DAT405 Assignment 2 – Group 95

Martin Blom - (5 hrs)
Jakob Windt - (5 hrs)

January 31, 2023

Problem 1

A

To find the linear regression model, we simply extracted the necessary data from the .CSV file, and used NumPy to plot the graph. There was no need for any data cleaning to arrive at the solution.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Load data
data = pd.read_csv('data_assignment2.csv')

area = data['Living_area']
price = data['Selling_price']

# Use NumPy to fit a linear regression model
coef = np.polyfit(area, price, 1)
poly1d_fn = np.poly1d(coef)

# Plot graph
plt.plot(area, price, 'yo', area, poly1d_fn(area), '--k')
plt.xlabel('Living area')
plt.ylabel('Selling price')
plt.title('Living area vs Selling price')
plt.show()
```

Listing 1: Python code – Problem A.

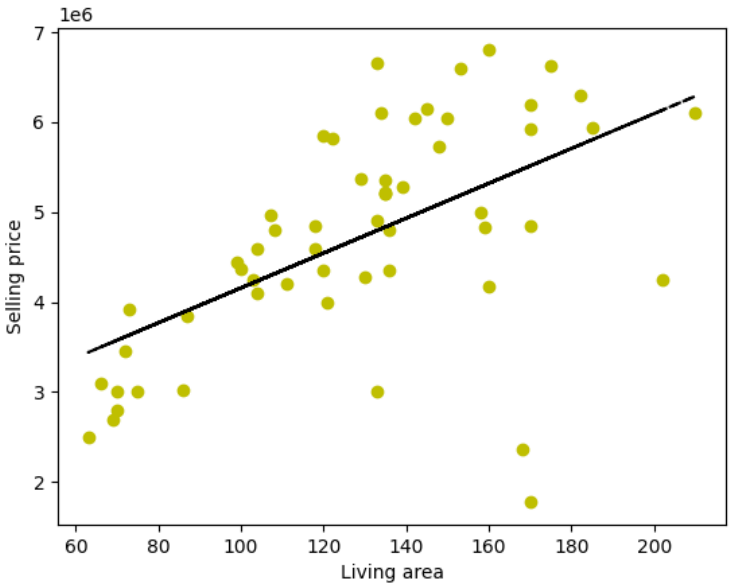


Figure 1: Linear regression model

B

To find the slope and intercept values, we simply used the NumPy polyfit function. The results are a slope of 19370 and an intercept at 2220603.

```
# Find slope and intercept
slope, intercept = coef

print("Slope: " + str(slope))
print("Intercept: " + str(intercept))
```

Listing 2: Python code – Problem B.

C

By using the NumPy poly1d function, we were able to predict the cost of different house sizes.

- 100.00 m^2: 4,157,617 kr
- 150.00 m^2: 5,126,124 kr
- 200.00 m^2: 6,094,630 kr

```
# Predict price for 100, 150, 200 sqm
print("100m^2 price: " + str(poly1d_fn(100)))
print("150m^2 price: " + str(poly1d_fn(150)))
print("200m^2 price: " + str(poly1d_fn(200)))
```

Listing 3: Python code – Problem C.

D

The graph shows the residual plot, which shows the difference between actual price and estimated price.

```
# Plot residual graph
diff = poly1d_fn(area) - price
plt.plot(area, diff, 'or')
plt.show()
```

Listing 4: Python code – Problem D.

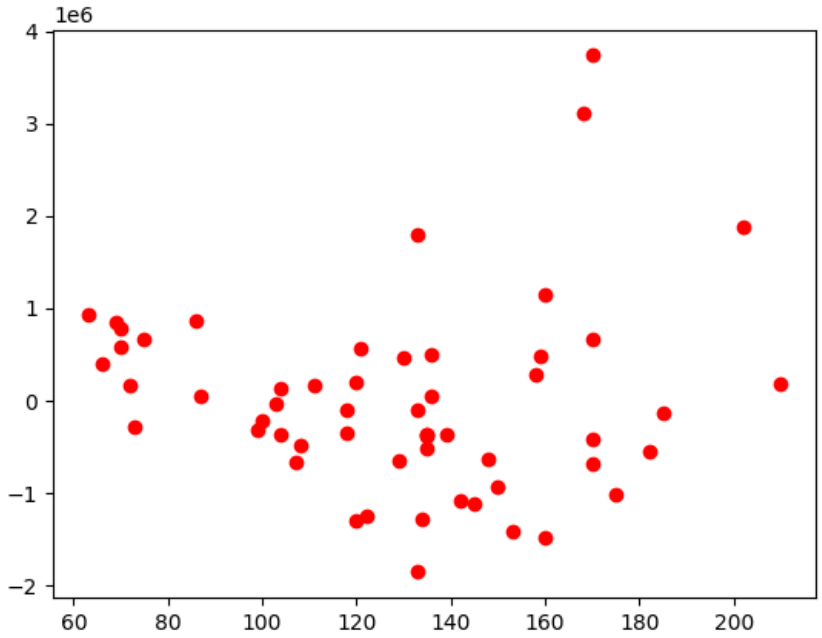


Figure 2: Residual plot

Problem 2

A

In this case the accuracy of the predictions was 100%. For the most part the training will result in almost perfect accuracy, however we've seen cases where its not. The lowest accuracy we've seen was around 85%.

