ASSIGNMENT 3

Karan Mangla
201301205

Q1.

- The average Accuracy and Std dev across 10 runs is – Accuracy: 0.880533333333 , Std dev: 0.197
- **Handling Ties** - I choose the class with the higher Prior Probability. i.e., If P(Class1)>P(Class2), choose Class1. This makes sense since when the posterior probabilities are equal, given all the known attributes of the class, we can still predict it with only 0.5 probability. Hence, we rely on the domain expert to choose which class it belongs to.
- If there are any **missing entries**, I include them as another type of the same feature. I can't distinguish between different 'unknown' quantities, but I do know that they are not present for that record. So, treating them as a new category in a feature is a valid move.
  There could be a case, where the input feature vector has a value for a column that didn't occur in training data. Then we don't have it's probability at all. In that case we ignore that value, and move to the next column in X.

Implementation:

For each run, the whole data was randomly permuted and divided into two equal halves, training and test data. From training data, for each feature, probabilities of occurrence were calculated for each value that the feature can take. Separately, the probabilities of each output class actually occurring was also calculated .

Each record was taken in the test data, and $P(X \mid w_i)$ was calculated for each w_i . For this, I simply multiplied the probability of feature j given w_i . Then this was multiplied with $P(w_i)$ calculated earlier. Finally each value was divided by the sum of the values over i, since P(X) is the summation of the numerator over all i's.

Finally, the values $P(w_i \mid X)$ were compared, and the class with highest value assigned to the the input vector. If it was wrong, error variable was increased.

Code:

```
import csv,random
import numpy as np

Count = {}
Count1 = {}
```

```
def readData():
	with open('bank-full.csv', 'rb') as f:
		reader = csv.reader(f)
		elems = list(reader)
		UseFeatures = []
		for row_c in xrange(1,len(elems)):
			elems[row_c][0] = elems[row_c][0].split(";")
			#print elems[row_c][0]
			temp = []
			for index in xrange(0,17):
				if(index == 0 or index == 5 or index ==9 or index == 11 or
index == 12 or index == 13 or index == 14):
					continue
				#print index
				ele = elems[row_c][0][index]
				try:
					ele = int(ele)
				except Exception:
					ele = ele.replace('"','')
				temp.append(ele)
			#print temp
			UseFeatures.append(temp)
	return UseFeatures

def randomize(data):
	for i in xrange(10):
		random.shuffle(data)
	return data

def NBTrain(TrData):
	Count['yes'] = 0
	Count['no'] = 0
	Count1['yes'] = {}
	Count1['no'] = {}
	global Attr
	Attr = []

	for i in xrange(0,len(TrData[0])-1):
		temp = []
```

```
		for j in TrData:
			if j[i] not in temp:
				temp.append(j[i])
		Attr.append(temp)
	#print Attr
	#print len(Attr)
	for i in TrData:
		Count[i[-1]] += 1
		for j in xrange(0,len(i)-1):
			try:
				Count1[i[-1]][i[j]] += 1
			except :
				Count1[i[-1]][i[j]] = 1


def AccuVerify(TsData):
	temp1 = Count['yes'] / ( Count['yes'] + Count['no'] + 0.0)
	temp2 = Count['no'] / ( Count['yes'] + Count['no'] + 0.0)
	Tot = 0
	Corr = 0.0
	for i in TsData:
		Ans1 = (temp1 + 0.0)
		for j in xrange(len(i)-1):
			Sum = 0
			for k in Attr[j]:
				try:
					Sum += Count1['yes'][k]
				except Exception:
					Sum += 0.0
			Sum += 0.0
			try:
				Sum =  (Count1['yes'][i[j]])/ Sum
			except Exception:
				Sum =0
			Ans1 = Ans1 * Sum

		Ans2 = (temp2 + 0.0)
		for j in xrange(len(i)-1):
			Sum = 0
```

```
                for k in Attr[j]:
                    try:
                        Sum += Count1['no'][k]
                    except Exception:
                        Sum += 0.0
                Sum += 0.0
                try:
                    Sum =  (Count1['no'][i[j]])/ Sum
                except Exception:
                    Sum =0
                Ans2 = Ans2 * Sum
            if Ans1 > Ans2:
                #print "yes"
                if ('yes' == i[-1]):
                    Corr +=1

            else:
                #print "no"
                if ('no' == i[-1]):
                    Corr +=1
            Tot +=1
        return Corr/Tot


if __name__ == '__main__':
    data = readData()

    #print len(data)
    accuracies = []
    for i in xrange(0,10):
        data = randomize(data)
        TsData = data[0:len(data)/2]
        TrData = data[len(data)/2:len(data)]
        #print len(TsData), len(TrData)
        NBTrain(TrData)

        #print len(Attr)
        accuracies.append(AccuVerify(TsData))
    print accuracies
```

```
print np.mean(accuracies)
print np.std(accuracies)
```

