In [1]:
```
import csv
import pandas as pd
import numpy as np
import scipy as sp
import collections
import math
import operator
import sklearn.metrics
from sklearn.metrics import confusion_matrix
import matplotlib.pylab as plt
from sklearn.model_selection import train_test_split
```

In [2]:
```
!pwd
!ls
```

```
/Users/xiasong/Documents/Class_2016/DSE/DSE220/homework/homework_2
Homework_2.pdf          mnist_test_data.csv    mnist_train_labels.csv
hw_2.ipynb              mnist_test_labels.csv
hw_2.pdf                mnist_train_data.csv
```

load data into python and check data structure

In [3]:
```
train = pd.read_csv('mnist_train_data.csv')
train_lab = pd.read_csv('mnist_train_labels.csv')
test = pd.read_csv('mnist_test_data.csv')
test_lab = pd.read_csv('mnist_test_labels.csv')
```

In [4]:
```
print (train.shape, train_lab.shape, test.shape, test_lab.shape)
```

```
(5999, 784) (5999, 1) (999, 784) (999, 1)
```

Question 1: Compute and report the prior probabilities πj for all labels. (10 marks)

In [5]:
```
train_lab['5'].head(2)
```

Out[5]:
```
0    0
1    4
Name: 5, dtype: int64
```

In [6]:
```
#count each class in train lab
lab_count1=dict(collections.Counter(train_lab['5']))
lab_count = collections.OrderedDict(sorted(lab_count1.items()))
print (lab_count, lab_count1)
print ("---------------------------------------------------------------------
#calculate the fraction of each digit
priors1=[]
for k, v in lab_count.items():
    a = (v/len(train_lab))
    a = round(a,4)
    priors1.append((k,a))
priors2 = dict(priors1)
priors = collections.OrderedDict(sorted(priors2.items()))
print ('The prior probabilties of πj are:')
priors
```

```
OrderedDict([(0, 592), (1, 671), (2, 581), (3, 608), (4, 623), (5, 513),
 (6, 608), (7, 651), (8, 551), (9, 601)]) {0: 592, 4: 623, 1: 671, 9: 60
1, 2: 581, 3: 608, 5: 513, 6: 608, 7: 651, 8: 551}
--------------------------------------------------------------------
The prior probabilties of πj are:
```

Out[6]:
```
OrderedDict([(0, 0.0987),
             (1, 0.1119),
             (2, 0.0968),
             (3, 0.1014),
             (4, 0.1039),
             (5, 0.0855),
             (6, 0.1014),
             (7, 0.1085),
             (8, 0.0918),
             (9, 0.1002)])
```

In [7]:
```
lab_count1[1]
```

Out[7]:
```
671
```

Question 2: For each pixel Xi and label j, compute Pji = P(Xi = 1|y = j) (Use the maximum likelihood estimate shown in class). Use Laplacian Smoothing for computing Pji. Report the highest Pji for each label j. (15 marks)

In [8]:
```
priors[1]
```

Out[8]:
```
0.1119
```

In [9]:
```
#calcualte the pji
df1 = pd.concat([train_lab, train], axis=1)
Pji = []
for j in range(10):
    for i in range(1,785):
        df2 = df1.ix[:,(0,i)] #from df1 extract the 0 and ith column
        df3 = df2[(df2.iloc[:,0] == j) & (df2.iloc[:,1] > 0.9)]
        pji = (len(df3)+1) / (lab_count1[j]+2)
        Pji.append((j,pji))
```

In [10]:
```
#select the highest Pji for each label
k = 784
Pj = []
for j in range(10):
    n = []
    for i in range (784):
        a = j * k + i
        b = Pji[a][1]
        b = round(b,4)
        n.append(b)
    c = max(n)
    Pj.append((Pji[a][0],c))
#Pj = dict(Pj)
print ("The hightest Pji for each label j are")
Pj
```

```
The hightest Pji for each label j are
```

Out[10]:
```
[(0, 0.8519),
 (1, 0.9851),
 (2, 0.729),
 (3, 0.8082),
 (4, 0.8496),
 (5, 0.7126),
 (6, 0.8492),
 (7, 0.7948),
 (8, 0.8752),
 (9, 0.8673)]
```

Question 3: Use naive bayes (as shown in lecture slides) to classify the test data. Report the accuracy. (5 marks)

In [11]:
```
# form the classifier
k = 784
argmax = []
for i in range(len(test)):
    a = test.iloc[i,:]
    am = []
    for j in range(10):
        am1 = []
        for m in range(784):
            d = k * j + m
            if a[m] > 0.9:
                am2 = math.log(Pji[d][1])
                am1.append(am2)
        am3 = math.log(priors[j]) + sum(am1)
        am.append(am3)
    idx, val = max(enumerate(am), key=operator.itemgetter(1))
    argmax.append((idx, val))
```

