DAT405 Assignment 2 – Group 11

Hannes Skoog - (7 hrs)
Gabriel Wendel - (7 hrs)

April 19, 2023

Problem 1 a.

First task was to fins a linear regression model for living area and selling price given a set of data.

```
df = pd.read_csv('data_assignment2.csv')
# 'x' is a ndarray containing data for living areas
x = df['Living_area'].to_numpy().reshape(-1,1) # reshaped to a single column ndarray
y = df['Selling_price']
model = LinearRegression().fit(x, y)
```

Listing 1: Linear regression for living area and selling price.

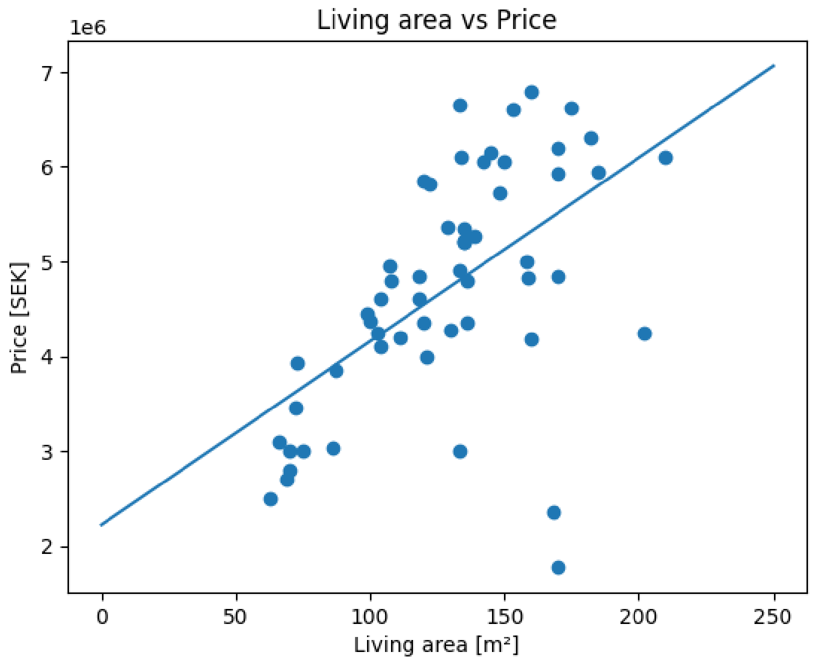

Figure 1: Linear plot.

The living area data is assigned to variable ”x”, while the selling price data is assigned to variable ”y”.

To enable comparison with ”y”, the shape of the ”x” ndarray was reshaped. The ”Model” is a linear regression model that establishes a relationship between the living area and the selling price.

Problem 1 b.

Coefficient and intersection was obtained by using *coef* and *intercept* functions:

```python
coef = model.coef_ # Finds coefficient of the linear model
interc = model.intercept_ # Interception of the linear model

print(f"Slope: {coef[0]}")
print(f"Intercept: {interc}")
```

Listing 2: Coefficient and interception for the linear regression.

Output:

```
Slope: 19370.138547331582
Intercept: 2220603.2433558684
```

Problem 1 c.

A function was constructed to predict selling prices:

```python
# Function that predicts selling prices given living area
def calc_price(area):
    return coef * area + interc

areas = [10,100,150,200,1000]

for area in areas:
    print(f"Area: {area}m2, Price: {calc_price(area)[0]} kr")
```

Listing 3: Function that predicts selling prices given living area.

Output:

```
Area: 10m2, Price: 2414304.628829184 kr
Area: 100m2, Price: 4157617.0980890263 kr
Area: 150m2, Price: 5126124.025455605 kr
Area: 200m2, Price: 6094630.952822184 kr
Area: 1000m2, Price: 21590741.79068745 kr
```

Problem 1 d.

Figure 2 illustrates a residual plot for the linear regression.

