Introduction

Abalone are a very common type of shellfish. Their flesh is considered to be a delicacy and their shells are popular in jewellery.

In this work I consider the problem of estimating the age of abalone given its physical characteristics. This problem is of interest since alternative methods of their age estimation are time-consuming. Therefore, if a statistical procedure proves reliable and accurate enough, hours of working hours could be saved.

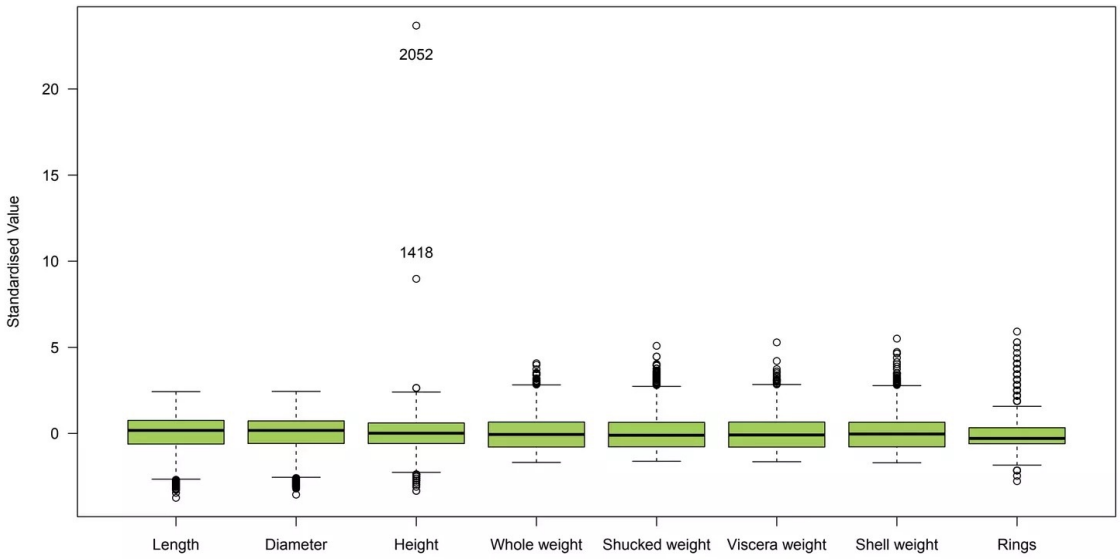
Abalone can live up to 50 years, depending on a species. The speed of their growth is primarily determined by environmental factors related to water flow and wave activity. Due to differences in food availability, those from sheltered waters typically grow more slowly than those from exposed reef areas [1]. Telling the age of abalone is therefore difficult mainly because their size depends not only on their age, but on the availability of food as well. Moreover, abalone sometimes form the so-called 'stunted' populations which have their growth characteristics very different from other abalone populations [2].

1 Data Overview

For purposes of abalone age prediction, I will work with a dataset coming from a biological study [3]. It includes observations on 8 variables of 4177 abalone and contains no missing values.

The dataset contains two obvious univariate outliers. All 8 variables are plotted in Figure 1 in their standardised form, that is with their means equal to 0 and variances equal to 1. Two observations clearly stand out on variable *Height*, observations 1418 and 2052. These observations are likely to be coding errors not representing true height measurements. Therefore they are discarded from all the analysis yet to follow and I continue with the remaining 4175 observations.

Figure 1: Identification of Outliers



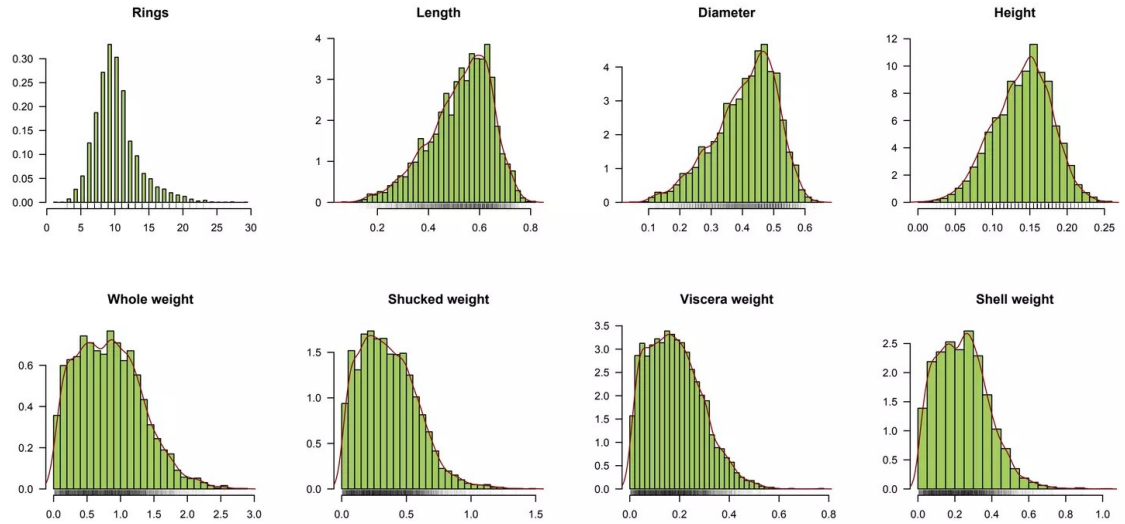
Two outliers (with indices 1418 and 2052) were identified in variable *Height*. They will not be included in any analysis in this paper.

1.1 Response

Most importantly, the dependent variable *Rings* is included in the dataset. It was measured as the number of rings observed after cutting and examining an abalone. Although it does not denote the age of a given abalone directly, it determines it more-or-less perfectly; the age of an abalone equals $Rings + 1$. Since this relationship holds reliably, in what follows I work with *Rings* as the dependent variable.

The top-left panel of Figure 2 displays the distribution of *Rings*. The number of rings measured in the data ranges from 1 to 29 and most of the abalone have between 5 and 15 rings. The distribution is slightly positively skewed as well but this does not pose any specific problems for the further analysis.

Figure 2: Relative Frequency Density of all Variables from the Dataset



Relative frequency density scale is constructed such that the product of the dimensions of any panel gives the relative frequency. Hence the total area under the histogram is 1. Estimate of the density is plotted in red on top of the histogram of each continuous variable. Rugs represent observations and are plotted semi-transparent below the horizontal axis.

1.2 Predictors

There are seven continuous predictors included in the dataset. The first three constitute shell measurements:

Length Longest shell measurement

Diameter Shell diameter perpendicular to the direction of *Length*

Height Height of the shell with meat inside

Although these variables were originally measured in millimetres, the dataset contains their scaled versions (see [4]). Distributions of these three variables are displayed in the top row of Figure 2. It is apparent that their distributions are all very similar, slightly skewed to the left. Their similarity is to be expected since these measurements are also likely to be correlated: shells small in *Length* are likely to be small in *Height* as well etc.

The other four explanatory variables are related to the weight of abalone:

Whole weight Weight of the whole abalone

Shucked weight Weight of meat of the abalone

Viscera weight Gut abalone weight (after bleeding)

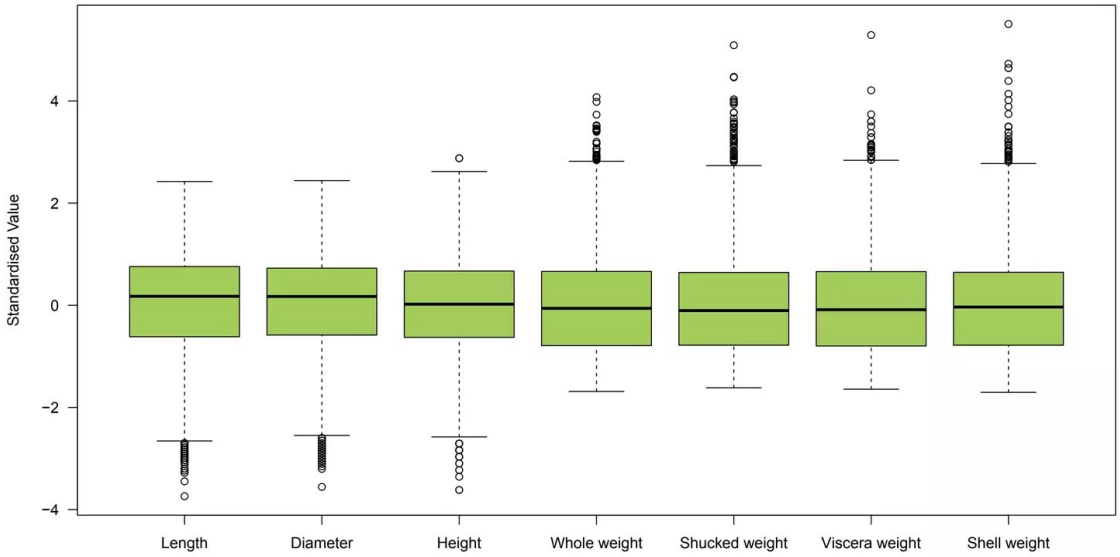
Shell weight Abalone weight after drying

These 4 variables were obtained by scaling the original measurements (in grams) analogously to shell measurements. The distributions of these variables can be explored in

Figure 2. It is easy to see that they are slightly positively skewed. Moreover, since they are all weight measurements they have very similar distributions. Heavy abalone are likely to be considered heavy whatever measure of weight is utilised.

For its use later in the analysis, the dataset needs to be normalised. Several options are available: simple standardisation, logarithmic transformation, Box-Cox transformation, a combination of these, or even some subject-matter based transformation. With regards to the first four of these methods (it will be commented on subject-matter based transformations in the next section), all were tried but the standardisation to mean 0 and variance 1 is not only the simplest but it also brings desirable results. Figure 3 plots the predictors after this transformation. Although the skewness is, of course, still present in the data, standardisation to the same variance ensures that these predictors have approximately equal a priori importance in the analysis.

Figure 3: Transformed Analysis-Ready Predictors



All predictors are plotted standardised to mean 0 and variance 1. They enter the analyses from Section 3 onwards in this form.

To sum up, the predictors relate on one hand to physical shell measurements and on the other hand to the weight of abalone. Given what has been mentioned in the introduction, it is rather unfortunate that the dataset includes only size-related variables. No information on the species of abalone, the region of its origin, food availability or even sex is included. This lack of information will make prediction of the age more difficult.

2 Principal Components Analysis

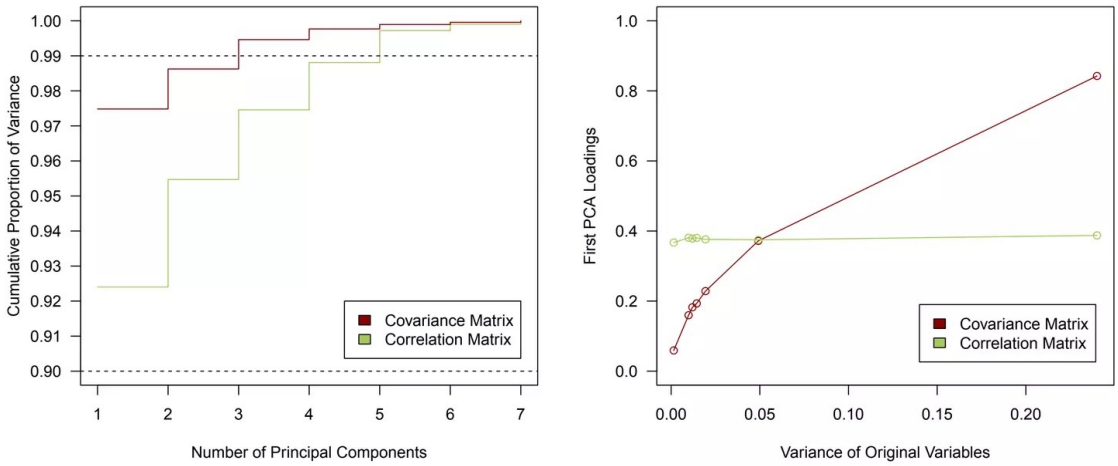
In order to decrease dimensionality of the data and to visualise it, principal component analysis (PCA) is performed. PCA identifies the lower-dimensional view of the data that captures most of its variance. It is based on eigenvalue decomposition of the data covariance matrix.

2.1 PCA with Covariance and Correlation Matrix

Statistical theory states that if variances of the variables differ markedly, PCA puts more weight on variables with the higher variance than on variables with lower variance. The resulting principal components (PCs) then tend to resemble variables with high variance more closely than those with low variance. In these cases, it is advisable to work with the correlation matrix, not the covariance matrix. This is because the correlation matrix provides standardisation relative to the variances of the variables. The strength of linear relationships between variables, not their variance, then determines the resulting PCs.

