Modelling and Analysis of Data

Exam no. 39

17. januar 2022

Indhold

1 Problem 1 - Maximum Likelihood Estimation

1.1

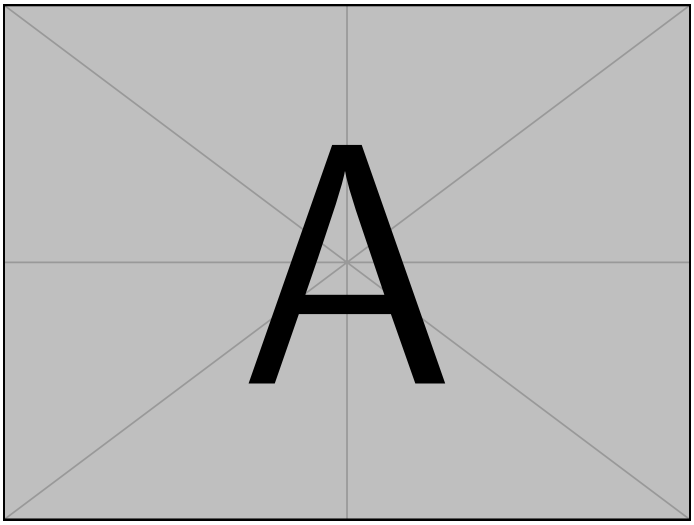
```
    def minted_for_code_here
```

2 Problem 2 - Statistics

2.1

This is my implementation of KNN

```
    def minted_for_code_here
```



Figur 1: Caption of figure here

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:

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

So in my case in order to calculate the determinant I have

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

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

3.3 3) - Two eigenvectors

4 Problem 4 - Regression

4.1

5 Problem 5

5.1 ...

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
```

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

    return t, X

# Loads the training data
X_train, t_train =
 ↪  loaddata('D:\\Uddannelse\\Datalogi\\KU\\2_aar\\MAD\\MAD2021\\2022_Exam\\exam_data\\data

# Loads the validation data
X_validation, t_validation =
 ↪  loaddata('D:\\Uddannelse\\Datalogi\\KU\\2_aar\\MAD\\MAD2021\\2022_Exam\\exam_data\\data
```

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 chosae 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.

6.4 Code for a) & b)

testestestest

```
# 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 2: Caption of figure here

7 Problem 7

7.1 K-means Clustering & Principal Component Analysis

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('D:\\Uddannelse\\Datalogi\\KU\\2_aar\\MAD\\MAD2021\\2022_Exam\\exam_data\\data

# Loads the validation data
X_validation, t_validation = loaddata('D:\\Uddannelse\\Datalogi\\KU\\2_aar\\MAD\\MAD2021\\2022_Exam\\exam_data\\data

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

```

```
38
39  '''
40  QUESTION 6 - B
41  '''
42  # Adding the parameters to test given in the exam question.
43  # Using these to find the optimal set of random forest
    ↪  classifier parameters.
44  random_f_parameters = {
45      'criterion'         : ['gini', 'entropy'],
46      'max_tree_depth'    : [2,5,6,10,15],
47      'max_features'      : ['sqrt', 'log2']
48      }
49
50  # Empty array for the result metrics
51  res = np.empty((0,3))
52
53  # Looping through the chosen(given) parameters and setting up
    ↪  the
54  # Random forest classifier each time.
55  for params in list(ParameterGrid(random_f_parameters)):
56      clf = RandomForestClassifier(
57          criterion    = params['criterion'],
58          max_depth    = params['max_tree_depth'],
59          max_features = params['max_features'])
60
61      # Training using the created classifier with the given
        ↪  parameters.
62      clf.fit(X_train, t_train)
63
64      # number of correctly classified validation samples
65      t_prediction = clf.predict(X_validation)
66      acc_score = accuracy_score(t_validation, t_prediction)
67
68      # probability associated with classification
69      t_probability = clf.predict_proba(X_validation)
70      probability_score = np.mean([t_probability[int(t_val)]
71                           for (t_probability, t_val)
72                           in zip(t_probability, t_validation)])
73
74      print("Accuracy score: %.2f"
75          %acc_score)
76      print("Average probability assigned to correct classes:
        ↪  %.2f"
77          %probability_score)
78
```

```
    # 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}
```