Högskolan I Halmstad
Design of Embedded and Intelligent Systems
DT8007

# Lab1

*Lab 1 Report*

| | | |
|---|---|---|
| Julian Kaduk | - | 19920911-T633 |
| Sivan Dawood | - | 19970614-8997 |
| Filip Kågesson | - | 19970320-8893 |
| Michael Forschlé | - | 19890731-4515 |

Supervisor/s:
Martin Daniel Cooney

Halmstad, Thursday 24th September, 2020

# 1. Lab: 1

## 1.1 task1

In this task we made our own KNN algorithm that can classify a dataset, we got a dataset from the site kaggle [kaggle., 2020]. The dataset is about diabetes, it haves 9 columns and 768 rows. The different columns are: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age and Outcome. The classification will we done on the outcome column. The KNN algorithm need some algorithms to make it work. We made one that calculate the euclidean distance and the other own to calculate the accuracy. when those two wore done could we do the KNN algorithm with help of them. The KNN algorithm uses the euclidean distance to calculate the nearest neighbour and that is also what KNN stands for K-nearest neighbour, the k is how many neighbour the data should look at before classifying the data point. The code was done in python and in the software Jupyter notebook.

```python
def Euclidean(dL1, dL2):
    sum = 0
    for x1, x2 in zip(dL1, dL2):
        sum += ((x1 - x2) * (x1 - x2))
    return sum ** .5
```

**Figure 1.1:** Euclidean distance

```python
def accuracy(Target_test, pre):
    sum = 0
    for x1,x2 in zip(Target_test,pre):
        if x1 == x2:
            sum +=1
    return (sum/len(Target_test))
```

**Figure 1.2:** Accuracy

```python
def Knn(Input_train, Target_train, Input_test, K):
    listD = []
    for q in Input_test:
        listD2 = []
        for x, y in zip(Input_train, Target_train):
            listD2.append((Euclidean(q, x), y))
        listD2.sort()
        nListD= []
        for z in listD2:
            nListD.append(int(z[1]))
        listD.append(Counter(nListD[:K]).most_common(1)[0][0])

    return listD
```

**Figure 1.3:** KNN

The Figures 1.1, 1.2 and 1.3, shows the full code of the working KNN algorithm, But to run this code some more code have to be done and that can be seen in Figure 1.4

```python
Data = np.loadtxt(open("diabetes.csv", "rb"), delimiter=";", skiprows=1)

Train_set, Test_set = train_test_split(Data, test_size=0.2)
Input_train = Train_set[:, :7]      # input features
Target_train = Train_set[:, 8]   # output labels
Input_test = Test_set[:, :7]
Target_test = Test_set[:, 8]

K = 3
pre = Knn(Input_train, Target_train, Input_test, K)
print(f'Accuracy:{accuracy(Target_test, pre)}')
```

**Figure 1.4:** Run the KNN with K=3

We also made a code that have cross validation and some parameter tuning and that code can be seen in Figure 1.5

```
Data = np.loadtxt(open("diabetes.csv", "rb"), delimiter=";", skiprows=1)
CVA = []
for K in range(3,8):
    for i in range(10):
        Train_set, Test_set = train_test_split(Data, test_size=0.2, random_state=i)
        Input_train = Train_set[:, :7]      # input features
        Target_train = Train_set[:, 8]   # output labels
        Input_test = Test_set[:, :7]
        Target_test = Test_set[:, 8]

        pre = Knn(Input_train, Target_train, Input_test, K)
        CVA.append(accuracy(Target_test, pre))

    CVA.sort()
    print('K = ', K, ' : ', statistics.mean(CVA) , ' +/-', int(np.std(CVA)*1000)/1000)
```

**Figure 1.5:** Cross validation with parameter tuning

The testing our code on the dataset showed a good result. with K=3 gave a accuracy of about 73% and with help of cross validation and parameter tuning the best result was almost the same as the one without parameter tuning, with this data and algorithm we are not able to achieve better result.

```
[[1.        , 0.38865855]
 [0.38865855, 1.        ]]
```

