The following lab was conducted using the programming language *Python* with libraries *matplotlib, scikit-learn, pandas* and *numpy*.

As a side-note, no cross-validation was performed during this project. By doing so, the results obtained will likely have higher variance and must thus be analysed with caution.

Part A

a) From exercise a) and b) in Part 1 of *Project 1*, the dataset had been constructed according to the median criterion of the *mpg* variable. Moreover, the features *cylinders, displacement, horsepower* and *weight* were selected for the classification.

The SVM was trained and evaluated using the *sci-kit learn* classifier function *svm.svc()*. Firstly, a test data ratio of 0.2 were selected, as it is commonly used. With all parameters set to their 'default values' in *sklearn*; i.e. a *penalty parameter* $C = 1.0$, $\gamma = \frac{1}{p}$, where $p =$ number of features; where applicable, the parameters and corresponding test and training errors (i.e. 1 – *accuracy*) were recorded as can be seen in *Tab. 1.*

Kernel	Penalty C	Gamma γ	Training Error	Test Error
Linear	1.0	-	0.105	0.114
RBF	1.0	$1/p$	0.019	0.139
Sigmoid	1.0	$1/p$	0.482	0.456

Tab. 1. *Training, test errors and parameters for each model.*

For the above default parameters, the test data ratio was set to different values between 0.05 and 0.3 for three different kernels, as can be seen in *Fig. 1-3.*

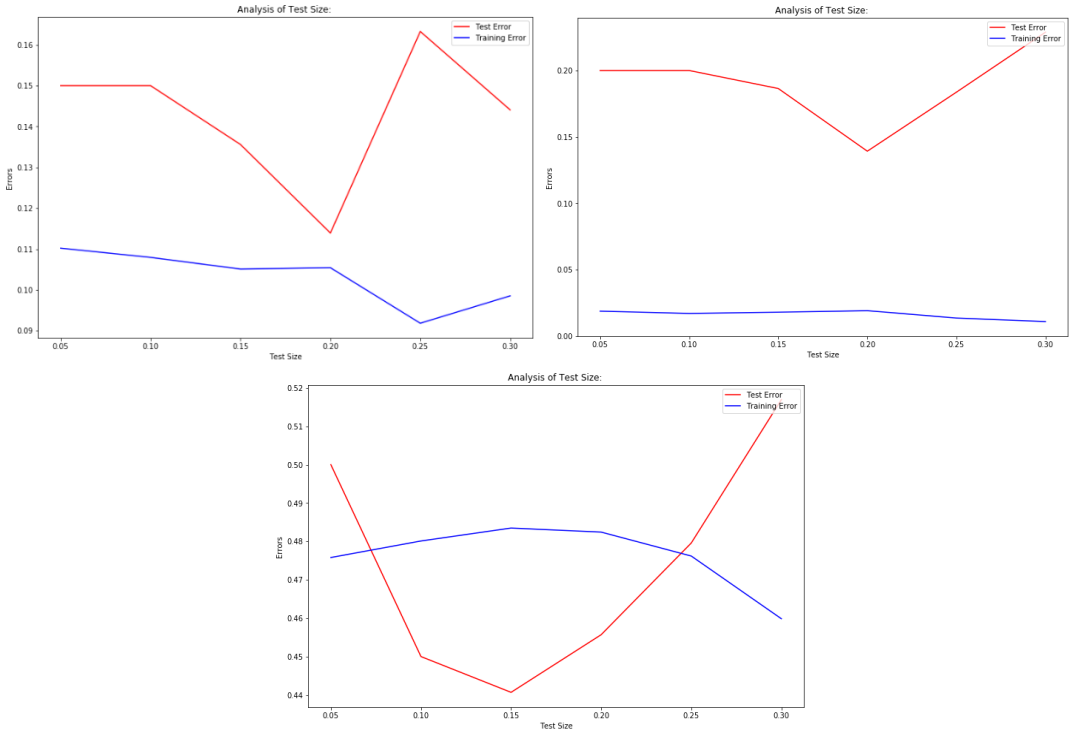

Fig. 1-3. *Test and training error for (from top left to right): linear, rbf, and sigmoid kernels, respectively.*

As to be expected, a test data ratio of approximately 0.2 seemed to yield the lowest training and test errors. For the *sigmoid* kernel, some abnormal behaviour could however be observed. The test size yielding the lowest errors seemed to be around 0.15, but the errors were vastly higher than the other two – around 0.45-0.5. As a brief conclusion, the sigmoid kernel is not suitable for this specific dataset.