In [12]:
```
# evaluate the model results with accuracy
labels_pre = []
for i in range(len(argmax)):
    a = argmax[i][0]
    labels_pre.append(a)
count = 0
for i in range(len(labels_pre)):
    if labels_pre[i] == test_lab.iloc[i,0]:
        count = count + 1
accuracy = count / len(test)

print ('The accuracy of model is %s' % accuracy)
```

```
The accuracy of model is 0.6706706706706707
```

Question 4: Compute the confusion matrix (as shown in the lectures) and report the top 3 pairs with most (absolute number) incorrect classications. (10 marks)

In [13]:
```
labels_tru = []
for i in range(len(test_lab)):
    a = test_lab.iloc[i,0]
    labels_tru.append(a)
```

In [14]:
```
confusion_matrix(labels_tru, labels_pre)
```

Out[14]:
```
array([[81,  0,  0,  0,  0,  0,  0,  0,  4,  0],
       [ 0, 60,  1,  2,  0,  0,  1,  0, 62,  0],
       [ 5,  0, 92,  0,  0,  0,  1,  0, 18,  0],
       [ 0,  0,  2, 91,  0,  1,  3,  0,  9,  1],
       [ 2,  0,  2,  0, 59,  0,  3,  0, 28, 16],
       [12,  0,  2, 18,  1, 10,  2,  1, 40,  1],
       [ 8,  0,  4,  0,  1,  0, 68,  0,  6,  0],
       [ 3,  0,  2,  3,  1,  0,  0, 58, 21, 10],
       [ 3,  0,  2,  8,  0,  0,  0,  0, 76,  0],
       [ 1,  0,  1,  2,  2,  0,  0,  0, 13, 75]])
```

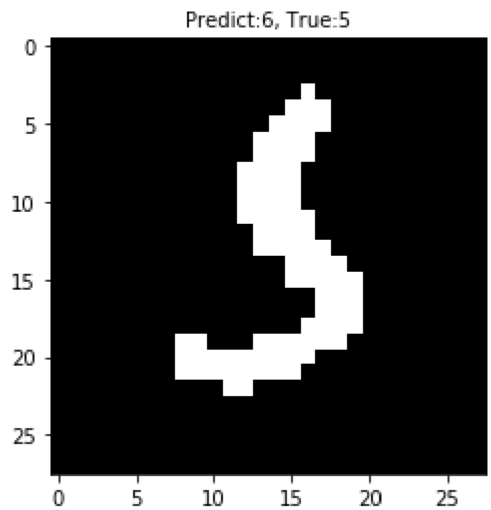
From the confusion_matrix we can tell that the top 3 pairs of incorrect classification are (8,1), (8,5) and (8,4).

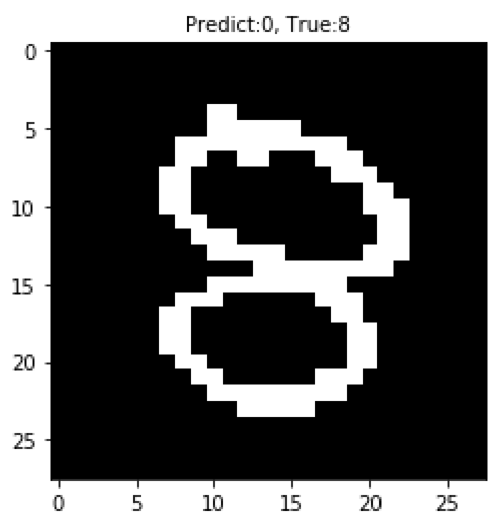
Question 5: Visualizing mistakes: Print two MNIST images from the test data that your classier misclassied. Write both the true and predicted labels for both of these misclassied digits. (10 marks)

In [15]:
```
#find the index where the predicted labels do not match with true labels
indicies = []
for i in range(len(labels_tru)):
    if labels_tru[i] != labels_pre[i]:
        indicies.append(i)
```

In [16]:
```
pixels1 = np.array(test.iloc[339,:], dtype='uint8')
pixels1 = np.array(pixels1, dtype='uint8')
pixels2 = np.array(test.iloc[267,:], dtype='uint8')
pixels2 = np.array(pixels2, dtype='uint8')
# Reshape the array into 28 x 28 array (2-dimensional array)
pixels1 = pixels1.reshape((28, 28))
pixels2 = pixels2.reshape((28, 28))
plt.title('Predict:%i, True:%i' %( labels_pre[339],labels_tru[339]), fontsiz
plt.imshow(pixels1, cmap='gray')
plt.show()

plt.title('Predict:%i, True:%i' %( labels_pre[267],labels_tru[267]), fontsiz
plt.imshow(pixels2, cmap='gray')
plt.show()
```





Now, we will implement Gaussian Mixture Model and Linear Discriminant Anal- ysis on the breast cancer data (sklearn.datasets.load breast cancer) available in sklean.datasets. Load the data and split it into train-validation-test (40-20-40 split). Don't shue the data, otherwise your results will be different.