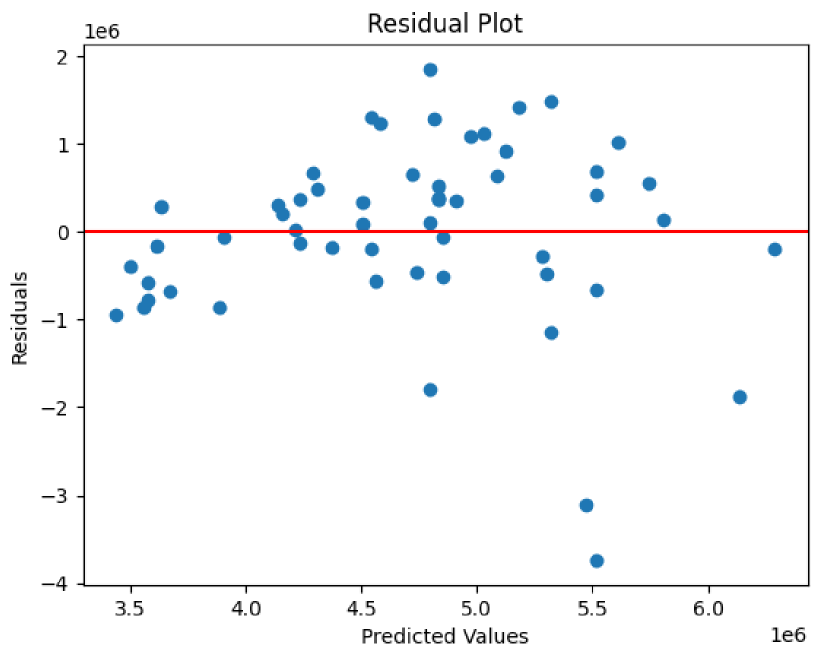

Figure 2: Residual plot.

Problem 1 e.

This model can be useful if you consider the factors that might make its use skewed. The residuals are unbalanced and we can see that some outliers have very high residuals. This essentially means that the data does not follow the model, which leads to lower usability. The big scattering of the data is likely due to that the living area isn't the only factor that is used to set the price. To make the model more usable, more factors need to be taken into account, some examples: location, age, and neighborhood. When taking these types of factors into account, the usability of the model is increased.

Another way to improve the model could be to narrow the data down. Instead of using data from all over Landvetter, data from only one block or one street would give a fairer model, since that narrows down other factors mentioned above. Making one or more factors constant for the data points would greatly narrow the model down since then you won't have to take those factors into account.

This model could be used for other areas but it most likely wouldn't make sense since that further increases the other factors' importance. For example, using this linear model in central Stockholm would make no sense since Landvetter and central Stockholm is two very different places and have very different price classes due to location, types of housing, and neighborhood. If you were to use data from central Stockholm to generate the linear model you would have a much different model, which in turn wouldn't work well in Landvetter.

Problem 2 a.

Dataframes was created for iris data and iris targets in order to scatter plot sepal length against sepal width, see Figure 3. Columns was added to the dataframe in order to display names of species in the legend.

```
# Load iris data and target values
iris = load_iris()
x = pd.DataFrame(iris.data)
y = pd.DataFrame(iris.target)

# Add columns to df and replace target values with names of species
x.columns = iris.feature_names
y = y.replace({0: 'setosa', 1: 'versicolor', 2: 'virginica'})

# Add target values to iris data
x['species'] = y

# Scatter plot data, sepal length vs sepal width
sns.scatterplot(data = x, x='sepal length (cm)', y='sepal width (cm)', hue='species')
```

Listing 4: Scatterplot for iris dataset.

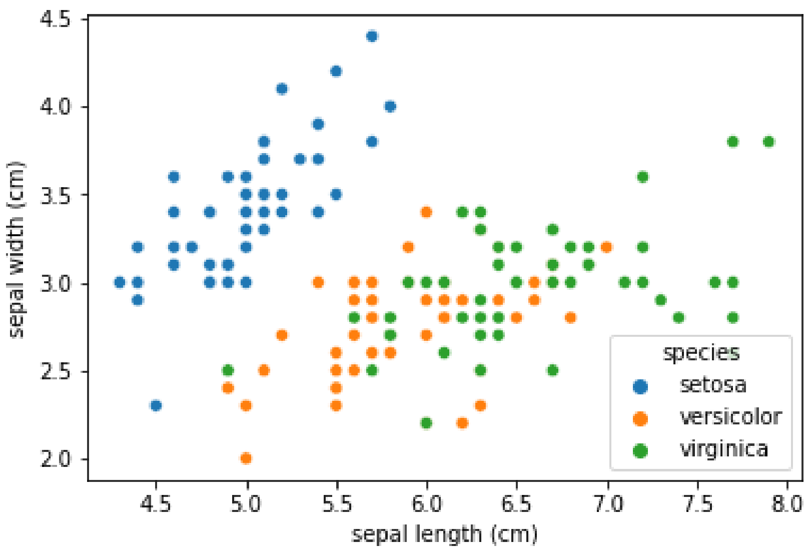

Figure 3: Scatter plot of Iris dataset.

