]:
```
for i in errorLin:
    print(f"{i[0]} Training Points, with {i[1]} error rate")

errorX = [pts[0] for pts in errorLin]
errorY = [pts[1] for pts in errorLin]

fig, ax = plt.subplots()
ax.plot(errorX, errorY, linewidth=1)
plt.title("Q3a")
plt.xlabel("# Training Points")
plt.ylabel("Error Rate")
plt.show()
```

```
100 Training Points, with 0.3325 error rate
200 Training Points, with 0.3218 error rate
500 Training Points, with 0.6465 error rate
1000 Training Points, with 0.3203 error rate
2000 Training Points, with 0.2143 error rate
5000 Training Points, with 0.1576 error rate
10000 Training Points, with 0.1423 error rate
30000 Training Points, with 0.1319 error rate
50000 Training Points, with 0.132 error rate
```



0.3.2 Part b

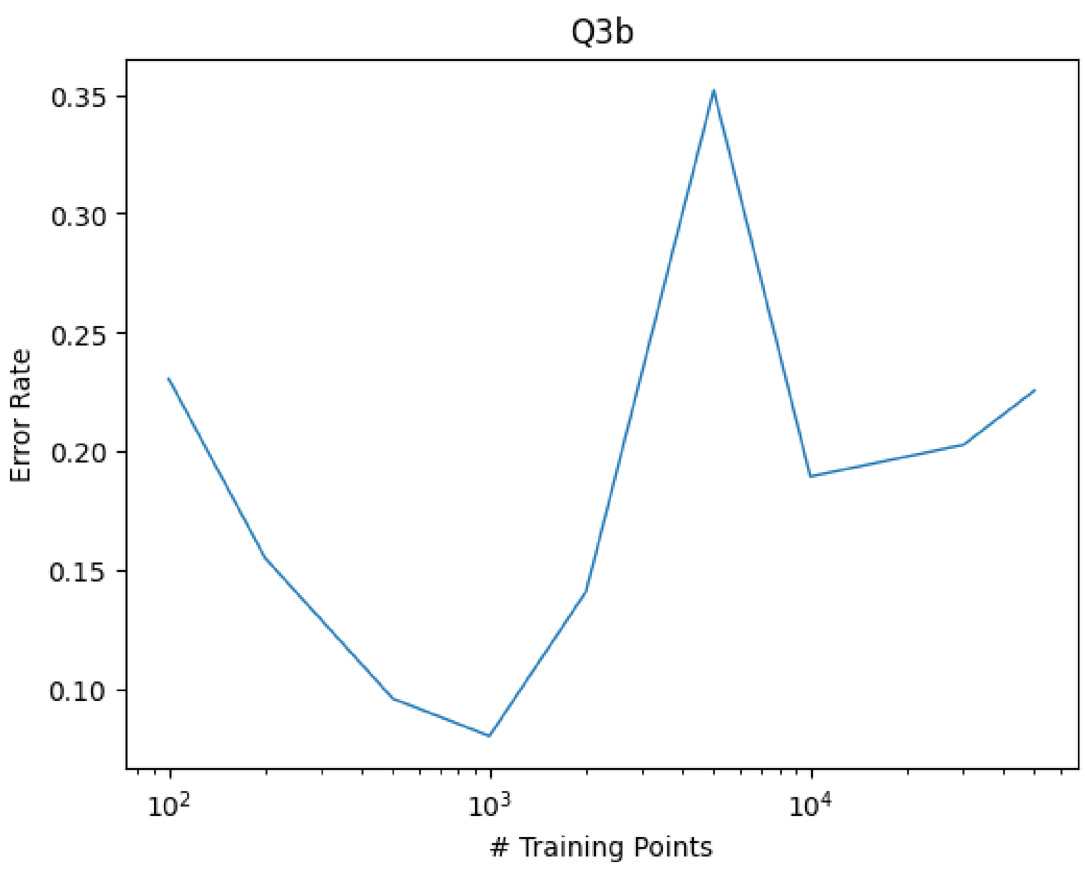
```
[ ]: def computeQDAClassVar(inputData, inputLabels, classMean, classCount):
         #QDA Estimation of Varience
         mnistTrainVar = np.zeros((10, 784, 784))
         mnistTrainQDAVar = np.zeros((10, 784, 784))

         for (data, label) in zip(inputData, inputLabels):
             meanDiff = data - classMean[label]
             mnistTrainVar[label] += np.outer(meanDiff, meanDiff)
             #print("mnistTrain: ", mnistTrainVar[label][300])

         for i in range(0,10):
             mnistTrainQDAVar[i] = (mnistTrainVar[i] / classCount[i])
             mnistTrainQDAVar[i] = mnistTrainQDAVar[i] + 1e-10*np.identity(784)
             #print(mnistTrainQDAVar)
             #print(mnistTrainQDAVar[i][2])
         return mnistTrainQDAVar
```

```
[ ]: def quadDiscrim(index, inverseQDAVar, pointMeanDiff):
         result = (np.linalg.multi_dot([pointMeanDiff[index].T,␣
      ↪inverseQDAVar[index], pointMeanDiff[index]]))/(-2)
         return result

     error = []
     for numTestPoints in [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]: #,␣
      ↪500, 1000, 2000, 5000, 10000, 30000, 30000]:
         #Train on i points
         trainSetX = mnistTrainX[:numTestPoints]
         trainSetY = mnistTrainY[:numTestPoints]

         # print(trainSetX.shape)
         # print(trainSetY.shape)

         classCount, classMean = computeMean(trainSetX, trainSetY)
         QDAClassVar = computeQDAClassVar(trainSetX, trainSetY, classMean,␣
      ↪classCount)

         preCalc = np.empty((10))
         inverseQDAVar = np.empty((10, 784, 784))
         pi = np.empty((10))

         for i in range(0,10):
             #print(np.linalg.slogdet(QDAClassVar[i])[1])
             preCalc[i]= (np.linalg.slogdet(QDAClassVar[i])[1])/(-2) + np.
      ↪log(classCount[i]/numTestPoints)
             inverseQDAVar[i] = np.linalg.inv(QDAClassVar[i])
             pi[i] = classCount[i]/numTestPoints

         # print("classmean: ", classMean.shape)
         # print(mnistTestX[testPointIndex].shape)
         #Using quadratic Disc
         #Test
         errorCount = 0
         correctCount = 0
         for testPointIndex in range(0, len(mnistTestX)):
             print(testPointIndex)
             point = mnistTestX[testPointIndex]
             pointMeanDiff = point - classMean
             # print("pointmeandiff: ", pointMeanDiff.shape)
             quadDisc = np.empty(10)
             for i in range(0,10):
                 quadDisc[i] = quadDiscrim(i, inverseQDAVar, pointMeanDiff)
             quadDisc = quadDisc + preCalc

             pred = np.argmax(quadDisc)
```

```
        if (pred != mnistTestY[testPointIndex]):
            errorCount += 1
        else:
            correctCount += 1
    print(errorCount/(errorCount+correctCount))
    error.append((numTestPoints, errorCount/(errorCount+correctCount)))
```

