Eksamen - MAD - 2021/2022

Eksamensnummer: 30

24. januar 2022

Indhold

Question 1

We start by rewriting $\hat{\mu}$

$$\begin{aligned}\hat{\mu} &= log(\sqrt[n]{x_1 \cdot x_2 \cdot \ldots \cdot x_n} \\ &= \log((x_1 \cdot x_2 \cdot \ldots \cdot x_n)^{\frac{1}{n}}) \\ &= \frac{1}{n}\log(x_1 \cdot x_2 \cdot \ldots \cdot x_n) \\ &= \frac{\log(x_1 \cdot x_2 \cdot \ldots \cdot x_n)}{n}\end{aligned}$$

Since the MLE for both μ and σ is the same as MLE for a normal distribution of the dataset $\ln(x_1), \ln(x_2), \ldots, \ln(x_n)$, we can write the maximum likelihood estimator μ as

$$\hat{\mu} = \frac{\sum_{i=1}^{n} \ln(x_i)}{n}$$

Since $\log(a) + \log(b) = \log(ab)$, we can write

$$\frac{log(x_1 \cdot x_2 \cdot \ldots x_n)}{n} = \frac{\sum_{i=1}^{n} \log(x_i)}{n}$$

and since log is the natural logarithm, we can write

$$\hat{\mu} = \frac{\sum_{i=1}^{n}(x_i)}{n} = \log(\sqrt[n]{x_1 \cdot x_2 \cdot \ldots \cdot x_n})$$

which proves the original claim.

Question 2

a)

We have a joint probability, and we want to compute this. We can do this by the following formula.

$$P_{joint}(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n}(x_i)$$

We can then write the likelihood as

$$P_{joint}(x_1, x_2) = \left(\frac{2}{\beta^2} \cdot (\beta - x_1)\right) \cdot \left(\frac{2}{\beta^2} \cdot (\beta - x_2)\right)$$

In order to compute the maximum likelihood estimator, we have to first find $f'(\beta)$, then solve the equation $f'(\beta) = 0$, then find $f''(\beta)$ and compute $f''(\beta)$ for each β value found, when solving $f'(\beta) = 0$.
We start by inserting the values of x_1 and x_2, and simplify the function.

$$P_{joint}(3, 4) = \left(\frac{2}{\beta^2} \cdot (\beta - 3)\right) \cdot \left(\frac{2}{\beta^2} \cdot (\beta - 4)\right) = \frac{4}{\beta^2} - \frac{28}{\beta^3} - \frac{48}{\beta^4}$$

We then find the derivative of the function

$$P'(\beta) = \frac{-8}{\beta^3} + \frac{84}{\beta^4} - \frac{192}{\beta^5}$$

Now we can solve the equation

$$\frac{-8}{\beta^3} + \frac{84}{\beta^4} - \frac{192}{\beta^5} = 0$$

which we do in maple, giving

$$\beta = \frac{21 + \sqrt{57}}{4} \approx 7.14 \quad \vee \quad \beta = \frac{21 - \sqrt{57}}{4} \approx 3.36$$

Since $3.36 < 4$ this cannot be the value, and we get

$$\beta = \frac{21 + \sqrt{57}}{4} \approx 7.14$$

We can now plugs this value into $f(\beta)$ and compute the maximum likelihood estimator for $x_1 = 3$ and $x_2 = 4$.

$$P(3) = \frac{2}{7.14^2} \cdot (7.14 - 3) = 0.162$$

$$p(4) = \frac{2}{7.14^2} \cdot (7.14 - 4) = 0.123$$

b)

I use the 6 steps from the essentials.