For the default parameters as stated in *Tab. 1*, for kernel RBF, the penalty C was set to different values between 0.05 and 10^4. It was observed that the test error converged to approximately 0.139 just below 1.0 and remained constant as C was increased. For penalties below 1.0, the test error increased, as can be seen in *Tab. 2*.

C	0.05	0.1	1.0	5.0	10	100	1 000	10 000
Test Error	0.456	0.456	0.139	0.139	0.139	0.139	0.139	0.139

Tab. 2. *Test errors for different values of C for the rbf kernel.*

Again, for the default parameters as stated in *Tab. 1*, for kernel RBF, the parameter γ was now set to different values between 0.001 and 100. It was observed that the test error reached a minimum of approximately 0.0633 around $\gamma = 0.01$ and increased as γ was increased or decreased, as can be seen in *Tab. 3*.

γ	0.001	0.01	0.1	0.25	0.5	1.0	10	100
Test Error	0.101	0.0633	0.127	0.139	0.152	0.241	0.278	0.278

Tab. 3. *Test errors for different penalties γ for the rbf kernel.*

In both of these cases, when varying only one parameter at a time and keeping the others constant, i.e. doing a grid search, the global minimum may be neglected and local minima may instead be found. Another approach to cope with this issue is to do a randomized search in the hyperparameter space, but this is outside of the scope.

Considering only a test size of 0.2, it can be observed that the *rbf* kernel yields the lowest test error, with parameters $C = 1.0$ and $\gamma = 0.01$ for a test error of 0.0633. Note that a more extensive study with different values of C and γ for other kernels could have resulted in another conclusion.

b) A set of $N = 1\ 000$ samples from a Gaussian distribution were generated with parameters

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \in N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 100 \end{bmatrix}\right)$$

500 of these points were shifted radially by scaling each vector with a constant $r = 10$. These were classified as *class 1* and the remaining points were classified as *class -1*.

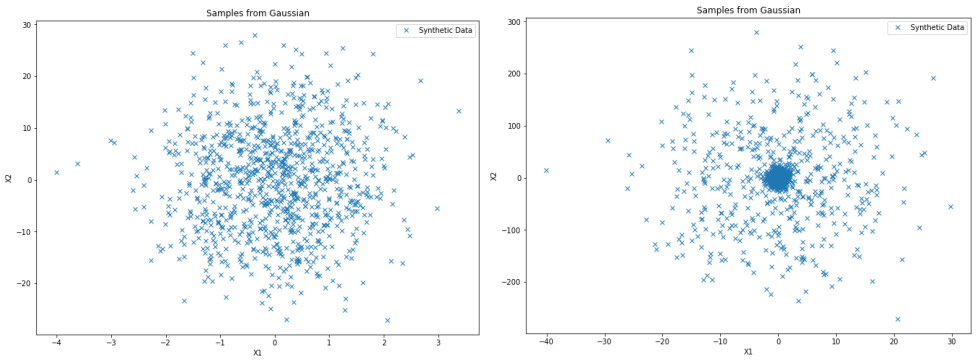

Fig. 4-5. *The set of X1, X2 parameters from a Gaussian distribution, before and after radially scaling half of the dataset.*

When selecting 100 points from each of the classes, i.e. corresponding to a test size of 0.2, different kernels and parameters were evaluated manually. The best performing model was obtained using a rbf kernel with $\gamma = 0.25$ and $C = 10$, yielding a test error of 0.035 and a training error of 0.0125.

Secondly, the procedure was repeated for $r = 100$, with the same parameters. This time, both the test and training errors were 0.

