Page 1

Part 1 Accuracies (10 points)

Setup	Cross-validation Accuracy
Unprocessed data	0.7647
0-value elements ignored	0.7320

Page 2

Part 1 Code Snippets (30 points)

1. Calculation of distribution parameters

```
# Find mean and variance arrays of each feature for a given dataset ‘dis’
# masked = True: take all data, masked = False: doesn’t take zeros in Col 3,4,6,8
def get_mean_and_var(dis, masked):
    if masked:
        for i in [2,3,5,7]:
            dis[dis[:, i] == 0, i] = np.nan
        mdis = np.nanmean(dis, axis=0)
        vdis = np.nanvar(dis, axis=0)
    else:
        mdis = np.mean(dis, axis=0)
        vdis = np.var(dis, axis=0)
    return mdis, vdis

def get_trained_model(dis, masked = False):
    dib = dis[dis[:, -1] != 0]
    ndib = dis[dis[:, -1] == 0]
    # mean and var of diabetes
    mdib, vdib = get_mean_and_var(dib[:,:-1], masked)
    # mean and var of non-diabetes
    mndib, vndib = get_mean_and_var(ndib[:,:-1], masked)
    total = dis.shape[0]
    ydib = dib.shape[0] / total
    lydib = np.log(ydib)
    yndib = 1 - ydib
    lyndib = np.log(yndib)
    return mdib, vdib, lydib, mndib, vndib, lyndib
```

2. Calculation of naive Bayes predictions

```
def predict(xdata, dib, vdib, lydib, mndib, vndib, lyndib):
    predictdib = np.array([norm.pdf(xdata[:, i], mdib[i], np.sqrt(vdib[i])) for i in range(len(mdib))]).T
    lndistdib = np.log(predictdib)
    nbayesdib = np.sum(lndistdib, axis=1)+lydib
    predictndib = np.array([norm.pdf(xdata[:, i], mndib[i], np.sqrt(vndib[i])) for i in range(len(mndib))]).T
    lndistndib = np.log(predictndib)
    nbayesndib = np.sum(lndistndib, axis=1)+lyndib
    ydata = np.zeros(xdata.shape[0])
    size = xdata.shape[0]
    for i in range(size):
        if nbayesdib[i] > nbayesndib[i]:
            ydata[i] = 1
        else:
            ydata[i] = 0
    return ydata # Array of predicted labels based on Gaussian Naive Bayes classifier

def evaluate(test, predictions):
    return np.sum(test == predictions)/test.shape[0] # Compare predicted labels to actual labels
```

3. Test-train split code: Please refer to function `def get_test_and_train(dis, trainsplit)` found on Page 7.

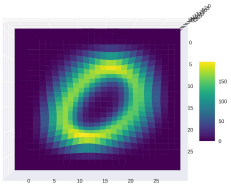
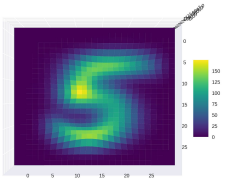
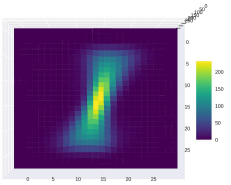
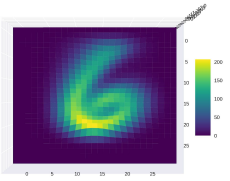
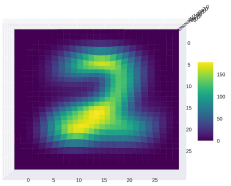
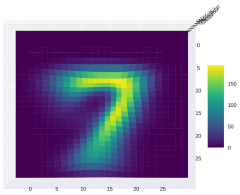
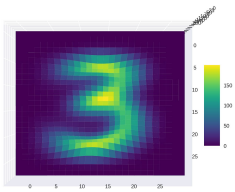
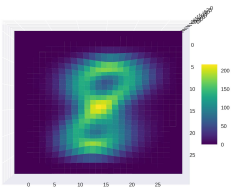
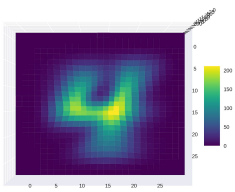
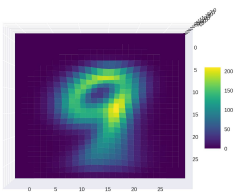
Page 3

Part 2 MNIST Accuracies (20 points)