1. We first find the model, which is

   $$X \sim \text{Bin}(n, \theta)$$

2. The hypothesis is given as

   $$H_0 : \theta = 0.5 \text{ and } H_1 : \theta \neq 0.5$$

3. The distribution under H_0 is

   $$\text{Bin}(20, 0.5) = \frac{20!}{x!(20-x)!} 0.5^x (1 - 0.5)^{20-x}$$

   To find the probability of getting 13 or more correct guesses, we sum the probabilities of 13 to 20 correct guesses and times it by 2, since we are doing a two-sided test.

   $$2 \cdot \text{Bin}(20, 0.5, 13) = 2 \cdot \sum_{i=13}^{20} \left(\frac{20!}{i!(20-i)!} 0.5^i (1 - 0.5)^{20-i} \right)$$

4. The significance leve is given as

   $$\alpha = 0.05$$

5. We calculate the rejection region, by using the formula from step 3, we have to try all values, to see, when the chance of it happening, is smaller than 5%. This gives us the rejection region:

$$\mathcal{R} = \{0, \ldots, 6\} \cup \{15, \ldots, 20\}$$

6. We compute the test result, which is

$$2 \cdot \text{Bin}(20, 0.5, 13) = 0.26318$$

this is not included in the rejection region, since $26\% > 5\%$ and the nul-hypothesis holds.

Question 3

Points								
coordinate x	0.6	0.5	1.1	-0.5	0.8	0.2	-0.1	1
coordinate y	1.1	1	2	0.2	-0.1	-0.1	-1.5	2.5

Given the above table, we calculate the mean of X and Y.

$$\text{Mean}_X = \frac{0.6 + 0.5 + 1.1 - 0.5 + 0.8 + 0.2 - 0.1 + 1}{8} = 0.45$$
$$\text{Mean}_Y = \frac{1.1 + 1 + 2 + 0.2 - 0.1 - 0.1 - 1.5 + 2.5}{8} = 0.6375$$

and the mean point is $(0.45, 0.6375)$

We then calculate the values for the normalized data.

Points								
coordinate x	0.15	0.05	0.65	-0.95	0.35	-0.15	-0.55	0.55
coordinate y	0.46	0.36	1.36	0.44	-0.74	-0.74	-2.14	1.86

We can now construct a covariance matrix by the given formula

$$\text{cov}(r) = \frac{1}{n} \sum_{i=1}^{n} p \cdot p^T$$

and we get the covariance matrix

$$\begin{bmatrix} \text{cov}(x,x) & \text{cov}(x,y) \\ \text{cov}(y,x) & \text{cov}(y,y) \end{bmatrix} = \begin{bmatrix} 0.2675 & 0.4394 \\ 0.4394 & 1.4398 \end{bmatrix}$$

We use the equation $\det(\Sigma - \lambda I) = 0$, to find the eigenvalues. The matrix looks like this.

$$\det \begin{bmatrix} 0.2675 - \lambda & 0.4394 \\ 0.4394 & 1.4398 - \lambda \end{bmatrix}$$

Computing this, we get the eigenvalues to be

$$\lambda_1 = 0.1211, \quad \lambda_2 = 1.5863$$

We use Gauss-Jordan Elimination on the matrix to get the eigenvectors, which gives

$$e_1 = \begin{bmatrix} -3.0014 \\ 1 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0.3318 \\ 1 \end{bmatrix}$$