Comparison of the usage of the correlation and the covariance matrix for the abalone data is visualised in Figure 4. In both panels the red line denotes the results of PCA using covariance matrix and the green line represents the results of PCA with the correlation matrix. The left-hand plot shows the cumulative proportions of the variance explained by a given number of PCs. It could seem that the covariance variant works better on this data since its first PC solely explains more than 97% of the data variance. On the other hand, the first PC of the correlation variant 'only' explains less than 93%.

Figure 4: Comparison of PCA with Covariance and Correlation Matrix



The left-hand panel plots the portion of the explained data variance as a function of the number of PCs. The right-hand plot shows how first component loadings for all seven variables differ when using the covariance and the correlation matrix. These loadings are actually all negative; for purposes of easier interpretation the sign of all of them was changed.

Examination of the explanatory potential of the first PC is important because, looking at the predictors, there is a reason to believe that the first PC should represent the size of abalone. It would make perfect sense since all seven predictors are related to either shell size or the weight of abalone. With this in mind it is easy to see that the first PC of the correlation variant represents size much better than the first PC of the covariance variant. Evidence is supplied in the right-hand plot of Figure 4. It plots the first PC loadings from the both analyses for all seven predictor variables.[1] The horizontal axis represents the variance of a given original (unscaled) variable. It can be observed that one predictor has the variance almost identical to 0, four other predictors have it also rather close to 0, one predictor has the variance above 0.05 and, finally, one has a relatively huge variance of more than 0.2.

In the PCA variant with correlation, all loadings are approximately the same (the green curve stays roughly constant). This is in accord with the hypothesis that the first principal component represents the size of abalone. No predictor should have substantially more influence on abalone size than any other. For example, it should not matter too much whether we measure the length or the width of abalone shells, they should both contribute similarly to whether we regard a given abalone as small or not.

On the contrary, the PCA variant with covariance shows a clear difference in factor loadings. The predictors with higher variance have higher loadings and thus higher influence on the first PC than those with lower variance (the red line monotonically increases). This has no substantial grounding. It is unclear why *Whole weight* should be more than 14 times as important for determining size of abalone than *Height*. This strange result is only caused by the large differences in the variances of the original variables. Therefore, the theory recommendations are completely relevant here and the analysis proceeds with the results of the PCA variant with the correlation matrix.

2.2 Examination and Interpretation of the Principal Components

Having solved the question which PCA to use, the resulting PCs should be examined. Looking back at the left-hand panel of Figure 4 it can be spotted that it takes five PCs to explain at least 99% of the data variance. As has been previously noted, the first PC explains the vast majority of the data variance (more than 92%). The previous paragraphs also support the hypothesis that the first PC represents the size of abalone.

We can further investigate this by looking at the biplot displayed at the left of Figure 5. The biplot was rescaled so that the relationships between the variables are clearly visible. The arrows represent the directions of the original variables projected into a two-dimensional space defined by the first two PCs. There are two variable clusters: shell-size measurements in one and 3 weight-related variables in the other. The fact that there are two clusters means that this first PC can be only roughly interpreted as the abalone size. If the first PC does represent size, the variable arrows should be more-or-less parallel with the horizontal axis (first PC). This follows from the general rule that the smaller the angle between two lines on which two arrows in a biplot are, the stronger the correlation

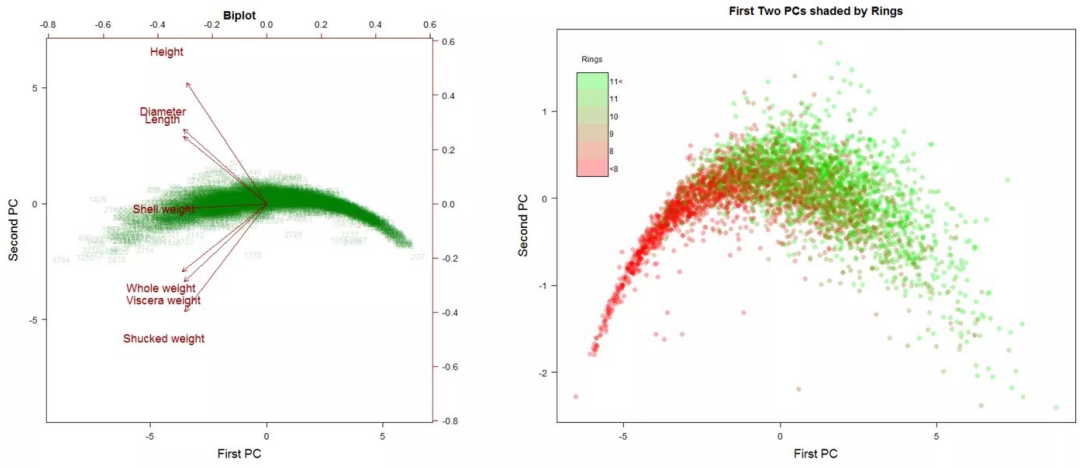
[1] These loadings were actually all negative; for purposes of easier interpretation their signs were changed.

between the two variables.

The right-hand panel of Figure 5 plots the first two PCs against each other. The sign of the loadings of the first PC was changed to positive numbers to simplify the interpretation as the size of abalone (rather than smallness). The colours in the plot range from bright red to bright green and denote the the number of rings observed (smallest and largest respectively). Moreover, since the points have high density, they are plotted semi-transparent.

The main message of the plot is two-fold. First, *Rings* are obviously related to the values of the first PC. Small abalone tend to be red, to have few rings. Large abalone tend to be green, to have many rings. The middle region is, however, mixed, with both colours often occurring. This means that size might be a very good predictor of the abalone age for very small and and very large abalone. However, the age may be much harder to predict for medium-sized abalone.

Figure 5: First Two PCA Components



The left hand plot is a standard biplot for the first two PCs with the observations plotted semi-transparent. It is rescaled to better see the relationships between variables. The right-hand plot is a plot of the first two PCs with the sign of the loadings of the first PC changed to positive to simplify the interpretation. The colours in the plot range from bright red to bright green and denote the number of rings observed (smallest and largest respectively). Large tail intervals were selected to achieve good separation of the cases in the middle of the range. Moreover, since the points have high density, they are plotted semi-transparent.

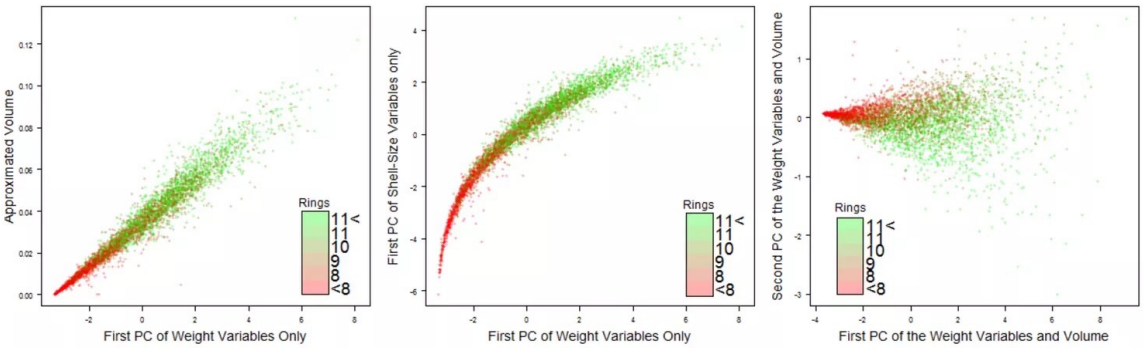
The second message is that there exists a curved relationship between the first and the second PC. This may seem surprising as PCs are by definition uncorrelated. However, it must be remembered that correlation is a measure of linear association only. Indeed, if we were to fit a line through the points, a constant line would appear. But how can we explain the paradoxical curvature of PCs?

I claim that the curvature is simply a result of the existing physical relationship between **weight** and **volume** that is known to be linear. While the dataset contains no variables measuring volume directly, it contains three one-dimensional measurements of shell size (*Length*, *Diameter*, *Height*). With a slight simplification, it could be argued that the product of these three variables well approximates the volume of abalone. The left-hand panel

of Figure 6 shows the plot of this 3-way product of the one-dimensional shell measurements against the first PC computed from the weight-related variables only. We see a strong linear, not curved, relationship. So the hypothesis of a linear volume-weight relationship is supported.

If we now look at the 3 one-dimensional shell measurements, extract the first PC from them and plot this PC again against the first PC from the weight-related variables only, we find a curved relationship (middle panel, Figure 6). This confirms the curved pattern between the first two PCs discovered in Figure 5. We see that the curved pattern is a direct consequence of the fact that we use weight variables together with one-dimensional shell-size measurements. **Since weight is linearly related to volume, not to individual measurements, the curvilinear pattern** between the variables in the dataset, and analogously **between the PCs**, **occurs**. Indeed, if we look at what happens if we run PCA on the weight-related variables together with the 3-way product representing the volume, the curvature disappears (the right-hand plot of Figure 6).

Figure 6: Further Investigation of the Curved PC Shape



The left-hand panel plots the 3-way product of shell-measurements (approximated volume) against the first PC computed from the weight-related variables only. The middle panel shows what happens if we take the one-dimensional shell measurements, extract the first PC from them and plot it again against the first PC from the weight-related variables only. The right-hand panel plots the first two PCs from the PCA on the weight-related variables and the 3-way product (volume).The colours in the plot range from bright red to bright green and denote the number of rings observed (smallest and largest respectively). Large tail intervals were selected to achieve good separation of the cases in the middle of the range. Moreover, since the points have high density, they are plotted semi-transparent.

After the detailed analysis of the PCA results, the data is understood quite well. If we work with volume, not the original variables, the first PC represents size more accurately. The values of the second PC spread with the higher values of size (the right-hand panel of Figure 6). It also seems to separate cases according to their colour rather well and therefore could be potentially useful for predictive analysis. Possibly it could be interpreted as food availability. With its high values, even the young abalone (red) grow large (are situated to the right of the plot). However, this remains only a speculation.

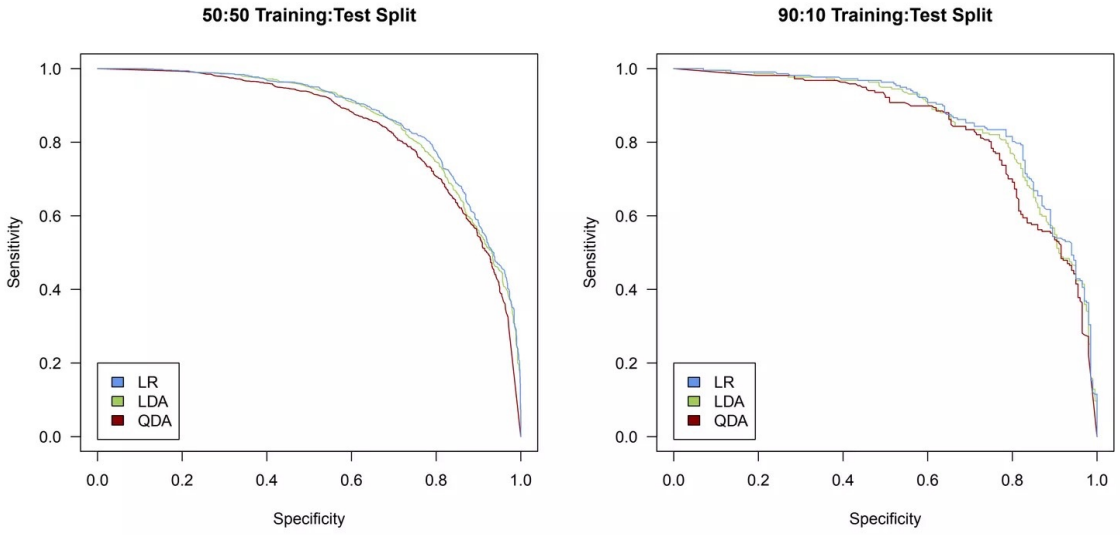
3 ROC Curves

In this section I perform classification of the observations into classes of the 'young' and 'old' abalone. The two classes are simply defined according to their number of *Rings*; observations with less than 10 rings are classified as 'young' and those with at least 10 rings as 'old'. Out of 4175 abalone in the dataset, 2095 are 'young' and 2080 are 'old'.