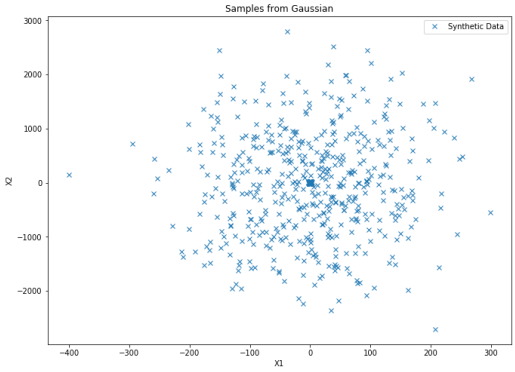

Fig. 6. *The set of X1, X2 parameters from a Gaussian distribution with half of the dataset radial scaling of r = 100.*

This indicates that a smaller value of r causes data from *class 1* to overlap with *class -1* and thus reduces the separability of the classes. This in turn leads to a higher test error.

The kernel function *rbf* performs best of the three kernels as expected, due to the construction of the kernel function. The distance from an input x to a point $\tilde{x}$ is connected to the value of the decision function f. Thus, points close to $\tilde{x}$ will be classified similarly and vice versa. This would explain why the model yielded low test errors, especially for larger values of r.

A *feature map* was created s.t. x_i, x_i^2 (elementwise), $i = 1,\ldots, N$, $x_i \in \mathbb{R}^2$. Together this feature map created data in $\mathbb{R}^4$. With a *support vector classifier* and default parameters $C = 1.0$ and a test size of 0.2 the following results were obtained:

	Test error	Training error
SVC, no feature map	0.410	0.400
SVC, feature map	0.225	0.286

Tab. 4. *Test and training errors for linear kernel with or without feature map applied.*

Since the *Bayes decision boundary* is non-linear, the non-linear feature map was expected to perform better in the SVC setting. This was confirmed, as to be seen in *Tab. 4*.

Lastly, the ratio k between between points in *class 1* and *-1*, respectively, (i.e. fraction of dataset to scale) was changed in $k \in (0.5, 1)$. With a scaling constant $r = 10$ and the rbf kernel with default parameters from *Tab. 1*, the training and test errors were obtained as can be seen in *Fig. 7*.

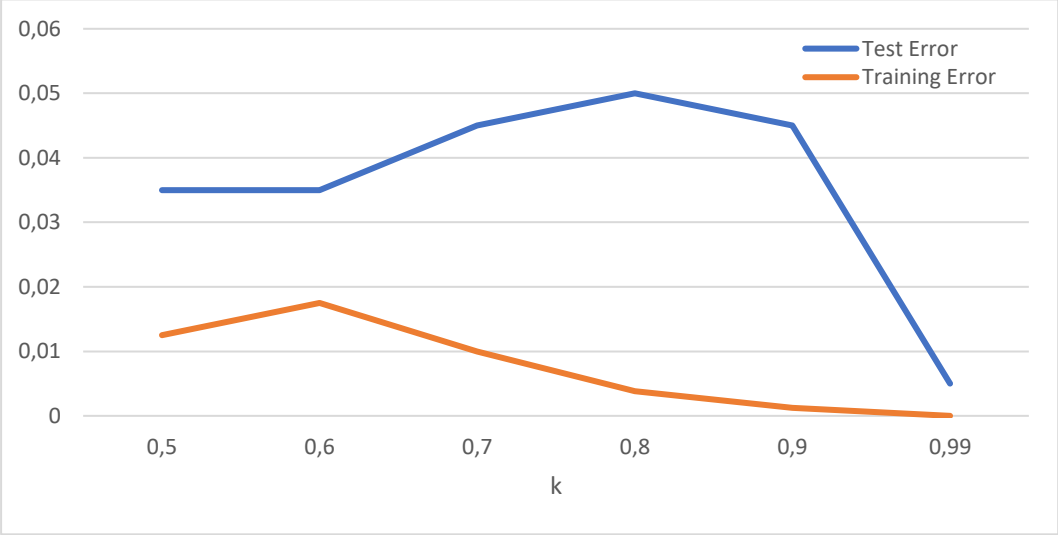

Fig. 7. *Test and training errors for different values of* k.

When increasing the value of k, the test errors initially increase and thereafter decrease as $k \to 1$. "An SVM classifier trained on an imbalanced dataset can produce suboptimal models which are biased towards the majority class and have low performance on the minority class like most of the other classification paradigms."[1] The test/training accuracy (= 1 – *test/training error*) is not a suitable metric for capturing this bias, which is why the test errors are low for high k's. E.g. a classifier which classifies all datapoints to *class 1* would have an accuracy of 0.99 if 99% of the dataset belongs to *class 1*. More suitable metrics would for instance be *recall* and *precision*. Moreover, the results become increasingly volatile as the imbalance increases between the classes.