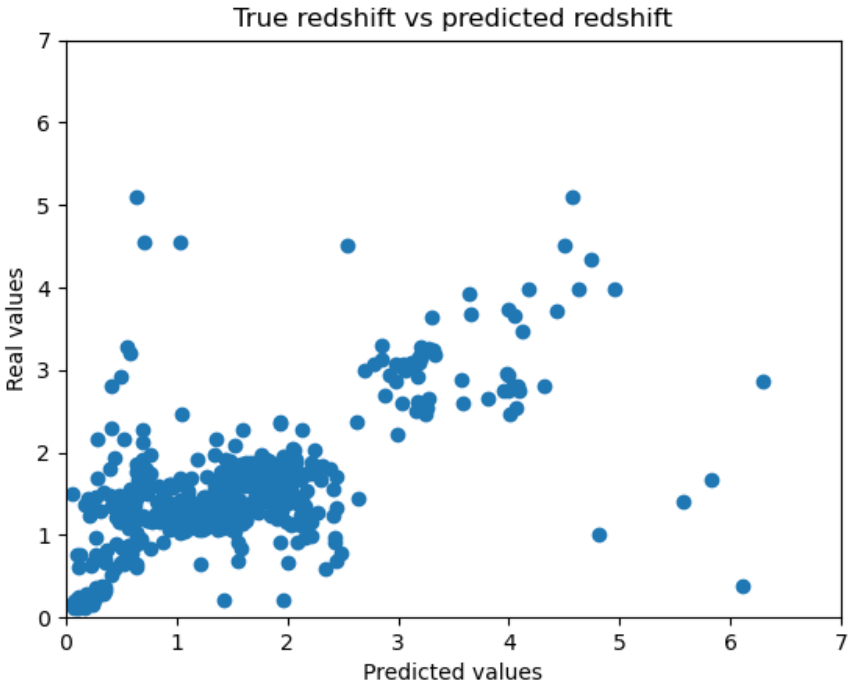
Question 4

1)

a)

The RMSE value is:

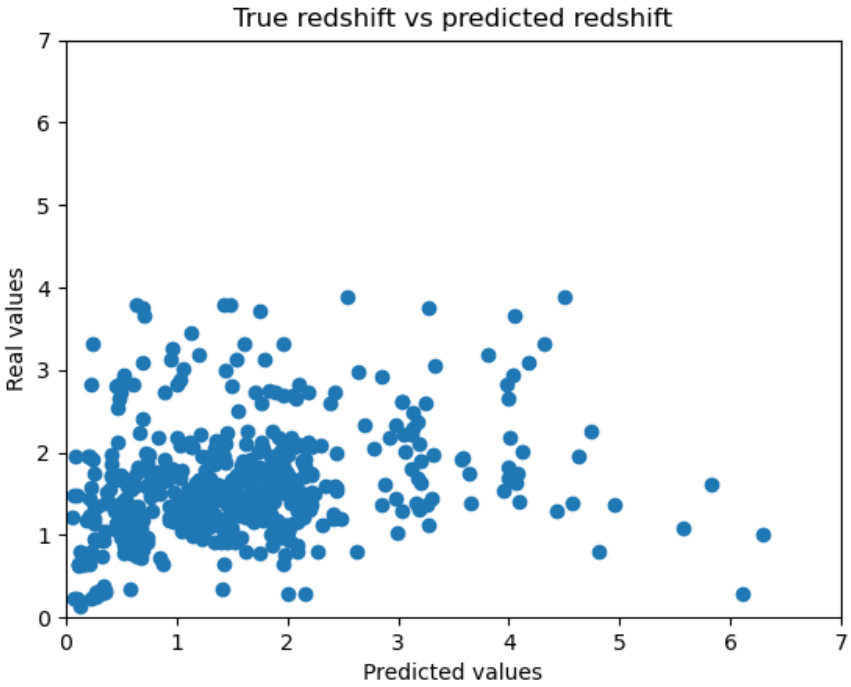
$$RMSE = 0.8243$$



b)

The RMSE value is:

$$RMSE = 1.0998$$

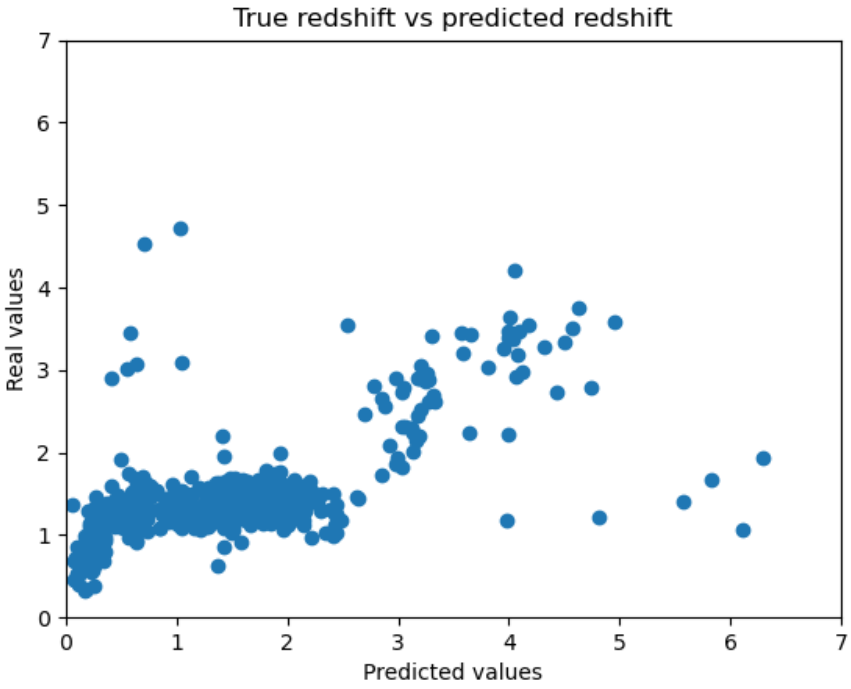


M makes us able to give each of the features a weight. This makes us able to make some features more important than others for the function. This is nice, since other functions are able to check if some features have a higher impact on the label, than others.

2)

The RMSE value is:

$$RMSE = 0.8249016977737486$$



Question 5

a)

To calculate the entropy gain we start by computing $H(t)$ of the initial set. The formula looks like this

$$H(t) = -\frac{\text{pos}}{\text{total}} \log_2 \left(\frac{\text{pos}}{\text{total}} \right) - \frac{\text{neg}}{\text{total}} \log_2 \left(\frac{\text{neg}}{\text{total}} \right)$$

When we then split the sets, by making the first branches of the tree, we calculate the $H(t)$ value of these new subsets. The gain by the given split is then computed by the following formula

$$\text{gain} = H(t) - H(s_1) - H(s_2)$$

where t is the original set and s_1 and s_2 are the subsets. To find the optimal splits of the set, we compute the gain of all possible splits, and select the split with the highest gain.

b)

Points												
feature	10	20	30	40	50	60	70	80	90	100	110	120
label	0	0	1	1	1	1	0	0	0	1	0	0

To get identify the potentiel thresholds, we only look at threshold where they have different class neighbors. These thresholds being:

$$\text{feature} < 25$$
$$\text{feature} < 65$$
$$\text{feature} < 95$$
$$\text{feature} < 105$$

c)

We calculate the entropy gain for the 4 thresholds, and select the threshold with the highest gain, as the optimal threshold.

$$H(T) = -\frac{5}{12}\log_2\left(\frac{5}{12}\right) - \frac{7}{12}\log_2\left(\frac{7}{12}\right) = 0.97987$$

Threshold at 25

$$\begin{aligned} H(S_{feature>25}) &= \frac{10}{12}\cdot\left(-\frac{5}{10}\cdot\log_2\left(\frac{5}{10}\right) - \frac{5}{10}\cdot\log_2\left(\frac{5}{10}\right)\right) \\ &= \frac{5}{6} \\ H(S_{feature<25}) &= -\frac{2}{12}\cdot\left(-\frac{2}{2}\cdot\log_2\left(\frac{2}{2}\right)\right) \\ &= 0 \\ \text{Gain} &= H(T) - H(S_{feature>25}) - H(S_{feature<25}) \\ &= 0.14654 \end{aligned}$$

