

Exam

Modelling and Analysis of Data Department of Computer Science University of Copenhagen

Exam nr. 87

January 22, 2023

1 Maximum Likelihood Estimation

We have the following pdf:

$$p_{\theta} = \frac{\theta}{x^2} \cdot \exp\left(-\frac{\theta}{x}\right)$$

The Likelihood-function can then be written as:

$$\mathbf{L}(\theta) = \prod_{n=1}^N \frac{\theta}{x_n^2} \cdot \exp\left(-\frac{\theta}{x_n}\right)$$

Since we want to maximize for θ , we'll take the logarithm of the Likelihood-function and differentiate with respect to θ :

$$l(\theta) = \ln(\mathbf{L}(\theta)) = \sum_{n=1}^N \ln(\theta) - 2\ln(x_n) - \frac{\theta}{x_n}$$

$$\frac{\partial l(\theta)}{\partial \theta} = \sum_{n=1}^N \left(\frac{1}{\theta} - \frac{1}{x_n}\right)$$

Now that we have the derivative of the Likelihood-function, we'll set it equal to zero and solve for θ

$$\sum_{n=1}^N \left(\frac{1}{\theta} - \frac{1}{x_n}\right) = 0$$

By summation rules, we can split the difference within the summation into two separate summations

$$\sum_{n=1}^N \left(\frac{1}{\theta}\right) - \sum_{n=1}^N \left(\frac{1}{x_n}\right) = 0$$

Moving $\sum_{n=1}^N \left(\frac{1}{x_n}\right)$ to the right side

$$\sum_{n=1}^N \left(\frac{1}{\theta}\right) = \sum_{n=1}^N \left(\frac{1}{x_n}\right)$$

Since we have $\frac{1}{\theta}$ N -times $\rightarrow \frac{1}{\theta} \cdot N = \frac{N}{\theta}$

$$\frac{N}{\theta} = \sum_{n=1}^N \left(\frac{1}{x_n} \right)$$

Multiplying θ on both sides

$$N = \theta \cdot \sum_{n=1}^N \left(\frac{1}{x_n} \right)$$

Dividing $\sum_{n=1}^N \left(\frac{1}{x_n} \right)$ on both sides gives the maximum likelihood estimator $\hat{\theta}$

$$\begin{aligned} \frac{N}{\sum_{n=1}^N \left(\frac{1}{x_n} \right)} &= \theta \\ \rightarrow \hat{\theta} &= \frac{N}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_N}} \end{aligned}$$

To prove that this is indeed a global maximum, we start by finding the second derivative with respect to θ i.e. $\frac{\partial^2 l(\theta)}{\partial \theta \partial \theta}$

From earlier we have that:

$$\frac{\partial l(\theta)}{\partial \theta} = \sum_{n=1}^N \left(\frac{1}{\theta} - \frac{1}{x_n} \right)$$

So the second derivative is:

$$\begin{aligned} \frac{\partial^2 l(\theta)}{\partial \theta \partial \theta} &= \sum_{n=1}^N \left(-\frac{1}{\theta^2} \right) \\ &\rightarrow -\frac{N}{\theta^2} \end{aligned}$$

and is always less than zero, which means that:

$$\rightarrow \hat{\theta} = \frac{N}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_N}}$$

is a global maximum

2 Statistics

a)

We have the following pdf, $f_{\beta}(x)$

$$f_{\beta}(x) = \begin{cases} \frac{2}{\beta^2} \cdot (\beta - x) & \text{if } 0 \leq x \leq \beta \\ 0 & \text{otherwise} \end{cases}$$

The likelihood of two given observations is the product of their individual pdf:

$$\mathbf{L}(\beta; x_1; x_2) = f_\beta(x_1) \cdot f_\beta(x_2) = \frac{2}{\beta^2} \cdot (\beta - x_1) \cdot \frac{2}{\beta^2} \cdot (\beta - x_2)$$

The likelihood-function with parameters β , x_1 and x_2 is:

$$\mathbf{L}(\beta; x_1; x_2) = \frac{4(\beta - x_1)(\beta - x_2)}{\beta^4}$$

Since we want to find the maximum likelihood estimator, when $x_1 = 3$ and $x_2 = 4$, we will first take the logarithm of the Likelihood-function, and then differentiate it with respect to β

$$l(\beta) = \ln(L(\beta)) = \ln\left(\frac{4(\beta - 3)(\beta - 4)}{\beta^4}\right)$$

