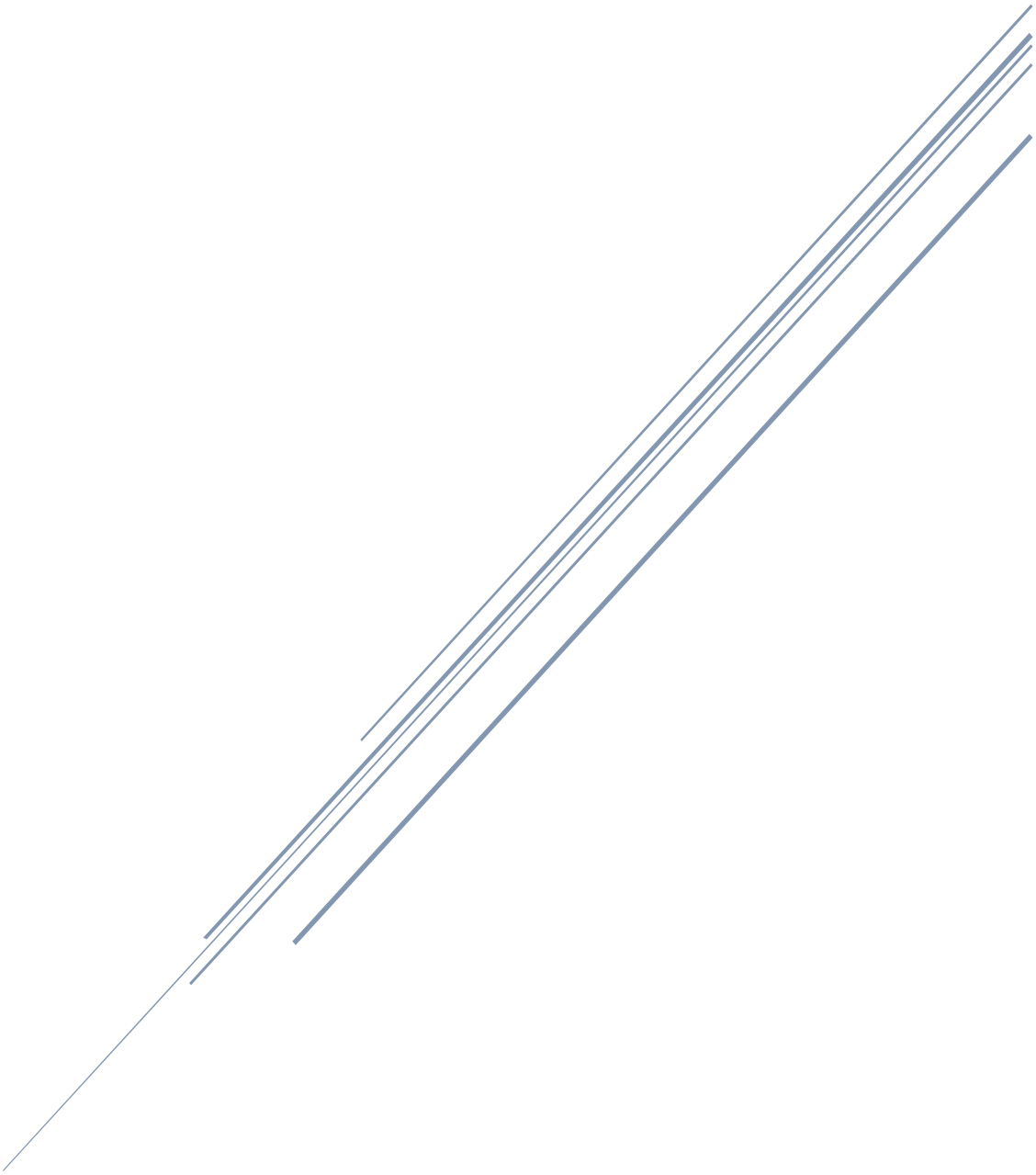
ASSIGNMENT 01

Machine Learning – Forår 2023



Erhvervsakademiet Aarhus
Mads Bjerg

Contents

Exercise 1

Exercise 1 - A

NN MLPClassifier med 12/8/4 neurons I HL:

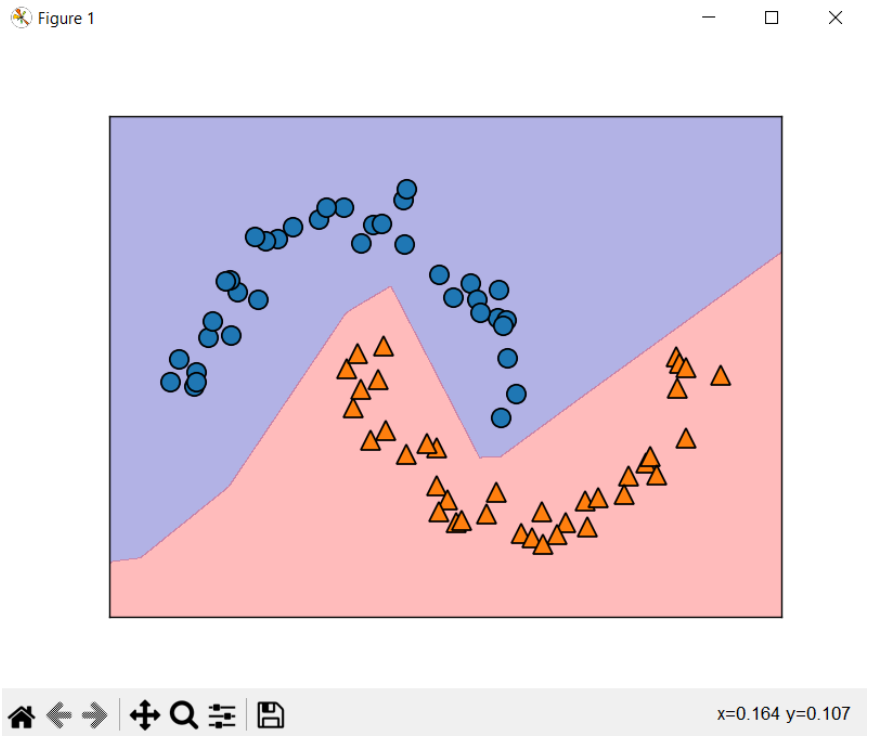


Figure 1 NN MLP 3 hidden layers

As seen on figure 1 is the plot of a neural network using MLPClassifier from sklearn, with three hidden layers. The layers are structured as follows.

First layer of 12 neurons.

Second layer of 8 neurons.

Third layer of 4 neurons.

Which to me seemed to give a good degree of precision while still maintaining a decent performance.

With 10 neurons in each of three hidden layers, it lost some of its as some blue started being in the red zone and some reds started being in the blue zone. Though the original solution of 2 hidden layers seemed to work alright, I'd prefer to use the solution above seeing as it's a bit more conforming to the shape of the moons, without reaching the territory of overfitting.