**Figure 1.6:** Correlation matrix

In Figure 1.6 we can see the correlation matrix from the result of our KNN algorithm. It shows that we can classify 0 as 0 perfectly the same for 1 as 1, but we have some misclassifying, sometimes we classify 0 as 1 and 1 as 0 and that's a 38 % that it can do it

## 1.2 Task 2

The purpose of the second task is to carry out some significant difference test. In this report the t-test has been selected as it is good when testing if two data sets are from the same distribution or not. To carry out this test two data sets have been used where one contains 50 random generated values in the range of zero to ten. This data set can be seen at the left column of table 1.1. The other data set used is containing 45 random values between zero to ten and five random generated values between 0 to 100 and this can be seen in the right column of table 1.1.
The hypothesis that will be tested in this test is to see if it is possible to say that these two data sets differ significantly. This can be interesting to see as there are five values in one of the data sets that is generated from a different distribution which means that the test will see if this is possible to detect or not. The null hypothesis of this test will be that the two distributions are the same. The significance level(alpha) is set to 5% which means that there will be a 5% probability that the null hypothesis will be rejected even if it is true.
The t-test results in a p-value of approximately 0.0423 which means that the null hypothesis will be rejected. This means that it is not possible to say with a significance level of 5% that the two data sets are equal.

| | |
|---|---|
| 15.4938185928636 | 7.82877856660123 |

| | |
|---|---|
| 57.7203682266502 | 2.20769223917607 |
| 12.3296308624523 | 4.6130449533076 |
| 86.7257434425042 | 3.23731537872843 |
| 15.3347559653543 | 5.25095414904006 |
| 9.80414046837108 | 9.20335915458963 |
| 1.48902091279811 | 3.78662869124287 |
| 1.18165192990337 | 4.34466787564573 |
| 9.10439689151401 | 7.1750290662562 |
| 8.64676165121708 | 5.9722560022896 |
| 1.59383219251718 | 6.27913883464776 |
| 3.98288482689485 | 0.079196268277779 |
| 9.39044548412291 | 2.75198877597704 |
| 0.203266236206815 | 2.58439173368476 |
| 9.02765838899281 | 7.56798199942826 |
| 7.08081547374178 | 7.4500500160891 |
| 1.98917367095459 | 3.11438137501894 |
| 1.46805651477065 | 1.40224639700556 |
| 2.11693775853778 | 4.79408181666517 |
| 7.74763064292111 | 4.39970127293829 |
| 8.62870865183286 | 1.70925868586955 |
| 6.65401255038118 | 0.583616513032744 |
| 5.55343213845899 | 1.92074513059208 |
| 2.1980606135441 | 2.69392023942655 |
| 6.50286935644344 | 2.83507466496013 |
| 0.958026245105143 | 9.03270370343821 |
| 7.64549474718895 | 7.39908828683856 |
| 4.32156991985046 | 2.75410573662216 |
| 7.01196792159436 | 2.82012798065856 |
| 3.33420540580309 | 8.52857329508517 |
| 5.43323619403153 | 5.32948556480939 |
| 6.49016383177432 | 1.2172860570074 |
| 3.1773508439439 | 2.10160379519595 |
| 2.63969053774885 | 1.78402100035046 |
| 8.20279273522769 | 2.14002003936183 |
| 5.68525168067746 | 4.59715207794756 |
| 9.04758042006095 | 7.82910736704418 |
| 4.14125193430427 | 6.62978633764226 |
| 7.56456780556786 | 4.6067792111427 |
| 8.97241454423825 | 2.12777291506682 |
| 2.54825565463779 | 6.46096202284712 |
| 2.30846327734978 | 9.84083903188022 |
| 3.43093366083391 | 0.094641546568351 |
| 8.0383978543114 | 6.02623818331415 |
| 3.83570498814478 | 5.44907714837199 |
| 4.3593594159875 | 7.97674711870369 |
| 7.56346240383968 | 2.91505732816391 |
| 7.18439479651867 | 1.51858763986422 |
| 8.04925755109233 | 2.48828881327282 |
| 5.23768544242248 | 8.51145533327088 |

**Table 1.1:** The dataset used for the t-test.

# Bibliography

[kaggle., 2020] kaggle. (2020). *diabetes.csv*. saurabh sinha. 1