[1] He, Ma, 2013 (Hoboken: Wiley): Imbalanced Learning: Foundations, Algorithms and Applications, *p.* 83

In conclusion, the performance of the *svm* becomes worse as class imbalance increases, but this is not reflected in the test and/or training error.

Part B

a) From exercise a) and b) in Part 1 of *Project 1*, the dataset had been constructed according to the median criterion of the *mpg* variable. Moreover, the features *cylinders, displacement, horsepower* and *weight* were selected for the classification, as before.

The decision tree was constructed and evaluated using the *sci-kit learn* classifier function *tree.DecisionTreeClassifier()*. With all parameters set to their 'default values' in *sklearn*; i.e. a *measure of node impurity* set to the *Gini index*, an *inf.* max-depth, *minimum samples per leaf* set to 1 and *minimum samples required for split* set to 2. This yielded a fully grown, unpruned tree.
With test sizes varying between 0.05 and 0.3, test and training errors were recorded as can be seen in *Fig. 8.* As an example, for test size 0.2, the maximum depth was 10.

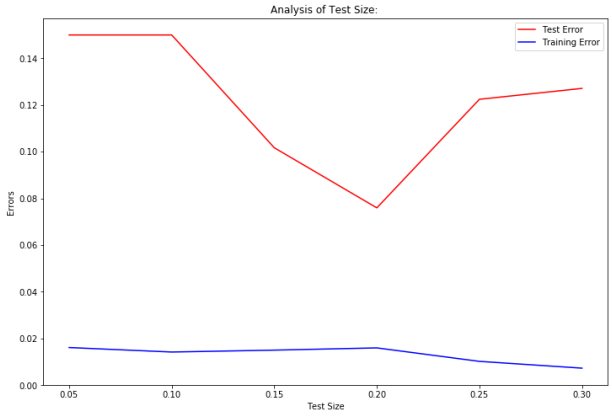

Fig. 8. *Test and training error for a fully grown, unpruned decision tree.*

It could be observed from *Fig. 8* that the lowest test error is obtained with the test size 0.2, as to be expected from previous results.

Secondly, a pruned tree was constructed with non-default parameters. The *maximum depth* was set to 5, the *minimum samples required for split* set to 5 and the *minimum samples per leaf* set to 3. This yielded a pruned tree, the results for which test and training errors were recorded as can be seen in *Fig. 9.*

The standard form of decision tree pruning using a *cost complexity function* is not implemented in *sklearn.* However, using a *"pre-pruning"* methodology by setting limits on hyperparameters in this fashion and not constructing the fully-grown tree beforehand, allows for *a* method of pruning. This comes with some obvious drawbacks. While reducing the time complexity, it does not allow for comparing the performance of the resulting tree in relation to other possible trees.

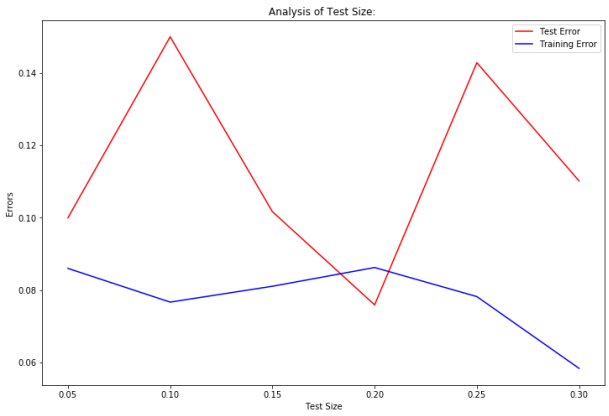

Fig. 9. *Test and training error for a pruned decision tree.*

It could be observed from *Fig. 9* that the lowest test error is obtained with the test size 0.2, again as to be expected from previous results.

When comparing the two sets of errors from the pruned and unpruned tree, respectively, the pruned tree produced marginally lower test errors while higher training errors. This is intuitive since pruning reduces overfitting.

b) For this exercise, the *mpg* column in the dataset was reverted to that of the original dataset, i.e. no longer binary. Moreover, the features *cylinders, displacement, horsepower* and *weight* were since previously selected to be used in the model.