Q2

Assignment 3

Karan Mangla
201301205

SM in AI

Q2. a) The univariate case

$p(x|\mu) \sim N(\mu, \sigma^2)$
$p(\mu) \sim N(\mu_0, \sigma_0^2)$

Case 1. $p(\mu|D)$

$D = \{ x_1, \dots, x_n \}$

$$p(\mu|D) = \frac{p(D|\mu) \cdot p(\mu)}{\int p(D|\mu) \cdot p(\mu) \cdot d\mu}$$

$$= \alpha \prod p(x_k|\mu) \cdot p(\mu)$$

normalisation factor

$$p(\mu|D) = \alpha \left[\frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{1}{2} \left(\frac{x_k - \mu}{\sigma} \right)^2 \right) \right] \left[\frac{1}{\sqrt{2\pi}\,\sigma_0} \exp \left[-\frac{1}{2} \left(\frac{\mu - \mu_0}{\sigma_0} \right)^2 \right] \right]$$

$$= \alpha'' \exp\left[-\frac{1}{2} \left[\left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \right) \mu^2 - 2 \left(\frac{1}{\sigma^2} \sum_{k=1}^{n} x_k + \frac{\mu_0}{\sigma_0^2} \right) \mu \right] \right]$$

Factors that don't depend on μ are absorbed

$\therefore$ $p(\mu|D)$ is an exp fn of a quadratic function of μ i.e, is a normal density.

If $p(\mu|D) \sim N(\mu_n, \sigma_n^2)$, we find μ_n, σ_n^2 from above equation

$p(\mu|D) = \frac{1}{\sqrt{2\pi}\,\sigma_n} e^{-\frac{1}{2}\left(\frac{\mu - \mu_n}{\sigma_n}\right.}$

$$\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}$$

$$\frac{\mu_n}{\sigma_n^2} = \frac{n}{\sigma^2} \underset{\text{sample mean}}{\hat{\mu_n}} + \frac{\mu_0}{\sigma_0^2}$$

$$\therefore \mu_n = \left(\frac{n\sigma_0^2}{n\sigma_0^2 + \sigma_0^2}\right)\hat{\mu_n} + \left(\frac{\sigma^2}{n\sigma_0^2+\sigma^2}\right)\mu_0$$

$$\sigma_n^2 = \frac{\sigma_0^2\sigma^2}{n\sigma_0^2+\sigma^2}$$

Case 2. $p(x|D) \sim N(\mu_n, \sigma^2+\sigma_n^2)$

$$p(x|D) = \int p(x|\mu)\cdot p(\mu|D)\cdot d\mu$$

$$= \int \frac{1}{\sqrt{2\pi}\sigma} e\left[\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \frac{1}{\sqrt{2\pi}\sigma_n} e\left[-\frac{1}{2}\left(\frac{\mu-\mu_n}{\sigma_n}\right)^2\right] d\mu$$

$$= \frac{1}{2\pi\sigma\sigma_n} e\left[-\frac{1}{2}\frac{(x-\mu_n)^2}{\sigma^2+\sigma_n^2}\right] f(\sigma,\sigma_n)$$

Here

$$f(\sigma,\sigma_n) = \int e\left[-\frac{1}{2}\left(\frac{\sigma^2+\sigma_n^2}{\sigma^2\sigma_n^2}\right)\left(\mu - \frac{\sigma_n^2 x + \sigma^2\mu_n}{\sigma^2+\sigma^2 n}\right)^2\right] d\mu$$

b) Multivariate case: $p(x|\mu) \sim N(\mu, \Sigma)$
$p(\mu) \sim N(\mu_0, \Sigma_0)$.

$D = \{x_1, \ldots x_n\}$
$\downarrow$
independent samples

$-$ $p(\mu | D) = \alpha \prod_{k=1}^{n} p(x_k | \mu) \cdot p(\mu)$.

$= \alpha'' \cdot e\left[-\frac{1}{2}(\mu - \mu_n)' \Sigma_n^{-1} (\mu - \mu_n)\right]$

$-$ $p(\mu | D) \sim N(\mu_n, \Sigma_n)$

$\rightarrow$ $\hat{\mu}_n = \frac{1}{n} \sum_{k=1}^{n} x_k$.
$\downarrow$
sample mean

$\rightarrow$ $\Sigma_n^{-1} = n \Sigma^{-1} \hat{\mu}_n + \Sigma_0^{-1} \mu_0$. $\swarrow$ solution

$\therefore$ $\Sigma_n = \Sigma_0 \left(\Sigma_0 + \frac{1}{n} \Sigma \right)^{-1} \frac{1}{n} \Sigma$

[using matrix identity
$(A^{-1} + B^{-1})^{-1}$
$= A(A+B)^{-1}B$.

x: sum of two mutually indep. random variables
with $p(\mu | D) \sim N(\mu_n, \Sigma_n)$.
$p(x | D) \sim N(\mu_n, \Sigma + \Sigma_n)$.

—— x ——

Q3.