Taking the logarithm of a fraction $\ln\left(\frac{a}{b}\right)$ is the same as the difference between the logarithm of the numerator and the logarithm of the denominator $\ln(a) - \ln(b)$

$$= \ln(4(\beta - 3)(\beta - 4)) - \ln(\beta^4)$$

The exponent of β^4 goes out of the logarithm

$$= \ln(4(\beta - 3)(\beta - 4)) - 4\ln(\beta)$$

Now we will find the derivative, we know that $(f - g)' = f' - g'$

$$\frac{\partial l(\beta)}{\partial \beta}(\ln(4(\beta - 3)(\beta - 4))) - \frac{\partial l(\beta)}{\partial \beta}(4\ln(\beta))$$

For the leftside derivative we can use the chain-rule

$$\frac{1}{4(\beta - 3)(\beta - 4)} \cdot \frac{\partial l(\beta)}{\partial \beta}(4(\beta - 3)(\beta - 4))$$

We can take out a constant factor

$$\frac{4 \frac{\partial l(\beta)}{\partial \beta}((\beta - 3)(\beta - 4))}{4(\beta - 3)(\beta - 4)}$$

Cancelling the 4's and using the product-rule for the numerator

$$\frac{\frac{\partial l(\beta)}{\partial \beta}(\beta - 3)(\beta - 4) + \frac{\partial l(\beta)}{\partial \beta}(\beta - 4)(\beta - 3)}{(\beta - 3)(\beta - 4)}$$

The derivative of a difference is equal to the product of the derivative of each variable

$$\frac{[\frac{\partial l(\beta)}{\partial \beta}(\beta) \frac{\partial l(\beta)}{\partial \beta}(-3)](\beta - 4) + [\frac{\partial l(\beta)}{\partial \beta}(\beta) \frac{\partial l(\beta)}{\partial \beta}(-4)](\beta - 3)}{(\beta - 3)(\beta - 4)}$$

The parts in the square-brackets disappear since the derivative of the differentiation variable is 1 and the derivative of a constant is 0

$$\frac{(\beta - 3) + (\beta - 4)}{(\beta - 3)(\beta - 4)}$$

$$= \frac{2\beta - 7}{(\beta - 3)(\beta - 4)}$$

Since $\frac{\partial l(\beta)}{\partial \beta}(4 \ln(\beta))$ simply equals to $\frac{4}{\beta}$ We can then write the derivative as

$$\frac{2\beta - 7}{(\beta - 3)(\beta - 4)} - \frac{4}{\beta}$$

Combining the two fractions into one

$$\frac{-2\beta^2 + 21\beta - 48}{\beta(\beta - 3)(\beta - 4)}$$

Setting the derivative to 0

$$\frac{-2\beta^2 + 21\beta - 48}{\beta(\beta - 3)(\beta - 4)} = 0$$

Since we have an equation of the following form: $\frac{f(\beta)}{g(\beta)} = 0$, we can exclude the denominator since $f(\beta)$ must be equal to zero

$$-2\beta^2 + 21\beta - 48 = 0$$

Now that our expression is quadratic, we can use the Quadratic Formula to find solutions, we start by finding the discriminant

$$D = 21^2 - 4(-2)(-48) = 57$$

Since D is larger than zero, we have two solutions

$$\frac{21 + \sqrt{D}}{4} \approx 3.4$$

$$\frac{21 - \sqrt{D}}{4} \approx 7.1$$

By the definition of the pdf, $x \leq \beta$, which means that 7.1 is the maximum likelihood estimator for when $x_1 = 3$ and $x_2 = 4$

3 Principal Component Analysis

After calculating all the y-values, the y-vector is

$$[0.7, 0.7, -4.4, 0.4, -1.6, 0.7]$$

1)

We start by calculating the mean of each coordinate vector

$$x_{mean} = \frac{1}{6} \sum_{n=1}^6 x_n = 0.33$$

$$y_{mean} = \frac{1}{6} \sum_{n=1}^6 y_n = -0.58$$

We now have a mean point

$$xy_{mean} = (0.33, -0.58)$$

2)

Before we calculate eigenvalues and eigenvectors, we must first compute the covariance matrix. First we must center our data to zero mean

$$x\text{-vector}_{centered} = x\text{-vector} - x_{mean} = [0.27, 0.77, -1.23, -0.03, -0.83, 1.07]$$

$$y\text{-vector}_{centered} = y\text{-vector} - y_{mean} = [1.28, 1.28, -3.82, 0.98, -1.02, 1.28]$$

Augmenting these two centered vectors together, we'll get a 2x6 points matrix, that we'll call *pMatrix*

$$pMatrix = \begin{bmatrix} 0.27 & 0.77 & -1.23 & -0.03 & -0.83 & 1.07 \\ 1.28 & 1.28 & -3.82 & 0.98 & -1.02 & 1.28 \end{bmatrix}$$