The plot indicates that sepal width and length values for versicolor and virginica are somewhat similar.

Problem 2 b.

In order to classify the iris data set using logistic regression, the data was partitioned into training and testing datasets. The model makes a prediction and the score is recorded and printed. The performance of classification can be visualized through a confusion matrix, see Figure 4. Note that the values 0, 1 and 2 represent setosa, versicolor and virginica.

```
# Split data into test- and train data
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size
    =0.5)

# RegressitraiThe mon model
log_reg = LogisticRegression(max_iter=500)
log_reg.fit(x_train, y_train)

# Model makes prediction
prediction = log_reg.predict(x_test)

# Evaluate score
score = log_reg.score(x_test, y_test)

print('Score: {:.4f}'.format(score))

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, prediction)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
disp.plot()

plt.show()
```

Listing 5: Confusion matrix for logistic regression classifier.

```
Score: 0.9733
```

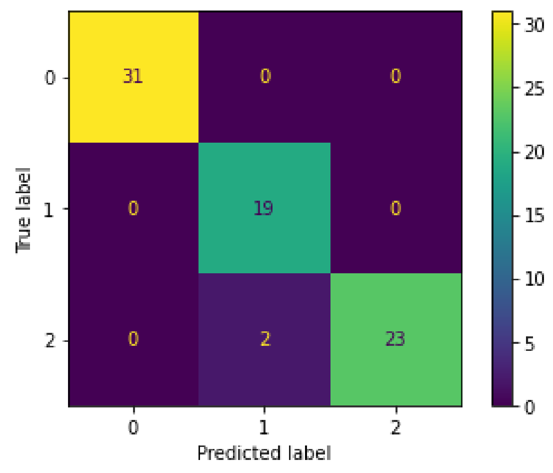

Figure 4: Confusion matrix for logistic regression classifier.

A 97% accuracy score for logistic regression on the iris dataset is quite good. It suggests that the model is able to correctly classify the majority of the data points in the dataset.

Problem 2 c.

A function was created to test different values of k and weights, including uniform and distance-based weights, to assess their impact on the classification performance.

```
# k-nearest neighbors, created a function in order to test different values for k and
    uniform and distance-based weights

def knn(k, weight):

    # Classifier
    k_nearest = KNeighborsClassifier(n_neighbors=k, weights=weight)

    # Training the regression model
    k_nearest.fit(x_train, y_train)

    # Model makes predictions
    prediction = k_nearest.predict(x_test)

    # Classifier score
    score = k_nearest.score(x_test, y_test)

    print('KNN-classifier score given k={} and {}-based weights: {:.5f}'.format(k,
    weight, score))

    # Confusion matrix for the classifier
    conf_matrix = confusion_matrix(y_test, prediction)
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
    disp.plot()
    plt.show()
```

Listing 6: Function that implements knn-classification.

The function was executed with varying values of k and both uniform and distance-based weights. Based on an analysis of the confusion matrix and scores, it can be concluded that higher values of k with distance-based weights yield the best performance.

When k grows larger the classification accuracy will first improve, but then it start to decrease as k becomes too large. This happens because as k grows larger, the algorithm considers more data points to make the classification decision, which may lead to overfitting [1].

When using uniform weights, the algorithm treats all k-nearest neighbors equally and assigns the class that is most common. This weight may work well when the data is uniformly distributed. However, when the data is skewed or imbalanced, this weigh method performs poorly.

When using distance-based weights, the algorithm assigns weights based on distance to the data point. This weight performs well when the data has clear clusters, but when the data is noisy or contains outliers, the algorithm produce unreliable results.

Problem 2 d.

The plot in Problem 2 a. indicates that the setosa species can be predicted more easily using a knn-classifier due to the clustered nature of the data. The data for versicolor and virginica species are relatively mixed together and overlaps, making it more challenging to predict using a knn-classifier.

References

[1] IBM. *What is the k-nearest neighbors algorithm?*. https://www.ibm.com/topics/knn. (Accessed: 2023-04-05).