[ ]:
```
for i in error:
    print(f"{i[0]} Training Points, with {i[1]} error rate")

errorX = [pts[0] for pts in error]
errorY = [pts[1] for pts in error]

fig, ax = plt.subplots()
ax.plot(errorX, errorY, linewidth=1)
plt.title("Q3b")
plt.xlabel("# Training Points")
plt.xscale("log")
plt.ylabel("Error Rate")
plt.show()
```

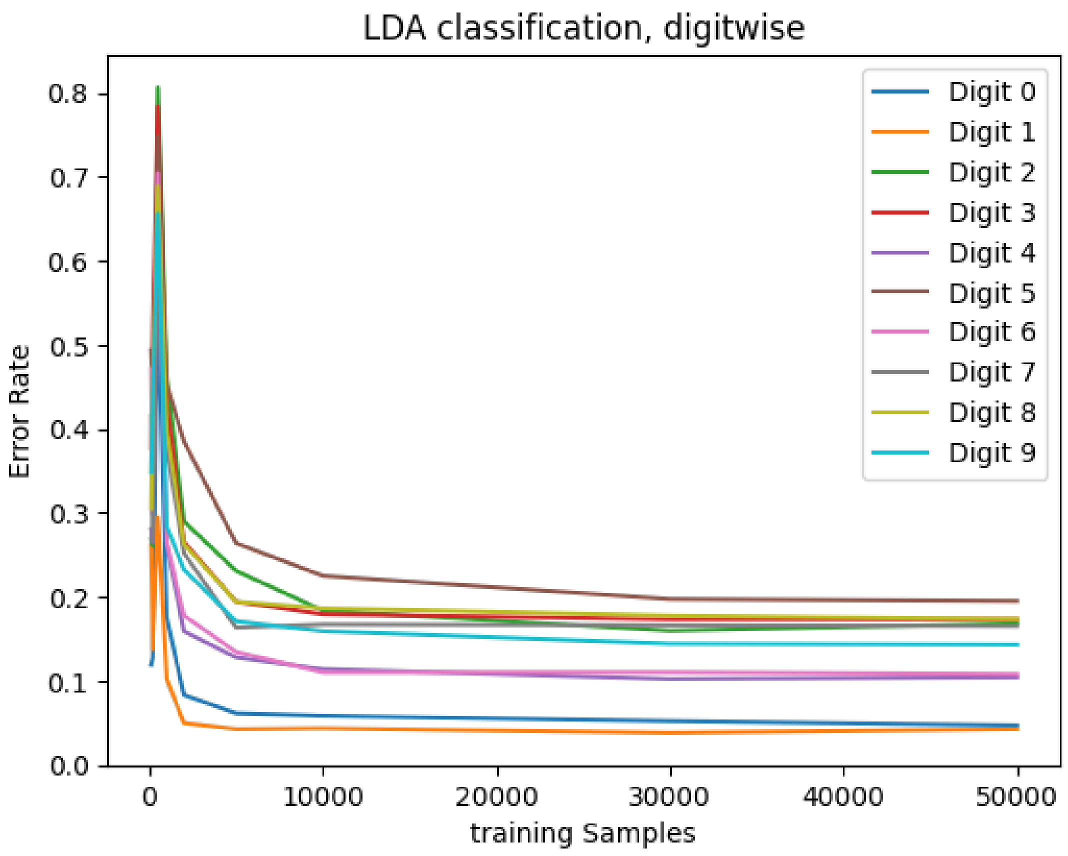
```
100 Training Points, with 0.2304 error rate
200 Training Points, with 0.1549 error rate
500 Training Points, with 0.0958 error rate
1000 Training Points, with 0.0801 error rate
2000 Training Points, with 0.1409 error rate
5000 Training Points, with 0.3518 error rate
10000 Training Points, with 0.1893 error rate
30000 Training Points, with 0.2026 error rate
50000 Training Points, with 0.2255 error rate
```



0.3.3 Part d

```
[ ]: errorRate = np.zeros((9, 10))
     numSample = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
     for index in range(0,9):
         errorRate[index] = errorLin[numSample[index]][0]/
      ↪errorLin[numSample[index]][1]
     print(errorRate)
     errorRate = errorRate.T
     for index in range(0,10):
         plt.plot(numSample, errorRate[index], label = f"Digit {index}")
         plt.legend()
     plt.title("LDA classification, digitwise")
     plt.xlabel("training Samples")
     plt.ylabel("Error Rate")
     plt.show()
```

```
[[0.11946447 0.26118721 0.26878613 0.37709773 0.28098291 0.49389567
  0.47157895 0.41580433 0.30498534 0.34851485]
 [0.1277034  0.13789954 0.261079   0.52517275 0.26495726 0.43840178
  0.29789474 0.26810913 0.46236559 0.45148515]
 [0.56024717 0.29497717 0.80635838 0.78381046 0.65277778 0.74805771
  0.70421053 0.60677328 0.68914956 0.65643564]
 [0.17610711 0.10228311 0.46339114 0.42941757 0.26282051 0.45394007
  0.26526316 0.37723424 0.3998045  0.28415842]
 [0.08341916 0.05022831 0.28998073 0.26554788 0.15918803 0.38512764
  0.17789474 0.25211665 0.2629521  0.23267327]
 [0.06179197 0.04292237 0.23121387 0.19447187 0.12820513 0.26415094
  0.13473684 0.16368768 0.1945259  0.17128713]
 [0.05870237 0.04383562 0.18400771 0.17966436 0.11431624 0.22530522
  0.11052632 0.16745061 0.18670577 0.15940594]
 [0.05252317 0.03835616 0.15992293 0.17374136 0.1025641  0.19755827
  0.11052632 0.16650988 0.17790811 0.14455446]
 [0.04737384 0.04292237 0.16859345 0.17374136 0.10470085 0.19533851
  0.10842105 0.16556914 0.17399804 0.14356436]]
```



0.4 Question 4

[ ]:
```
#Train on i points
trainSetX = mnistTrainX[:1000]
trainSetY = mnistTrainY[:1000]

# print(trainSetX.shape)
# print(trainSetY.shape)

classCount, classMean = computeMean(trainSetX, trainSetY)
QDAClassVar = computeQDAClassVar(trainSetX, trainSetY, classMean, classCount)

preCalc = np.empty((10))
inverseQDAVar = np.empty((10, 784, 784))
pi = np.empty((10))

for i in range(0,10):
    #print(np.linalg.slogdet(QDAClassVar[i])[1])
    preCalc[i]= (np.linalg.slogdet(QDAClassVar[i])[1])/(-2) + np.
  ↪log(classCount[i]/numTestPoints)
    inverseQDAVar[i] = np.linalg.inv(QDAClassVar[i])
    pi[i] = classCount[i]/numTestPoints

# print("classmean: ", classMean.shape)
# print(mnistTestX[testPointIndex].shape)
#Using quadratic Disc
#Test
errorCount = 0
correctCount = 0
pred = []
for testIndex in range(0, len(mnistTestReshaped)):
    print(testIndex)
    point = mnistTestReshaped[testIndex]
    pointMeanDiff = point - classMean
    quadDisc = np.empty(10)
    for i in range(0,10):
        quadDisc[i] = quadDiscrim(i, inverseQDAVar, pointMeanDiff)
    quadDisc = quadDisc + preCalc

    pred.append(np.argmax(quadDisc))
```

[ ]:
```
import pandas as pd
list = [*range(1, len(pred)+1)]
outputDict = {"Id":list, "Category": pred}
df = pd.DataFrame(outputDict)
df.to_csv('mnistResult.csv', index=False)
```