Now we will compute the covariance matrix, by multiplying *pMatrix* with itself transposed, and then multiply the resulting matrix with a scalar. The value 0.2 comes from $\frac{1}{N-1}$

$$\begin{aligned} covMatrix &= (pMatrix \cdot pMatrix^T) \cdot 0.2 \\ &\approx \begin{pmatrix} 0.8 & 1.6 \\ 1.6 & 4.3 \end{pmatrix} \end{aligned}$$

From the covariance-matrix, we can find eigenvalues, by solving $\det(covMatrix - \lambda I) = 0$, where λ is a eigenvalue and I is the identity-matrix

$$\begin{aligned} \det(covMatrix - \lambda I) &= 0 \\ \rightarrow \det \left(\begin{pmatrix} 0.8 & 1.6 \\ 1.6 & 4.3 \end{pmatrix} - \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \right) &= 0 \\ \rightarrow \det \begin{pmatrix} 0.8 - \lambda & 1.6 \\ 1.6 & 4.3 - \lambda \end{pmatrix} &= 0 \end{aligned}$$

To find the determinant of a 2x2 matrix, we can multiply each diagonal and take the difference

$$\rightarrow (0.8 - \lambda)(4.3 - \lambda) - 1.6 \cdot 1.6 = 0$$

Expanding this expression, it becomes quadratic

$$\lambda^2 - 5.1\lambda + 0.88 = 0$$

Finding the discriminant

$$D = (-5.1)^2 - 4(1)(0.88) = 22.49$$

Since $D > 0$, we have two solutions(eigenvalues)

$$\lambda_1 = \frac{-(-5.1) + \sqrt{22.49}}{2} \approx 4.9$$

$$\lambda_2 = \frac{-(-5.1) - \sqrt{22.49}}{2} \approx 0.2$$

3)

To find the eigenvectors, that are associated with the found eigenvalues, we can solve the following equation for each eigenvalue:

$$\text{covMatrix} \cdot e_i = \lambda_i \cdot e_i$$

Inserting our covariance matrix and λ_1

$$\begin{pmatrix} 0.8 & 1.6 \\ 1.6 & 4.3 \end{pmatrix} \begin{pmatrix} e_{1,1} \\ e_{1,2} \end{pmatrix} = 4.9 \begin{pmatrix} e_{1,1} \\ e_{1,2} \end{pmatrix}$$

↓

$$0.8e_{1,1} + 1.6e_{1,2} = 4.9e_{1,1}$$

$$1.6e_{1,1} + 4.3e_{1,2} = 4.9e_{1,2}$$

Moving everything to the left side

$$-4.1e_{1,1} + 1.6e_{1,2} = 0$$

$$1.6e_{1,1} - 0.6e_{1,2} = 0$$

It should be noted that for whichever expression we choose, the relation between $e_{1,1}$ and $e_{1,2}$ is the same when we isolate for $e_{1,2}$. Isolating the first expression for $e_{1,2}$

$$1.6e_{1,2} = 4.1e_{1,1}$$

$$e_{1,2} = \frac{4.1}{1.6}e_{1,1} \approx 2.6e_{1,1}$$

Which means that the first eigenvector is

$$\text{eigVec}_1 = \begin{pmatrix} 1 \\ 2.6 \end{pmatrix}$$

Lastly we will normalize the eigenvector

$$\|\text{eigVec}_1\| = \sqrt{1^2 + 2.6^2} \approx 2.8$$

$$eigVec_{1norm} = \frac{eigVec_1}{||eigVec_1||} = \begin{pmatrix} 0.36 \\ 0.93 \end{pmatrix}$$

Now we will perform the same procedure for the second eigenvalue, λ_2

$$\begin{pmatrix} 0.8 & 1.6 \\ 1.6 & 4.3 \end{pmatrix} \begin{pmatrix} e_{2,1} \\ e_{2,2} \end{pmatrix} = 0.2 \begin{pmatrix} e_{2,1} \\ e_{2,2} \end{pmatrix}$$