Threshold at 65

$$\begin{aligned} H(S_{feature>65}) &= \frac{6}{12}\cdot\left(-\frac{4}{6}\cdot\log_2\left(\frac{4}{6}\right) - \frac{2}{6}\cdot\log_2\left(\frac{2}{6}\right)\right) \\ &= 0.45915 \\ H(S_{feature<65}) &= \frac{6}{12}\cdot\left(-\frac{1}{6}\cdot\log_2\left(\frac{1}{6}\right) - \frac{5}{6}\cdot\log_2\left(\frac{5}{6}\right)\right) \\ &= 0.32501 \\ \text{Gain} &= H(T) - H(S_{feature>65}) - H(S_{feature<65}) \\ &= 0.19571 \end{aligned}$$

Threshold at 95

$$\begin{aligned} H(S_{feature>95}) &= \frac{3}{12} \cdot \left(-\frac{1}{3} \cdot \log_2 \left(\frac{1}{3} \right) - \frac{2}{3} \cdot \log_2 \left(\frac{2}{3} \right) \right) \\ &= 0.22957 \\ H(S_{feature<95}) &= \frac{9}{12} \cdot \left(-\frac{4}{9} \cdot \log_2 \left(\frac{4}{9} \right) - \frac{5}{9} \cdot \log_2 \left(\frac{5}{9} \right) \right) \\ &= 0.74331 \\ \text{Gain} &= H(T) - H(S_{feature>95}) - H(S_{feature<95}) \\ &= 0.00699 \end{aligned}$$

Threshold at 105

$$\begin{aligned} H(S_{feature>105}) &= \frac{2}{12} \cdot \left(-\frac{2}{2} \cdot \log_2 \left(\frac{2}{2} \right) \right) \\ &= 0 \\ H(S_{feature<105}) &= \frac{10}{12} \cdot \left(-\frac{5}{10} \cdot \log_2 \left(\frac{5}{10} \right) - \frac{5}{10} \cdot \log_2 \left(\frac{5}{10} \right) \right) \\ &= \frac{5}{6} \\ \text{Gain} &= H(T) - H(S_{feature>105}) - H(S_{feature<105}) \\ &= 0.14654 \end{aligned}$$

Since feature < 65 has the highest gain, this is the optimal threshold.

Question 6

a)

The code is provided below. We use the RandomForestClassifier from sklearn.ensemble, to make the random forest. We then train the forest with the our training data, and return the predictor.

```
def __randomForests(X, t):
    predictor = RandomForestClassifier()
    predictor.fit(X, t)
    return predictor

predictor = __randomForests(X, t)

accuracyTrain = predictor.score(X, t)
print("accuracy a: ", accuracyTrain * 100, "%")
```

We calculate the accuracy of our predictor on the training set, by using the function score. This takes the features and labels as arguments, and returns how many it got right in procent. The accuracy of out predictor trained with the training data, and the run on the training data is 100%.

b)

To find the optimal values, we started by making a new function, which returns the accuracy of the prediction, and the predicted probability of each class. The predicted probability matrix is then run through to get the correct probabilities, which we then take an average of. The two function are provided below.

```
def randomForests(X, t, XVal, tVal, criterion, depth, samples):
    predictor = RandomForestClassifier(criterion = criterion,
                        max_depth = depth, max_features = samples)
    predictor.fit(X, t)
    accuracy = predictor.score(XVal, tVal)
    pred_proba = predictor.predict_proba(XVal)
    return accuracy, pred_proba

def optimalValues():
    optimalParameters = []
    optimalGuess = [0, 0]
    for criterion in ["entropy", "gini"]:
        for depth in [2, 5, 7, 10, 15]:
            for feature in ["sqrt", "log2"]:
                accuracy, pred_proba_init = randomForests(X, t,
                            XVal, tVal, criterion, depth, feature)
                accuracy = accuracy * XVal.shape[0]
                pred_proba = []
                for i in range(pred_proba_init.shape[0]):
                    temp = pred_proba_init[i]
                    pred_proba.append(temp[int(tVal[i] + 0.0005)])
                pred_proba = sum(pred_proba) / len(pred_proba)
                if (accuracy > optimalGuess[0] or (accuracy == optimalGuess[0]
                                    and pred_proba > optimalGuess[1])):
                    optimalParameters = [criterion, depth, feature]
                    optimalGuess = [accuracy, pred_proba]
                    print("Improvement. New values:\n")
                    print("Criterion = ", criterion)
                    print("max_depth = ", depth)
                    print("feature = ", feature)
                    print("average probability = ", pred_proba * 100, "%")
                    print("num of correct guesses = ", int(accuracy + 0.0005))
                else:
                    print("De nye værdier er ikke bedre, beholder de gamle\n")
    return optimalParameters, optimalGuess

optimalParameters, optimalGuess = optimalValues()
print("De optimale parametre er: ", optimalParameters)
print("Korrekt gættet: ", int(optimalGuess[0] + 0.0005))
```

```
print("Gennemsnitlig sandsynlighed: ", optimalGuess[1] * 100, "%")
```