The regression tree was constructed and evaluated using the *sci-kit learn* regression function *tree.DecisionTreeRegressor()*. With all parameters set to their 'default values' in *sklearn*; i.e. a *loss function* set to *mean squared error*, an *inf.* max-depth, *minimum samples per leaf* set to 1 and *minimum samples required for split* set to 2. This yielded a fully grown, unpruned tree. With test sizes varied between 0.05 and 0.3, R^2 values for test and training data were recorded as can be seen in *Fig. 10.* As an example, for test size 0.2, the maximum depth was 15.

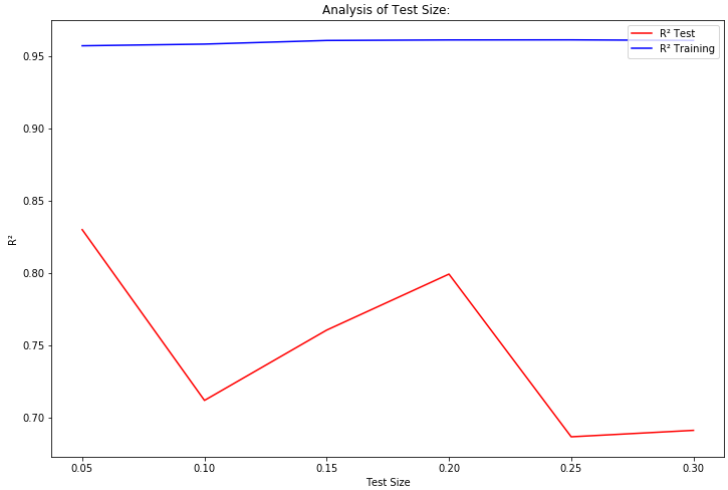

Fig. 10. *R^2 values for a fully grown, unpruned regression tree.*

Secondly, a pruned tree was constructed with different parameters. The *maximum depth* was set to 5, the *minimum samples required for split* set to 5 and the *minimum samples per leaf* set to 3. This yielded a pruned tree, the results for which R^2 values were recorded as can be seen in *Fig. 11.*

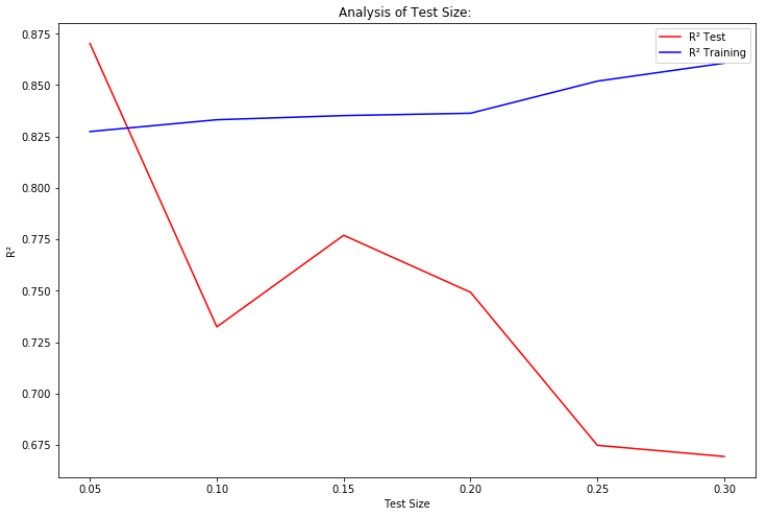

Fig. 11. *R^2 values for a pruned regression tree.*

In this case, the difference in goodness of fit was not as apparent between the pruned and unpruned tree, as in the classification setting. However, the R^2 for the training data decreased when pruning the tree, indicating that the unpruned tree suffered from overfitting.

From an interpretability perspective, the predictions of the trees are in certain ways more transparent in the regression case. This follows since a car will not only be classified binarily as having high or low mpg, but rather how high or low the predicted mpg is. If desirable, this could moreover be compared to the median and thus be made binary again.
Regarding analysis of how different features contribute to the prediction, the two models are similar. For instance, *feature importance*[2] can be obtained in both cases.

[2] For the interested reader; information about feature importance can be obtained here: *https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor.feature_importances_*