In [39]:
```
#download data
from sklearn import datasets
cancer = datasets.load_breast_cancer()
X = cancer.data
y = cancer.target
type(y)
```

Out[39]: numpy.ndarray

In [40]:
```
X_tra_val, X_tes, y_tra_val, y_tes = train_test_split(X, y, test_size=0.4, r
X_tra, X_val, y_tra, y_val = train_test_split(X_tra_val, y_tra_val, test_siz
```

Question 6: Implement Gaussian Mixture model on the data as shown in class. Tune the covariance type parameter on the validation data. Use the selected value to compute the test accuracy. As always, train the model on train+validation data to compute the test accuracy. (10 mark)

In [41]:
```
from sklearn.mixture import GaussianMixture
from sklearn.metrics import accuracy_score

# Initialize Gaussian Naive Bayes
for cov_type in ('full', 'tied', 'diag', 'spherical'):
    gm = GaussianMixture(n_components=3,covariance_type = cov_type)
    # Train the classifier
    gm.fit(X_val, y_val)
    # Make predictions on test data
    y_pre = gm.predict(X_tes)
    accuracy = accuracy_score(y_pre, y_tes)
    print ('Test accuracy = ' + str(accuracy) + ' at covariance_tpye = ' + c
```

```
Test accuracy = 0.175438596491 at covariance_tpye = full
Test accuracy = 0.0614035087719 at covariance_tpye = tied
Test accuracy = 0.859649122807 at covariance_tpye = diag
Test accuracy = 0.179824561404 at covariance_tpye = spherical
```

In [43]:
```
#train the model on train+validation compute the test accuracy
gm = GaussianMixture(n_components=3,covariance_type = 'diag')
# Train the classifier
gm.fit(X_tra_val, y_tra_val)
# Make predictions on test data
y_pre = gm.predict(X_tes)
accuracy = accuracy_score(y_pre, y_tes)
print ('Test accuracy = ' + str(accuracy))
```

```
Test accuracy = 0.736842105263
```

Question 7: Apply Linear Discriminant Analysis model on the train+validation data and report the accuracy obtained on test data. Report the transformation matrix (w) along with the intercept. (5 mark)

In [45]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf = LinearDiscriminantAnalysis()
# Train
clf.fit(X_tra_val, y_tra_val)
# Test
y_pre = clf.predict(X_tes)

