Modelling and Analysis of Data

Exam no. 39

22. januar 2022

Indhold

1 Problem 1 - Maximum Likelihood Estimation

1.1

Assessing the following:

$$f(x)\begin{cases}\frac{1}{\sigma\sqrt{2\pi}}\cdot\frac{1}{x}\cdot exp^{\left(-\frac{1}{2}\left(\frac{log(x)-\mu}{\sigma}\right)^2\right)} & \text{for} \quad x \in \mathbb{R}_+ \\ 0 & \text{otherwise}\end{cases}$$

Since I can fix the parameter σ I can express the likelihood L of observing the given dataset $x_1,...,X_N$ of N i.i.d samples from X as the joint distribution all the N i.i.d samples conditioning on μ as:

$$L = p(x_1,...,x_n|\mu) = \prod_{n=1}^{N} p(x_n|\mu) = \prod_{n=1}^{N} f(x_n;\mu)$$

The PDF is now considered a function of both x_n and μ I'm going to make take the logarithm of the expression, to make it more managable, without changing the estimate. When trying to find the optimal $\hat{\mu}$ that maximizes the likelihood L.

$$\begin{aligned} logL &= log\left(\prod_{n=1}^{N} f(x_n;\mu)\right) \\ &= \sum_{n=1}^{N} log(f(x_n;\mu)) \\ &= \sum_{n=1}^{N}\left[log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) + log\left(\frac{1}{x_n}\right) + log\left(exp\left(-\frac{1}{2}\left(\frac{log(x)-\mu}{\sigma}\right)^2\right)\right)\right] \\ &= \sum_{n=1}^{N}\left[log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) + log\left(\frac{1}{x_n}\right) + -\frac{1}{2}\left(\frac{log(x)-\mu}{\sigma}\right)^2\right] \end{aligned}$$

To find the MLE $\hat{\mu}$ for the parameter μ I optimizr L by solving $\frac{\partial logL}{\partial\mu} = 0$ When taking derivatives I'm only interesed in terms that involve μ and so I have the following expression.

$$\begin{aligned} \frac{\partial LogL}{\partial\mu} &= \frac{\partial}{\partial\mu}\left[\sum_{n=1}^{N} -\frac{1}{2}\left(\frac{log(x)-\mu}{\sigma}\right)^2\right] \\ &= \sum_{n=1}^{N} -\left(\frac{log(x)-\mu}{\sigma}\right)\cdot\left[\frac{log(x)-\mu}{\sigma}\cdot\frac{\partial}{\partial\mu}\right] \quad (\textit{Chain rule}) \\ &= \sum_{n=1}^{N} -\left(\frac{log(x)-\mu}{\sigma}\right)\cdot\left(-\frac{1}{\sigma}\right) \\ &= \frac{1}{\sigma^2}\sum_{n=1}^{N} log(x_n)-\mu \end{aligned}$$

solving $\frac{\partial logL}{\partial \mu} = 0$ with respect to μ

$$0 = \frac{1}{\sigma^2}\sum_{n=1}^{N} log(x_n) - \mu = \sum_{n=1}^{N} log(x_n) - \mu = \left[\sum_{n=1}^{N} log(x_n)\right] - N\mu$$

$$\Updownarrow$$

$$N\mu = \sum_{n=1}^{N} log(x_n)$$

$$\Updownarrow$$

$$\mu = \frac{1}{N}\sum_{n=1}^{N} log(x_n) = \frac{1}{N} log(x_1 \cdot x_2 \cdot ... \cdot x_N) = log\left((x_1 \cdot x_2 \cdot ... \cdot x_N)^{\frac{1}{N}}\right)$$
$$= log(\sqrt[N]{x_1 \cdot x_2 \cdot ... \cdot x_N})$$

The above calculations is a neccesary condition in order to find an optimal solution. Furthermore, to check if I'm dealing with a global maximum or a global minium I have to take the second derivative of my function.