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	0.7778	0.7914
2	Gaussian + stretched	0.8375	0.8447
3	Bernoulli + untouched	0.8154	0.8271
4	Bernoulli + stretched	0.8138	0.8280
5	10 trees + 4 depth + untouched	0.7450	0.7626
6	10 trees + 4 depth + stretched	0.7082	0.7012
7	10 trees + 16 depth + untouched	0.9956	0.9479
8	10 trees + 16 depth + stretched	0.9947	0.9477
9	30 trees + 4 depth + untouched	0.8118	0.8081
10	30 trees + 4 depth + stretched	0.7476	0.7701
11	30 trees + 16 depth + untouched	0.9976	0.9627
12	30 trees + 16 depth + stretched	0.9972	0.9658

Page 4

Part 2A Digit Images (10 points)

Digit	Mean Image	Digit	Mean Image
0		5	
1		6	
2		7	
3		8	
4		9	

Page 5

Part 2 Code (30 points)

The page should contain snippets of code demonstrating:

- Calculation of the Normal distribution parameters: Please refer to function `def get_trained_model(traindata, trainlabel, model = True)` found on Page 8 with `model = True`
- Calculation of the Bernoulli distribution parameters: Please refer to function `def get_trained_model(traindata, trainlabel, model = True)` found on Page 8 with `model = False`
- Calculation of the Naive Bayes predictions

```
def norm_predict(testdata, nmean, nvar, lamt):
  predict = np.empty(10, dtype=object)
  for j in range(len(nmean)): # labels
      predict[j] = np.log(np.array([norm.pdf(testdata[:, k], nmean[j][k], np.sqrt(nvar[j][k])) for k in
range(len(nmean[0]))]).T) # predict[i][j] contains norm pdf of 28x28 features of label i
  nbayes = np.array([np.sum(predict[i], axis=1) + lamt[i] for i in range(len(nmean))]).T
  predictions = np.array([np.argmax(nbayes, axis=1)])
  return predictions

def ber_predict(testdata, nmean, lamt):
  predict = np.empty(10, dtype=object)
  for j in range(len(nmean)): # labels
      predict[j] = np.array([bernoulli.logpmf(testdata[:, k]/255, nmean[j][k]/255, loc=0) for k in
range(len(nmean[0]))]).T # predict[i][j] contains Bernoulli prob of 28x28 features of label i
  nbayes = np.array([np.sum(predict[i], axis=1) + lamt[i] for i in range(len(nmean))]).T
  predictions = np.array([np.argmax(nbayes, axis=1)])
  return predictions

def evaluate(testlabel, predictions):
  return np.sum(testlabel == predictions)/testlabel.shape[0]
```

- Training of a decision tree & Calculation of a decision tree predictions

```
from sklearn.ensemble import RandomForestClassifier

print("With untouch data")
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=10,max_depth=4)
rfc.fit(A, b)
print("Random Forest with tree =10, depth = 4  : ", rfc.score(X, y))
[...]
```

See Part 2 -2 on Page 10 for other Random Forest classifier settings

Page 6+

Part 1

In [0]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import math

filename = 'pima-indians-diabetes.csv'
raw = np.genfromtxt(filename, dtype=float, delimiter=',')

def get_mean_and_var(dis, masked):
    if masked:
        for i in [2,3,5,7]:
            dis[dis[:, i] == 0, i] = np.nan
        mdis = np.nanmean(dis, axis=0)
        vdis = np.nanvar(dis, axis=0)
    else:
        mdis = np.mean(dis, axis=0)
        vdis = np.var(dis, axis=0)
    return mdis, vdis

def get_trained_model(dis, masked = False):
    dib = dis[dis[:, -1] != 0]
    ndib = dis[dis[:, -1] == 0]

    # mean and var of diabetes
    mdib, vdib = get_mean_and_var(dib[:,:-1], masked)

    # mean and var of non-diabetes
    mndib, vndib = get_mean_and_var(ndib[:,:-1], masked)

    total = dis.shape[0]

    ydib = dib.shape[0] / total
    lydib = np.log(ydib)

    yndib = 1 - ydib
    lyndib = np.log(yndib)

    return mdib, vdib, lydib, mndib, vndib, lyndib

def predict(xdata, dib, vdib, lydib, mndib, vndib, lyndib):
    predictdib = np.array([norm.pdf(xdata[:, i], mdib[i], np.sqrt(vdib[i])) for i in range(len(mdib))]).T
    lndistdib = np.log(predictdib)
    nbayesdib = np.sum(lndistdib, axis=1)+lydib
    predictndib = np.array([norm.pdf(xdata[:, i], mndib[i], np.sqrt(vndib[i])) for i in range(len(mndib))]).T

    lndistndib = np.log(predictndib)
    nbayesndib = np.sum(lndistndib, axis=1)+lyndib
    ydata = np.zeros(xdata.shape[0])
    size = xdata.shape[0]
    for i in range(size):
        if nbayesdib[i] > nbayesndib[i]:
            ydata[i] = 1
        else:
```

```
            ydata[i] = 0
    return ydata

def evaluate(test, predictions):
    return np.sum(test == predictions)/test.shape[0]

# 'dis' is a raw data array and 'trainsplit' is the fraction used for training
def get_test_and_train(dis, trainsplit):
    np.random.shuffle(dis)
    size = trainsplit*dis.shape[0]
    split = math.ceil(size)
    train = dis[:split,:]
    test = dis[split:,:]
    return test, train

# Get avg accuracy of 10 test-train splits
acc = np.zeros(10)
for i in range(10):
    test, train = get_test_and_train(raw, 0.8) # 80% training data
    testlabel = test[:,-1]
    mdib, vdib, lndistdib, mndib, vndib, lndistndib = get_trained_model(train)
    predictions = predict(test, mdib, vdib, lndistdib, mndib, vndib, lndistndib)
    acc[i] = evaluate(testlabel, predictions)
print("All data: ", np.mean(acc))

# Get avg accuracy of 10 test-train split with 0's in Col 3,4,6,8 ignored
acc_masked = np.zeros(10)
for i in range(10):
    test, train = get_test_and_train(raw, 0.8) # 80% training data
    testlabel = test[:,-1]
    mdib, vdib, lndistdib, mndib, vndib, lndistndib = get_trained_model(train, True)
    predictions = predict(test, mdib, vdib, lndistdib, mndib, vndib, lndistndib)
    acc_masked[i] = evaluate(testlabel, predictions)
print("Ignore 0: ", np.mean(acc_masked))
```

All data: 0.7647
Ignore 0: 0.7320

Part 2 - 1

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mlxtend.data import loadlocal_mnist
from scipy.stats import norm
from scipy.stats import bernoulli
import math

# training set
A, b = loadlocal_mnist(
        images_path='drive/My Drive/CS 498 AML/data/train-images.idx3-ubyte',
        labels_path='drive/My Drive/CS 498 AML/data/train-labels.idx1-ubyte')
```

```
b = np.array(b)
b = b.astype(np.float)
# test set
X, y = loadlocal_mnist(
        images_path='drive/My Drive/CS 498 AML/data/t10k-images.idx3-ubyte',
        labels_path='drive/My Drive/CS 498 AML/data/t10k-labels.idx1-ubyte')
y = np.array(y)
y = y.astype(np.float)

trainret = cv2.threshold(A,128,255,cv2.THRESH_BINARY)
testret = cv2.threshold(X,128,255,cv2.THRESH_BINARY)

traindata = np.array(trainret[1])
testdata = np.array(testret[1])

def resize(data):
  resized = np.empty((len(data), 400), dtype=np.uint8)
  for i in range(len(data)):
    img = data[i].reshape(28,28)
    x,y,w,h = cv2.boundingRect(img)
    num_only = img[y:y+h,x:x+w]
    stretched = cv2.resize(num_only, (20, 20))
    resized[i] = np.array(np.ravel(stretched))
  return (resized)

def get_trained_model(traindata, trainlabel, model = True): # traindata is (number of samples , features), model =
True: Gaussian, model = False: Bernoulli
    if model:
      white = np.full((10,len(traindata[0])), -2000) # A good guess
      black = np.full((10,len(traindata[0])), 2000)
      traindata = np.concatenate((traindata, white), axis=0)
      traindata = np.concatenate((traindata, black), axis=0)
      for i in range(2):
        dummy = np.array(range(10))
        trainlabel = np.concatenate((trainlabel, dummy), axis=0)
      num = np.empty(10, dtype=np.ndarray) # num has 10 objects (i), each object contains samples of i
      amt = np.empty(10, dtype=float) # amt keeps track of how many samples are in each i
      lamt = np.empty(10, dtype=float) # log of amt
      nmean = np.empty(10, dtype=np.ndarray) # nmean has 10 objects (i), each object contains means of each i
      nvar = np.empty(10, dtype=np.ndarray) # nvar has 10 objects (i), each object contains var of each i
      ly = np.empty(10, dtype=np.ndarray) # ly keeps track of fraction of each i from total sample count
      for i in range(10):
        num[i] = traindata[trainlabel == i] # num[i] contains all numbers for label i
        amt[i] = len(num[i])
      total = np.sum(amt)
      for j in range(num.shape[0]):
        nmean[j] = np.array(np.mean(num[j], axis=0)) # nmean[i] contains all mean of each feature of i
        nvar[j] = np.var(num[j], axis=0) # nvar[i] contains all var of each feature of i
      amt = np.array(amt)
      lamt = np.log(np.divide(amt, total))
      return nmean, nvar, lamt
    else:
      white = np.full((10,len(traindata[0])), 1) # A good guess
      traindata = np.concatenate((traindata, white), axis=0)
      for i in range(1):
        dummy = np.array(range(10))
        trainlabel = np.concatenate((trainlabel, dummy), axis=0)
```

```
      num = np.empty(10, dtype=np.ndarray) # num has 10 objects (i), each object contains samples of i
      amt = np.empty(10, dtype=float) # amt keeps track of how many samples are in each i
      lamt = np.empty(10, dtype=float) # log of amt
      nmean = np.empty(10, dtype=np.ndarray) # nmean has 10 objects (i), each object contains means of each i
      ly = np.empty(10, dtype=np.ndarray) # ly keeps track of fraction of each i from total sample count
      for i in range(10):
        num[i] = traindata[trainlabel == i] # num[i] contains all numbers for label i
        amt[i] = len(num[i])
      total = np.sum(amt)
      for j in range(num.shape[0]):
        nmean[j] = np.array(np.mean(num[j], axis=0)) # nmean[i] contains all mean of each feature of i
      amt = np.array(amt)
      lamt = np.log(np.divide(amt, total))
#       print("nmean: \n", nmean)
      return nmean, lamt

def norm_predict(testdata, nmean, nvar, lamt):
  predict = np.empty(10, dtype=object)
  for j in range(len(nmean)): # labels
      predict[j] = np.log(np.array([norm.pdf(testdata[:, k], nmean[j][k], np.sqrt(nvar[j][k])) for k in
range(len(nmean[0]))]).T) # predict[i][j] contains norm pdf of 28x28 features of label i
  nbayes = np.array([np.sum(predict[i], axis=1) + lamt[i] for i in range(len(nmean))]).T
  predictions = np.array([np.argmax(nbayes, axis=1)])
  return predictions

def ber_predict(testdata, nmean, lamt):
  predict = np.empty(10, dtype=object)
  for j in range(len(nmean)): # labels
      predict[j] = np.array([bernoulli.logpmf(testdata[:, k]/255, nmean[j][k]/255, loc=0) for k in
range(len(nmean[0]))]).T # predict[i][j] contains norm pdf of 28x28 features of label i
  nbayes = np.array([np.sum(predict[i], axis=1) + lamt[i] for i in range(len(nmean))]).T
  predictions = np.array([np.argmax(nbayes, axis=1)])
  return predictions

def evaluate(testlabel, predictions):
  return np.sum(testlabel == predictions)/testlabel.shape[0]

bounded = resize(A)
trainret = cv2.threshold(bounded,128,255,cv2.THRESH_BINARY)
btrain = np.array(trainret[1])

boundedtest = resize(X)
trainrettest = cv2.threshold(boundedtest,128,255,cv2.THRESH_BINARY)
btest = np.array(trainrettest[1])
```

```
# Normal Untouched
nmean, nvar, lamt = get_trained_model(traindata, b)
predictions = norm_predict(testdata, nmean, nvar, lamt)
acc = evaluate(y, predictions)
print("Normal Dist Untocuhed: ", acc)
```

```
# Bernoulli Untouched
nmean, lamt = get_trained_model(traindata, b, False)
predictions = ber_predict(traindata, nmean, lamt)
```

```
acc = evaluate(b, predictions)
print("Bernoulli Dist Untocuhed: ", acc)
```

```
# Normal Bounded
nmean, nvar, lamt = get_trained_model(btrain, b)
predictions = norm_predict(btest, nmean, nvar, lamt)
acc = evaluate(y, predictions)
print("Normal Dist Stretched Bounding Box: ", acc)
```

```
# Bernoulli Bounded
nmean, lamt = get_trained_model(btrain, b, False)
predictions = ber_predict(btest, nmean, lamt)
acc = evaluate(y, predictions)
print("Bernoulli Dist Stretched Bounding Box: ", acc)
```