\downarrow

$$0.8e_{2,1} + 1.6e_{2,2} = 0.2e_{2,1}$$

$$1.6e_{2,1} + 4.3e_{2,2} = 0.2e_{2,2}$$

Moving everything to the left side

$$0.6e_{2,1} + 1.6e_{2,2} = 0$$

$$1.6e_{2,1} + 4.1e_{2,2} = 0$$

Isolating the second expression for $e_{2,2}$

$$4.1e_{2,2} = -1.6e_{2,1}$$

$$e_{2,2} = \frac{-1.6}{4.1}e_{2,1} \approx -0.4e_{2,1}$$

Which means that the second eigenvector is

$$eigVec_2 = \begin{pmatrix} 1 \\ -0.4 \end{pmatrix}$$

Lastly we will normalize the eigenvector

$$||eigVec_2|| = \sqrt{1^2 + (-0.4)^2} \approx 1.1$$

$$eigVec_{2norm} = \frac{eigVec_2}{||eigVec_2||} = \begin{pmatrix} 0.91 \\ -0.36 \end{pmatrix}$$

We now have the following two eigenvectors and their corresponding eigenvalues.

$$eigVec_{1norm} = \begin{pmatrix} 0.36 \\ 0.93 \end{pmatrix}; \lambda_1 = 4.9$$

$$eigVec_{2norm} = \begin{pmatrix} 0.91 \\ -0.36 \end{pmatrix}; \lambda_2 = 0.2$$

To determine which of these are the first and second principal components, we will arrange the eigenvalues in descending order.

$$eigVec_{1norm} = \begin{pmatrix} 0.36 \\ 0.93 \end{pmatrix}; \lambda_1 = 4.9$$

$$eigVec_{2norm} = \begin{pmatrix} 0.91 \\ -0.36 \end{pmatrix}; \lambda_2 = 0.2$$

And find that $eigVec_{1norm}$ is the first principal components and $eigVec_{2norm}$ is the second principal component

4 kNN

The code associated with this assignment is located within the "kNN_students_2022.py" file

a)

```
Current Neighbors: 1; accuracy_score: 0.62; rmse_score: 0.6819090848492928
Current Neighbors: 2; accuracy_score: 0.56; rmse_score: 0.724568837309472
Current Neighbors: 3; accuracy_score: 0.65; rmse_score: 0.648074069840786
Current Neighbors: 4; accuracy_score: 0.69; rmse_score: 0.6633249580710799
Current Neighbors: 5; accuracy_score: 0.74; rmse_score: 0.6204836822995429
Current Neighbors: 6; accuracy_score: 0.75; rmse_score: 0.648074069840786
Current Neighbors: 7; accuracy_score: 0.72; rmse_score: 0.6204836822995429
Current Neighbors: 8; accuracy_score: 0.71; rmse_score: 0.6557438524302001
Current Neighbors: 9; accuracy_score: 0.71; rmse_score: 0.6324555320336759
Current Neighbors: 10; accuracy_score: 0.68; rmse_score: 0.6519202405202649
```

Figure 1: Accuracy- and rmse-score for $k = 1 \dots 10$, neighbors

Since we would like our model to have the highest accuracy-score when predicting, the optimal number of neighbors is 6

The following plot is the dependency between k-Neighbors and the associated accuracy_score

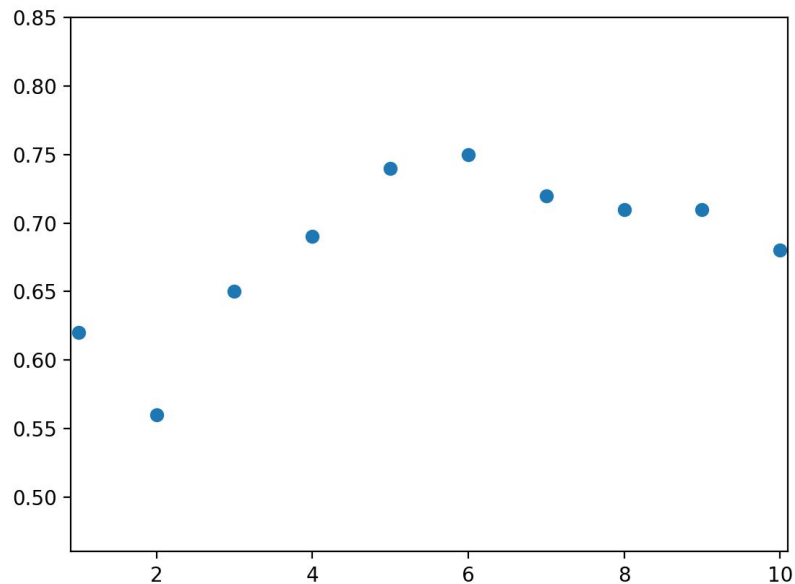


Figure 2: Dependency-plot between k-Neighbors and its associated accuracy score

b)

```

Current weights: [1, 0.01]; accuracy_score: 0.72; rmse_score: 0.5958187643906492
Current weights: [1, 0.025]; accuracy_score: 0.74; rmse_score: 0.5873670062235365
Current weights: [1, 0.05]; accuracy_score: 0.77; rmse_score: 0.6
Current weights: [1, 0.1]; accuracy_score: 0.75; rmse_score: 0.6