The classification is performed using three data mining techniques: Linear Discriminant Analysis (LDA), Qudratic Discriminant Analysis (QDA) and Logistic Regression (LR). LDA assumes that conditionally on class label, the distribution of predictors is multivariate normal. Moreover, the within-class covariance matrices are assumed to be identical. Given these conditions, LDA finds linear boundaries that best discriminate between the classes. QDA is a direct generalisation of LDA which does not assume equality of the within-class covariance matrices. It can draw boundaries much more complex than LDA, but it is in general also more likely to overfit. LR does not model the class boundaries, but it directly models the probabilities of class membership for every case. With virtually no assumptions, it has been shown to be a very reliable and robust method.

In order to assess performance of the methods, the dataset needs to be split into a training set and a test set. Two methods are considered. First, the classifier is trained on half of the data and tested on the other half. Under the second design, the classifier is trained on 90% of the original dataset and tested on the remaining 10%. Naturally, the observations are picked to either set set at random. As was mentioned in the first section, the predictors enter this analysis (and all of the subsequent analyses as well) in their standardised form.

Figure 7: ROC Curves of LDA, QDA and LR



The ROC curves are plotted for three methods: Linear Discriminant Analysis (LDA), Qudratic Discriminant Analysis (QDA) and Logistic Regression (LR). The left-hand plot was produced by splitting the dataset into a 50% training set and a 50% test set. The right-hand plot was produced training the classifiers on 90% of the data and testing it on the remaining 10%.

Figure 7 visualises the results of the classifications using Receiver Operating Curves (ROC) curves. ROC curves plot sensitivity (estimated probability of classifying an 'old' abalone correctly) of a binary classifier against its specificity (estimated probability of classifying a 'young' abalone correctly). Good classifiers should both have high sensitivity and specificity. Therefore, if an ROC curve for some method lies above the curve of another method, the first method should be preferred.

The left-hand plot displays the ROC curves for the 50:50 split of the data between the training and test set. It can be easily seen that almost the whole red curve, representing QDA, lies below the other two curves. This means that if the level of specificity is given then LR and LDA have higher sensitivity than QDA. Analogously, sensitivity given, LDA and LR both have higher specificity than QDA. We can conclude that QDA performs worst of all the methods tried. The green (LDA) and the blue (LR) curve are very close to each other which means that they perform similarly well. It, however, seems that LR slightly outperforms LDA since the blue curve is, in general, located slightly above the green curve.

Of course, the true ROC curves are unknown and the plotted curves represent only their estimates calculated from the observed values and predicted classes. Therefore, shape of the curves depends on sample characteristics such as test sample size. This fact can be well illustrated by the right-hand plot. It shows the ROC curves for the classifiers trained on randomly chosen 90% of the data and tested on the remaining 10%. Estimating the ROC curve from 417 observations (as opposed to 2087 observations in the 50:50 split) substantially increases the variance of the estimated ROC curves. Indeed, the curves in the right-hand panel of Figure 7 are much less smooth than the previously estimated ROC curves. Although the overall ordering of the methods remains the same, there is much more uncertainty now. For instance, this time the red curve (QDA) in some local areas crosses the other curves and therefore sufficiently well compares with the other two methods.

If we were to use Fisher's LDA to reduce the dimensionality of the data, we would use one discriminant direction. This is simply because here the aim is to separate the data into two classes. Using one discriminant direction is therefore adequate.

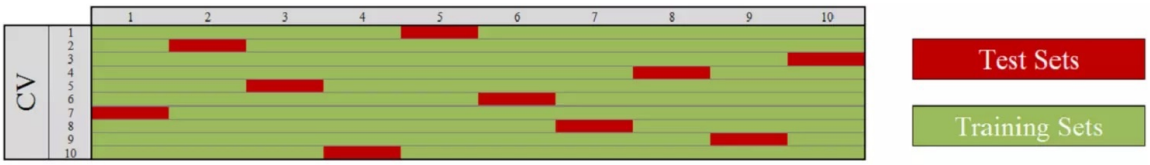
4 k Nearest Neighbours

In this section and the sections to come I work with the response classified into three categories with roughly equal frequencies. The abalone with 8 rings or less are coded as 'young', those with 9-10 rings as 'adult' and those with at least 11 rings as 'old'. These three categories contain 1406, 1322 and 1447 observations respectively.

The k Nearest Neighbours algorithm (k-NN) is a very simple and intuitive supervised non-parametric classifier. Every observation is classified according to the class that is prevalent in its 'neighbourhood' (with ties broken at random). The 'neighbourhood' is specified via parameter k and stands for the number of closest points in the training data that are considered when classifying a new observation.

Choosing the optimal k a priori is difficult if not impossible. Therefore, in order to maximise the predictive performance of k-NN, it has to be fitted for all relevant k's and the optimal value subsequently determined. Since this procedure requires testing a trained model on unseen data, a cross-validation design is often implemented. In this case, I choose to perform 10-fold cross-validation. The whole dataset is split 10 times in the 90:10 ratio (see Figure 8). For every k, k-NN is then trained on each of the larger (green) datasets and tested on the corresponding smaller datasets (red). This way the misclassification error is obtained 10 times for every value of k. The mean of these values serves as an indicator of k-NN performance for a given k. The minimum of the mean misclassification values over all k determines the optimal number of neighbours.

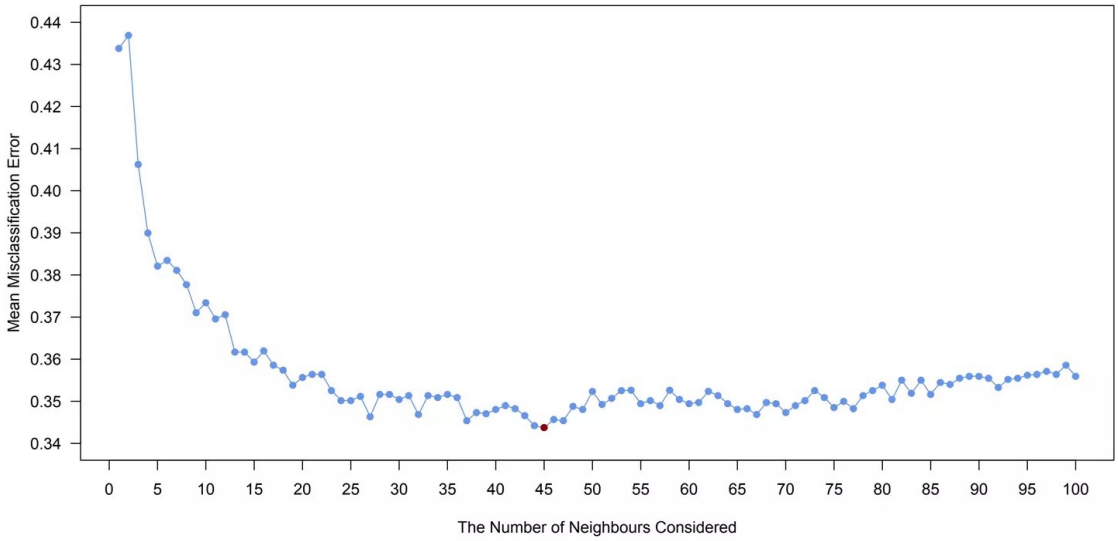
Figure 8: 10-Fold Cross-Validation



Each horizontal bar represents the entire dataset. The test sets (red parts) always account for 10% of the data and the training sets (green parts) for 90%. Each model (with a given number of neighbours) is fitted 10 times (10 horizontal bars).

The procedure explained above was applied to the abalone dataset with the results indicated in Figure 9. Values of k in 1-100 range were checked. It is clear from the plot that the mean misclassification error decreases rapidly until approximately 25 neighbours. It continues to decrease with the increasing k until it reaches the minimum at 45 neighbours. More than 45 neighbours lead to a slow increase in the mean misclassification error. Based on this analysis, the value of 45 should be chosen as the optimal value for the number of neighbours. The mean misclassification error is in this case slightly lower than 34.4%.

Figure 9: Selection of the Optimal Number of Neighbours



The vertical axis represents the mean misclassification error from 10-fold cross-validation. The minimum is achieved with 45 nearest neighbours (highlighted in red).

Two important notes should be added here. First, although value 45 is chosen here, many other values could possibly emerge with the same data. This is related to the very flat character of the curve in Figure 9. For example if we changed the cross-validation design it would be likely to get different values. Or if we simply created the 10 cross-validation divisions at random again, different esitmate for the optimal k could emerge. Strictly speaking, nothing in the cross-validation design or the data sets needs to be changed. A small amount of randomness is included in the k-NN algorithm itself (in case of ties). Even with the optimal number of neighbours close to 45, ties are very likely to happen in large samples and they may sometimes change the final choice of the optimal k.

Second, the value 34.4% does not provide a good enough estimate of the generalisation error of k-NN. This is because test data is here used to optimize over k and, therefore, the classifier's performance on this very test set is artificially improved. In order to accurately estimate the generalisation error, the information contained in the test set should not be used in any way during the training process. In section 5, I develop a more complex cross-validation design ehich takes this aspect into account.

5 Comparing Methods

In this section I work with the same quantisation of the response as in the previous section, that is into the three categories ('young', 'adult' and 'old' abalone). 7 different methods are used to predict this variable: linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), logistic regression (LR), k nearest neighbours (k-NN), classification trees (CT), random forests (RF) and neural networks (NN).

The first four of the above mentioned methods (LDA, QDA, LR and k-NN) have already been introduced in the previous section. Classification trees partition the predictors' space into disjoint subsets and propose the same prediction for all observations falling into that subset. The partitioning is done sequentially, based on simple decision rules of whether a given predictor is larger or smaller than a constant. The more complicated CT is grown, the better its explanatory power is on a training set. However, CTs too large often overfit the training data and are likely to provide unsatisfactory results on test data. The so-called 'pruning' procedure provides the way out. It trains CTs of many sizes on the same training data and then picks the smallest tree size providing predictive capacity on the test set only marginally worse than the optimal size.

Random forests constitute a generalisation of CTs. Basically, they fit many classification trees on the same training data set and average the predictions over all of them. Moreover, not all variables are typically searched at each splitpoint but only a subset. The size of this subset is the only real 'tuning' parameter of RFs (no pruning is necessary). The optimal value is determined by choosing the one giving, on average, lowest classification errors.

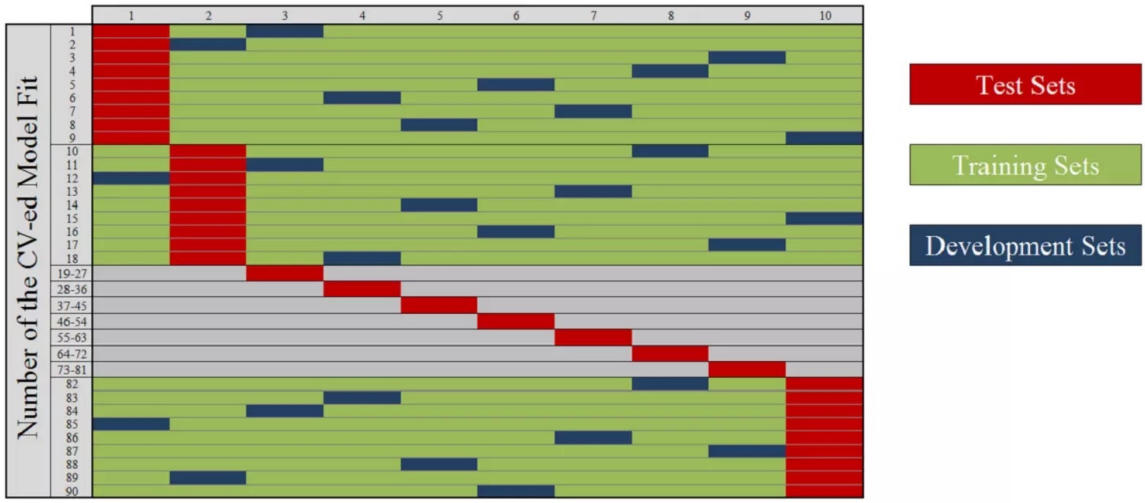
Last, neural networks are a method that combines many linear relationships them into a potentially very complicated non-linear classifier. The variables are combined through a structure of layers. In this paper, single-hidden-layer NNs are used; only one layer of artificial variables exists between the input variables and the resulting predictions. Two parameters can be optimised over. The first one is size, the number of units in the hidden layer, the second one being decay, a regularisation penalty on the weights ('NN coefficients'). Here, decay is set to 0.01 which typically helps to avoid overfitting. The size parameter is then optimised by fitting NNs of different sizes and subsequently choosing the value that minimises classification errors.