```
# Load data
dataset = load_iris()
df = pd.DataFrame(dataset.data,columns=dataset.feature_names)
df['target'] = pd.Series(dataset.target)
df['target_names'] = df['target'].apply(to_target)

# Define predictor and predicted datasets
X = df.drop(['target','target_names'], axis=1).values
y = df['target_names'].values

# split taining and test set
X_train, X_test, y_train, y_test = skm.train_test_split(X, y)

# train the model at logistic regression
model_lr = LogisticRegression().fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)

# Plot confusion matrix
confusion_matrix = sk.confusion_matrix(y_test, y_pred_knn)
cm_display = sk.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix)

cm_display.plot()
plt.show()
```

Listing 5: Python code – Problem A.

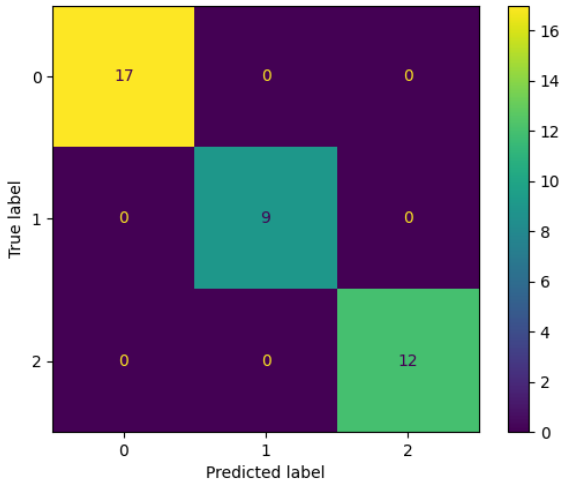


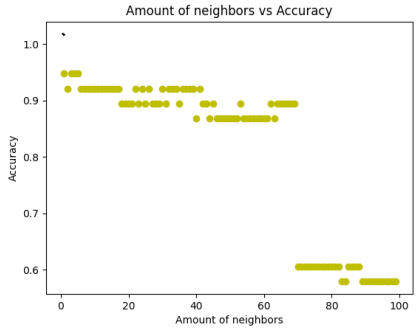
Figure 3: Confusion graph

B

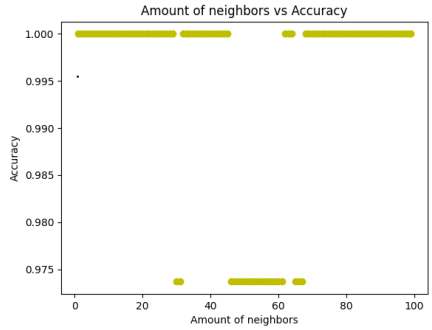
When we tested both the uniform and distance-based weights we saw that they differed quite a bit. The two graphs below show that the uniform weights drop when reaching about 70 neighbors while distance based doesn't drop in accuracy when the amount of neighbors increases. The reason behind this is because with uniform weights all points are weighted equally while distance-based weights its points depending on the distance between neighbors. The closer the point is to one another, the greater the influence it has.

```
arr = []
for x in range(1,100):
    model_knn = skn.KNeighborsClassifier(n_neighbors=x, weights="distance").fit(
    X_train, y_train)
    y_pred_knn = model_knn.predict(X_test)
    temp = model_knn.score(X_test, y_test)
    print("KNN model accuracy for " + str(x) + " neighbors: " + str(temp))
    arr.append(temp)

coef = np.polyfit(range(1, 100), arr, 1)
poly1d_fn = np.poly1d(coef)

plt.plot(range(1,100), arr, 'yo', arr, poly1d_fn(arr), '--k')
plt.xlabel('Amount of neighbors')
plt.ylabel('Accuracy')
plt.title('Amount of neighbors vs Accuracy')
plt.show()
```

Listing 6: Python code – Problem B.



(a) Uniform-based weight



(b) Distance-based weight

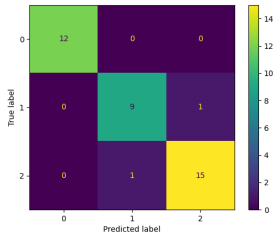
Figure 4: Graphs of prediction accuracy

C

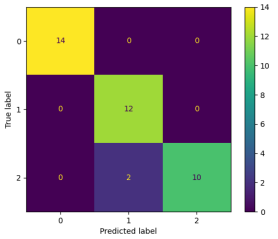
As shown in the confusion matrices below, the performance varies a lot between the different settings and classifiers. For the K-Neighbors classifier, distance-based weights perform better compared to uniform, especially when the amount of neighbors increase.

The logistic regression classifier performance the same, with a decent accuracy. Since it doesn't depend on neighbors etc, the result is always the same.

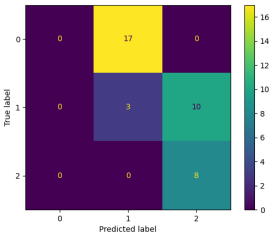
In our opinion, as shown in this lab, we recommend the use of a K-Neighbors classifier, with a distance-based weight system.



(a) 1 Neighbor

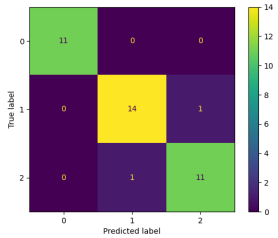


(b) 50 Neighbors

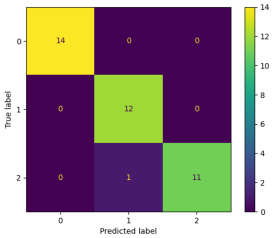


(c) 100 Neighbors

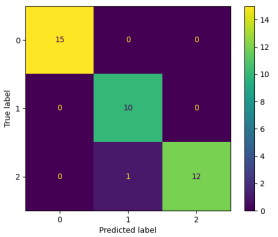
Figure 5: Uniform - 1, 50 and 100 Neighbors



(a) 1 Neighbor



(b) 50 Neighbors



(c) 100 Neighbors

Figure 6: Distance - 1, 50 and 100 Neighbors