PCA Principle Component Analysis is method of dimensionality reduction before applying any classifier on the data. This is useful when we have dimensions or features in the order of 1000s. For PCA, our training data set has n (100) data points each with d (10000) dimensions. First we calculate the covariance matrix for the data set.
Then, for the dXd matrix formed, we find the eigenvectors and eigenvalues. Then we sort the eigenvalues in descending order, and take the top k corresponding eigen vectors where k is 10, 100 and 1000. Then we multiply the nXd training matrix and dXk matrix of top k eigenvectors. Finally we have an nXk projection of the original nXd data matrix. As we can see, k<<d. Same is done for the test data set. Now we find the mean and variance over each of the k features, column wise for each class separately.

LDA Linear Discriminant Analysis is also a dimensionality reduction algorithm which reduces the d dimensional data to 1 dimension. First we compute the means of all the features over all data points for each class separately .
We have to maximize the between class separation and minimize the within class scatter. To achieve this, take the eigenvector corresponding to the maximum eigenvalue of the matrix [inverse(SW)*SB] .

Code :

```
%Assignment 3 SMAI
clear all;
clc;
%Reading the training data
traindata = importdata('arcene_train.data');
trainlabels = importdata('arcene_train.labels');

%Reading the validation data
Validdata = importdata('arcene_valid.data');
Validlabels = importdata('arcene_valid.labels');
traindata = [traindata;Validdata];

%Calculating PCA
```

```
mew = mean(traindata);          % Mean of the data
temp = traindata-repmat(mew,size(traindata,1),1);   % X-M
S = temp'*temp;         %Scatter Matrix = Sigma (xk-m)*(xk-m)'
S = S/size(traindata,1);

%[V D]=eig(S);          %Eigen Vector V & Eigen Value D of Scatter Matrix
V = load('eien_values.mat');
d = load('eigen_vectors.mat');

dl = flipud(D);
vl = fliplr(V);
k=10;
vl1 = vl(:,1:k);
new_matrix = traindata*vl1;
train = new_matrix(1:100,:);
testing = new_matrix(101:200,:);
class1 = [];
class2 = [];
for i=1:100
   if trainlabels(i) == -1
      class1 = [class1;train(i,:)];
   else
      class2 = [class2;train(i,:)];
   end
end
mu1 = mean(class1);
cov1 = cov(class1);
pw1 = size(class1,1)/size(train,1);
mew2 = mean(class2);
cov2 = cov(class2);
pw2 = size(class2,1)/size(train,1);
count = 0;
for i=1:100
   p1 = -0.5*log(det(cov1)) - 0.5*((testing(i,:)-mu1) * inv(cov1) * (testing(i,:)-mu1)') +
log(pw1);
   p2 = -0.5*log(det(cov2)) - 0.5*((testing(i,:)-mew2) * inv(cov2) * (testing(i,:)-mew2)') +
log(pw2);
   if p1 > p2
      if Validlabels(i) ~= -1
```

```
            count = count + 1;
        end
    end
    if p2 > p1
        if Validlabels(i) ~= 1
            count = count + 1;
        end
    end
end
count
```

%Computing LDA

```
class_1 = find(trainlabels == 1);
class_2 = find(trainlabels == -1);

m1 = mean(traindata(class_1,:),1);
m2 = mean(traindata(class_2,:),1);

new = traindata(class_1,:)-repmat(m1,size(class_1,1),1);  %X-M1
S1 = new'*new;  %S1 Matrix

new = traindata(class_2,:)-repmat(m2,size(class_2,1),1); %X-M2
S2 = new'*new;  %S2 Matrix

SW = S1 + S2;   %Within class scatter
%SW = SW\eye(size(SW));

w = inv(SW)*(m1-m2)';

Y1 = traindata(class_1,:)*w;    %Final 1D LDF
Y2 = traindata(class_2,:)*w;
```

Handling Ties Again ties have been handled as in question 1. If the P(Wi) is the same for both classes, then we have assigned the class that has the higher prior probability. In case that also comes out to same, we arbitrarily assign class 1 (that is label ‘1’) to the input vector and calculate error.
We have to maximize the between class separation and minimize the within class scatter. To achieve this, take the eigenvector corresponding to the maximum eigenvalue of the matrix [inverse(SW)*SB].

Observations :

- Best value of k found ~= 20
- After performing the LDA, each class has one coefficient associated to describe it.
- Mean accuracy is low because of random features added to dataset, as mentioned in the file for the dataset. This leads to random features hindering the classifier, in both cases.
- If the matrix is singular, inverse can’t be calculated so in that case we calculate the pseudo inverse of the matrix, in case of LDA.
- One alternative way could be perform PCA first and on top of that perform Fisher’s LDA. So, the training data which was earlier 100x10000 becomes 100xk after PCA (where say k =100). This reduces the runtime a lot. The resulting Sw matrix which is S1+S2 is of dimensions 100x100. Calculating the inverse of this matrix is much easier.
- Gaussian classifier’s basis is the assumption that the features follow a gaussian distribution, which is wrong, because of the random features added