Estimates of the generalised errors for LDA, QDA and LR are computed in a 10-fold cross-validation design as it was explained in the previous section. However, more complicated cross-validation designs are necessary for estimation of the generalisation errors of the other four methods (k-NN, CT, RF, NN). All of these four methods contain parameters that have to optimised using misclassification rates on unseen data. As I already explained in the k-NN section, if optimisation of these parameters is done on the same data that is used for testing, generalisation error estimates are artificially decreased. This is because information about the optimal parameter values for the given test set is supplied to the training procedure. The test set then no longer constitutes new (yet unseen) data.

A valid cross-validation design must determine the optimal value of all parameters without taking the information in the test data into account. The dataset therefore needs to be divided into: *Training Data* where the values of all 'standard' parameters are determined,

Development Data where the optimal values of 'tuning' parameters are determined by fitting models with their different values, and *Test Data* where the final models are tested.

Figure 10: Cross-Validation Design with Development Sets



Each horizontal bar represents the entire dataset. The test sets (red parts) always account for 10%, the development sets (blue parts) also account for 10% and the training sets (green parts) for 80%. For each of the 10 test sets, 9 different development sets are used to optimise the 'tuning' parameter (f.e. number of neighbours in k-NN). After the 'tuning' parameter value has been determined, it is used to fit the model on the union of the corresponding training and development set, and the resulting classifier is tested on the corresponding test dataset. The optimal parameter estimates from the development process are thus allowed to be different for each of the test sets.

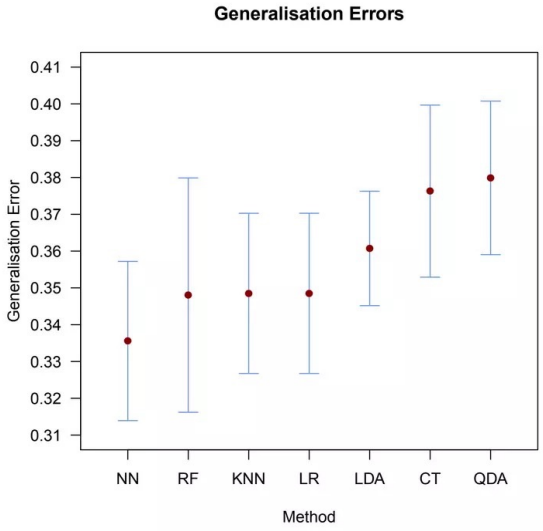
In order to use the available data most efficiently, this dataset division is here encompassed in the standard 10-fold cross-validation design (see Figure 10). To estimate the generalisation errors for all of the four methods:

1. The data set is split into 10 parts *(10 bars in all horizontal sections of Figure 10)*.
2. Each part serves once as a test set *(each stacked red bar in Figure 10)*.
3. For a given test set, each one of the remaining 9 parts is once used as the development set *(blue)* and the 8 other parts then serve as the training set *(green)*.
4. For a given test set and a given development set *(that is, for each one of the 90 rows of Figure 10)*, every model is fitted once for every 'tuning' parameter value on the corresponding 8 training data *(green)* parts.
5. For a given test set, the optimal value of the 'tuning' parameter is decided based on the mean error from the corresponding 9 development sets *(rows 1-9, . . . , 82-90)*.
6. For a given test set, the optimal 'tuning' parameter value is used to fit the final model on the 8 training data *(green)* parts and 1 development data part *(blue)* combined. The misclassification rate is computed based on this model *(we now get one misclassification rate for the rows 1-9, one for the rows 10-18 and so on)*.
7. The generalisation error of the method is then given by the mean of the 10 misclassification rates computed on the 10 different test sets.

This design is the same for all four methods. However, since the methods differ, small variations are present in the way the optimal values for their 'tuning' parameters are determined. For each development set, values of 1-100 are tried for k in k-NN, values of 1-7 in the case of RF's mtry and values 1-20 in the case of NN's size (decay is held constant on 0.01 throughout the analysis). For a given test set, the optimal parameter value is then determined as the one given the lowest mean classification rate over all development set for a given test set. Slightly different approach is preferred for CTs. The optimal size of a CT is determined not necessarily by the best performing CT, but by the simplest one with the misclassification rate within one standard deviation from the optimal model

All in all, 9000 k-NNs, around 600 CTs (the exact number varies), 630 RFs (each based on 500 CTs) and 1800 NNs are fitted in the development phase. Each of the seven methods is then fitted 10 additional times to compute the generalisation errors.

Figure 11: Comparison of the Seven Methods



Means and standard deviations of the generalisation errors for each method are plotted.

The resulting estimates of the generalisation errors are plotted in red in Figure 11. NNs perform best, with the estimated error of approximately 33.6%. The optimal sizes of the hidden layer are in the range 3-12, with the median of 5.5. Therefore, rather simple NN proves to have the largest generalisable predictive potential.

RFs, k-NN and LR perform almost exactly the same, with the generalised error rate of approximately 34.8%. Remarkably, the optimal number of variables for selection (mtry) in RFs is equal to 1 in all 10 data sets. This provides substantial evidence that indeed 1 is the value with the greatest predictive potential. The optimal number of neighbours for k-NN varies a lot (what is not surprising given the very flat character of its misclassification error curve) from 28 to 65 with the median of 40.5.

LDA comes fifth with the estimate generalised error rate of around 36.1%, the sixth are CTs (37.3%). The optimal size of a CT is almost always equal to 5; this means that rather

simple trees are the ones most appropriate for this data. QDA performs worst, with the generalised error rate equal to 38.0%.

The differences between some of the methods are actually very small in comparison to the standard deviations of the generalisation errors obtained. RF, k-NN and LR all fall within one standard deviation from the estimated generalisation error of NN. These methods therefore perform comparably well. On the contrary, LDA, CT and QDA do worse than NN even when uncertainty expressed by its standard deviation is taken into account. It should also be noted that the standard deviation of the generalisation error estimates is clearly largest in the case of RFs. Accuracy of RF predictions for new data is more variable than that of the other methods.

To sum up, the generalisation error estimates tell us to prefer NN. Contrary to LDA, CT and QDA, the k-NN, LR also perform satisfactory well. In a real-life situation, other factors than prediction accuracy, like computational costs or interpretability, may play an important role. In that scenario LR would be suggested as a computationally cheap method with very good prediction accuracy, virtually no assumptions, simple interpretation and no need for 'tuning'.

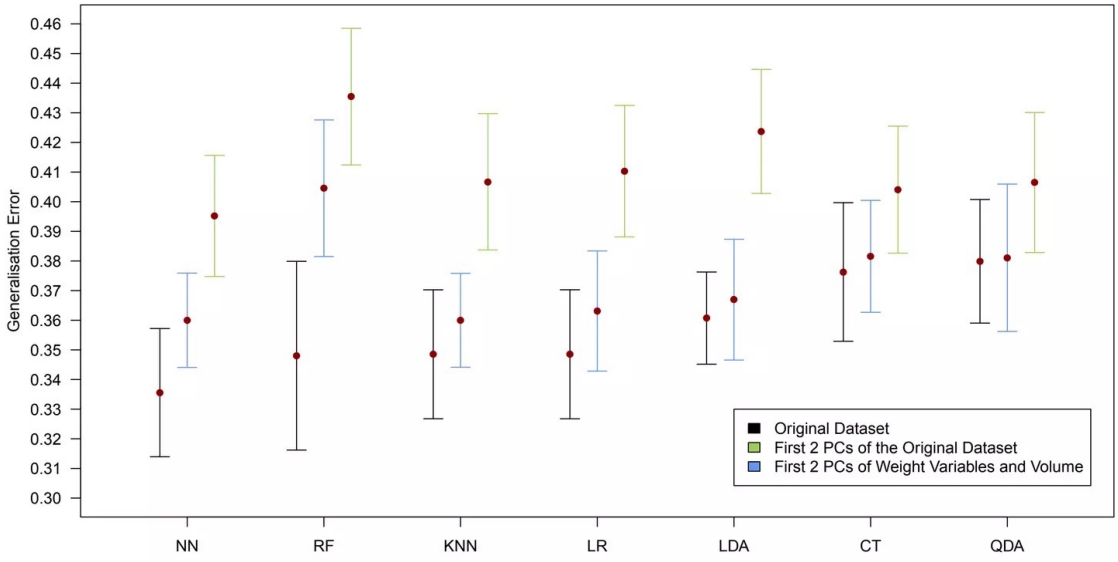
6 Extensions

In the last section I look at two different problems. First, I examine the performance of the 7 methods in a 2-dimensional space spanned by the first two PCs. Recalling the results from Section 2, I compare and contrast usage of the first two PCs of the (standardised) original variables and the usage of the first two PCs of the weight-related variables and volume only (not the original size-related measurements). I show that the understanding of the data gained in Section 2 dramatically improves prediction accuracy here. Second, I compare NN, RF and CT in their accuracy of predicting the response coded as a numerical variable.

6.1 Classification in 2 Dimensions and Visualisation of the Classifiers

For purposes of dimensionality reduction and visualisation, the (standardised) original dataset is reduced into two dimensions using the first two PCs. Exactly the same analysis as in section 5 is then applied to this two-dimensional dataset. Figure 12 shows the resulting generalisation error estimates plotted with green standard deviation bars. They can be easily compared to the results from the previous section, which are plotted with black bars here.

Figure 12: Performance of Classifiers on Three Variants of Data



Means and standard deviations of the generalisation errors for each method and each dataset are plotted.

It is easy to see that the generalisation error increased substantially after applying this dimensionality reduction. All of the estimated mean generalisation errors for the two-dimensional dataset lie above the upper standard deviation bar of the error from the original dataset. This means that the two original PCs leave out a substantial portion of the original data variance which is useful for classification. Most strikingly, the estimated

generalisation error of RF increased from 34.8% to 43.5%. Apparently, RF can make good use of the variance used in the third and the following PCs. Although the best performance is again obtained by NN, the generalisation error increased by almost 6%. It can be concluded that the first two original PCs leave out a substantial portion of the data variance that is useful for prediction. Using them would markedly decrease the performance of the tested methods.

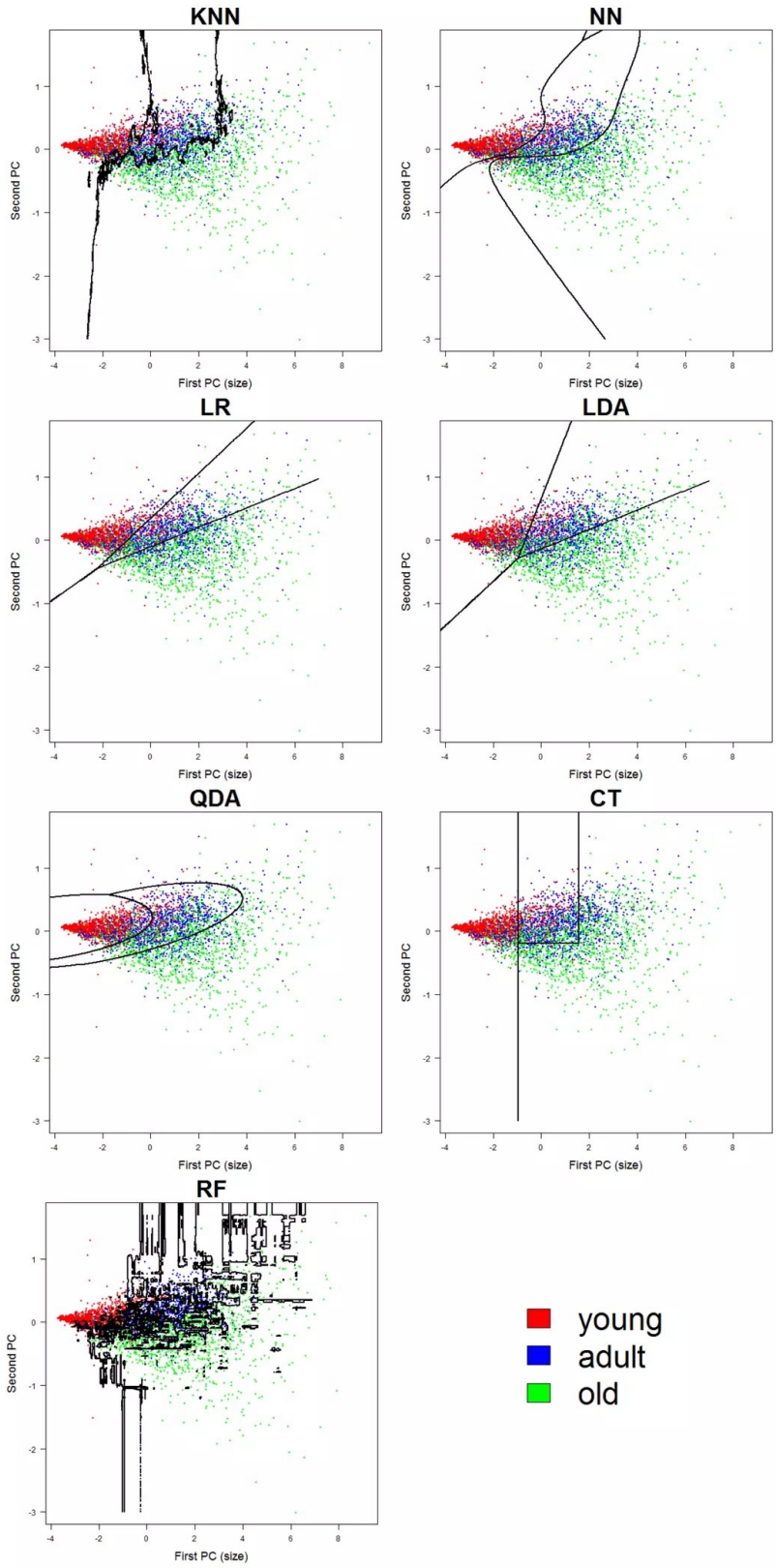
A very different story appears if we work with the first 2 PCs proposed in Section 2 which are based on 5 variables: 4 weight-related variables and 1 variable approximating volume (the 3-way product of *Lenght*, *Diameter* and *Height*). As I explained in Section 2, this constitutes a sensible transformation of the original variables, it explains the curvature between the first 2 PCs of the original dataset, and it appears to separate the classes rather well.

Figure 12 provides substantial support for using these weight-and-volume based PCs (from now on called WVB PCs) instead of the original ones. The generalisation error estimates plotted with blue bars refer to the results of applying the same cross-validation procedure as was applied in Section 5 to the first 2 WVB PCs. The estimated error rates are substantially smaller for these PCs than for the ones from the original data. For all methods, the mean generalised errors are much higher than the mean generalised errors on the first 2 WVB PCs. What is even more, If we adjust the WVB PC results for their standard deviations, we see that the original PCs lead to errors that are higher than the upper blue whiskers. Additionally, for all methods with the exceptions of NN and RF, the WVB PCs lead to comparable estimates to the ones from the whole dataset. The small amount of discarded information present in WVB PCs can only be used by NN and RF to improve the predictive performance. And only RF seems to do substantially worse than it did in the whole original dataset.

Since WVB PCs improve dimensionality reduction dramatically, I use them for plotting the classifiers (Figure 13). All 7 methods are plotted, with the best performing k-NN in the upper-left panel, the second best performing NN in the panel of its right and so on until the worst performing RF in the bottom panel. All points are plotted, differing in colour based on the class label, and semi-transparent to simplify examination of high-density regions. The boundaries plotted are based on the training on all 4175 observations with the 'tuning' parameters determined from the previous analysis (k = 45 in k-NN, size of NN = 7 with decay = 0.01, size of CT = 3, mtry in RF is 1).

The boundaries of the first two best performing methods (k-NN and NN) are similar in their shape in high-density regions. They both seem to have found classification boundaries close to the 'true' classification boundaries. The advantage of NN would be the high degree of smoothness. LR and LDA have their boundaries linear, rather similar to each other and they also seem to discriminate between the classes well. The next comes QDA with its typical elliptical shapes. CT is simple, separating the two-dimensional space into three regions. Finally, RF decision boundaries are very complex, as it is also anticipated since they are based on 500 different trees.

Figure 13: Visualisations of the Classifiers on the First 2 WVB PCs

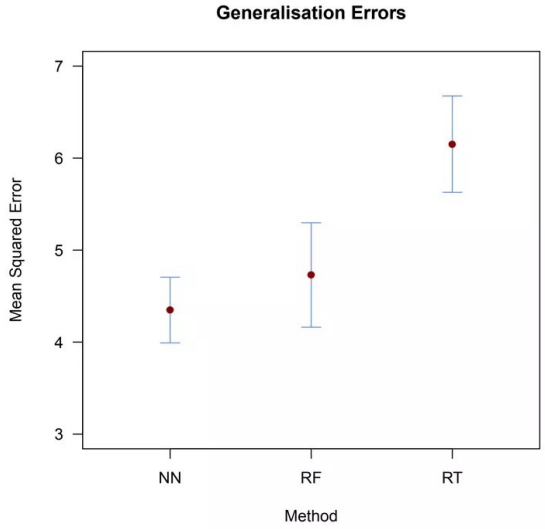


All 4175 points are plotted with their colour based on the class label, and semi-transparent to simplify examination of high-density regions. 'The first 2 WVB PCs' stands for the first 2 PCs from the dataset of 5 variables: the 4 weight-related variables and the 3-way product of *Length*, *Diameter* and *Height* approximating the volume.

6.2 Regression with NN, RF and RT

Lastly, approaching the response as a continuous variable, not a factor, is attempted. For this purpose, three methods are used for prediction: NN, RF and regression trees (RT, the continuous equivalent of CT). Exactly the same cross-validation design is implemented for the estimation of the generalisation error now as was implemented in Section 5. The only difference being that the mean square error, rather than the misclassification rate, is used to estimate the generalisation error.

Figure 14: Comparison of Regression Methods



Means and standard deviations of the generalisation errors for each method are plotted.

Figure 14 plots the means and the standard deviations obtained for the three regression methods. The results closely resemble the results from classification. NNs (of median size 4) do best, with the estimated mean square error of approximately 4.35. RFs (with median mtry of 1) perform slightly worse (4.73) but the difference is not too big. On the contrary, the prediction of RTs (of median size 6.5) is, on average, worst. The estimated mean square error is 6.15 which lies high above the estimates for the other two methods, even if the whiskers are used to take uncertainty into account.

7 Conclusion

The task to predict the age of abalone was systematically approached in this paper. First, the dataset was explored, the outliers deleted and the variables appropriately standardised. Then, PCA was used to further investigate the variance structure in the data. Further investigation of the non-linear relationship between the first two PCs lead to a specific transformation when the approximated shell volume was used instead of three one-dimensional measurements. This finding lead to a much better separation of the classes.

ROC curves were used to compare LR, LDA and QDA in terms of their specificity and sensitivity when the response was treated as a binary variable. LR and LDA did much better than QDA. For the three-class quantisation of the response, it was showed how cross-validation can be used to determine the optimal k value for k-NN. In this case, the estimate was very variable with the best value at $k = 45$.

Assessment of the generalisation error of 7 classifiers was performed using a complex cross-validation design with nested loops and development sets. NN proved to have the lowest generalisation error, with RF, k-NN and LR coming closely second. The WVB PCs discovered in the PCA section were used to reduce dimensionality of the data and visualise the classification boundaries. Lastly, the problem was tackled as a regression, with NN performing best, RF second best and RT worst.

References

[1] Victorian Abalone Divers Association (2010). *Anatomy - Abalone (Haliotis Sp.)* Online text, checked on 20.4.2013, [http://www.vada.com.au/Anatomy.html].

[2] Department of Fisheries, Government of Western Australia (2011). *Fisheries Fact Sheet: Abalone.* Online text, checked on 20.4.2013, [http://www.fish.wa.gov.au/Documents/recreational_fishing/fact_sheets/fact_sheet_abalone.pdf].

[3] Warwick J Nash, Tracy L Sellers, Simon R Talbot, Andrew J Cawthorn and Wes B Ford (1994). *The Population Biology of Abalone (Haliotis species) in Tasmania. I. Blacklip Abalone (H. rubra) from the North Coast and Islands of Bass Strait.* Sea Fisheries Division, Technical Report, No. 48. Tokyo.

[4] Center for Machine Learning and Intelligent Systems (1995). *Data Set Description* Online text, checked on 20.4.2013, [http://archive.ics.uci.edu/ml/datasets/Abalone].

Appendix: R Code

```
#################### 1 Pre-processing ####################

library(MASS); library(nnet); library(rpart); library(randomForest);
library(plotrix); library(class)


### Reading data in

shellfish <- data.frame(read.table("X.txt"),read.table("Y.txt"))
colnames(shellfish) <- c("Length", "Diameter", "Height", "Whole weight",
                         "Shucked weight", "Viscera weight", "Shell weight", "Rings")


### Basic check of the type and values of the variables. There are no missing data.

summary(shellfish); str(shellfish)


### 1.1: Search for outliers in a boxplot of standardised variables

pdf(file = "outliers.pdf", width = 12.5, height = 6)
par(mar = c(2, 4, 0, 2) + 0.1)
boxplot(scale(shellfish), col = "darkolivegreen3", ylab = "Standardised Value", las = 1)
text(x = c(3,3), y = c(10.5,22), labels = c("1418","2052"))
dev.off()


### 1.2: Identification and deletion of the outliers

identify(shellfish$Height) # observations 1418 and 2052
shellfish <- shellfish[-c(1418,2052),]


### Histograms of all 8 variables (with the 2 outliers deleted)

pdf(file = "histograms.pdf", width = 12, height = 6)
par(mfrow = c(2,4))
for (i in c(8,1:7)){
  truehist(shellfish[,i], main = names(shellfish)[i], xlab = "",
           col = "darkolivegreen3", las = 1)
  rug(shellfish[,i], col = rgb(0,0,0,0.05))
  if (i<8) lines(density(shellfish[,i]), col = "darkred")
}
dev.off()


### 1.3 Boxplot of the standardised variables (using scale())

pdf(file = "boxplots.pdf", width = 12.5, height = 6)
par(mar = c(2, 4, 0, 2) + 0.1)
boxplot(scale(shellfish[,1:7]), col = "darkolivegreen3",
        ylab = "Standardised Value", las = 1)
```

```
dev.off()
#################### 2 PCA ####################

### 2.1 Computing the two PCA variants.

pca1 <- princomp(shellfish[,1:7], cor = FALSE)
pca2 <- princomp(shellfish[,1:7], cor = TRUE)

### 2.1 & 2.2 Plot of the cumulative explained variance for the two PCA variants

pdf(file = "pca.pdf", width = 12, height = 5)
par(mfrow = c(1,2), mar = c(4, 4, 1, 2) + 0.1)

plot(cumsum(summary(pca2)$sdev^2/sum(summary(pca2)$sdev^2)), type = "s",
     col = "darkolivegreen3", xlab = "Number of Principal Components",
     ylab = "Cumulative Proportion of Variance", yaxp = c(0.90, 1, 10),
     ylim = c(0.9,1),las = 1)
lines(cumsum(summary(pca1)$sdev^2/sum(summary(pca1)$sdev^2)), type = "s",
      col = "darkred")
abline(0.9,0, lty = 2); abline(0.99,0, lty = 2)
legend(4.5,0.92, legend = c("Covariance Matrix", "Correlation Matrix"),
       fill = c("darkred", "darkolivegreen3"))

### 2.1 Plot of the first PC loadings against the variance of the original variables

plot(-pca1$loadings[,1][order(sapply(shellfish[,1:7], var))] ~
       sort(sapply(shellfish[,1:7], var)), type = "o", col = "darkred", las = 1,
    ylim = c(0,1), xlab = "Variance of Original Variables", ylab = "First PCA Loadings")
lines(-pca2$loadings[,1][order(sapply(shellfish[,1:7], var))] ~
        sort(sapply(shellfish[,1:7], var)), type = "o", col = "darkolivegreen3")
legend(0.14,0.19, legend = c("Covariance Matrix", "Correlation Matrix"),
       fill = c("darkred", "darkolivegreen3"))
dev.off()

### 2.3 The biplot and the first two PCs plotted against each other, coloured
### according to their number of rings.

pdf(file = "components1.pdf", width = 12, height = 5)
par(mfrow = c(1,2), mar = c(4, 4, 4, 2) + 0.1)

biplot(pca2, cex = c(0.2, 0.5), xlab = "First PC", ylab = "Second PC", main = "Biplot",
     col= c(rgb(0,0.5,0, 0.2),"darkred"), pch = 16, las = 1, cex.axis = 0.75, scale = 0)
plot(-pca2$scores[,1], pca2$scores[,2], main = "First Two PCs shaded by Rings",
     col= rgb(255-50*(unclass(cut(shellfish$Rings,
            c(0, 7.5, 8.5, 9.5, 10.5, 11.5, 30)))-1), 50*(unclass(cut(shellfish$Rings,
         c(0, 7.5, 8.5, 9.5, 10.5, 11.5, 30)))-1), rep(0,4175), 120, maxColorValue=255),
      pch = 16, las = 1, cex = 0.4, xlab = "First PC", ylab = "Second PC")
     color.legend(-6.5,0.25,-5.5,1.45, legend = c("<8","8","9","10","11","11<"),
                  gradient = "y", align = "rb",
                  rect.col = c(rgb(255,0,0,120, maxColorValue=255),
                               rgb(205,50,0,120, maxColorValue=255),
                               rgb(155,100,0,120, maxColorValue=255),
                               rgb(105,150,0,120, maxColorValue=255),
                               rgb(55,200,0,120, maxColorValue=255),
                               rgb(5,250,0,120, maxColorValue=255)))
text(-6,1.6, "Rings")
```

```
dev.off()
### 2.3 Plots of the relationships of the variables and the volume

pdf(file = "shape.pdf", width = 12, height = 4)
par(mfrow = c(1,3), mar = c(4, 4, 3, 2) + 0.1)

### 2.3 Plot of the first PC of weight variables against the volume

plot(princomp(-shellfish[,4:7], cor = TRUE)$scores[,1], shellfish[,1]*shellfish[,2]*shellfish[,3],
    col= rgb(255-50*(unclass(cut(shellfish$Rings, c(0, 7.5, 8.5, 9.5, 10.5, 11.5, 30)))-1),
             50*(unclass(cut(shellfish$Rings, c(0, 7.5, 8.5, 9.5, 10.5, 11.5, 30)))-1),
               rep(0,4175), 120, maxColorValue=255), cex.lab = 1.2,
     pch = 16, las = 1, cex = 0.4, xlab = "First PC of Weight Variables Only",
     ylab = "Approximated Volume")
color.legend(6,-0,7,0.04, legend = c("<8","8","9","10","11","11<"),
             gradient = "y", align = "rb",
         rect.col = c(rgb(255,0,0,120, maxColorValue=255), rgb(205,50,0,120, maxColorValue=255),
                rgb(155,100,0,120, maxColorValue=255), rgb(105,150,0,120, maxColorValue=255),
                rgb(55,200,0,120, maxColorValue=255), rgb(5,250,0,120, maxColorValue=255)))
text(6.5,0.044, "Rings")

### 2.3 Plot of the first PC of the weight variables and the first PC of the shell measurements

plot(princomp(-shellfish[,4:7], cor = TRUE)$scores[,1],
     princomp(shellfish[,1:3], cor = TRUE)$scores[,1],
   col= rgb(255-50*(unclass(cut(shellfish$Rings, c(0, 7.5, 8.5, 9.5, 10.5, 11.5, 30)))-1),
             50*(unclass(cut(shellfish$Rings, c(0, 7.5, 8.5, 9.5, 10.5, 11.5, 30)))-1),
               rep(0,4175), 120, maxColorValue=255), cex.lab = 1.2,
     pch = 16, las = 1, cex = 0.4, xlab = "First PC of Weight Variables Only",
     ylab = "First PC of Shell-Size Variables only")
color.legend(6,-6.2,7,-3, legend = c("<8","8","9","10","11","11<"),
             gradient = "y", align = "rb",
         rect.col = c(rgb(255,0,0,120, maxColorValue=255), rgb(205,50,0,120, maxColorValue=255),
                rgb(155,100,0,120, maxColorValue=255), rgb(105,150,0,120, maxColorValue=255),
                rgb(55,200,0,120, maxColorValue=255), rgb(5,250,0,120, maxColorValue=255)))
text(6.5,-2.7, "Rings")

### Plot of the first 2 PC of the weight variables and volume combined

plot(princomp(data.frame(shellfish[,4:7],shellfish[,1]*shellfish[,2]*shellfish[,3]),
              cor = TRUE)$scores[,1],
     princomp(data.frame(shellfish[,4:7],shellfish[,1]*shellfish[,2]*shellfish[,3]),
              cor = TRUE)$scores[,2],
   col= rgb(255-50*(unclass(cut(shellfish$Rings, c(0, 7.5, 8.5, 9.5, 10.5, 11.5, 30)))-1),
             50*(unclass(cut(shellfish$Rings, c(0, 7.5, 8.5, 9.5, 10.5, 11.5, 30)))-1),
               rep(0,4175), 120, maxColorValue=255),
     pch = 16, las = 1, cex = 0.4, cex.lab = 1.2,
     xlab = "First PC of the Weight Variables and Volume",
     ylab = "Second PC of the Weight Variables and Volume")
color.legend(-3.1,-3,-2,-1.7, legend = c("<8","8","9","10","11","11<"),
             gradient = "y", align = "rb",
         rect.col = c(rgb(255,0,0,120, maxColorValue=255), rgb(205,50,0,120, maxColorValue=255),
                rgb(155,100,0,120, maxColorValue=255), rgb(105,150,0,120, maxColorValue=255),
                rgb(55,200,0,120, maxColorValue=255), rgb(5,250,0,120, maxColorValue=255)))
text(-2.5,-1.52, "Rings")
dev.off()
```

```
#################### 3 Recevier Operating Curve (ROC) ####################

### 3.1 Creating the new binary variable

shellfish <- data.frame(scale(shellfish[,1:7]), shellfish[,8], ifelse(shellfish$Rings<10, 0, 1))
colnames(shellfish)[8:9] <- c("Rings","Rings > 9")
table(shellfish[,"Rings > 9"])

### 3.1 Splitting the dataset into a training and a test set (50:50)

set.seed(2013)
train.indices <- sample(1:4175, 2088)
shellfish.train <- shellfish[train.indices,]
shellfish.test <- shellfish[-train.indices,]

### 3.1 LDA, QDA and LR model fitting

lda1 <- lda(shellfish.train[,1:7], shellfish.train[,9])
qda1 <- qda(shellfish.train[,1:7], grouping = shellfish.train[,9])
lr1  <- glm(shellfish.train[,9] ~ ., family = binomial(link = "logit"), data = shellfish.train[,1:7])

### 3.1 Computing class predictions for the test set

proba.lda <- predict(lda1, newdata = shellfish.test[,1:7])$posterior[,2]
proba.qda <- predict(qda1, newdata = shellfish.test[,1:7])$posterior[,2]
proba.lr  <- predict(lr1, newdata = shellfish.test[,1:7], type = "response")

### 3.3 Splitting the dataset 90:10 into a training and a test set

train.indices.90.10 <- sample(1:4175, 3758)
shellfish.train.90.10 <- shellfish[train.indices.90.10,]
shellfish.test.90.10 <- shellfish[-train.indices.90.10,]

### 3.3 LDA, QDA and LR model fitting

lda1.90.10 <- lda(shellfish.train.90.10[,1:7], shellfish.train.90.10[,9])
qda1.90.10 <- qda(shellfish.train.90.10[,1:7], grouping = shellfish.train.90.10[,9])
lr1.90.10  <- glm(shellfish.train.90.10[,9] ~ ., family = binomial(link = "logit"),
                  data = shellfish.train.90.10[,1:7])

### 3.3 Computing class predictions in the 50% test set

proba.lda.90.10 <- predict(lda1.90.10, newdata = shellfish.test.90.10[,1:7])$posterior[,2]
proba.qda.90.10 <- predict(qda1.90.10, newdata = shellfish.test.90.10[,1:7])$posterior[,2]
proba.lr.90.10  <- predict(lr1.90.10, newdata = shellfish.test.90.10[,1:7], type = "response")
```

```
### 3.1 Plotting ROC curves

### 3.1-3.3 Setting up the function to compute the sensitivity and specificity estimates

cvec <- seq(0.000, 1, length = 1001)

compute.SS <- function(pred, true){
  specif <- numeric(length(cvec))
  sensit <- numeric(length(cvec))
  for (i in 1:length(cvec)){
    sensit[i] <- sum(pred> cvec[i] & true==1)/sum(true==1)
    specif[i] <- sum(pred<=cvec[i] & true==0)/sum(true==0)
  }
  data.frame(sensit, specif)
}

### 3.1-3.3 Computing specificities and sensitivities for both splits

sensit1 <- compute.SS(proba.lda, shellfish.test[,9])[,1]
specif1 <- compute.SS(proba.lda, shellfish.test[,9])[,2]
sensit2 <- compute.SS(proba.qda, shellfish.test[,9])[,1]
specif2 <- compute.SS(proba.qda, shellfish.test[,9])[,2]
sensit3 <- compute.SS(proba.lr, shellfish.test[,9])[,1]
specif3 <- compute.SS(proba.lr, shellfish.test[,9])[,2]

sensit1.90.10 <- compute.SS(proba.lda.90.10, shellfish.test.90.10[,9])[,1]
specif1.90.10 <- compute.SS(proba.lda.90.10, shellfish.test.90.10[,9])[,2]
sensit2.90.10 <- compute.SS(proba.qda.90.10, shellfish.test.90.10[,9])[,1]
specif2.90.10 <- compute.SS(proba.qda.90.10, shellfish.test.90.10[,9])[,2]
sensit3.90.10 <- compute.SS(proba.lr.90.10, shellfish.test.90.10[,9])[,1]
specif3.90.10 <- compute.SS(proba.lr.90.10, shellfish.test.90.10[,9])[,2]

### 3.1-3.3 Plots of the ROC curves for the both cases

pdf(file = "roc.pdf", width = 12, height = 6)
par(mfrow = c(1,2), mar = c(5, 4, 4, 2) + 0.1)

## The 50:50 split

plot(specif1,sensit1, xlab = "Specificity", ylab = "Sensitivity", type = "l", las = 1,
     pty = "s", col = "darkolivegreen3", main = "50:50 Training:Test Split")
points(specif2, sensit2, type = "l", col = "darkred")
points(specif3, sensit3, type = "l", col = "cornflowerblue")
legend(0, 0.2, legend = c("LR","LDA","QDA"), fill = c("cornflowerblue","darkolivegreen3", "darkred"))

### The 90:10 split

plot(specif1.90.10,sensit1.90.10, xlab = "Specificity", ylab = "Sensitivity", type = "l", las = 1,
     pty = "s", col = "darkolivegreen3", main = "90:10 Training:Test Split")
points(specif2.90.10, sensit2.90.10, type = "l", col = "darkred")
points(specif3.90.10, sensit3.90.10, type = "l", col = "cornflowerblue")
legend(0, 0.2, legend = c("LR","LDA","QDA"), fill = c("cornflowerblue","darkolivegreen3", "darkred"))

dev.off()
```

```
#################### 4 k-NN ####################

### Creating the new dependent variable with 3 classes

shellfish <- data.frame(shellfish, factor(ifelse(shellfish$Rings<9, 1,
                                                 ifelse(shellfish$Rings<11, 2, 3))))
colnames(shellfish)[10] <- "Rings (3 classes)"
table(shellfish[,"Rings (3 classes)"])

### Preparing the indexing vector for 10-fold cross-validation

indices <- numeric(0)
for (i in 1:10){
  indices <- c(indices, rep(i, 417 + i%%2))
}
indices <- factor(sample(indices)); rm(i)

### 4-6 This function finds the optimal number of neighbours as the number that minimises
### the mean misclassificaiton error in cross-validation. Predictors and the corresponding true
### classes are supplied first. 'cv.factor' is a factor vector that defines membership to
### different cross-validation groups for every case (in case of 10-fold crossvalidation
### it contains 10 classes). The maximum number of neighbours to be checked for is supplied
### in parameter 'max.k' (the search always starts from 1 neighbour). The function outputs the
### calculated mean missclassification errors for every value of k for every test set.

optimise.knn <- function(predictors, true.classes, cv.factor, max.k){
  misclass.rate <- matrix(NA, length(levels(cv.factor)), max.k)
    for (j in 1:max.k){
      for (d in as.numeric(levels(droplevels(cv.factor)))){
        classes <- knn(train = predictors[cv.factor != d,],
                       test = predictors[cv.factor == d,],
                       cl = true.classes[cv.factor != d],
                       k = j)
        misclass.rate[d,j] <- (sum(true.classes[cv.factor == d] != classes)
                               / nrow(predictors[cv.factor == d,]))
        rm(classes)
      }
    }
  apply(misclass.rate, 2, mean, na.rm = TRUE)
}

### Optimise the number of neighbours using the above function

cv.knn1 <- optimise.knn(shellfish[,1:7], shellfish[, "Rings (3 classes)"], indices, 100)

### Plot of the mean misclassification error as a function of the number of neighbours used

pdf(file = "knn.pdf", width = 12, height = 6)
par(mfrow = c(1,1), mar = c(4, 4, 2, 2) + 0.1)
plot(cv.knn1, type = "o", pch = 16, xlab = "The Number of Neighbours Considered",
     ylab = "Mean Misclassification Error", las = 1, col = "cornflowerblue",
     xaxp = c(0,100,20), yaxp = c(0.34, 0.44, 10), ylim = c(0.34, 0.44))
points(match(min(cv.knn1), cv.knn1),cv.knn1[match(min(cv.knn1), cv.knn1)],
       col = "darkred", pch = 16)
dev.off()
```

```
#################### 5 Comparing Methods ####################

### 5 & 6 This function finds the optimal value of the 'mtry' parameter in random forests.
### It is determined as the number that minimises the mean misclassification error in
### cross-validation. Predictors and the corresponding true classes are supplied first.
### 'cv.factor' is a factor vector that defines membership to different cross-validation groups
### for every case (in case of 10-fold crossvalidation it contains 10 classes).
### The maximum value of 'mtry' to be checked for is supplied in parameter 'max.mtry'
### (the search always starts from 1). The function outputs the mean misclassification errors
### from cross-validation for all values of 'mtry'.

optimise.rf <- function(predictors, true.classes, cv.factor, max.mtry = 7){
  max.mtry <- min(max.mtry, ncol(predictors))
  misclass.rate <- matrix(NA, length(levels(cv.factor)), max.mtry)
    for (j in 1:max.mtry){
      for (d in as.numeric(levels(droplevels(cv.factor)))){
        classes <- predict(randomForest(predictors[cv.factor != d,],
                                        true.classes[cv.factor != d],
                                        mtry = j),
                           predictors[cv.factor == d,])
        misclass.rate[d,j] <- (sum(true.classes[cv.factor == d] != classes)
                               / nrow(predictors[cv.factor == d,]))
        rm(classes)
      }
    }
  apply(misclass.rate, 2, mean, na.rm = TRUE)
}

### Find the optimal value of 'mtry'

cv.rf1 <- optimise.rf(shellfish[,1:7], shellfish[, "Rings (3 classes)"], indices)

### 5 & 6 This function finds the optimal value of the 'size' parameter in classification trees.
### It is determined as the smallest size with the mean misclassification error lying within
### one standard deviation from the minimised mean misclassification error (from cross-validation).
### Predictors and the corresponding true classes are supplied first. 'cv.factor' is a factor
### vector that defines membership to different cross-validation groups for every case
### (in 10-fold crossvalidation it contains 10 classes). The function outputs a list of the
### with size and the corresponding cp value (see ?rpart.control form more details).

optimise.ct <- function(predictors, true.classes, cv.factor){
  cpinfo <- rpart(true.classes ~ . , data = predictors,
                  control = rpart.control(xval = unclass(droplevels(cv.factor))))$cptable
  e <- rep(NA, dim(cpinfo)[1]-1)
  for (j in 1:(dim(cpinfo)[1]-1)){
    e[j] <- cpinfo[j,4]-cpinfo[dim(cpinfo)[1],4]-cpinfo[dim(cpinfo)[1],5]
  }
 cp <- ifelse(all(e>0), cpinfo[dim(cpinfo)[1],1], cpinfo[which.max(ifelse(e>0,NA,e)), 1])
  size <- cpinfo[match(cp, cpinfo[,1]),2]
  list(cp,size)
}


### Find the optimal CT size
```

```
cv.ct1 <- optimise.ct(shellfish[,1:7], shellfish[, "Rings (3 classes)"], indices)

### 5 & 6 This function finds the optimal value of the size parameter in neural networks.
### It is determined as the number that minimises the mean misclassification error in
### cross-validation. Predictors and the corresponding true classes are supplied first.
### 'cv.factor' is a factor vector that defines membership to different cross-validation groups
### for every case (in case of 10-fold crossvalidation it contains 10 classes).
### The maximum size to be checked for is supplied in parameter 'max.size' (the search always
### starts from 1). The function outputs the mean misclassification errors from cross-validation.

optimise.nn <- function(predictors, true.classes, cv.factor, max.size = 20){
  misclass.rate <- matrix(NA, length(levels(cv.factor)), max.size)
  for (j in 1:max.size){
    for (d in as.numeric(levels(droplevels(cv.factor)))){
      classes <- max.col(predict(nnet(true.classes[cv.factor != d] ~.,
                                      data = predictors[cv.factor != d,],
                                      size = j, decay = 0.01),
                                 predictors[cv.factor == d,]))

      misclass.rate[d,j] <- (sum(true.classes[cv.factor == d] != classes)
                             / nrow(predictors[cv.factor == d,]))
      rm(classes)
    }
  }
  apply(misclass.rate, 2, mean, na.rm = TRUE)
}

### Find the optimal NN size

cv.nn1 <- optimise.nn(shellfish[,1:7], shellfish[, "Rings (3 classes)"], indices)

### 5-6 A function that compares the generalised errors for the methods. The inputs are analogous
### to to optimise functions described above. The function outputs a list with the information
### on generalization error estimates from each of the cross-validation test sets. It also
### outputs the means of these values and all the values max.k, max.mtry and max.size selected.

compare.methods <- function(predictors, true.classes, cv.factor,
                            max.k = 100, max.mtry = 7, max.size = 20){

  ### Definition of development parameter vectors

  chosen.k    <- rep(NA, length(levels(cv.factor)))
  chosen.mtry <- rep(NA, length(levels(cv.factor)))
  chosen.cp   <- rep(NA, length(levels(cv.factor)))
  chosen.size <- rep(NA, length(levels(cv.factor)))
  chosen.ct.pars <- rep(list(NA), length(levels(cv.factor)))
  chosen.nnsize  <- rep(NA, length(levels(cv.factor)))

  ### Definition of output vectors

  misclass.rate.lda <- rep(NA, length(levels(cv.factor)))
  misclass.rate.qda <- rep(NA, length(levels(cv.factor)))
  misclass.rate.lr  <- rep(NA, length(levels(cv.factor)))
  misclass.rate.knn <- rep(NA, length(levels(cv.factor)))
  misclass.rate.rf  <- rep(NA, length(levels(cv.factor)))
  misclass.rate.ct  <- rep(NA, length(levels(cv.factor)))
```

```
misclass.rate.nn  <- rep(NA, length(levels(cv.factor)))

### The development phase. Optimising the 'tuning' parameters on the development sets using
### the optimise functions described above

for (v in 1:length(levels(cv.factor))){
 choosing.k      <- optimise.knn(predictors[cv.factor != v,], true.classes[cv.factor != v],
                                 cv.factor[cv.factor != v], max.k)
  chosen.k[v]       <- match(min(choosing.k), choosing.k)

 choosing.mtry    <- optimise.rf(predictors[cv.factor != v,], true.classes[cv.factor != v],
                                 cv.factor[cv.factor != v])
  chosen.mtry[v]    <- match(min(choosing.mtry), choosing.mtry)

 chosen.ct.pars[[v]] <- optimise.ct(predictors[cv.factor != v,], true.classes[cv.factor != v],
                                    cv.factor[cv.factor != v])
  chosen.cp[v]      <- chosen.ct.pars[[v]][[1]]
  chosen.size[v]    <- chosen.ct.pars[[v]][[2]]

 choosing.nnsize  <- optimise.nn(predictors[cv.factor != v,], true.classes[cv.factor != v],
                                 cv.factor[cv.factor != v])
  chosen.nnsize[v] <- match(min(choosing.nnsize), choosing.nnsize)
}

### Computation of predictions for each test set

for (v in 1:length(levels(cv.factor))){

 classes.lda <- predict(lda(predictors[cv.factor != v,], true.classes[cv.factor != v]),
                        newdata = predictors[cv.factor == v,])$class
 classes.qda <- predict(qda(predictors[cv.factor != v,], true.classes[cv.factor != v]),
                        newdata = predictors[cv.factor == v,])$class
  classes.lr  <- predict(multinom(true.classes[cv.factor != v] ~ .,
                                  data = predictors[cv.factor != v,]),
                        newdata = predictors[cv.factor == v,])
  classes.knn <- knn(train = predictors[cv.factor != v,],
                     test = predictors[cv.factor == v,],
                     cl = true.classes[cv.factor != v],
                     k = chosen.k[v])
  classes.rf  <- predict(randomForest(predictors[cv.factor != v,],
                                      true.classes[cv.factor != v],
                                      mtry = chosen.mtry[v]),
                         predictors[cv.factor == v,])
  classes.ct <- max.col(predict(rpart(true.classes[cv.factor != v] ~.,
                                      data = predictors[cv.factor != v,],
                                control = rpart.control(cp = chosen.cp[v], xval = 1)),
                                 predictors[cv.factor == v,]))
  classes.nn <- max.col(predict(nnet(true.classes[cv.factor != v] ~.,
                                     data = predictors[cv.factor != v,],
                                     size = chosen.nnsize[v], decay = 0.01),
                        predictors[cv.factor == v,]))

  ### Computation of the misclassification errors in each test set

  misclass.rate.lda[v] <- ( sum(true.classes[cv.factor == v] != classes.lda)
                            / nrow(predictors[cv.factor == v,]) )
```

```
    misclass.rate.qda[v] <- ( sum(true.classes[cv.factor == v] != classes.qda)
                              / nrow(predictors[cv.factor == v,]) )
    misclass.rate.lr[v]  <- ( sum(true.classes[cv.factor == v] != classes.lr)
                              / nrow(predictors[cv.factor == v,]) )
    misclass.rate.knn[v] <- (sum(true.classes[cv.factor == v] != classes.knn)
                             / nrow(predictors[cv.factor == v,]))
    misclass.rate.rf[v]  <- (sum(true.classes[cv.factor == v] != classes.rf)
                             / nrow(predictors[cv.factor == v,]))
    misclass.rate.ct[v]  <- (sum(true.classes[cv.factor == v] != classes.ct)
                             / nrow(predictors[cv.factor == v,]))
    misclass.rate.nn[v]  <- (sum(true.classes[cv.factor == v] != classes.nn)
                             / nrow(predictors[cv.factor == v,]))

   rm(classes.lda, classes.qda, classes.lr, classes.knn, classes.rf, classes.ct, classes.nn)
  }

  ### Formatting of the output

 error.rates <- c(mean(misclass.rate.lda), mean(misclass.rate.qda), mean(misclass.rate.lr),
             mean(misclass.rate.knn), mean(misclass.rate.rf), mean(misclass.rate.ct),
                  mean(misclass.rate.nn))
  names(error.rates) <- c("lda", "qda", "lr", "knn", "rf", "ct", "nn")

 all.error.rates <- data.frame(misclass.rate.lda, misclass.rate.qda, misclass.rate.lr,
                               misclass.rate.knn, misclass.rate.rf, misclass.rate.ct,
                                misclass.rate.nn)
  colnames(all.error.rates) <- c("lda", "qda", "lr", "knn", "rf", "ct", "nn")

