Linear, Logistic regression and KNN

Elmar Hajizada

1 Task 1

For the first task, we gathered the living area and selling price of houses sold in Landvetter less than 6 months ago from Hemnet and wrote them as a csv file. After this, we read in the data from the csv file with x being the living area, and y being the selling price. We then fitted a linear regression model from the package sklearn.linear_model and made a prediction line yfit based on x using the following code.

```
from sklearn.linear_model import LinearRegression
    #Import data
    house = pd.read_csv('House.csv')
    #Create model
    model= LinearRegression()
    x= house['Area']
    y= house['Price']
    model.fit(x[:, np.newaxis],y)
    #Create prediction
    yfit = model.predict(x[:, np.newaxis])
```

The linear regression and the data was then plotted which can be seen in figure 1.

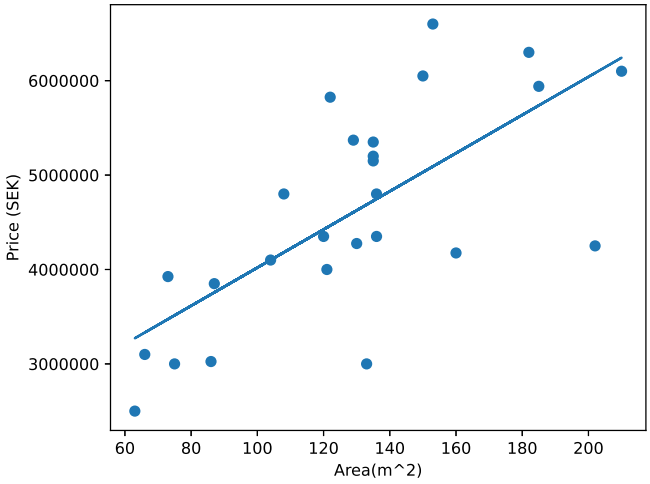

Figure 1: A scatter plot of the living area and selling prices, as well as the linear regression that came from these

We then read the slope and interception of the model using model.coef_ and model.intercept_, respectively. The slope was 20213 SEK/m^2, and the interception was 1998200 SEK. We also see in the figure that there appears to be some correlation. By calculating the correlation coefficient with the code

```
print(np.corrcoef(x,y)),
```

we find that the correlation coefficient between the living area and price is about 0.7.

We then tried to predict the prices for 100, 150 and 200m^2 living areas using the following piece of code.

```
predictions = np.linspace(100, 200, 3)
print(model.predict(predictions[:,np.newaxis]))
```

This ended up giving us that the selling price was 4019500, 5030200 and 6040900 SEK for each of the three living areas respectively.

Next, by plotting x against y-yfit instead of against y, we got a plot of the residuals which can be seen in figure 2.

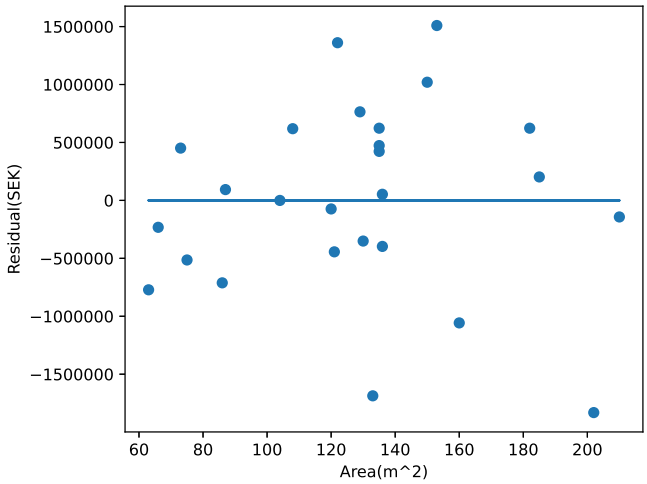

Figure 2: Plot of the residuals from the linear regression

We can see in the residual plot that the difference between the regression and the actual data is very large due to the correlation not being too close to one. The residuals reach 1.5 million SEK when the prices themselves are of a similar order. This could be because in our model, we only consider the living area and no other things like the number of rooms in the house or the land area outside of the house itself. Thus, a better approach could be to estimate the price based on multivariate linear regression with both the living area and the land area aside from the living area taken into account in the model. This could lead to a correlation between weighted sums of living and non-living area and the sales price that is larger than the correlation between living area and the sales price.

2 Task 2

For this task, we used logistic regression from sklearn to classify the iris data set also found in sklearn. To do this, we first imported the iris data set and put the data in the variable x and the target in the variable y.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from sklearn.datasets import load_iris

#Load the iris data set.
iris = load_iris()
X= iris.data
Y= iris.target
class_names = iris.target_names
```

Next, we split x and y into training and testing sets using the method train_test_split to let only 75% of the set be used to train the set, with the rest used to test the set. We fitted the logistic regression model to the training set. In order to have the model be as it was in the lectures, we let the model handle multiple classes using the "one versus rest" approach.

```
XTrain, XTest, YTrain, YTest = train_test_split(X,Y)
model= LogisticRegression(multi_class='ovr')
model.fit(XTrain,YTrain)
```

After this, we plotted a confusion matrix of the model applied to the testing set, which can be seen in figure 3.

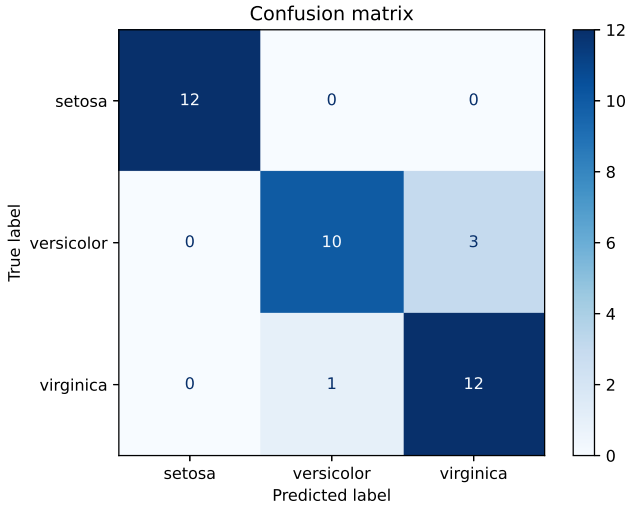

Figure 3: The confusion matrix when using logistic regression on the iris data set

We see that out of the 38 tests performed, 34 of them gave the correct answer while four of them were wrong, giving an accuracy of $\frac{17}{19} \approx 0.895$. Furthermore, the classifier only confused virginica and versicolor for each other, with versicolor having more false positives than false negatives. For versicolor, the F-measure was

$$F = \frac{2TP}{2TP + FP + FN} = \frac{20}{20 + 1 + 3} = \frac{5}{6}, \tag{1}$$

meaning that the mean of the precision and sensitivity is $\frac{5}{6}$, a good balance between the rate of positives and positives that are accurate.

3 Task 3

In this task, our objectives were the following:

- split train and test set
- build model and plot confusion matrix
- apply K-fold cross validation technique to see more general evaluation of model score
- take different k values in range, plot model accuracy with respect to K values and find optimal K.

We have used k-nearest neighbours model from sklearn library in order to classify the iris data with some different values for k and to analyze how model performance changes depending on value of k. So in the first case we imported and used default class constructor in order to create model. As a default values, k is assigned to 5 and weights are 'uniform' which simply means that all neighbours will have the same vote and we'll look only for distances between them. We also split the data into train and test set, we the took size of test set as default value 0.25 of whole data.

```
def question3_train(x_train, y_train, x_test, y_test):
    model = KNeighborsClassifier() # create model with k = 5 default value
    model.fit(x_train, y_train)
    title = "Confusion matrix with k=5 and uniform weight"
    disp = plot_confusion_matrix(model, x_test, y_test,
                                 display_labels=iris.target_names,
                                 cmap=plt.cm.Blues)
    disp.ax_.set_title(title)
    plt.savefig('confusion_matrix_default.png')
    plt.show()
```

So in the first try, we plotted our confusion matrix on test set so that we can see how was our model performance.

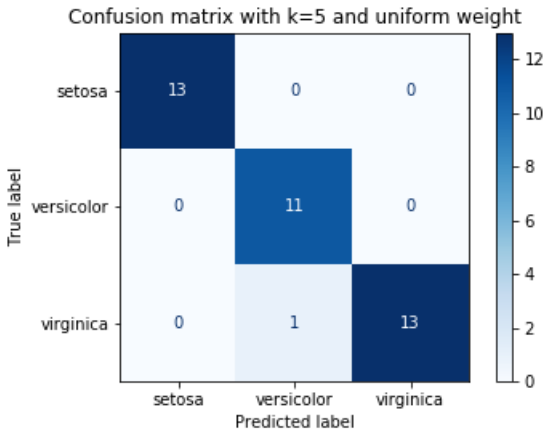

Figure 4: Plot of the confusion matrix from the KNN model with k=5

In order to have more generalized score we tried to apply K-fold cross validation technique where data randomly partitioned into n overlapping sets. One block becomes test set and others become train set. We repeat this operation n times. So in this task we took n as 5 and on the other hand for each partitioned we put K values in a range between 1 and 31. So what we are looking was what is the optimal value with highest accuracy.

```
def find_optimal_k(x, y, weights='uniform'):
    '''
    apply k-fold cross validation and take different k, plot the error to the corresponding k and find ↩
        optimal k.
    '''
    k_range = list(range(1, 31))
    k_scores = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k, weights=weights)
        score = (cross_val_score(knn, x, y, cv=5, scoring='accuracy'))
        k_scores.append(score.mean())
    k_scores, k_range = zip(*sorted(zip(k_scores, k_range)))
    optimal_k = k_range[-1]
```

In order to see how was our model performance based on each k value, we plotted the model accuracy corresponding to each k values. Moreover, we trained the model both with uniform and distance weights. So we plotted the accuracy with its corresponding k values. As can be seen from the graph K values such as 10, 11, 12 are optimal K values for both uniform and distance weights. After K value 12 model performance started to decrease based on accuracy. The number K should be chosen wisely, If we give k = 1, model would act like an over fitting and it gives non-smooth decision boundary. However, if we put k as higher value such as 30, model would be under fitting and decision boundary will be smooth. So we choose k by giving different values and look for the best accuracy as described in plots.

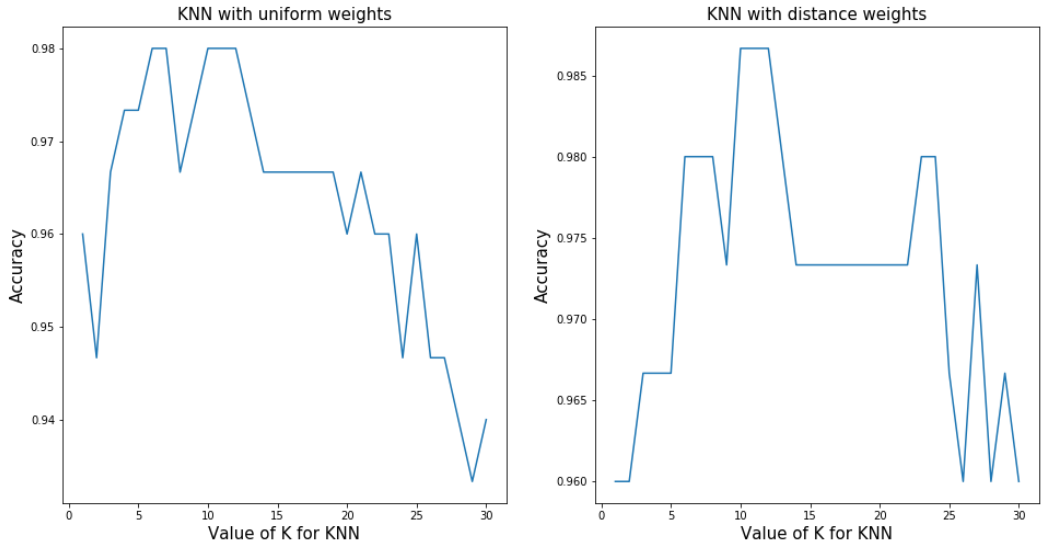

Figure 5: Plot of the confusion matrix from the KNN model with k=5

As we mentioned before we tried with different values of k and used different combination of k with both uniform and distance weights. For instance lets take a label for arbitrary point in the test data as 0 and k is 10, so if we look at our 10 closes points we can for instance come up with a situation where 6 of points are labelled as 1 and other 4 points are labelled as 0. So as a result of this model we would have answer for desirable point as 1, but the result is wrong indeed. This is because when the weights are uniform, model takes the label with high frequency in the neighbourhood. But if the weights would be distance based, the label for that desirable point could be 0 because actually the 4 points, which are labelled as 0, are closest points to our desirable point. So by giving weights to our distances we can reach out to true label. The points that are closest to our desirable point, are given more weights.

4 Task 4

We can differentiate regression and K Nearest Neighbour based on the way of algorithm that they have. Both of them are used for classification but K Nearest Neighbour looks for k closest neighbours and votes are taken among all neighbours. The class occurring with highest number of time is assigned to that class. However, logistic regression uses sigmoid function to predict probabilities of classes for desired point.

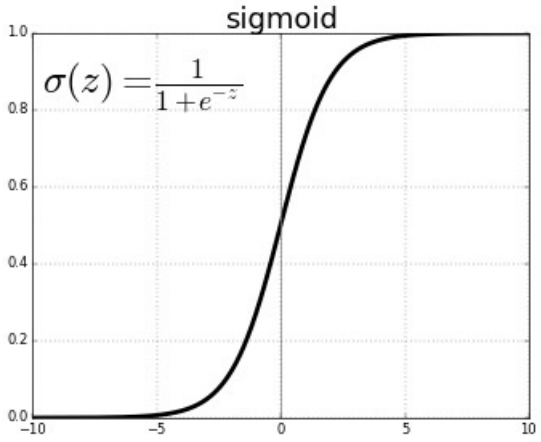

Figure 6: Graph of the sigmoid function

Moreover, it should be mentioned out that training of the model is slower for Logistic regression than KNN, since you need to optimise the likelyhood of the data appearing from the regression while doing Logistic regression, while you just need to store the data points and target values for KNN. However, since Logistic regression only calculates the value of one function per class when classifying a point while KNN requires the distance to every training point to be calculated, KNN is slower when actually classifying data points. We can sum it up by saying that Logistic regression uses eager learning, which generalises the data before receiving a point, while KNN uses lazy learning to only generalize the data when it gets a point.

By looking at the confusion matrices we can say that there is not huge difference between these models, but KNN with optimal k value achieved better accuracy rather than Logistic regression. There are more false positives and false negatives in logistic regression according to confusion matrix which is shown in Figure 3 and Figure 4.

5 Task 5

The reason why it is important to let the testing set be separate from the training set is that it is already clear that the training set will be classified correctly by the classifier. Generally, when we classify sets, we want to be able to classify more sets than just the sets that used to train the set, meaning that in order to see if our model can classify other data point that is not in the training set, we need to test a set which is separate from the training set.