Exercise 1 - B

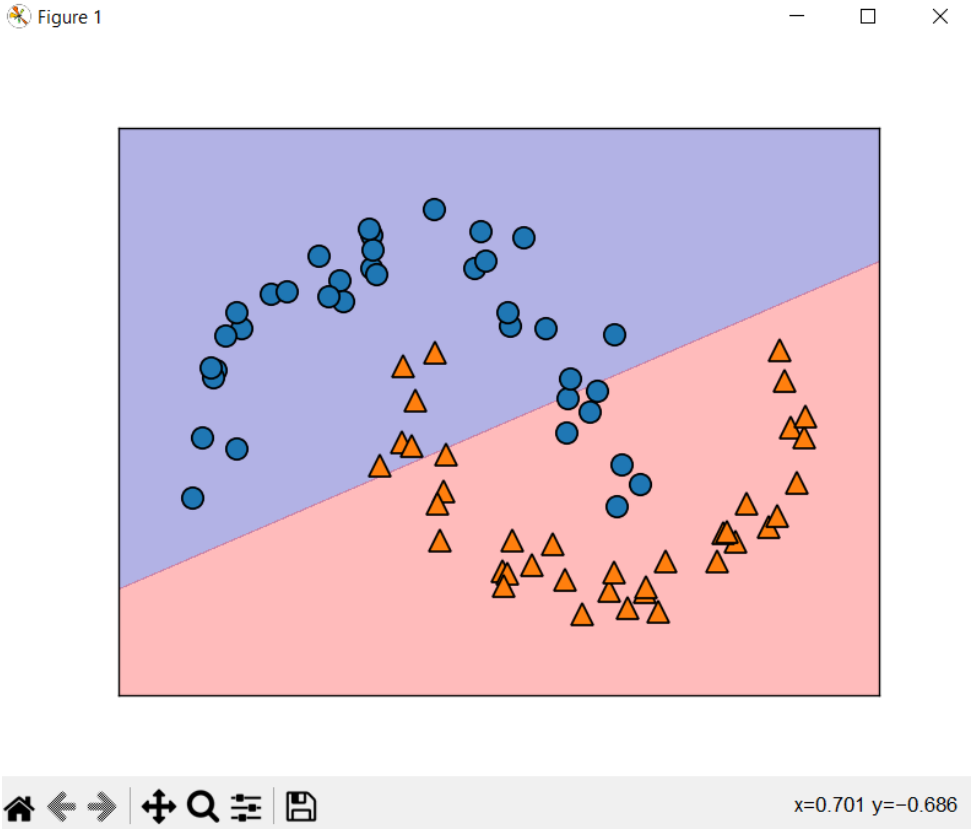


Figure 2 Logistic Regression moons attempt

As seen on figure 2 logistic regression will just try to draw a line across the two moons, however since the moons are overlapping, it is unable to make a good solution, and the plot on figure 2 is the result of this.

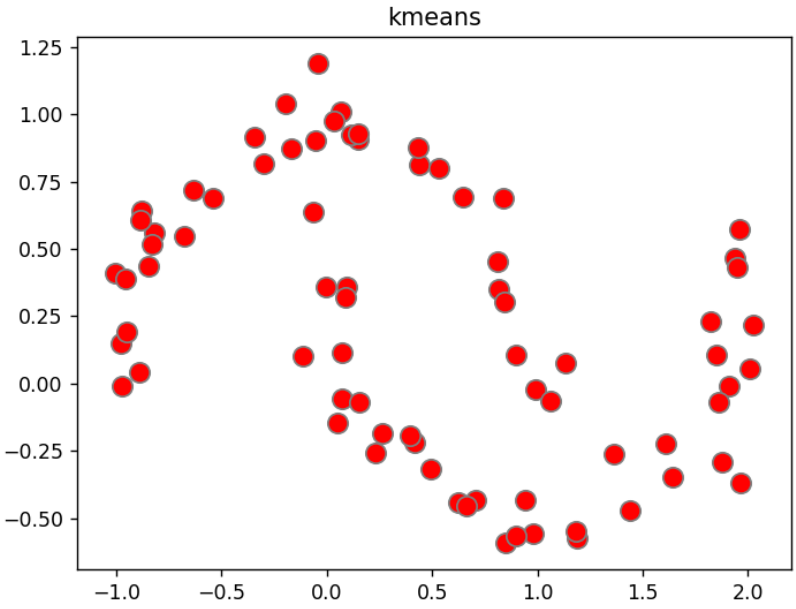


Figure 3 kmeans set to 2 clusters

As seen on figure 3, despite having given the parameter n_clusters the value 2 when creating the kmeans classifier, it is still adamant on making only one cluster. I do have a suspicion this is because I have made an error when making the colormapping though, as when I've talked to peers doing the same exercise, they have indeed been able to make it do two clusters.