# print the accuracy
print ('Test accuracy = ' + str(np.sum(y_pre == y_tes)/len(y_tes)))
```

```
Test accuracy = 0.964912280702
```

In [46]:
```
clf.transform(X_tra_val)
```

Out[46]:
```
array([[ 2.26351958],
       [ 1.50315616],
       [-2.32016749],
       [ 1.16950535],
       [ 0.2848613 ],
       [-1.64129414],
       [-4.67284807],
       [ 1.07672183],
       [ 1.54651486],
       [ 0.64701382],
       [ 2.6571233 ],
       [ 2.6550835 ],
       [ 2.33859685],
       [-1.94301274],
       [-0.62044859],
       [ 1.56540432],
       [-2.78728741],
       [-1.88634026],
       [-3.01128946],
       [ 2.01215895]
```

Question 8: Load the digits dataset (scikit-learn's toy dataset) and take the last 1300 samples as your test set. Train a K-Nearest Neighbor (k=5, linf distance) model and then without using any scikit-learn method, report the nal values for Specicity, Sensitivity, TPR, TNR, FNR, FPR, Precision and Recall for Digit 3 (this digit is a positive, everything else is a negative). (15 marks)

In [2]:
```
from sklearn import datasets
digits= datasets.load_digits()
X = digits.data
y = digits.target
```

In [3]:
```
len(X)
```

Out[3]:
```
1797
```

In [4]:
```
#split dataset into train and test
X_tra = X[0:497]
X_tes = X[498:]
y_tra = y[0:497]
y_tes = y[498:]
```

In [5]:
```
#train the model in f classes
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(5)
clf.fit(X_tra, y_tra)
#predict test data
pred = clf.predict(X_tes)
```

In [6]:
```
#Calculate TPR, Sensitivity and Recall
TP = 0
FN = 0
for i in range(len(y_tes)):
    if ((y_tes[i] == 3) & (pred[i] == 3)):
        TP = TP + 1
    elif ((y_tes[i] == 3) & (pred[i] != 3)):
        FN = FN + 1
```

In [7]:
```
#Calculate TPR, Sensitivity and Recall
TPR = TP / (TP + FN)
TPR
```

Out[7]: 0.8769230769230769

In [9]:
```
#Calculate TNR, Specificity
TN = 0
FP = 0
for i in range(len(y_tes)):
    if ((y_tes[i] != 3) & (pred[i] != 3) & (y_tes[i] == pred[i])):
        TN = TN + 1
    elif ((y_tes[i] != 3) & (pred[i] == 3)):
        FP = FP + 1
```

In [10]:
```
#Calculate TNR, Specificity
TNR = TN / (TN + FP)
TNR
```

Out[10]: 0.9880294659300184

In [11]:
```
#Calculate FNR
FNR = FN / (TP + FN)
FNR
```

Out[11]: 0.12307692307692308

In [12]:
```
#Calculate FPR
FPR = FP / (FP + TN)
FPR
```

Out[12]: 0.011970534069981584

In [13]:
```
#Calculate precisions
Precisions = TP / (TP + FP)
Precisions
```

Out[13]: 0.8976377952755905

An ablation experiment consists of removing one feature from an experiment, in order to assess the amount of additional information that feature provides above and beyond the others. For this section, we will use the diabetes dataset from scikit-learn's toy datasets. Split the data into training and testing data as a 90-10 split with random state of 10.

In [58]:
```
from sklearn import datasets
diabetes= datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
```

In [59]:
```
X_tra, X_tes, y_tra, y_tes = train_test_split(X, y, test_size=0.1, random_st
```

Question 9: Perform least squares regression on this dataset. Report the mean squared error and the mean absolute error on the test data. (5 marks)

In [60]:
```
from sklearn.linear_model import LinearRegression
```

In [61]:
```
# Least squares regression
theta,residuals,rank,s = np.linalg.lstsq(X_tra, y_tra)
# Make predictions on the test data
predictions = np.dot(X_tes, theta)
```

In [14]:
```
from sklearn.linear_model import LinearRegression
lin = LinearRegression()
lin.fit(X_tra, y_tra)
predictions = lin.predict(X_tes)
```

```
/Users/xiasong/anaconda2/envs/py36/lib/python3.6/site-packages/scipy/lina
lg/basic.py:1018: RuntimeWarning: internal gelsd driver lwork query erro
r, required iwork dimension not returned. This is likely the result of LA
PACK bug 0038, fixed in LAPACK 3.2.2 (released July 21, 2010). Falling ba
ck to 'gelss' driver.
  warnings.warn(mesg, RuntimeWarning)
```

In [63]:
```
# Mean squared error calculation
from sklearn.metrics import mean_squared_error
print (mean_squared_error(y_tes, predictions))
```

```
2155.96465103
```

In [64]:
```
# Mean absolute error calculation
from sklearn.metrics import mean_absolute_error
print (mean_absolute_error(y_tes, predictions))
```

```
36.3181336987
```

Question 10: Repeat the experiment from Question 10 for all possible values of ablation (i.e., removing the feature 1 only, then removing the feature 2 only, and so on). Report all MSEs. (10 marks)

In [65]:
```
len(X_tra)
```

Out[65]: 397

In [66]:
```
a = np.delete(X_tra, np.s_[1],1)
len(a)
```

Out[66]: 397

In [67]:
```
for i in range(10):
    X_tra_rem = np.delete(X_tra, np.s_[i],1)
    X_tes_rem = np.delete(X_tes, np.s_[i],1)
    lin = LinearRegression()
    lin.fit(X_tra_rem, y_tra)
    predictions = lin.predict(X_tes_rem)
    a = mean_squared_error(y_tes, predictions)
    print ('MSE = ' + str(a) + ' when remove the %i feature' %i)
```

```
MSE = 2152.80664218 when remove the 0 feature
MSE = 2259.13307937 when remove the 1 feature
MSE = 2783.51448185 when remove the 2 feature
MSE = 2424.772348 when remove the 3 feature
MSE = 2187.59951938 when remove the 4 feature
MSE = 2167.51760615 when remove the 5 feature
MSE = 2159.15148251 when remove the 6 feature
MSE = 2153.06317113 when remove the 7 feature
MSE = 2335.17338461 when remove the 8 feature
MSE = 2165.86619219 when remove the 9 feature
```

Question 11: Based on the MSE values obtained from Question 11, which fea- tures do you deem the most/least signicant and why? (5 marks)

According to the results of questions 10, we can draw the conclusion that the second is the most significant and the 7th feature is the least significant feature. Actully, we can tell the importance of each feature from the ESMs change before and after removing these features. When we remove the 2nd feature, the magnititude change of MSE is the largest, and when we remove the 7th feature, the magnititude change of MSE is least. Therefore, the second feature is most significant feature as to this linear regression and the 7th feature is the least significant feature to this linear regression.

In [ ]: