

assignment_2_final

September 14, 2021

1 Assignment 2: Regression and classification

1.0.1 Worked hours: Sidner 15h, Lukas 15h

1.0.2 Authors: Sidner Magnéli, Lukas Martinsson

```
[48]: # global imports and configuration
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# set plot style
sns.set_style("darkgrid")
```

1.1 Part 1: Hemnet dataset

The dataset associated to this assignment was downloaded from www.hemnet.se on 2020-10-18. The data contains information about selling prices of villas in Landvetter that were sold in the past 12 months

Load and inspect dataset One immediately notice that the Biarea column contains NaN values as well as the presence of outliers in the scatterplot which should probably be addressed.

```
[49]: df_raw = pd.read_csv('datasets/data_assignment2.csv')

#Checks that there are no villas have 0 or negative number of rooms, since they
→would be errors in the data. Same could be done for other columns but seems
→redudant.
print((df_raw['Rooms'] < 1).any())

df_raw.head(5)
```

False

```
[49]:
```

	ID	Living_area	Rooms	Land_size	Biarea	Age	Selling_price
0	1	104	5.0	271.0	25.0	33	4600000
1	2	99	5.0	1506.0	6.0	88	4450000
2	3	133	6.0	486.0	NaN	44	4900000
3	4	175	7.0	728.0	NaN	14	6625000
4	5	118	6.0	1506.0	NaN	29	4600000

```
[50]: # Show an overview of the data before making a linear regression
ax = sns.scatterplot(x=df_raw['Selling_price'], y=df_raw['Living_area'])
ax.set_title('Living area vs Selling price');
```



1.1.1 1.a) Find a linear regression model that relates the living area to the selling price. If you did any data cleaning step(s), describe and explain why you did that.

```
[51]: # Remove any rows where either of the two features are NaN, as they only
      ↪ provide useful insight together.
df = df_raw.dropna(how='any', subset=['Living_area', 'Selling_price'])

#Alternative solution (was not use for the sake of learning)
#sns.regplot(x="Living_area", y="Selling_price", data=df);

# Select features for the model
```

```

x_area = df.loc[:,['Living_area']].values
y_selling = df['Selling_price'].values

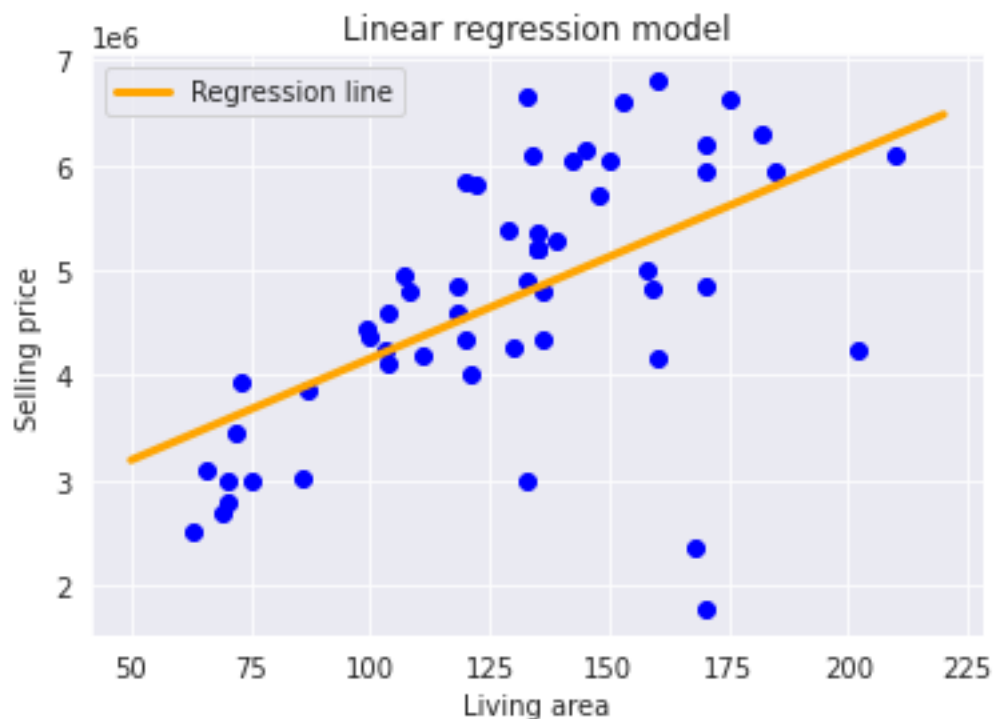
# Create an linear regression model and fit it
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(x_area,y_selling)

x_fit = np.array([50,220])
y_fit = model.predict(x_fit[:,np.newaxis])

# Plotting the model
plt.scatter(x_area,y_selling, color='blue')
plt.plot(x_fit,y_fit,color='orange', linewidth=3, label = 'Regression line')

plt.xlabel("Living area")
plt.ylabel("Selling price")
plt.title('Linear regression model')
plt.legend()
plt.show()

```



As mentioned above, the scatterplots show that the dataset contains a couple of outliers. Depending on whether these are errors or not, they could either skew the data, decreasing the accuracy of the model, or provide valuable insight into the nature of the data. In short *Sometimes outliers are bad data, and should be excluded, such as typos. Sometimes they are Wayne Gretzky or Michael*

Jordan, and should be kept. Hence, simply removing outliers might actually harm the accuracy of the model, under the assumption that they are valid observations. A way to address this dilemma is to use a different model, such as **Huber regression** which accounts for outliers by assigning them less weight than other observations in the dataset.

We chose to remove the outliers to illustrate the differences. We originally wanted to filter outliers by at least three standard deviations, however there were no such points. Therefore, we limited ourselves to two standard deviations, this was mainly to illustrate the point of outliers even if one could argue that in this case, there are no outliers.

Furthermore, we removed every NaN value for 'Living area' and 'Selling price' as such data could possibly influence the data.

```
[52]: # Getting the mean and std
x_mean = df['Living_area'].mean()
y_mean = df['Selling_price'].mean()
x_2std = df['Living_area'].std()*2
y_2std = df['Selling_price'].std()*2

df_area = df['Living_area']
df_price = df['Selling_price']

# Creating two dataframes, one with the outliers and one without
df_outlier = df[((df_area > x_mean + x_2std) | (df_area < x_mean - x_2std) |
    →(df_price > y_mean + y_2std) | (df_price < y_mean - y_2std))]
df_cleaned = df[((df_area < x_mean + x_2std) & (df_area > x_mean - x_2std) &
    →(df_price < y_mean + y_2std) & (df_price > y_mean - y_2std))]

# As before, we select the features and create a model
x_area = df_cleaned.loc[:,['Living_area']].values
y_selling = df_cleaned['Selling_price'].values

x_area_outlier = df_outlier.loc[:,['Living_area']].values
y_selling_outlier = df_outlier['Selling_price'].values

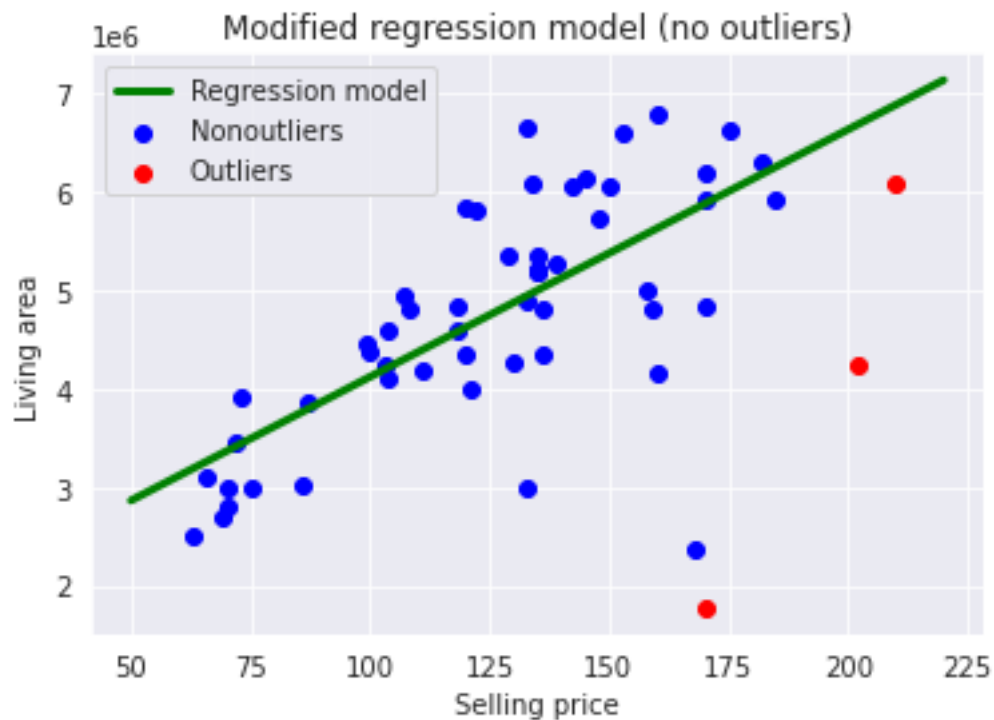
model2 = LinearRegression().fit(x_area,y_selling)

xfit = np.array([50,220])
yfit = model2.predict(xfit[:,np.newaxis])

# plot data with outliers
plt.scatter(x_area,y_selling, color='blue', label = 'Nonoutliers')
plt.scatter(x_area_outlier,y_selling_outlier, color='red', label = 'Outliers')
plt.plot(xfit,yfit,color='green',linewidth=3, label = 'Regression model')

plt.xlabel("Selling price")
plt.ylabel("Living area")
plt.title("Modified regression model (no outliers)")
```

```
plt.legend(loc='upper left')
plt.show()
```



As one can see, three outliers were filtered out, and the regression line adjusted appropriately.

1.1.2 1.b) What are the values of the slope and intercept of the regression line?

```
[53]: # Slope of the line:
print('Slope of the line is:', model2.coef_)

# Intercept of the line:
print('Intercept at:', model2.intercept_)
```

Slope of the line is: [25160.26380939]

Intercept at: 1606573.8212559996

1.1.3 1.c) Use this model to predict the selling prices of houses which have living area 100 m2, 150 m2 and 200 m2.

```
[54]: # Create an array with the specified living areas
values = np.asarray([100,150,200])

# Let the model predict the selling prices for those input values
```

```

value = model.predict(values[:,np.newaxis])

print('Selling price at 100 m^2:', int(value[0]) )
print('Selling price at 159 m^2:', int(value[1]) )
print('Selling price at 200 m^2:', int(value[2]) )

```

Selling price at 100 m²: 4157617
 Selling price at 159 m²: 5126124
 Selling price at 200 m²: 6094630

1.1.4 1.d) Draw a residual plot.

```

[55]: # Using seaborn a residual plot was created
ax = sns.residplot(x=x_area,y=y_selling, lowess=True, color="g")
ax.set_title('Residual plot');

```



1.1.5 1.e) Discuss the results, and how the model could be improved.

```

[56]: plt.scatter(x_area,y_selling, color='blue', label = 'Nonoutliers')
plt.scatter(x_area_outlier,y_selling_outlier, color='red', label = 'Outliers')
plt.plot(xfit,yfit,color='green', label = 'Regression model (mod)')
plt.plot(x_fit,y_fit,color='orange', label = 'Regression model')

plt.xlabel("Selling price")

```

```

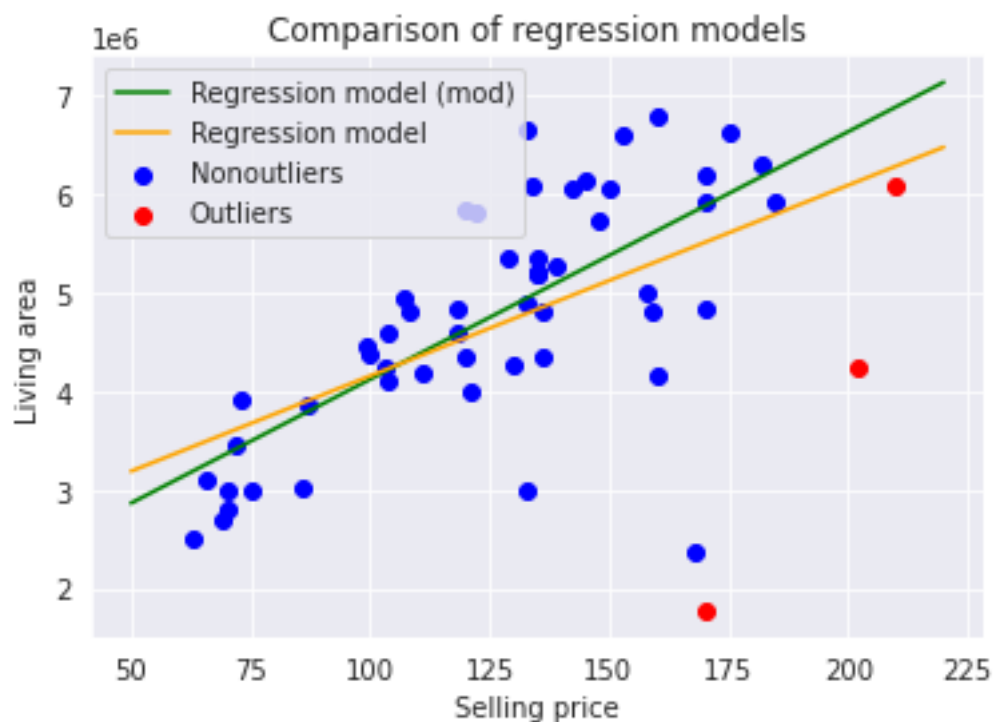
plt.ylabel("Living area")

#Slope of the lines:
print('Slope of the line is (original):', model.coef_)
print('Slope of the line is (modified):', model2.coef_)

#Intercept of the lines:
print('Intercept at (original):', model.intercept_)
print('Intercept at (modified):', model2.intercept_)
plt.title('Comparison of regression models')
plt.legend()
plt.show()

```

Slope of the line is (original): [19370.13854733]
 Slope of the line is (modified): [25160.26380939]
 Intercept at (original): 2220603.243355869
 Intercept at (modified): 1606573.8212559996



By removing outliers, we can see that the regression lines slope shifts and intercepts at different point. Depending on the selling price/living area one wants to predict, these two different lines can give different predictions. It is hard to show which one is better than the other without using test data but it illustrates how difficult it is to choose which points are outliers and which are not. As mentioned above in **1.a**), using a more robust model, such as **Huber regression** might also yield better results.

Furthermore, as has been touched on above, adding additional features that one would expect to have an effect on the selling prices may also improve the performance of the model. For example by combining the biarea with the living area.

Also plotting the average residual line (setting lowess) gives an indication if the data conforms to random errors or not, which could be crucial in determining whether the model is valid

```
[57]: # Create an array with the values chosen
values2 = np.asarray([75, 100, 150, 200])

# Let the model predict the selling prices at those points
value1 = model.predict(values2[:,np.newaxis])
value2 = model2.predict(values2[:,np.newaxis])
model_type = ['model', 'model2']

print('Selling price at 75 m^2:', ('Original:'), int(value1[0]), ('Modified:'),
      ↪int(value2[0]) )
print('Selling price at 100 m^2:', ('Original:'), int(value1[1]), ('Modified:
      ↪'), int(value2[1]) )
print('Selling price at 150 m^2:', ('Original:'), int(value1[2]), ('Modified:
      ↪'), int(value2[2]) )
print('Selling price at 200 m^2:', ('Original:'), int(value1[3]), ('Modified:
      ↪'), int(value2[3]) )
```

```
Selling price at 75 m^2: Original: 3673363 Modified: 3493593
Selling price at 100 m^2: Original: 4157617 Modified: 4122600
Selling price at 150 m^2: Original: 5126124 Modified: 5380613
Selling price at 200 m^2: Original: 6094630 Modified: 6638626
```

As a final note: As the code above illustrates the predictions can be extremely close or far apart depending on the selling price chosen. This goes to show that discarding outliers can have a large effect on the final model and it's prediction, and should be done with care.

1.2 Part 2: Iris dataset

In this question, you will use the Iris data set (“from sklearn.datasets import load_iris”)

1.2.1 2.a) Use a confusion matrix to evaluate the use of logistic regression to classify the iris data set.

```
[58]: from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

# create the df from load_iris
iris = load_iris()
```



```

X = iris.data
y = iris.target

# splits it into test and training with a test of 25% of the total
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0,
    ↳test_size=0.25)

# generate a logistic regression multiclass model
# setting multi_class='ovr' uses the One vs Rest schema, for which a separate
    ↳model is trained for each class
# predicting whether an observation is that class or not (thus making it a
    ↳binary classification problem).
model = LogisticRegression(solver='liblinear', multi_class='ovr').
    ↳fit(X_train,y_train)

# make classification
y_pred = model.predict(X_test)

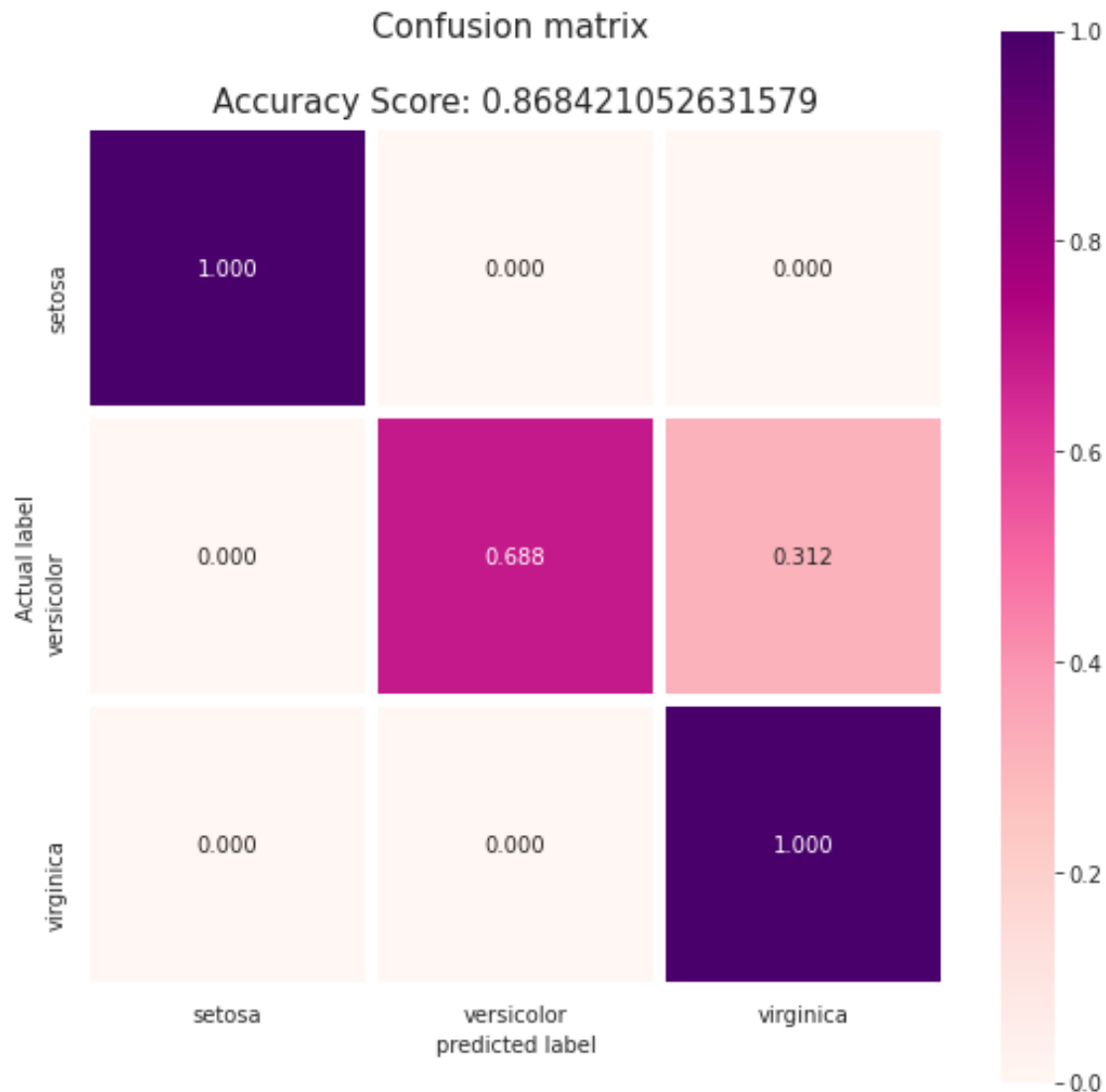
# calculate model score (accuracy)
score = model.score(X_test, y_test)

# create a confusion matrix and plots it
cm = confusion_matrix(y_test, y_pred, normalize='true')

# plot confusion matrix
fig, ax = plt.subplots(figsize=(9,9))
sns.heatmap(cm, annot = True, fmt=".3f", linewidths=5, square=True, cmap='RdPu',
    ↳)
ax.set_xticklabels(iris.target_names)
ax.set_yticklabels(iris.target_names)
all_sample_title = 'Confusion matrix \n\nAccuracy Score: {0}'.format(score)
plt.ylabel('Actual label')
plt.xlabel('predicted label')
plt.title(all_sample_title, size = 15)

plt.show()

```



As the confusion matrix shows the accuracy is 87%. Both **setosa** and **virginica** can be predicted with 100% accuracy, however **versicolor** is not as accurate, only at 69%, misinterpreted as **virginica** 31% of the times.

1.2.2 2.b) Use k-nearest neighbours to classify the iris data set with some different values for k, and with uniform and distance-based weights. What will happen when k grows larger for the different cases? Why?

```
[59]: from sklearn.neighbors import KNeighborsClassifier
      from sklearn import metrics

      # Creating a for loop that create a plot with different weight and k to
      # illustrate what happens when k grows larger
```

```

# and how it differs between 'distance' and 'uniform'
plt.figure(figsize=(10,10))
plt.xlabel('Number of k')
plt.ylabel('Accuracy Score')
plt.title('Accuracy for different numbers of k', size = 15)

l = list(range(1, 113))
for weight in ['distance','uniform']:
    accuracy_score = []
    for i in l:
        # Create KNN Classifier
        knn = KNeighborsClassifier(n_neighbors=i,weights=weight )

        #Train the model using the training sets
        knn.fit(X_train, y_train)

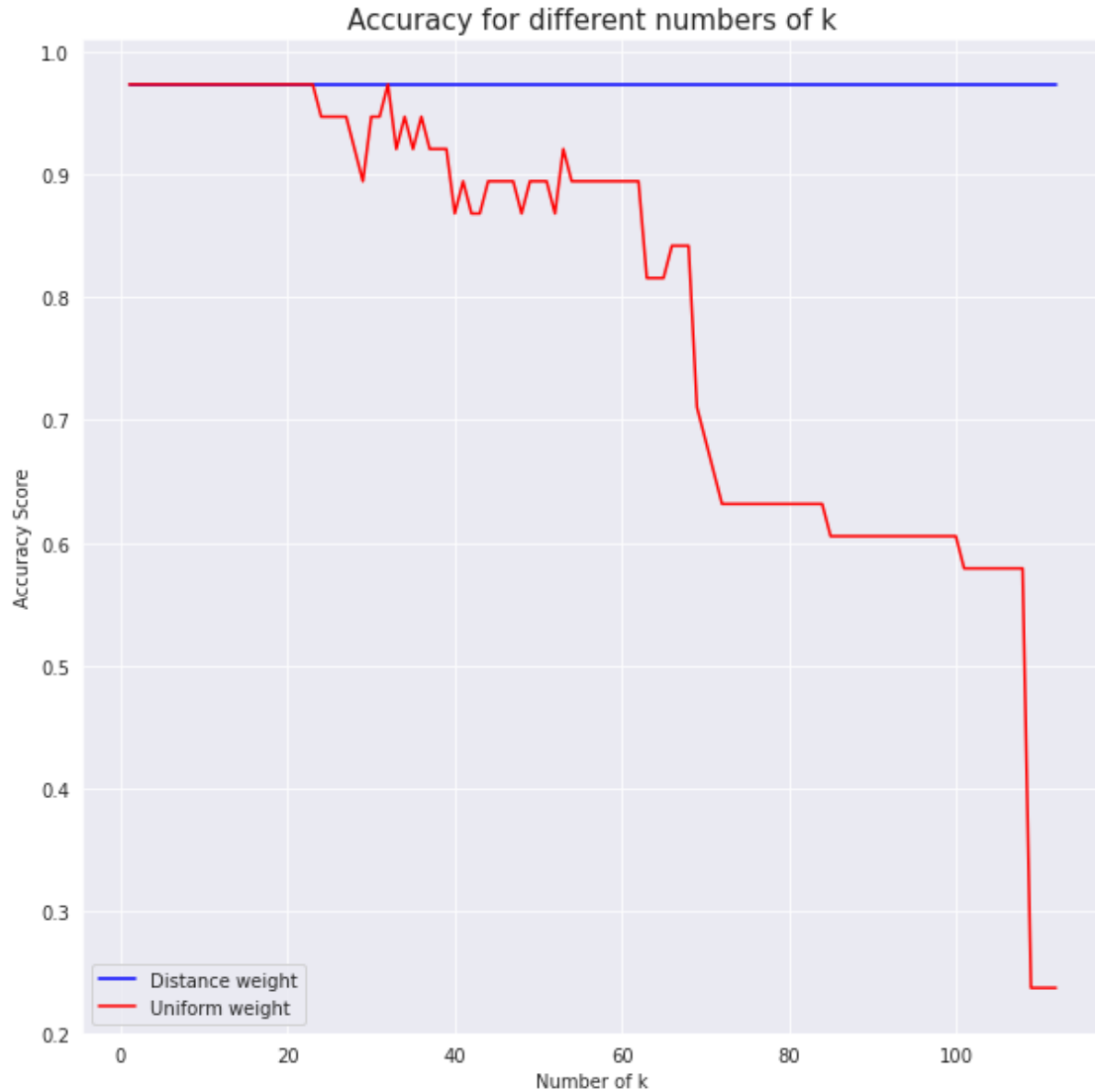
        #Predict the response for test dataset
        y_pred = knn.predict(X_test)

        # Model Accuracy, how often is the classifier correct?
        accuracy_score.append(metrics.accuracy_score(y_test, y_pred))

    if weight == 'uniform':
        plt.plot(l,accuracy_score, label = 'Uniform weight', color = 'red')
    else:
        plt.plot(l,accuracy_score, label = 'Distance weight', color = 'blue')

plt.legend()
plt.show()

```



As the plot illustrates the distance based weight gives a linear and high accuracy for every k. Uniform based however differs greatly depending on k. Distance based is probably better through and through since it makes more sense to give a closer k a higher weight then a k far away. this is especially true when the k is high. This is because a high k probably results in a k really far away making as much a contribution to the prediction as a close k. Therefore, the uniform weight loses accuracy the higher k becomes.

1.2.3 2.c) Compare the classification models for the iris data set that are generated by k-nearest neighbours (for the different settings from question 2b) and by logistic regression. Calculate confusion matrices for these models and discuss the performance of the various models.

Here we chose to create 4 different models by k-nearest neighbors. 2 uniform weight based and 2 distance weight based with the k value set to 35 and 75 respectively

```
[60]: #Create KNN Classifier
uniform35 = KNeighborsClassifier(n_neighbors=35,weights='uniform')
uniform75 = KNeighborsClassifier(n_neighbors=75,weights='uniform')
distance35 = KNeighborsClassifier(n_neighbors=35,weights='distance')
distance75 = KNeighborsClassifier(n_neighbors=75,weights='distance')

#Train the models using the training sets
for classifier in [uniform35, uniform75, distance35, distance75]:
    classifier.fit(X_train, y_train)

# predict the response for test datasets
uniform35_predict = uniform35.predict(X_test)
uniform75_predict = uniform75.predict(X_test)
distance35_predict = distance35.predict(X_test)
distance75_predict = distance75.predict(X_test)

# model accuracy, how often is the classifier correct?
uniform35_score = uniform35.score(X_test,y_test)
uniform75_score = uniform75.score(X_test,y_test)
distance35_score = distance35.score(X_test,y_test)
distance75_score = distance75.score(X_test,y_test)

# create confusion matrixes
uniform35_cm = confusion_matrix(y_test, uniform35_predict, normalize = 'true')
uniform75_cm = confusion_matrix(y_test, uniform75_predict, normalize = 'true')
distance35_cm = confusion_matrix(y_test, distance35_predict, normalize = 'true')
distance75_cm = confusion_matrix(y_test, distance75_predict, normalize = 'true')

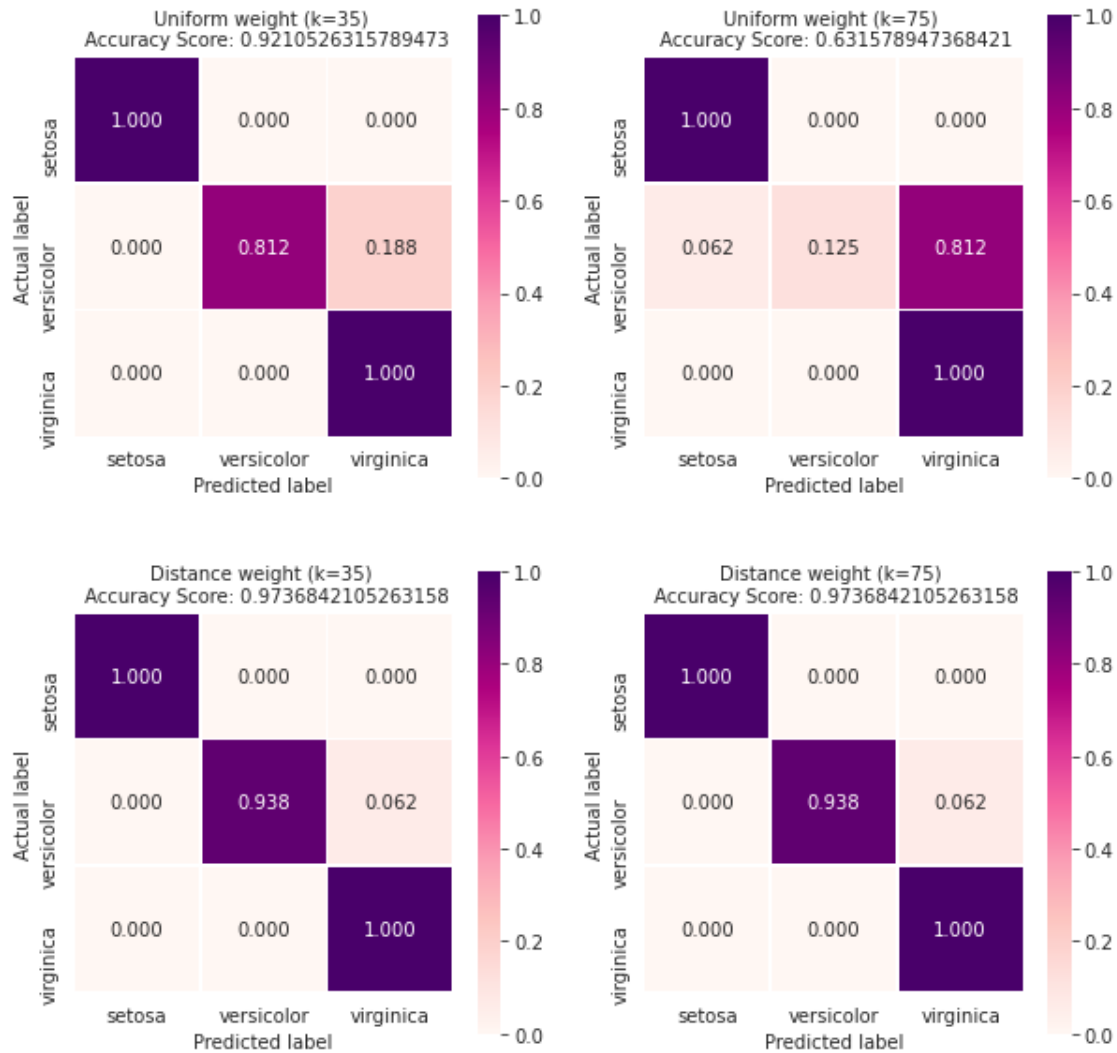
# plot confusion matrixes
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize=(10,10))
sns.heatmap(uniform35_cm, ax = ax1, annot = True, fmt=".3f", linewidths=.5,
    ↳square=True, cmap='RdPu' )
sns.heatmap(uniform75_cm, ax = ax2, annot = True, fmt=".3f", linewidths=.5,
    ↳square=True, cmap='RdPu' )
sns.heatmap(distance35_cm, ax = ax3, annot = True, fmt=".3f", linewidths=.5,
    ↳square=True, cmap='RdPu' )
sns.heatmap(distance75_cm, ax = ax4, annot = True, fmt=".3f", linewidths=.5,
    ↳square=True, cmap='RdPu' )

# set labels
ax1.set_xlabel('Predicted label')
ax1.set_ylabel('Actual label')
ax1.set_title('Uniform weight (k=35) \n Accuracy Score: {0}'.
    ↳format(uniform35_score), size=10)
ax1.set_xticklabels(iris.target_names)
ax1.set_yticklabels(iris.target_names)
ax2.set_xlabel('Predicted label')
```

```

ax2.set_ylabel('Actual label')
ax2.set_title('Uniform weight (k=75) \n Accuracy Score: {0}'.
    ↪format(uniform75_score), size=10)
ax2.set_xticklabels(iris.target_names)
ax2.set_yticklabels(iris.target_names)
ax3.set_xlabel('Predicted label')
ax3.set_ylabel('Actual label')
ax3.set_title('Distance weight (k=35) \n Accuracy Score: {0}'.
    ↪format(distance35_score), size=10)
ax3.set_xticklabels(iris.target_names)
ax3.set_yticklabels(iris.target_names)
ax4.set_xlabel('Predicted label')
ax4.set_ylabel('Actual label')
ax4.set_title('Distance weight (k=75) \n Accuracy Score: {0}'.
    ↪format(distance75_score), size=10)
ax4.set_xticklabels(iris.target_names)
ax4.set_yticklabels(iris.target_names)
plt.show()

```



As one can see, the confusion matrix for the distance-based weight is the same while the uniform based differs. This more or less illustrates the same thing as the plot above however this shows 4 concert examples of this. Furthermore, this demonstrates that only versicolor is the one to get predicted wrongly, indicating that its points are probably less cluttered together then the other 2. One could argue that since it usually gets wrong assumed to be virginica it has to the same must be true for virginica, but since the accuracy is 100% across every confusion matrix the only conclusion is that virginica points are close together and versicolor is not, however the versicolor points are probably closer to virginica than setosa.

Lastly when comparing knn to logistic regression (2a) one could see that knn has a higher accuracy 97% (at least if we use distances weights which vastly outperforms uniform on higher k values) compared to logistic regression which only had a accuracy of 87%

1.3 Part 3: Explain why it is important to use a separate test (and sometimes validation) set

Since when training a model, one usually tweaks it and for example removes outliers to improve accuracy. And let's say that one was not to use a test set then the model after tweaking could theoretically achieve 100% accuracy on the set used. However, if one were to use it as a prediction model for another set the accuracy could be extremely low. Instead by using a test set one could tweak the model on the training set and then test it on test set. Then go back to the training set and tweak it further (which is basically a validation set, it seems that test and validation set are used somewhat lose). After a few iterations the model would then be able to more accurately predict data on both the test set and thus also new data.

Furthermore, when designing, let's say a classification system, data scientists usually evaluate the performance of several different models with the validation dataset before selecting the best candidate. However, if they were to also evaluate the real-world performance of the system using that same dataset, the system would score much higher than if it were to be presented with new data as the model was selected because it performed well on that specific dataset.