Normal Dist Untocuhed: 0.7914
Bernoulli Dist Untocuhed: 0.8374666666666667
Normal Dist Stretched Bounding Box: 0.8271
Bernoulli Dist Stretched Bounding Box: 0.828

Part 2 - 2

In [0]:

```
from sklearn.ensemble import RandomForestClassifier

print("With untouch data")
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=10,max_depth=4)
rfc.fit(A, b)
print("Random Forest with tree =10, depth = 4  : ", rfc.score(X, y))
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=10,max_depth=16)
rfc.fit(A, b)
print("Random Forest with tree =10, depth = 16 : ", rfc.score(X, y))
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=30,max_depth=4)
rfc.fit(A, b)
print("Random Forest with tree =30, depth = 4  : ", rfc.score(X, y))
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=30,max_depth=16)
rfc.fit(A, b)
print("Random Forest with tree =30, depth = 16 : ", rfc.score(X, y))

print("\nWith stretched data")
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=10,max_depth=4)
rfc.fit(btrain, b)
print("Random Forest with tree =10, depth = 4  : ", rfc.score(btest, y))
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=10,max_depth=16)
rfc.fit(btrain, b)
print("Random Forest with tree =10, depth = 16 : ", rfc.score(btest, y))
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=30,max_depth=4)
rfc.fit(btrain, b)
print("Random Forest with tree =30, depth = 4  : ", rfc.score(btest, y))
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=30,max_depth=16)
rfc.fit(btrain, b)
print("Random Forest with tree =30, depth = 16 : ", rfc.score(btest, y))
```

With untouch data
Random Forest with tree =10, depth = 4 : 0.7841
Random Forest with tree =10, depth = 16 : 0.9469
Random Forest with tree =30, depth = 4 : 0.8014
Random Forest with tree =30, depth = 16 : 0.9611

With stretched data
Random Forest with tree =10, depth = 4 : 0.7102
Random Forest with tree =10, depth = 16 : 0.9501
Random Forest with tree =30, depth = 4 : 0.7645
Random Forest with tree =30, depth = 16 : 0.9644

Appendix 1: Getting map of mean as shown on Page 4

In [0]:

```
def get_trained_model(traindata, trainlabel, model = True):
    if model:
      white = np.full((10,len(traindata[0])), -2000) # Apply two “backgrounds” to make all variances non-zero
      black = np.full((10,len(traindata[0])), 2000)
      traindata = np.concatenate((traindata, white), axis=0)
      traindata = np.concatenate((traindata, black), axis=0)
      for i in range(2):
        dummy = np.array(range(10))
        trainlabel = np.concatenate((trainlabel, dummy), axis=0)
      num = np.empty(10, dtype=np.ndarray) # num has 10 objects (i), each object contains samples of i
      amt = np.empty(10, dtype=float) # amt keeps track of how many samples are in each i
      lamt = np.empty(10, dtype=float) # log of amt
      nmean = np.empty(10, dtype=np.ndarray) # nmean has 10 objects (i), each object contains means of each i
      nvar = np.empty(10, dtype=np.ndarray) # nvar has 10 objects (i), each object contains var of each i
      ly = np.empty(10, dtype=np.ndarray) # ly keeps track of fraction of each i from total sample count
      for i in range(10):
        num[i] = traindata[trainlabel == i] # num[i] contains all numbers for label i
        amt[i] = len(num[i])
      total = np.sum(amt)
      for j in range(num.shape[0]):
        nmean[j] = np.array(np.mean(num[j], axis=0)) # nmean[i] contains all mean of each feature of i
        nvar[j] = np.var(num[j], axis=0) # nvar[i] contains all var of each feature of i
      amt = np.array(amt)
      lamt = np.log(np.divide(amt, total))
      return nmean, nvar, lamt
    else:
      white = np.full((10,len(traindata[0])), 1) # A good guess
      traindata = np.concatenate((traindata, white), axis=0)
      for i in range(1):
        dummy = np.array(range(10))
        trainlabel = np.concatenate((trainlabel, dummy), axis=0)
      num = np.empty(10, dtype=np.ndarray) # num has 10 objects (i), each object contains samples of i
      amt = np.empty(10, dtype=float) # amt keeps track of how many samples are in each i
      lamt = np.empty(10, dtype=float) # log of amt
      nmean = np.empty(10, dtype=np.ndarray) # nmean has 10 objects (i), each object contains means of each i
      ly = np.empty(10, dtype=np.ndarray) # ly keeps track of fraction of each i from total sample count
      for i in range(10):
        num[i] = traindata[trainlabel == i] # num[i] contains all numbers for label i
        amt[i] = len(num[i])
      total = np.sum(amt)
      for j in range(num.shape[0]):
        nmean[j] = np.array(np.mean(num[j], axis=0)) # nmean[i] contains all mean of each feature of i
      amt = np.array(amt)
```