```

Figure 3: Accuracy- and rmse-score for the different weights. Each line is computed with 5 neighbors

5 Random Forests

Instead of randomly picking a threshold value and testing its information gain, we should first consider the following:

- Since we are given a definite list of labels, it does not make sense to select a threshold-value outside of said list, as we need to classify the labels within the list
- If we pick two adjacent features, 60 and 70, no matter how close to either feature, the same amount of information will still be yielded. Therefore we shall always pick the average point between two features, which in this example is 65
- No matter how many different features we have, it does not make sense to select a threshold that is in between two labels of the same class. Therefore we shall only pick threshold values that lie between labels of two different classes

With these considerations, we pick 15, 35, 65, 95 and 105 as potential optimal thresholds.

Now we will calculate the entropy on each side of the threshold, as well as for the entire set, for each selected threshold. After said calculation for each threshold, we will also calculate the information gain based on said calculated entropy.

$$xThreshold_{15} = 15$$

$$entropy_{left} = H(1) = -1 \cdot \log_2(1) = 0$$

$$entropy_{right} = H(7, 3, 1) = -0.64 \cdot \log_2(0.64) - 0.27 \cdot \log_2(0.27) - 0.09 \cdot \log_2(0.09) = 1.23$$

$$entropy_{WholeSet} = H(7, 3, 2) = -0.58 \cdot \log_2(0.58) - 0.25 \cdot \log_2(0.25) - 0.17 \cdot \log_2(0.17) = 1.39$$

$$Gain(features, xThreshold_{15}) = 1.39 - \frac{11}{12}1.23 - \frac{1}{12}0 = 0.2625$$

↓

$$xThreshold_{35} = 35$$

$$entropy_{left} = H(2, 1) = -0.66 \cdot \log_2(0.66) - 0.33 \cdot \log_2(0.34) = 0.92$$

$$entropy_{right} = H(5, 3, 1) = -0.56 \cdot \log_2(0.56) - 0.33 \cdot \log_2(0.33) - 0.11 \cdot \log_2(0.11) = 1.35$$

$$Gain(features, xThreshold_{35}) = 1.39 - \frac{9}{12}1.35 - \frac{3}{12}0.92 = 0.1475$$

↓

$$xThreshold_{65} = 65$$

$$entropy_{left} = H(3, 2, 1) = -0.5 \cdot \log_2(0.5) - 0.33 \cdot \log_2(0.33) - 0.17 \cdot \log_2(0.17) = 1.46$$

$$entropy_{right} = H(5, 1) = -0.83 \cdot \log_2(0.83) - 0.17 \cdot \log_2(0.17) = 0.66$$

$$Gain(features, xThreshold_{65}) = 1.39 - \frac{6}{12}1.46 - \frac{6}{12}0.66 = 0.33$$

↓

$$xThreshold_{95} = 95$$

$$entropy_{left} = H(5, 3, 1) = 1.35$$

$$entropy_{right} = H(2, 1) = 0.92$$

$$Gain(features, xThreshold_{95}) = 1.39 - \frac{9}{12}1.35 - \frac{3}{12}0.92 = 0.1475$$

↓

$$xThreshold_{105} = 105$$

$$entropy_{left} = H(5, 3, 2) = -0.5 \cdot \log_2(0.5) - 0.3 \cdot \log_2(0.3) - 0.2 \cdot \log_2(0.2) = 1.49$$

$$entropy_{right} = H(1, 1) = -0.5 \cdot \log_2(0.5) - 0.5 \cdot \log_2(0.5) = 1$$

$$Gain(features, xThreshold_{105}) = 1.39 - \frac{10}{12}1.49 - \frac{2}{12}1 = 0$$

We now have the following thresholds and their corresponding information gain

Threshold	15	35	65	95	105
InfoGain	0.2625	0.1475	0.33	0.1475	0

From this table, we can see that the optimal threshold is 65

6 Classification and Validation

The code associated with this assignment is located within the "ClassVal.py" file

a)

These two functions help convert the categorical features to numerical

```
def catGenderConverter( genderString ):
    if (genderString == "M"):
        return 1
    else:
        return 0

def catPainConverter( painTypeString , array ):
    tempArray = array
    if (painTypeString == "ATA"):
        tempArray[5] = 1
    if (painTypeString == "NAP"):
        tempArray[6] = 1
    if (painTypeString == "ASY"):
        tempArray[7] = 1
    if (painTypeString == "TA"):
        tempArray[8] = 1
    return tempArray
```