We run the function and prints the optimal parameters and what number of correct guesses and the predicted probability, these parameters corresponces to.

c)

Here is a list with all the improvements, we have found, when running the code:

```
Improvement. New values:
Criterion =  entropy
max_depth =  2
feature =  sqrt
average probability =  37.291726835561214 %
num of correct guesses =  39

Improvement. New values:
Criterion =  entropy
max_depth =  2
feature =  log2
average probability =  37.41706219684989 %
num of correct guesses =  41

Improvement. New values:
Criterion =  entropy
max_depth =  5
feature =  sqrt
average probability =  52.09841362120895 %
num of correct guesses =  59

Improvement. New values:
Criterion =  entropy
max_depth =  7
feature =  sqrt
average probability =  56.33927872817577 %
num of correct guesses =  61

Improvement. New values:
Criterion =  entropy
max_depth =  15
feature =  log2
average probability =  57.3116883116883 %
num of correct guesses =  62
```

Running the code again, can change this result due to the random nature of randomForest. Running this multiple times, we can see, that we often get the best result, at a depth of 7 or 10, suggesting, we might begin to overfit, at a depth of 15. We see way more entropy criterion, when

running the code, this does not mean that it is much better however, since we start by trying with all the values on entropy.

Question 7

a)

I have implemented the following function, to normalize the data.

```
def normalize(samples):
    means = samples.mean(axis = 0)
    deviations = []
    norm = np.zeros(samples.shape)

    for c in samples.T:
        sampleStandardDeviation = statistics.stdev(c)
        deviations.append(sampleStandardDeviation)

    for i in range(samples.shape[0]):
        for j in range(samples.shape[1]):
            norm[i, j] = (samples[i, j] - means[j]) / deviations[j]
    return norm
```

b)