$$\frac{\partial^2 logL}{\partial \mu^2} = -\frac{N}{\sigma^2}$$

Assessing the result from the second derivative, I can see that my function will alwas be negative. Thus by the second deratuive test

$$\hat{\mu} = log(\sqrt[N]{x_1 \cdot x_2 \cdot ... \cdot x_N})$$

is a global maximum

2 Problem 2 - Statistics

3 Problem 3 - Principal Component Analysis

3.1 1) - Mean point of the data

Step one is to calculate the mean point of the data with respect to both x and y

$$\bar{x} = \frac{1}{8}(0.6 + 0.5 + 1.1 + (-0.5) + 0.8 + 0.2 + (-0.1) + 1.0) = 0.4500$$

$$\bar{y} = \frac{1}{8}(1.1 + 1.0 + 2.0 + 0.2 + (-0.1) + (-0.1) + (-1.5) + 2.5) = 0.6375$$

Step two is to normalize the dataset. This is done by taking each original point and subtract the mean. $p - [\bar{x}, \bar{y}]$

Normalizing each point

$$p1 = (0.6, 1.1) - [0.4500, 0.6375] = [0.1500, 0.4625] \quad (1)$$
$$p2 = (0.5, 1.0) - [0.4500, 0.6375] = [0.0500, 0.3625] \quad (2)$$
$$p3 = (1.1, 2.0) - [0.4500, 0.6375] = [0.6500, 1.3625] \quad (3)$$
$$p4 = (-0.5, 0.2) - [0.4500, 0.6375] = [-0.9500, -0.4375] \quad (4)$$
$$p5 = (0.8, -0.1) - [0.4500, 0.6375] = [0.3500, -0.7375] \quad (5)$$
$$p6 = (0.2, -0.1) - [0.4500, 0.6375] = [-0.2500, -0.7375] \quad (6)$$
$$p7 = (-0.1, -1.5) - [0.4500, 0.6375] = [-0.5500, -2.1375] \quad (7)$$
$$p8 = (1.0, 2.5) - [0.4500, 0.6375] = [0.5500, 1.8625] \quad (8)$$

Step three is to calculate the covariances for XX, XY and YY using the normalized values for p1-p8

$$cov(xx) = \frac{1}{8}\Big(0.15^2 + 0.05^2 + 0.65^2 + (-0.95)^2 + (-0.25)^2 + (-0.55)^2 + 0.55^2\Big) = 0.2675 \quad (9)$$

$$cov(yy) = \frac{1}{8}\Big(0.4625^2 + 0.3625^2 + 1.3625^2 + (-0.4375)^2 + (-0.7375)^2 + (-0.7375)^2 + (-2.1375)^2 + 1.8625^2\Big) = 1.439843750 \quad (10)$$

$$cov(xy) = \frac{1}{8}\Big((0.15 \cdot 0.4625 + 0.05 \cdot 0.3625 + 0.65 \cdot 1.3625 + ((-0.95) \cdot (-0.4375)) + 0.35 \cdot (-0.7375) + ((-0.25) \cdot (-0.7375)) + ((-0.55) \cdot (-2.1375)) + 0.55 \cdot 1.8625)\Big) = 0.439375 \quad (11)$$

having done the three calucaltions I can now form the covariance matrix inserting the calculated values into their respectable place:

$$cov = \begin{pmatrix} xx & xy \\ xy & yy \end{pmatrix} =$$

$$cov = \begin{pmatrix} 0.2675 & 0.439375 \\ 0.439375 & 1.439843750 \end{pmatrix}$$

3.2 2) - Two eigenvalues

I order to find the eigenvalues and the eigenvectors i need to solve ***Characteristic equation*** of the covariance matrix. The Characteristic equation is denoted as:

$$det(A - \lambda I) = 0$$

To calculate the determinant you calculate the following:

$$determinant = \begin{pmatrix} a & b \\ c & c \end{pmatrix} = ad - bc$$

Solving the system equation that I get from uising the Characteristics equation on the covariance matrix i get the following eigenvalues.

$$\begin{pmatrix} 0.2675 - \lambda & 0.439375 \\ 0.439375 & 1.439843750 - \lambda \end{pmatrix} = 0$$

First I get the equation by calulating the determinant

$$det(cov) \rightarrow ((0.2675 - \lambda) \cdot (1.439843750 - \lambda)) - (0.439375 \cdot 0.439375)$$

Then setting the equation equal to zero and solving it will give me the two values for λ which is the eigenvalues.

$$((0.2675 - \lambda) \cdot (1.439843750 - \lambda)) - (0.439375 \cdot 0.439375) = 0$$

$$eigenvalue_1 = 0.1211093468 \qquad eigenvalue_2 = 1.586234403$$

3.3 3) - Two eigenvectors

I order to find the eigenvectors I use the definition of an eigenvector $A\bar{\mathbf{x}} = \lambda\bar{\mathbf{x}}$ and solve this this equation.

$$cov \cdot \bar{x} = \lambda \cdot \bar{x}$$

I'm going to do the calculations in Maple since I'm dealing with odd numbers. I also going to reduce some of the decimals by rounding off just for readabilitys sake.

$$eigenvector_1 = [-0.9487, -0.3161] \qquad eigenvector_2 = [0.3161, -0.9487]$$

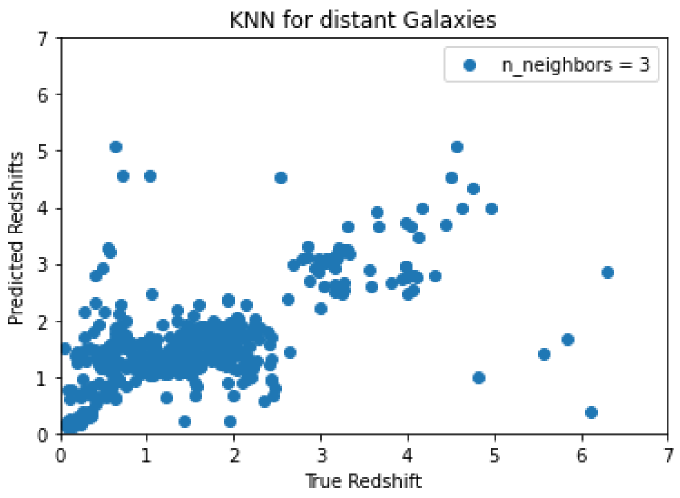
note that signs of the eigenvector can be different. e.g. $eigenvector_1$ could be $[0.9487, 0.3161]$ []

4 Problem 4 - Regression

4.1 a)

Using the euclidean distance I get an RSME value of

$$RMSE = 0.82430$$

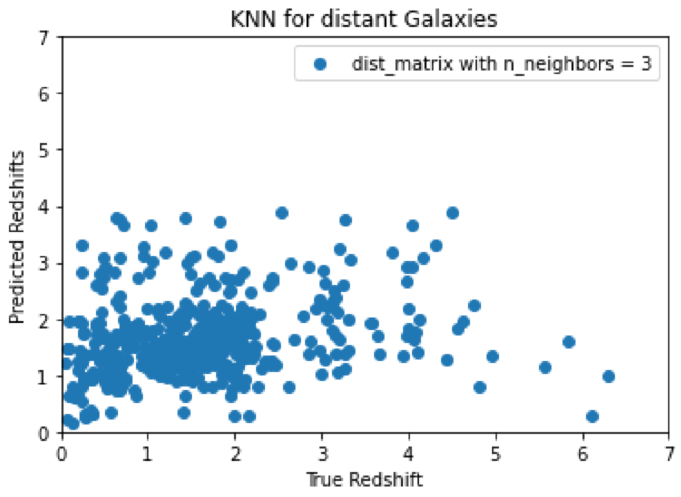


Figur 1: Plot of kNN with k = 3

4.2 b)

Using the other distance calculation as given in the exam set I get the following RS-ME value

$$RMSE = 1.100519$$



Figur 2: Plot of kNN with k = 3 and distance matrix

The distance matrix will lower/lessen the influence of all the features/attributes except for the last two, since they are set to 1.0.

4.3 c)

5 Problem 5 - Random Forests

5.1 Finding Thresholds

There are four places that I should take into consideration as possible thresholds.

option	x_i	x_{i+1}
1	20	30
2	60	70
3	90	100
4	100	110

There hare four positions that could be possible threshold, however, I'm going to ignore optinon 4, since it will be the same as option 1, except with reversed values. These four places has been found using the same approatch as show in our lecture, and lecture video (L10_Classification_Regression_2_Part3, starting at 17min 50sec.). This means, that I'm only going to look at positions where there is a "change"in labels for a given datapoint compared to its predecessor or successor. Since there is no reason to look at possible thresholds where the neigbors to a given feature has the same labels.

This process can be formalized:

$$x_i + (\frac{x_{i+1} - x_i}{2}) : y_{i+1} \neq y_i$$

As stated, option 4 is the same as option 1, so I will only perform the calculations for option 1-3.

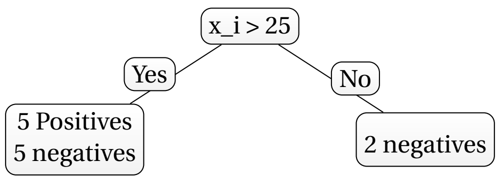
$$\text{Option } 1 = 20 + \frac{(30-20)}{2} = 25$$

$$\text{Option } 2 = 60 + \frac{(70-60)}{2} = 65$$

$$\text{Option } 3 = 90 + \frac{(100-90)}{2} = 95$$

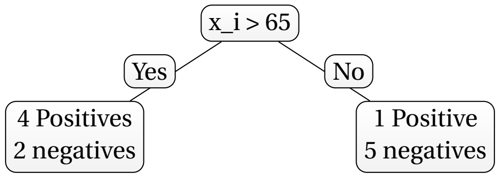
Using my calculated values for for possible thresholds I have three different decision trees to take into consideration.

Tree 1:



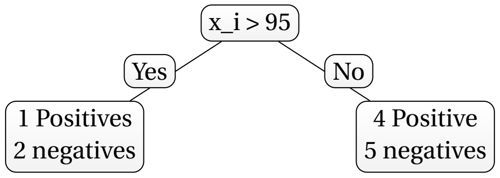
Figur 3: Decision three with threshold = 25

Tree 2:



Figur 4: Decision three with threshold = 65

Tree 3:



Figur 5: Decision three with threshold = 95

From here, I will start by calculating the Entropy values for splits in my trees. Aswell as the information gain using the functions ginven in our lecture slides (L10_Classification_Regression_2_Part3 p. 8)

$$\text{Entropy H(T)} = -p_+ \cdot log_2 \cdot p_+ - p_- \cdot log_2 \cdot p_-$$

$$\text{Gain (T, j)} = \text{H(T)} - \sum_{i=0,1} \frac{|T_i|}{|T|} \text{H}(T_i)$$

Calculating the information gain for tree 1 where the threshold is = 25:

$$\begin{aligned}\text{Gain(T, j)} &= \text{H(5 pos, 7 neg)} - \frac{10}{12}\text{H(5 pos, 5 neg)} - \frac{2}{12}\text{H(0 pos, 2 neg)} \\ &= 0.979 - \frac{10}{12}1 - \frac{2}{12}0 \approx 0.145\end{aligned}$$

Calculating the information gain for tree 2 where the threshold is = 65:

$$\begin{aligned}\text{Gain(T, j)} &= \text{H(5 pos, 7 neg)} - \frac{6}{12}\text{H(4 pos, 2 neg)} - \frac{6}{12}\text{H(1 pos, 5 neg)} \\ &= 0.979 - \frac{10}{12}0.918 - \frac{2}{12}0.650 \approx 0.195\end{aligned}$$

Calculating the information gain for tree 3 where the threshold is = 95:

$$\begin{aligned}\text{Gain(T, j)} &= \text{H(5 pos, 7 neg)} - \frac{6}{12}\text{H(4 pos, 2 neg)} - \frac{6}{12}\text{H(1 pos, 5 neg)} \\ &= 0.979 - \frac{10}{12}0.918 - \frac{2}{12}0.991 \approx 0.0062\end{aligned}$$

Since the goal is to find the tree which gives us the highest information gain. I can see that the second decision tree with a threshold of 65 is the best one.

Thus, the optimal threshold is 65.

6 Problem 6 - Classification & Validation

NOTE that I've included the full python code in the appendix

6.1 a) - Implement random forest training

I've implemented the Random Forest Training using the existing library from ***Sikit-learn***, more specific the package: **sklearn.ensemble.RandomForestClassifier**. Below I've added the most notable part from my code in order to load the data. I'm reusing the "loaddata"function that's been handed out, though I've modified it slightly in line 12 in the code snippet.

```
1  # Modified the loaddata function to make the training features
    ↪ into 2d np.array, more specifically the 't value'
2  def loaddata(filename):
3      """Load the balloon data set from filename and return t, X
4          t - N-dim. vector of target (temperature) values
5          X - N-dim. vector containing the inputs (lift) x for
    ↪ each data point
6      """
7      # Load data set from CSV file
8      Xt = np.loadtxt(filename, delimiter=',')
9
10     # Split into data matrix and target vector
11     X = Xt[:,0] # Matrix
12     t = Xt[:,1:] # Vector
13
14     return t, X
15
16 # Loads the training data
17 X_train, t_train =
    ↪ loaddata("./accent-mfcc-data_shuffled_train.txt")
18
19 # Loads the validation data
20 X_validation, t_validation =
    ↪ loaddata("./accent-mfcc-data_shuffled_validation.txt")
```

To see how I've chosen to set up the Random Forest Classifier, training the algorithm using the it to predict - see the submitted code in section 6.4

6.2 b) - Finding the optimal set of random forest classifier parameters

I've chosen to use two extra packages in order to implement my Random Forest algorithm. It hasn't been specified that I'm not allowed to use other packages from the **Sklean** library so I assume that this is a possible solution.

I've made use of **ParameterGrid** from the ***sklearn.model_selection*** and **accuracy_score** from the ***sklearn.metrics***.
I'm using ParameterGrid to generate a list of all the possible permutations of the specified parameters used for a given iteration to perform the optimized search. With each iteration the parameters and the resulting performance metrics are added to a list which can be sorted easily. I'm currently using the accuracy_score as the first metric.

see the submitted code in section 6.4

6.3 c) - My results

Looking at the results, it's possible to see that the performance of the algorithm increases when tree depth, number of features that's taken into consideration and the complexity of the criterion is also increased. However it comes with the cost of a more computational complexity aswell. The second listing in section 6.4 shows the terminal output from the optimization loop. I've appended a sorted print of the permutated parameters in the appendix.

Assessing the result, I get the optimal results from the algorithm with the parameters:

```
{'criterion': 'entropy', 'max_tree_depth': 10, 'max_features': 'sqrt'}
```

6.4 Code for a) & b)

testestestest

```python
# Adding the parameters to test given in the exam question.
↪ Using these to find the optimal set of random forest
↪ classifier parameters.
random_f_parameters = {
    'criterion'          : ['gini', 'entropy'],
    'max_tree_depth'     : [2,5,6,10,15],
    'max_features'       : ['sqrt', 'log2']
    }

# Empty array for the result metrics
res = np.empty((0,3))

# Looping through the chosen(given) parameters and setting up
↪ the Random forest classifier each time.
for params in list(ParameterGrid(random_f_parameters)):
    clf = RandomForestClassifier(
        criterion    = params['criterion'],
        max_depth    = params['max_tree_depth'],
        max_features = params['max_features'])

    # Training using the created classifier with the given
    ↪  parameters.
    clf.fit(X_train, t_train)

    # number of correctly classified validation samples
    t_prediction = clf.predict(X_validation)
    acc_score = accuracy_score(t_validation, t_prediction)

    # probability associated with classification
    t_probability = clf.predict_proba(X_validation)
    probability_score = np.mean([t_probability[int(t_val)]
                            for (t_probability, t_val)
                            in zip(t_probability,
                             ↪  t_validation)])

    print("Accuracy score: %.2f"
        %acc_score)
    print("Average probability assigned to correct classes:
    ↪  %.2f"
        %probability_score)

    # print the parameters if new ones are more optimal than
    ↪  previously tried ones.
    if len(res) > 0 and (acc_score > res[-1,1]
                            or (acc_score == res[-1,1]
                                and probability_score >
                                ↪  res[-1,2])):
        print(params)

    # accumulate results
    res = np.append(res, np.array([[params, acc_score,
    ↪  probability_score]]), axis=0)
    # sort the results in ascending order
    res = res[np.lexsort((res[:,1], res[:,2]))]
```

```
Accuracy score: 0.48
Average probability assigned to correct classes: 0.36
Accuracy score: 0.66
Average probability assigned to correct classes: 0.48
{'criterion': 'gini', 'max_features': 'sqrt', 'max_tree_depth': 5}
Accuracy score: 0.73
Average probability assigned to correct classes: 0.51
{'criterion': 'gini', 'max_features': 'sqrt', 'max_tree_depth': 6}
Accuracy score: 0.77
Average probability assigned to correct classes: 0.56
{'criterion': 'gini', 'max_features': 'sqrt', 'max_tree_depth': 10}
Accuracy score: 0.77
Average probability assigned to correct classes: 0.56
{'criterion': 'gini', 'max_features': 'sqrt', 'max_tree_depth': 15}
Accuracy score: 0.48
Average probability assigned to correct classes: 0.36
Accuracy score: 0.69
Average probability assigned to correct classes: 0.48
Accuracy score: 0.71
Average probability assigned to correct classes: 0.51
Accuracy score: 0.82
Average probability assigned to correct classes: 0.56
{'criterion': 'gini', 'max_features': 'log2', 'max_tree_depth': 10}
Accuracy score: 0.79
Average probability assigned to correct classes: 0.57
{'criterion': 'gini', 'max_features': 'log2', 'max_tree_depth': 15}
Accuracy score: 0.52
Average probability assigned to correct classes: 0.37
Accuracy score: 0.73
Average probability assigned to correct classes: 0.51
Accuracy score: 0.74
Average probability assigned to correct classes: 0.54
Accuracy score: 0.81
Average probability assigned to correct classes: 0.57
{'criterion': 'entropy', 'max_features': 'sqrt', 'max_tree_depth': 10}
Accuracy score: 0.81
Average probability assigned to correct classes: 0.58
{'criterion': 'entropy', 'max_features': 'sqrt', 'max_tree_depth': 15}
Accuracy score: 0.51
Average probability assigned to correct classes: 0.38
Accuracy score: 0.75
Average probability assigned to correct classes: 0.51
Accuracy score: 0.78
Average probability assigned to correct classes: 0.54
Accuracy score: 0.81
Average probability assigned to correct classes: 0.58
Accuracy score: 0.78
Average probability assigned to correct classes: 0.57
```

Figur 6: Caption of figure here

7 Problem 7 - K-means Clustering & Principal Component Analysis

7.1 a)

I've decided to make my own function to normalize my data. It normalizes the input value x by subtracting the mean and dividion by the standard deviation.

```python
def normalizeData(x):
    '''
    Normalizes the input using mean and the standard deviation.
    Normalization is done on all the columns in the 2d
↪   numpy-array 'x'

    Returns the normalized 'x'
    '''
    return (x - np.mean(x, axis=0)) / np.std(x, axis=0)
```

7.2 b)

Below is a codesnippet of my implementation of the k-mean clustering. I've definde multiple helperfunctions, which can be found in the source code or in the appendix.

```python
def k_mean_clustering(k, data, centroids):
    """
    K-means clustering

    Params:
    ------
    k : number of clusters
    data : (n_samples, n_features) (normalized) data matrix

    Returns:
    --------
    assignments, intra_cluster_dist : tuple

    """
    current_assignments = assign_datapoints_to_centroids(data,
    ↪   centroids)

    new_assignments = [] # initial empty value

    while not np.array_equal(current_assignments,
    ↪   new_assignments):
        # repeat until assignments does not change
        centroids = calculate_new_centroids(data,
         ↪  current_assignments, centroids)
        current_assignments = new_assignments
        new_assignments = assign_datapoints_to_centroids(data,
         ↪  centroids)

    intra_cluster_dist = compute_sum_intra_cluster_dist(data,
     ↪  current_assignments, centroids)

    return current_assignments, intra_cluster_dist
```

The results of the k-means cluster algorithm with k = 3. The algorithm has been implemented with ***np.random.default_rng*** which constructs a random generator so that a desired "seed_value"can be reproduced.

7.3 c)

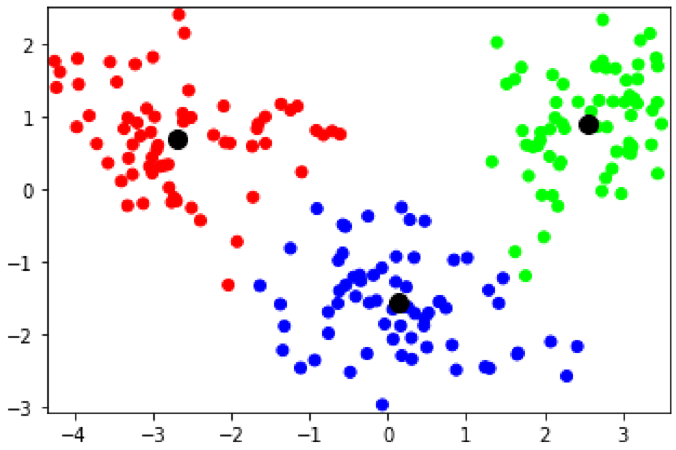
Here's a print of the values that I get from the k-means clustering algorithm, when using seed: 1

```
Calculations with seed value: 1
Smallest Intra-Cluster Distance: 268.5510
Number of samples in cluster 0: 67
Number of samples in cluster 1: 66
Number of samples in cluster 2: 67
```

7.4 d)

I've reused code that has been handed out in the assignments aswell as my own implementation of PCA from Assignment 5.

7.5 e)



Figur 7: Caption of figure here

8 Appendix A

8.1 ...

```
def minted_for_code_here
```

9 Appendix B

9.1 ...

```
    def minted_for_code_here
```

10 Appendix Q6

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.model_selection import ParameterGrid
from sklearn.ensemble import RandomForestClassifier

# Modified the loaddata function to make the training features
↪ into
# 2d np.array, more specifically the 't value'
def loaddata(filename):
    """Load the balloon data set from filename and return t, X
        t - N-dim. vector of target (temperature) values
        X - N-dim. vector containing the inputs (lift) x for
↪ each data point
    """
    # Load data set from CSV file
    Xt = np.loadtxt(filename, delimiter=',')

    # Split into data matrix and target vector
    X = Xt[:,0] # Matrix
    t = Xt[:,1:] # Vector

    return t, X

'''
QUESTION 6 - A
'''
# Loads the training data
X_train, t_train =
↪ loaddata("./accent-mfcc-data_shuffled_train.txt")

# Loads the validation data
X_validation, t_validation =
↪ loaddata("./accent-mfcc-data_shuffled_validation.txt")

print("Shape of train targets: %s" %str(t_train.shape))
print("Shape of train features: %s" %str(X_train.shape))
print("Shape of val targets: %s" %str(t_validation.shape))
print("Shape of val features: %s" %str(X_validation.shape))
print()


'''
```

```
QUESTION 6 - B
'''
# Adding the parameters to test given in the exam question.
# Using these to find the optimal set of random forest
 ↪  classifier parameters.
random_f_parameters = {
    'criterion'         : ['gini', 'entropy'],
    'max_tree_depth'    : [2,5,6,10,15],
    'max_features'      : ['sqrt', 'log2']
    }

# Empty array for the result metrics
res = np.empty((0,3))

# Looping through the chosen(given) parameters and setting up
 ↪  the
# Random forest classifier each time.
for params in list(ParameterGrid(random_f_parameters)):
    clf = RandomForestClassifier(
        criterion    = params['criterion'],
        max_depth    = params['max_tree_depth'],
        max_features = params['max_features'])

    # Training using the created classifier with the given
     ↪  parameters.
    clf.fit(X_train, t_train)

    # number of correctly classified validation samples
    t_prediction = clf.predict(X_validation)
    acc_score = accuracy_score(t_validation, t_prediction)

    # probability associated with classification
    t_probability = clf.predict_proba(X_validation)
    probability_score = np.mean([t_probability[int(t_val)]
                          for (t_probability, t_val)
                          in zip(t_probability, t_validation)])

    print("Accuracy score: %.2f"
        %acc_score)
    print("Average probability assigned to correct classes:
     ↪  %.2f"
        %probability_score)

    # print the parameters if new ones are more optimal
    # than previously tried ones.
    if len(res) > 0 and (acc_score > res[-1,1]
                         or (acc_score == res[-1,1]
                             and probability_score >
                              ↪  res[-1,2])):
        print(params)

    # accumulate results
    res = np.append(res, np.array([[params, acc_score,
     ↪  probability_score]]), axis=0)
    # sort the results in ascending order
    res = res[np.lexsort((res[:,1], res[:,2]))]

for x in res:
    print(x[0])
```

11 Appendix X

11.1 Q6_c

This is a sorted print of the permutated parameters from the Random Forest algorithm. which which yields matrics that are increacinly better.

I've rearranged the terminal output slightly for better readability.

```
{'criterion': 'gini'   , 'max_features': 'sqrt', 'max_tree_depth': 2}
{'criterion': 'gini'   , 'max_features': 'log2', 'max_tree_depth': 2}
{'criterion': 'entropy', 'max_features': 'sqrt', 'max_tree_depth': 2}
{'criterion': 'entropy', 'max_features': 'log2', 'max_tree_depth': 2}
{'criterion': 'gini'   , 'max_features': 'sqrt', 'max_tree_depth': 5}
{'criterion': 'gini'   , 'max_features': 'log2', 'max_tree_depth': 5}
{'criterion': 'entropy', 'max_features': 'log2', 'max_tree_depth': 5}
{'criterion': 'entropy', 'max_features': 'sqrt', 'max_tree_depth': 5}
{'criterion': 'gini'   , 'max_features': 'log2', 'max_tree_depth': 6}
{'criterion': 'gini'   , 'max_features': 'sqrt', 'max_tree_depth': 6}
{'criterion': 'entropy', 'max_features': 'log2', 'max_tree_depth': 6}
{'criterion': 'entropy', 'max_features': 'sqrt', 'max_tree_depth': 6}
{'criterion': 'gini'   , 'max_features': 'sqrt', 'max_tree_depth': 10}
{'criterion': 'gini'   , 'max_features': 'log2', 'max_tree_depth': 10}
{'criterion': 'gini'   , 'max_features': 'sqrt', 'max_tree_depth': 15}
{'criterion': 'gini'   , 'max_features': 'log2', 'max_tree_depth': 15}
{'criterion': 'entropy', 'max_features': 'sqrt', 'max_tree_depth': 10}
{'criterion': 'entropy', 'max_features': 'log2', 'max_tree_depth': 15}
{'criterion': 'entropy', 'max_features': 'log2', 'max_tree_depth': 10}
{'criterion': 'entropy', 'max_features': 'sqrt', 'max_tree_depth': 15}
```