This code-snippet loops through all samples in the data, and convert the gender-feature, as well as removing the categorical chest-pain-type and adds four new numerical features, corresponding to each chest-pain-type

```
trainData_combined_converted_new = np.array ([])
for i in range( trainData_combined_converted.shape[0] ):
    trainData_combined_converted[ i ][4] = catGenderConverter(
        trainData_combined_converted[ i ][4] )
    chestPainType = trainData_combined_converted[ i ][5]
    tempSubArray = np.array ( [ trainData_combined_converted[ i ][ :5 ] ] )
    for i in range(4):
        tempSubArray = np.append( tempSubArray , 0 )
    tempSubArray = catPainConverter( chestPainType , tempSubArray )
    trainData_combined_converted_new = np.append(
        trainData_combined_converted_new , tempSubArray )
trainData_combined_converted_new = trainData_combined_converted_new.reshape(
    trainData_combined_converted.shape[0] , 9)
```

b)

This code-snippet splits the heart_simplified_train data into two sets, one being the training data and the other being the testing data. The same procedure is done with the labels corresponding to the samples.

```
trainingData_combined_data = trainData_combined_converted_new[:250]
trainingData_combined_labels = trainData_labels[:250]
testingData_combined_data = trainData_combined_converted_new[250:]
testingData_combined_labels = trainData_labels[250:]
```

```
RandomForest = RandomForestClassifier(n_estimators=100)
RandomForest.fit(trainingData_combined_data,
                  np.ravel(trainingData_combined_labels))
predictions = RandomForest.predict(testingData_combined_data)
modelAccuracy = accuracy_score(testingData_combined_labels, predictions)
print(modelAccuracy)
```

With the aforementioned training and testing data, the random forest regressor has the following accuracy

accuracy: 0.664

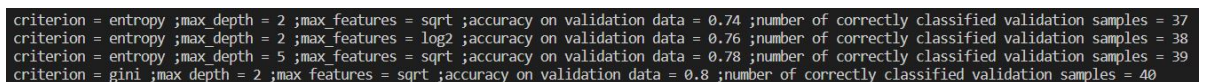
c) and d)

This code-snippet goes through all different parameter combinations and prints each time it gets a superior configuration to the previous configuration

```
currentBestAccuracy = 0
```

```
for criterion in criterionList:
    NewRandomForest.criterion = criterion
    for Depth in treeDepthList:
        NewRandomForest.max_depth = Depth
        for featureAmount in nFeatureList:
            NewRandomForest.max_features = featureAmount
            NewRandomForest.fit(validationDataConv_combined_data, np.ravel(validationDataConv_combined_labels))
            NewPredictions = NewRandomForest.predict(validationDataConvTesting_combined_data)
            prediction_nCorrectAmount = ComputeNCorrect(NewPredictions, validationDataConvTesting_combined_labels)
            prediction_accuracy = accuracy_score(validationDataConvTesting_combined_labels, NewPredictions)
            if (prediction_accuracy > currentBestAccuracy or currentBestAccuracy == 0):
                currentBestAccuracy = prediction_accuracy
                FormatPrint(criterion, Depth, featureAmount, prediction_accuracy, prediction_nCorrectAmount)
```

The following screenshot are the results from the exhaustive search for optimal parameters



```
criterion = entropy ;max_depth = 2 ;max_features = sqrt ;accuracy on validation data = 0.74 ;number of correctly classified validation samples = 37
criterion = entropy ;max_depth = 2 ;max_features = log2 ;accuracy on validation data = 0.76 ;number of correctly classified validation samples = 38
criterion = entropy ;max_depth = 5 ;max_features = sqrt ;accuracy on validation data = 0.78 ;number of correctly classified validation samples = 39
criterion = gini ;max_depth = 2 ;max_features = sqrt ;accuracy on validation data = 0.8 ;number of correctly classified validation samples = 40
```

Figure 4: Each output represent the optimal parameters at the current time of the iteration. The last line represents the optimal parameters

We find the optimal parameters to be

criterion = "gini"

max_depth = 2

max_features = square-root of max_features

With these parameters, we reached the following accuracy and amount of correctly classified samples

accuracy = 0.8

Number of correctly classified samples = 40

Running multiple tests reveal that entropy primarily the best criterion early on, but gini usually gives the accuracy in the end. From the max_depths, we can also see that a low max_depth is usually better as a better accuracy is only ever reached when the max_depth is set 5 or lower. The max_features does not seem to tend to either "sqrt" or "log2" as it regularly switches between the two