We have made a class, to do the Kmeans search. This class contains methods to calculate the mean of a centroid and the samples assigned to it. Then it contains a method, to calculate the KMeans, and a method to calculate the optimal clusters, with a specified number of iterations.

```
class KMean():
    def __init__(self, sample, nClusters = 5, nIterations = 5):
        self.samples = sample
        self.nClusters = nClusters
        self.nIterations = nIterations

    def normalize(self):
        means = self.samples.mean(axis = 0)
        deviations = []
        norm = np.zeros(self.samples.shape)

        for c in self.samples.T:
            sampleStandardDeviation = statistics.stdev(c)
            deviations.append(sampleStandardDeviation)

        for i in range(self.samples.shape[0]):
            for j in range(self.samples.shape[1]):
                norm[i, j] = (self.samples[i, j] - means[j]) / deviations[j]
```

```
        self.normalizedData = norm

    def calculateMean(self, centroid, assignedTo):
        temp = []
        for j in range(self.normalizedData.shape[0]):
            if assignedTo[j] == centroid:
                temp.append(self.normalizedData[j])
        return sum(temp) / len(temp)

    def kMeans(self):
        centroids = np.zeros((self.nClusters, self.normalizedData.shape[1]))
        for i in range(self.nClusters):
            randomRow = np.random.randint(self.normalizedData.shape[0] - 1)
            centroids[i] = self.normalizedData[randomRow]
        assigntedToComp = np.zeros(self.normalizedData.shape[0])
        assigntedTo = []
        while(1):
            for i in range(self.normalizedData.shape[0]):
                distances = np.zeros(self.nClusters)
                for j in range(self.nClusters):
                    distances[j] = (np.linalg.norm(self.normalizedData[i]-centroids[j]))
                closest = np.argmin(distances)
                assigntedTo.append(closest)
            if np.array_equal(assigntedToComp, assigntedTo):
                samplesInClusters = []
                for i in range(self.nClusters):
                    samplesInClusters.append(assigntedTo.count(i))
                return centroids, np.array(assigntedTo), np.array(samplesInClusters)
            for centroid in range(self.nClusters):
                centroids[centroid] = self.calculateMean(centroid, assigntedTo)
            assigntedToComp = assigntedTo
            assigntedTo = []

    def KMeansIntra(self):
        optimalDist = sys.maxsize
        optimalClusters = np.zeros((self.nClusters, self.normalizedData.shape[1]))
        optimalAssignedTo = []
        for i in range(self.nIterations):
            centroids, assignedTo, samplesInClusters = self.kMeans()
            clusterDist = 0
            for i in range(1):
                for j in range(self.normalizedData.shape[0]):
                    if assignedTo[i] == assignedTo[j]:
                        clusterDist = clusterDist + np.linalg.norm(
                                self.normalizedData[i]-self.normalizedData[j])
```

```
            avgIntraClusterDist = clusterDist / (self.normalizedData.shape[0])
            if avgIntraClusterDist < optimalDist:
                optimalDist = avgIntraClusterDist
                optimalClusters[:] = centroids
                optimalAssignedTo = assignedTo
        self.optimalClusters = optimalClusters
        self.optimalAssignedTo = optimalAssignedTo
        self.optimalDistance = optimalDist
        self.samplesInClusters = samplesInClusters
```

c)

We ran the code with the below code.

```
Kmeans = KMean(sample = X, nClusters = 3)
Kmeans.normalize()
Kmeans.KMeansIntra()

print("Samples in each cluster: ", Kmeans.samplesInClusters)
```

The number of samples we found in each cluster running the code from b.

$$\begin{bmatrix} 70 & 67 & 63 \end{bmatrix}$$

Running the code again, could change the result of this, since clusters are created randomly.

d)

We have used the implementation of the PCA function from assignment 5.

```
def __PCA(data):
    meanTrainingFeatures = np.mean(data.T, axis = 1)

    data_cent = data.T - meanTrainingFeatures.reshape((-1, 1))
    data_cent = np.cov(data_cent)
    PCevals, PCevecs = np.linalg.eigh(data_cent)
    PCevals = np.flip(PCevals, 0)
    PCevecs = np.flip(PCevecs, 1)
    return PCevals, PCevecs
```

e)

We have used the implementation of the transformData function from assignment 5.

```
def __transformData(features, PCevecs):
    return np.dot(features,  PCevecs[:, 0:2])
```

Plotting the graph, using the visualizeLabels function from assignment 5, modified a bit, to make it able to plot the centroids aswell.

```
def __visualizeLabels(features, referenceLabels, centroids):
    plt.figure()
    cmap_bold  = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
    y = referenceLabels

    plt.scatter(features[:, 0], features[:, 1], c = y, cmap = cmap_bold, s = 10)
    # added scatter for centroids.
    plt.scatter(centroids[:, 0], centroids[:, 1], color="black", s = 100)
    plt.show()
```

The plot of the points and the centroids