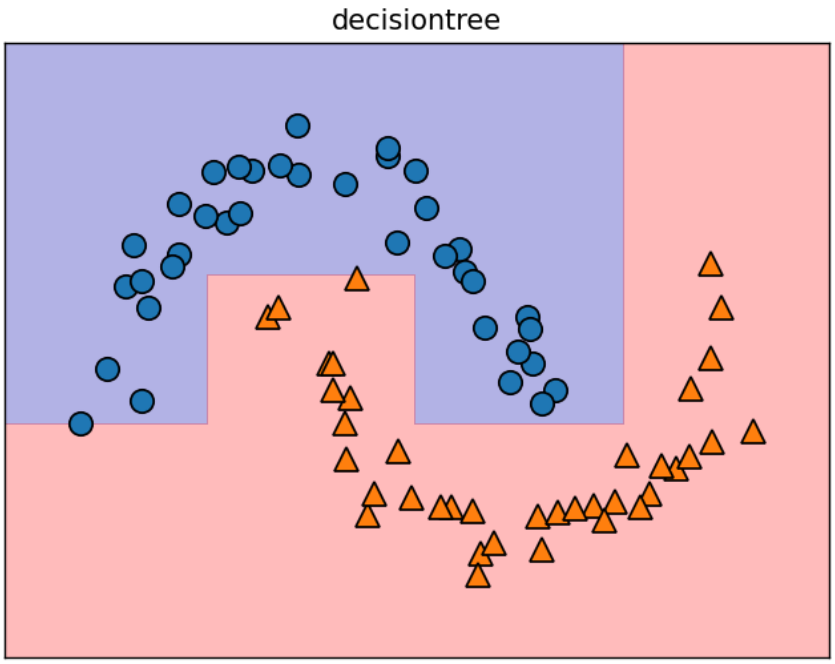


Figure 4 decision tree with max_depth 5

On figure 4 is seen the output after a decision tree model with max_depth set to 5. In this specific one the decision tree model actually does almost follow the moons. However in about 50% of the cases it would be off by quite a bit. In comparison to the SVM with rbf kernel and the MLP Classifier, I would pick either of those over the decision tree.

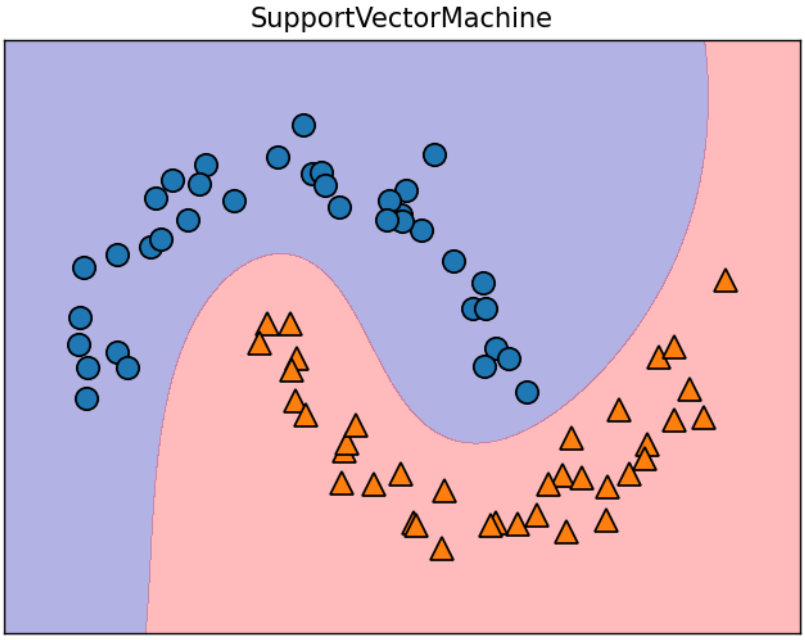


Figure 5 SVM with rbf kernel and C=100

As seen on figure 5, a Support Vector Machine using the rbf kernel does an even better job than the finetuned neural network from Exercise 1 A. It is seen perfectly following the moons as it is finding the middle between the zones, and does not appear to be overfitting either.

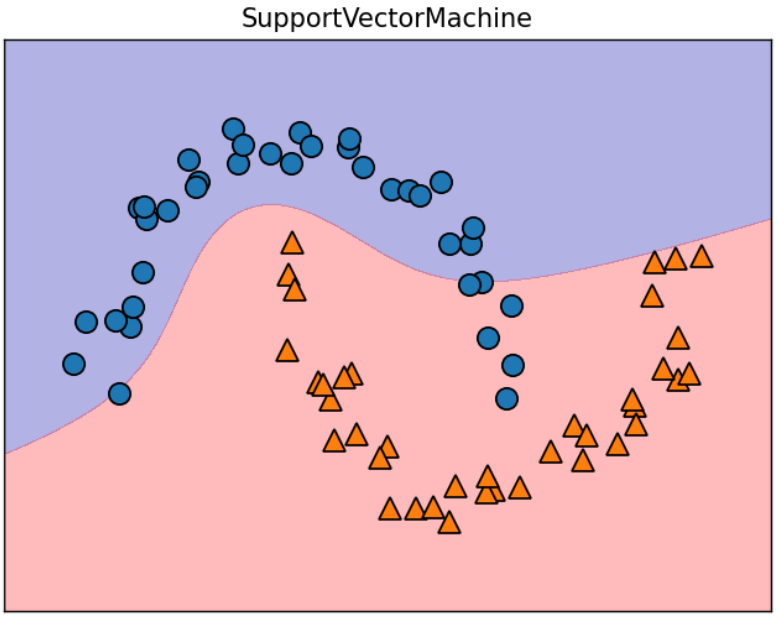


Figure 6 SVM poly kernel, C=100, 3 degrees

However as seen on figure 6, if we change the kernel to poly, with the default degrees of 3, it will still try to make the curves for the moon, it will however fail, though it does get quite close.

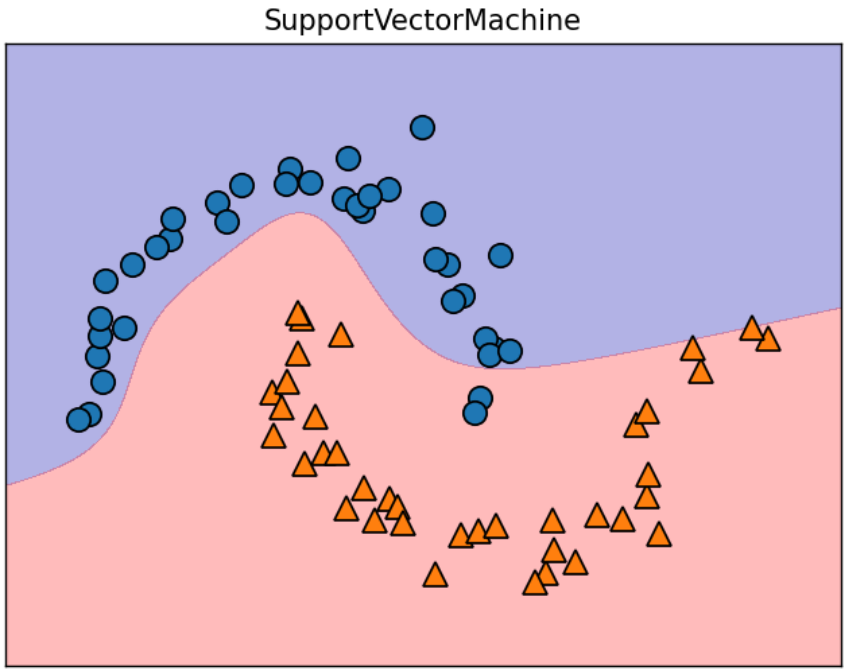


Figure 7 SVM poly kernel, C=100, degree=5

If we however up the polynomial degree to 5, it will seemingly get closer to a proper result, though it is still off by a small margin, it is not disastrous, but for this particular dataset using the rbf kernel seems to be the smarter choice.

Exercise 1 – C

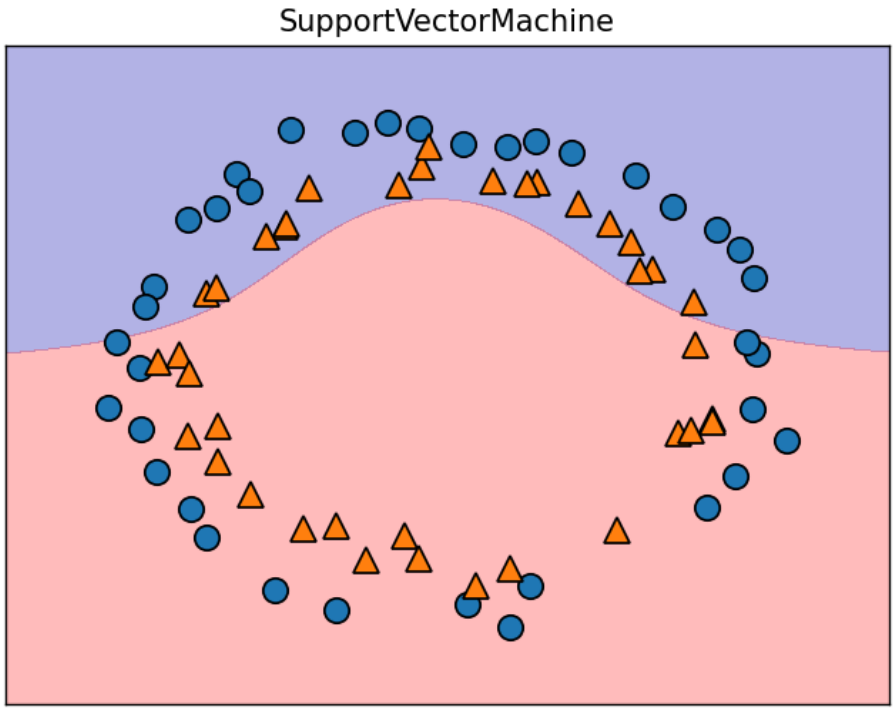


Figure 8 SVM poly kernel, C=100, degree = 3

As seen on figure 8 when using the poly kernel, it will not really split it up any good at all, and thus I'd strongly recommend against using this kernel for a circular dataset like this.

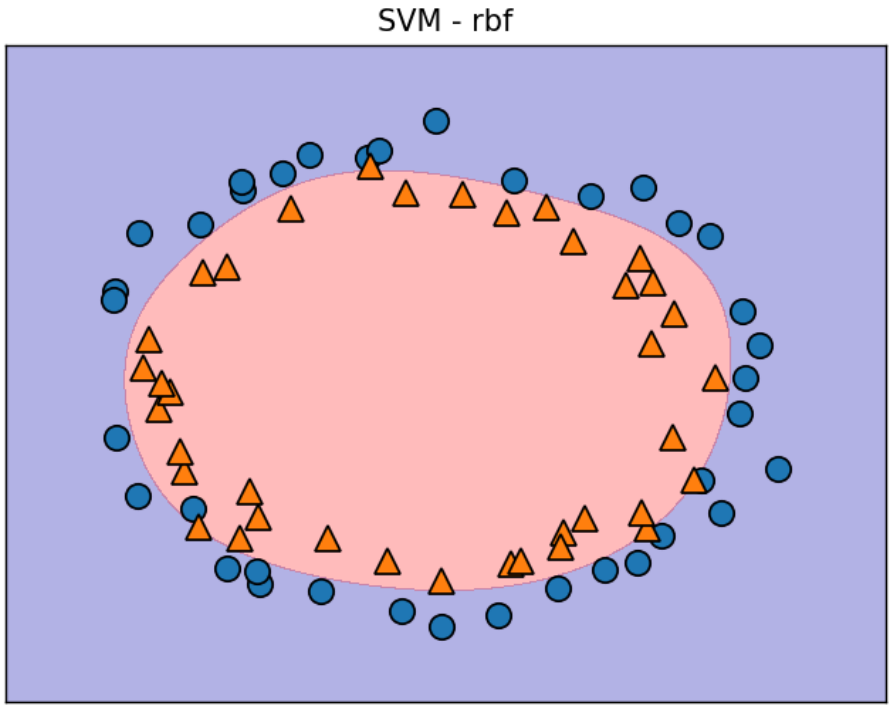


Figure 9 SVM rbf kernel, C=100

However, again the rbf kernel splits the data pretty much perfectly, with only 1 or 2 outliers. Without a confusion matrix it is still possible to see that the accuracy of the rbf kernel far outshines the poly kernel, and even the sigmoid kernel which can be seen below in figure 10.

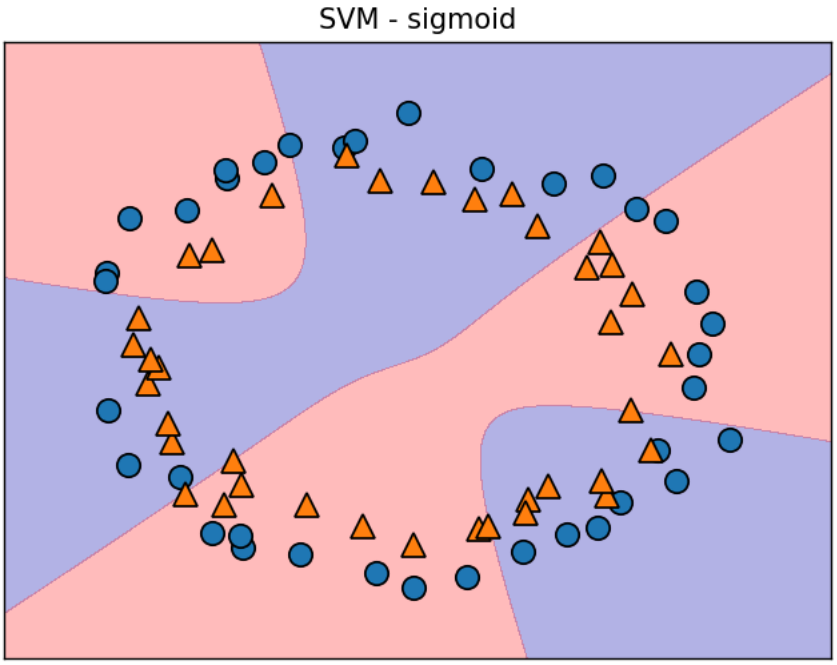


Figure 10 SVM sigmoid kernel, C=100

Exercise 2

Exercise 2 A

What kind of machine learning problem are we dealing with?

I would argue that we are dealing with a binary classification problem, considering that we want to train a machine learning algorithm to predict whether a specific passenger goes into the survived or not survived category, 1 and 0 respectively.

This is because we can classify passengers based on whether or not they did survive and make that classification to a certain degree of accuracy by using the other useful data that resides in the dataset.

How many features are there? (And how will I clean the data)

The completely useless features since they don't have any influence over the survivability is name and passengerId.

The rest of the features and the useful ones for the X matrix are:

- Passenger class
- Sex (male or female) – Will turn this into a 1 or 0
- Age
- Sibsp (# siblings onboard)
- Parch (# parents/children onboard)
- Fare
- Ticketnumber could be a feature, depending on whether the ticket number has anything to do with which cabin someone recides in, people in neighbouring cabins who had it worse might have something to say, it may also be completely irrelevant and is definitely worth testing.
- Cabin might make ticketnumber obsolete, but still would like to test
- Embarked (at which location the passenger embarked) – might turn this into a numerical value during training

These are the features of which I think may hold importance, other than that there are also passengerId and passenger name, both of which I believe have no importance to whether a passenger survived or not.

How many of these features will be in the Y data?

This question confuses me a bit, since I'd argue that none of them would considering we want to classify them between 1 and 0, I'd only put the Survived feature into the y vectors for both training and testing.

Exercise 2 B

Cleaning the data

After further inspection of ticket numbers those will be removed seeing as a lot of them appear to contain letters and not only numbers as previously expected, and I can't find a good way to change them from chars to numerical values.

As previously stated passenger IDs and Names will also be removed.

If sibsp is undefined it will be set to 0.

If parch is undefined it will be set to 0.

Having looked further into the data, a lot of the Cabin's seem to be undefined and those that aren't contain latin characters, which makes it hard to fill out with made up data. So that column will be dropped.

Sex will be replaced with 1 and 0.

Embarked will be replaced with 0, 1 and 2 for S, C and Q respectively.

If survived is undefined we will assume they survived.

Looking through the fare all of them have a value, which leads me to believe 0 means that they worked aboard the ship, in which case I will leave that as it is.

On top of that I will also be using sklearn's Standard Scaler.

Exercise 2 C

For the classification I've decided to use a neural network, specifically sklearns MLP Classifier with three hidden layers of 12, 8 and 4 neurons respectively. The reason for this is that while it may not be as simple to set up as an SVM, though both are supported learning models, I was unsure on whether there is a suitable kernel for the SVM, so I decided to use the neural network as it can be tweaked and customized more fluidly by changing the amount of neurons and hidden layers.

Exercise 2 D

	precision	recall	f1-score	support
0	0.81	0.86	0.83	98
1	0.75	0.68	0.71	62
accuracy			0.79	160
macro avg	0.78	0.77	0.77	160
weighted avg	0.79	0.79	0.79	160

Figure 11 performance

As seen on figure 11 we have a precision of 81% on determining if someone would die, and a precision of 75% determining if someone survived. This gives us an average precision of 78%. Our recall had an average of 77%. Our average f1 score lies on 77% which is also pretty decent. Though as seen on the figure the model is far better at predicting deaths than it is at predicting survival.

```
[[78 20]
 [17 45]]
tp: 45
tn: 78
fp: 20
fn: 17
```

Figure 12 confusion matrix

On figure 12 I extracted the confusion matrix and the values there in for more clarity, as seen, our true positives are 45, true negatives are 78 and fp and np are 20 and 17 respectively.

This means that 123 out of 160 were classified correctly. However, it had more false positives than false negatives which means it's better, as we established earlier, at correctly predicting death.

The percentage of correct predictions, also known as the accuracy is then determined as 123 divided by 160. (tp + tn) / (tp + fp + tn + fn) = 0.768 or 76.8% accuracy of this run.

Exercise 2 E

SVM (SVC, rbf kernel, C=100)

```
[[98  0]
 [61  1]]
tp: 1
tn: 98
fp: 0
fn: 61
Accuracy: 0.61875
              precision    recall  f1-score   support

           0       0.62      1.00      0.76        98
           1       1.00      0.02      0.03        62

    accuracy                           0.62       160
   macro avg       0.81      0.51      0.40       160
weighted avg       0.77      0.62      0.48       160
```

Figure 13 classification report and confusion matrix

As seen in the classification report in figure 13, we get a pretty poor accuracy when it comes to using the support vector classifier using the rbf kernel. As it seemingly wants everyone to die, even though the rbf kernel has proven itself in previous exercises to be a good fit at complex shapes like moons and circles, it would not have made a good model and kernel for this exercise.

Exercise 3

Exercise 3 A

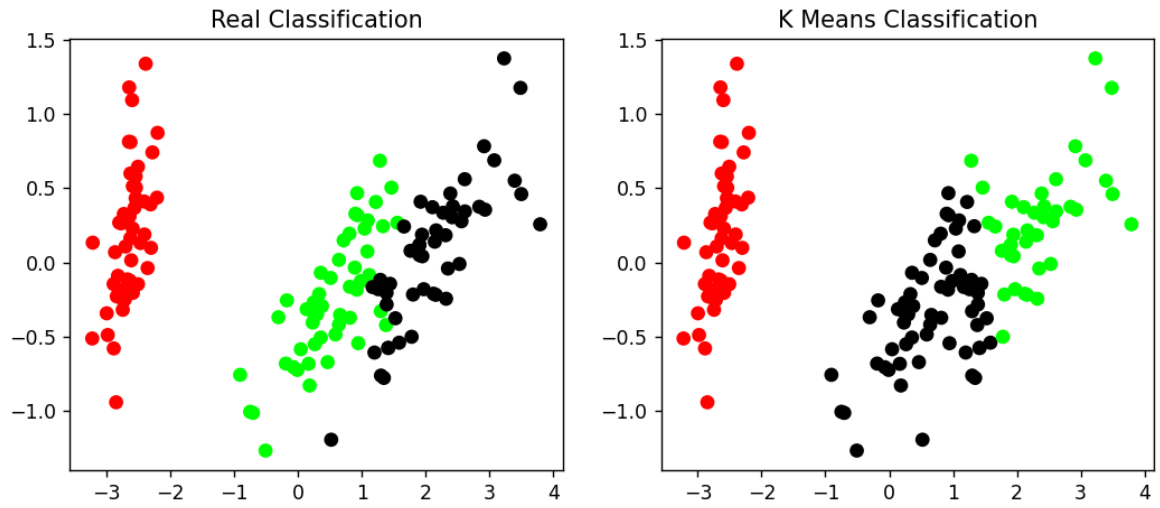


Figure 14 iris set with k means 3 clusters and PCA(2)

Exercise 3 B

Why PCA can be useful in larger datasets

In large datasets when using unsupervised training models such as k-means it can be hard to visualize the data if it exists in a higher dimension, as many larger datasets often do. The idea is to take for example a 5 dimensional dataset and reduce it to 2 dimensions while still retaining the characteristics of the higher dimensions.

In short it means we trade a little bit of accuracy for a simpler model which for example can be easier to visualize.

For example in figure 14 we can see the 2 dimensional version of the iris dataset after putting it through PCA. This gives us an overview of how well k means has managed to cluster our data in a simplified way compared to if we visualized it in 2 dimensions, but still had clusters being made based on a hidden 3rd dimension we can't visualize. Having the hidden 3rd or even 4th dimension would make it much harder to track how the algorithm has actually performed.