7 Clustering

The code associated with this assignment is located within the "Clustering.py" file

a)

This code-snippet, calculates the mean and standard deviation of each feature and appends them to an array

```
meanStdList = np.array([])
for i in range(len(loopList)):
    tempData = data[loopList[i]].values
    featureMean = np.mean(tempData)
    featureDeviation = np.std(tempData)
    meanStdList = np.append(meanStdList, [featureMean, featureDeviation])
meanStdList = meanStdList.reshape(9, 2)
```

And this code-snippet uses the mean and standard deviation on each data-sample to normalize the data

```
for j in range(housingData.shape[1]):
    for i in range(housingData.shape[0]):
        housingData[i][j] = ((housingData[i][j] - housingDataMeanStd[j][0])
                             /housingDataMeanStd[j][1])
```

b)

This function splits the given data into two clusters. This is used for each iteration in the k-means clustering procedure.

```
def DetermineClusters(data, firstCentroid, secondCentroid):
    firstCentroidCluster = np.array([])
    secondCentroidCluster = np.array([])
    for dataSample in data:
        distDataSampleFirst = EuclideanDistance(firstCentroid, dataSample)
        distDataSampleSecond = EuclideanDistance(secondCentroid, dataSample)
        if (distDataSampleFirst < distDataSampleSecond):
```

```

    firstCentroidCluster = np.append(firstCentroidCluster , dataSample)
else:
    secondCentroidCluster = np.append(secondCentroidCluster , dataSample)
firstCentroidCluster = firstCentroidCluster.reshape(int(firstCentroidCluster.shape[0]/9), 9)
secondCentroidCluster = secondCentroidCluster.reshape(int(secondCentroidCluster.shape[0]/9), 9)
return firstCentroidCluster , secondCentroidCluster

```

This function is recursive, and controls the top-down hierarchical approach of clustering. Each time it's called, it runs k-means clustering, until the clusters don't change/update anymore, and then it calls itself with each cluster found from the DetermineClusters function.

```

def Hierarchical_kMeans(data , maxSubDivisions):
    K0InitialIndex , K1InitialIndex = GenerateInitialIndices(data.shape[0])
    K0Centroid = data[K0InitialIndex]
    K1Centroid = data[K1InitialIndex]
    K0PreviousCentroid = [0,0,0,0,0,0,0,0,0]
    K1PreviousCentroid = [0,0,0,0,0,0,0,0,0]
    while (not np.array_equal(K0Centroid , K0PreviousCentroid) and not np.array_equal(K1Centroid , K1PreviousCentroid)):
        K0CentroidCluster , K1CentroidCluster = DetermineClusters(data , K0Centroid , K1Centroid)
        K0PreviousCentroid = K0Centroid
        K1PreviousCentroid = K1Centroid
        K0Centroid = FindMeanCentroid(K0CentroidCluster)
        K1Centroid = FindMeanCentroid(K1CentroidCluster)
    if (maxSubDivisions == 0):
        return K0CentroidCluster , K1CentroidCluster
    else:
        return np.array(Hierarchical_kMeans(K0CentroidCluster , maxSubDivisions - 1), dtype=object),
               np.array(Hierarchical_kMeans(K1CentroidCluster , maxSubDivisions - 1), dtype=object)

```

From the 5 kMeans test, the best cluster was found with the following samples in each cluster




Figure 5: Number of samples in the best cluster from the 5 kMeans test

c)

I use the PCA-class from sklearn. I instantiate the PCA-class with `n_components=2`, meaning that it should reduce the dimensions of the given data to 2 dimensions. Then I flatten the best cluster I found from b) and call `ComputePCA`

```

def ComputePCA(data):
    pcaComponent = PCA(n_components=2)
    return pcaComponent.fit(data)

flatBestCluster = np.concatenate((kMeansBestCluster[0][0], kMeansBestCluster[0][1],
                                   kMeansBestCluster[1][0], kMeansBestCluster[1][1]))
PCA_flatBestClusterData = ComputePCA(flatBestCluster)

```

d)