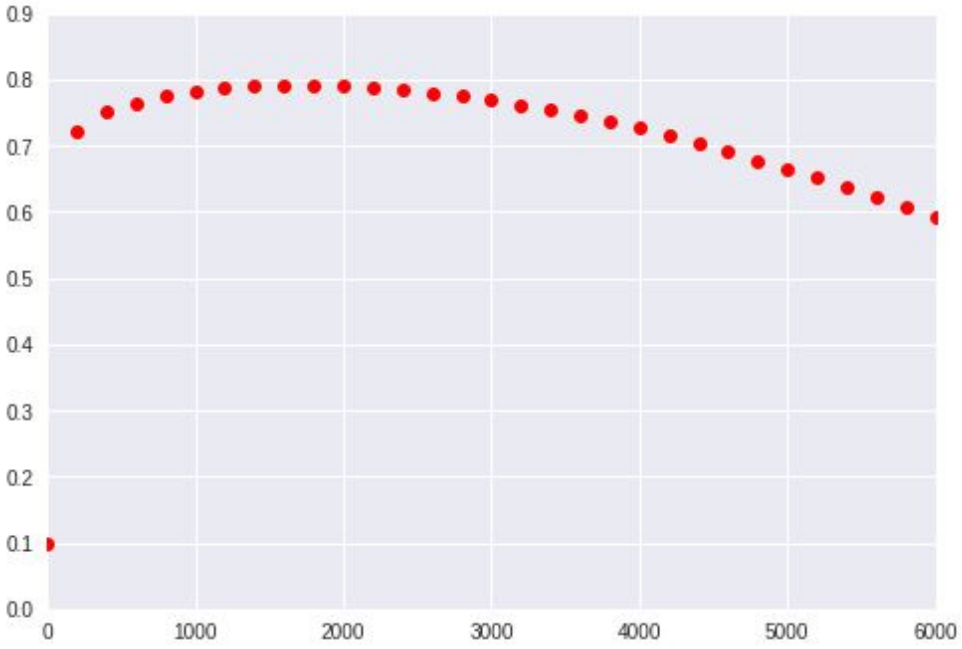
```
      lamt = np.log(np.divide(amt, total))
#       print("nmean: \n", nmean)
      return nmean, lamt

nmean, nvar, lamt = get_trained_model(traindata, b)
samplev = nmean[9].reshape(28,28)
def prepare(data):
  x = np.empty((len(data),len(data)))
  y = np.empty((len(data),len(data)))
  z = np.empty((len(data),len(data)))
  for i in range(len(data)):
    x[i] = np.full(len(data), i)
    y[i] = np.arange(len(data))
  for i in range(len(data)):
    for j in range(len(data)):
      z[i][j] = data[i][j]
  x = x.ravel()+1
  y = y.ravel()+1
  z = z.ravel()
  return x, y, z

def plotavg(data,number):
  fig = plt.figure()
  ax = fig.add_subplot(111, projection='3d')
  x, y, z = prepare(data)
  X, Y = np.meshgrid(np.arange(28), np.arange(28))
  ax = fig.gca(projection='3d')
  surf = ax.plot_surface(X, Y, data, cmap=plt.cm.viridis, linewidth=0)
  ax.view_init(elev=-90, azim=-90)
  ax.dist=6
  fig.colorbar(surf, shrink=0.5, aspect=5)
  plt.savefig('drive/My Drive/CS 498 AML/HW1/no%s.png' % number, dpi=300, format='png')
  plt.show()

## PLOT
for i in range(len(nmean)):
  samplev = nmean[i].reshape(28,28)
  plotavg(samplev, i)
```

Appendix 2: Plot of Gaussian Naive Bayes training accuracy with different background value in `def get_trained_model(traindata, trainlabel, model = True)`



The maximum probability is achieved at background value ranging from 1,200 to 2,000.