This code-snippet converts the coordinates of the clusters into x-coordinates and y-coordinates, and plots them in a scatterplot

```
cluster002D = PCA_flatBestClusterData.transform(kMeansBestCluster[0][0])
cluster012D = PCA_flatBestClusterData.transform(kMeansBestCluster[0][1])
cluster102D = PCA_flatBestClusterData.transform(kMeansBestCluster[1][0])
cluster112D = PCA_flatBestClusterData.transform(kMeansBestCluster[1][1])

def GetClusterXAndY(leafCluster):
    xCoords = np.array([])
    yCoords = np.array([])
    for i in range(len(leafCluster)):
        xCoords = np.append(xCoords, leafCluster[i][0])
        yCoords = np.append(yCoords, leafCluster[i][1])
    return xCoords, yCoords

cluster002DxCoords, cluster002DyCoords = GetClusterXAndY(cluster002D)
cluster012DxCoords, cluster012DyCoords = GetClusterXAndY(cluster012D)
cluster102DxCoords, cluster102DyCoords = GetClusterXAndY(cluster102D)
cluster112DxCoords, cluster112DyCoords = GetClusterXAndY(cluster112D)

def Get2DClusterCentroidCoords(coordListX, coordListY):
    return np.mean(coordListX), np.mean(coordListY)

cluster002DCentroidxCoord, cluster002DCentroidyCoord = Get2DClusterCentroidCoords(cluster002DxCoords, cluster002DyCoords)
cluster012DCentroidxCoord, cluster012DCentroidyCoord = Get2DClusterCentroidCoords(cluster012DxCoords, cluster012DyCoords)
cluster102DCentroidxCoord, cluster102DCentroidyCoord = Get2DClusterCentroidCoords(cluster102DxCoords, cluster102DyCoords)
cluster112DCentroidxCoord, cluster112DCentroidyCoord = Get2DClusterCentroidCoords(cluster112DxCoords, cluster112DyCoords)

plt.scatter(cluster002DxCoords, cluster002DyCoords, c="red", s = 15)
plt.scatter(cluster002DCentroidxCoord, cluster002DCentroidyCoord, c="black")
plt.scatter(cluster012DxCoords, cluster012DyCoords, c="aqua", s = 15)
plt.scatter(cluster012DCentroidxCoord, cluster012DCentroidyCoord, c="black")
plt.scatter(cluster102DxCoords, cluster102DyCoords, c="green", s = 15)
plt.scatter(cluster102DCentroidxCoord, cluster102DCentroidyCoord, c="black")
plt.scatter(cluster112DxCoords, cluster112DyCoords, c="orange", s = 15)
plt.scatter(cluster112DCentroidxCoord, cluster112DCentroidyCoord, c="black")
plt.show()
```

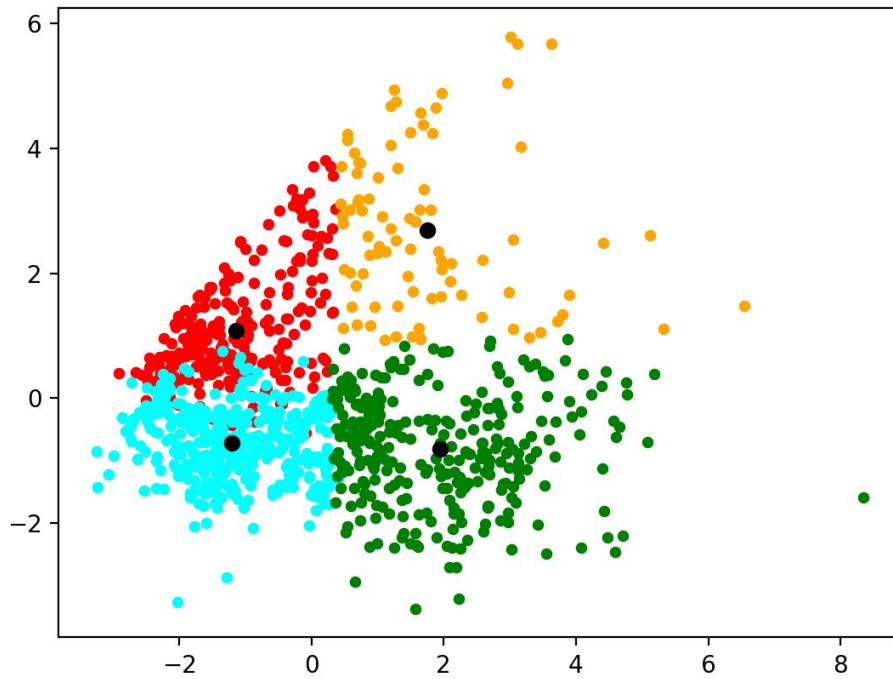


Figure 6: Scatter-plot of the clusters

From this plot, we can see that we have normalized the data correctly, since it's centered around $(0,0)$. We can also see that the clustering is working correctly, since we are getting 4 pretty evenly spaced clusters, and for the most part no overlap