 list(error.rates, all.error.rates, data.frame(Chosen.k = chosen.k, Chosen.mtry = chosen.mtry,
                                    Chosen.Size = chosen.size, Chosen.cp = chosen.cp,
                                              Chosen.nnsize = chosen.nnsize))
}

### 5-6 Computation of the generalisation errors for all methods

analysis.output <- compare.methods(shellfish[,1:7], shellfish[, "Rings (3 classes)"],
                                   indices, max.k = 100)

### A simple function to produce error bars

error.bar <- function(x, y, upper, lower=upper, length=0.1,...){
  arrows(x,y+upper, x, y-lower, angle=90, code=3, length=length, ...)}

### 5 Plot of the mean generalisation errors of the methods and their standard deviations

pdf(file = "comparison.pdf", width = 6, height = 6)

plot(sort(analysis.output[[1]]), axes = FALSE, xlab = "Method", ylab = "Generalisation Error",
     col = "darkred", pch = 16, xlim = c(0.5,7.5), ylim = c(0.31, 0.41), pty = "s",
     main = "Generalisation Errors")
box();axis(2, at = 1:11/100+0.3, las = 1)

axis(1,at = 1:7,labels = toupper(names(sort(analysis.output[[1]]))))
error.bar(1:7, sort(analysis.output[[1]]),
       apply(analysis.output[[2]], 2,sd)[order(analysis.output[[1]])], col = "cornflowerblue")
points(1:7, sort(analysis.output[[1]]), col = "darkred",pch = 16)
```

```
dev.off()
################## 6 Extensions ###################

### Compare the methods in the space spanned by the first 2 original PCs

pca2predictors <-  data.frame(pca2$scores[,1:2])

analysis.output.pca2 <- compare.methods(pca2predictors, shellfish[, "Rings (3 classes)"], indices)

### Compare the methods in the space spanned by the first 2 WVB PCs

pca4predictors <-  data.frame(princomp(data.frame(shellfish2[,4:7],
                                       shellfish2[,1]*shellfish2[,2]*shellfish2[,3]),
                                      cor = TRUE)$scores[,1:2])

analysis.output.pca4 <- compare.methods(pca4predictors, shellfish[, "Rings (3 classes)"], indices)

### 6.1 Plot of the mean generalisation errors and their standard deviations for all methods
### on three datasets

pdf(file = "comparisonpca2a.pdf", width = 12, height = 6)
par(mfrow = c(1,1), mar = c(2, 4, 1, 2) + 0.1 )

plot(sort(analysis.output[[1]]), axes = FALSE, xlab = "Method", ylab = "Generalisation Error",
     col = "darkred", pch = 16, xlim = c(0.5,7.5), ylim = c(0.30, 0.46), type = "n")
box();axis(2, at = 1:17/100+0.29, las = 1)
axis(1,at = 1:7,labels = toupper(names(sort(analysis.output[[1]]))))

error.bar(1:7-0.2, sort(analysis.output[[1]]),
        apply(analysis.output[[2]], 2,sd)[order(analysis.output[[1]])], col = "black")
points(1:7 -0.2, sort(analysis.output[[1]]), col = "darkred",pch = 16)

error.bar(1:7, analysis.output.pca4[[1]][order(analysis.output[[1]])],
       apply(analysis.output.pca4[[2]], 2,sd)[order(analysis.output[[1]])], col = "cornflowerblue")
points(1:7, analysis.output.pca4[[1]][order(analysis.output[[1]])], col = "darkred",pch = 16)

error.bar(1:7+0.2, analysis.output.pca2[[1]][order(analysis.output[[1]])],
          apply(analysis.output.pca2[[2]], 2,sd)[order(analysis.output[[1]])],
          , col = "darkolivegreen3")
points(1:7+0.2, analysis.output.pca2[[1]][order(analysis.output[[1]])], col = "darkred",pch = 16)

legend(5,0.33, legend = c("Original Dataset", "First 2 PCs of the Original Dataset",
                          "First 2 PCs of Weight Variables and Volume"),
       fill = c("black", "darkolivegreen3", "cornflowerblue"))
dev.off()

### 6.1 Create a grid for the plotting surface

x <- seq(-9,7,0.01)
y <- seq(-3,2,0.01)
z <- as.matrix(expand.grid(x,y),0)
colnames(z) <- c("Comp.1", "Comp.2")
m <- length(x)
n <- length(y)
```

```
### 6.1 Compute predictions for all methods on the new dataset (First 2 WVB PCs)

ldap2 <- predict(lda(pca4predictors, shellfish[,"Rings (3 classes)"]), z)$class
qdap2 <- predict(qda(pca4predictors, shellfish[,"Rings (3 classes)"]), z)$class
lrp2  <- predict(multinom(shellfish[,"Rings (3 classes)"] ~ ., data = pca4predictors), z)
knnp2 <- knn(train = pca4predictors, test = z, cl = shellfish[,"Rings (3 classes)"], k = 45)
rfp2  <- predict(randomForest(pca4predictors, shellfish[,"Rings (3 classes)"], mtry = 1), z)
ctp2 <- max.col(predict(rpart(shellfish[,"Rings (3 classes)"] ~.,data = pca4predictors,
                        control = rpart.control(cp = 0.01, xval = 1)), data.frame(z)))
nnp2 <- max.col(predict(nnet(shellfish[,"Rings (3 classes)"] ~.,data = pca4predictors,
                             size = 7, decay = 0.01), data.frame(z)))

### 6.1 Classes are 1,2 and 3, so set contours at 1.5 and 2.5

pdf("visx.pdf", width = 8, height = 16)
par(mfrow = c(4,2), mar = c(4, 4, 3, 2) + 0.1 )

plot(pca4predictors, pch = 16, las = 1, cex = 0.35, pty = "s", xlab = "First PC (size)",
   ylab = "Second PC", main = "KNN", col= ifelse(shellfish[,"Rings (3 classes)"] == 1, "red",
                ifelse(shellfish[,"Rings (3 classes)"] == 2, "blue", "green")))
contour(x,y,matrix(knnp2,m,n), levels=c(1.5,2.5), add=TRUE, d=FALSE, lwd =1, col = "black")

plot(pca4predictors, pch = 16, las = 1, cex = 0.35, pty = "s", xlab = "First PC (size)",
   ylab = "Second PC", main = "NN", col= ifelse(shellfish[,"Rings (3 classes)"] == 1, "red",
                ifelse(shellfish[,"Rings (3 classes)"] == 2, "blue", "green")))
contour(x,y,matrix(nnp2,m,n), levels=c(1.5,2.5), add=TRUE, d=FALSE, lwd =1, col = "black")

plot(pca4predictors, pch = 16, las = 1, cex = 0.35, pty = "s", xlab = "First PC (size)",
   ylab = "Second PC", main = "LR", col= ifelse(shellfish[,"Rings (3 classes)"] == 1, "red",
                ifelse(shellfish[,"Rings (3 classes)"] == 2, "blue", "green")))
contour(x,y,matrix(lrp2,m,n), levels=c(1.5,2.5), add=TRUE, d=FALSE, lwd =1, col = "black")

plot(pca4predictors, pch = 16, las = 1, cex = 0.35, pty = "s", xlab = "First PC (size)",
   ylab = "Second PC", main = "LDA", col= ifelse(shellfish[,"Rings (3 classes)"] == 1, "red",
                ifelse(shellfish[,"Rings (3 classes)"] == 2, "blue", "green")))
contour(x,y,matrix(ldap2,m,n), levels=c(1.5,2.5), add=TRUE, d=FALSE, lwd =1, col = "black")

plot(pca4predictors, pch = 16, las = 1, cex = 0.35, pty = "s", xlab = "First PC (size)",
   ylab = "Second PC", main = "QDA", col= ifelse(shellfish[,"Rings (3 classes)"] == 1, "red",
                ifelse(shellfish[,"Rings (3 classes)"] == 2, "blue", "green")))
contour(x,y,matrix(qdap2,m,n), levels=c(1.5,2.5), add=TRUE, d=FALSE, lwd =1, col = "black")

plot(pca4predictors, pch = 16, las = 1, cex = 0.35, pty = "s", xlab = "First PC (size)",
   ylab = "Second PC", main = "CT", col= ifelse(shellfish[,"Rings (3 classes)"] == 1, "red",
                ifelse(shellfish[,"Rings (3 classes)"] == 2, "blue", "green")))
contour(x,y,matrix(ctp2,m,n), levels=c(1.5,2.5), add=TRUE, d=FALSE, lwd =1, col = "black")

plot(pca4predictors, pch = 16, las = 1, cex = 0.35, pty = "s", xlab = "First PC (size)",
   ylab = "Second PC", main = "RF", col= ifelse(shellfish[,"Rings (3 classes)"] == 1, "red",
                ifelse(shellfish[,"Rings (3 classes)"] == 2, "blue", "green")))
contour(x,y,matrix(rfp2,m,n), levels=c(1.5,2.5), add=TRUE, d=FALSE, lwd =0.5, col = "black")

plot(1:10, type = "n", axes = FALSE, xlab = "", ylab = "")
legend(2.5,8,legend = c("young", "adult", "old"), fill = c("red", "blue", "green"),cex = 2,bty = "n")
```

```
dev.off()
### 6.2 Regression Methods

### Regression analogies of the classification optimise functions. The mean squared error is
### used instead of the misclassification error. The default parameters are set to lower values
### to reduce computational costs.

### Random forests

optimise.rrf <- function(predictors, true.values, cv.factor, max.mtry = 3){
  max.mtry <- min(max.mtry, ncol(predictors))
  mse <- matrix(NA, length(levels(cv.factor)), max.mtry)
  for (j in 1:max.mtry){
    for (d in as.numeric(levels(droplevels(cv.factor)))){
      predicted <- predict(randomForest(predictors[cv.factor != d,],
                                        true.values[cv.factor != d],
                                        mtry = j),
                           predictors[cv.factor == d,])
      mse[d,j]  <- (mean(true.values[cv.factor == d] - predicted)^2)
      rm(predicted)
    }
  }
  apply(mse, 2, mean, na.rm = TRUE)
}

### Classification trees

optimise.rct <- function(predictors, true.values, cv.factor){

  cpinfo <- rpart(true.values ~ . , data = predictors,
              control = rpart.control(xval = unclass(droplevels(cv.factor))))$cptable

  e <- rep(NA, dim(cpinfo)[1]-1)
  for (j in 1:(dim(cpinfo)[1]-1)){
    e[j] <- cpinfo[j,4]-cpinfo[dim(cpinfo)[1],4]-cpinfo[dim(cpinfo)[1],5]
  }
 cp <- ifelse(all(e>0), cpinfo[dim(cpinfo)[1],1], cpinfo[which.max(ifelse(e>0,NA,e)), 1])
  size <- cpinfo[match(cp, cpinfo[,1]),2]
  list(cp,size)
}

### Neural networks

optimise.rnn <- function(predictors, true.values, cv.factor, max.size = 10){
  mse <- matrix(NA, length(levels(cv.factor)), max.size)
  for (j in 1:max.size){
    for (d in as.numeric(levels(droplevels(cv.factor)))){
    predicted <- predict(nnet(true.values[cv.factor != d] ~., data = predictors[cv.factor != d,],
                                        size = j, decay = 0.01, linout = TRUE),
                                 predictors[cv.factor == d,])
      mse[d,j]  <- mean((true.values[cv.factor == d] - predicted)^2)
      rm(predicted)
    }
  }
  apply(mse, 2, mean, na.rm = TRUE)
}
```

```
### 6.2 Regression analogy of the compare.methods function

compare.regression.methods <- function(predictors, true.values, cv.factor,
                                       max.mtry = 3, max.size = 10){

  ### Definition of development parameter vectors

  chosen.mtry <- rep(NA, length(levels(cv.factor)))
  chosen.cp   <- rep(NA, length(levels(cv.factor)))
  chosen.size <- rep(NA, length(levels(cv.factor)))
  chosen.ct.pars <- rep(list(NA), length(levels(cv.factor)))
  chosen.nnsize  <- rep(NA, length(levels(cv.factor)))

  ### Definition of output vectors

  mse.rf  <- rep(NA, length(levels(cv.factor)))
  mse.ct  <- rep(NA, length(levels(cv.factor)))
  mse.nn  <- rep(NA, length(levels(cv.factor)))

 ### The development phase. Optimising the 'tuning' parameters on the development sets using
  ### the optimise functions described above

  for (v in 1:length(levels(cv.factor))){

   choosing.mtry    <- optimise.rrf(predictors[cv.factor != v,], true.values[cv.factor != v],
                                    cv.factor[cv.factor != v])
    chosen.mtry[v]    <- match(min(choosing.mtry), choosing.mtry)

   chosen.ct.pars[[v]] <- optimise.rct(predictors[cv.factor != v,], true.values[cv.factor != v],
                                       cv.factor[cv.factor != v])
    chosen.cp[v]      <- chosen.ct.pars[[v]][[1]]
    chosen.size[v]    <- chosen.ct.pars[[v]][[2]]

   choosing.nnsize  <- optimise.rnn(predictors[cv.factor != v,], true.values[cv.factor != v],
                                    cv.factor[cv.factor != v])
    chosen.nnsize[v]       <- match(min(choosing.nnsize), choosing.nnsize)

  }

  ### Computation of predictions for the test sets

  for (v in 1:length(levels(cv.factor))){

    predictions.rf  <- predict(randomForest(predictors[cv.factor != v,],
                                            true.values[cv.factor != v],
                                            mtry = chosen.mtry[v]),
                               predictors[cv.factor == v,])
    predictions.ct <- predict(rpart(true.values[cv.factor != v] ~.,
                                    data = predictors[cv.factor != v,],
                                  control = rpart.control(cp = chosen.cp[v], xval = 1)),
                              predictors[cv.factor == v,])
    predictions.nn <- predict(nnet(true.values[cv.factor != v] ~.,
                                   data = predictors[cv.factor != v,],
                                  size = chosen.nnsize[v], decay = 0.01, linout = TRUE),
                              predictors[cv.factor == v,])
```

```
    ### Computation of mean square errors for the test sets

    mse.rf[v]  <- mean((true.values[cv.factor == v] - predictions.rf)^2)
    mse.ct[v]  <- mean((true.values[cv.factor == v] - predictions.ct)^2)
    mse.nn[v]  <- mean((true.values[cv.factor == v] - predictions.nn)^2)

    rm(predictions.rf, predictions.ct, predictions.nn)
  }

  ### Formatting the output

  mean.mses <- c(mean(mse.rf), mean(mse.ct), mean(mse.nn))
  names(mean.mses) <- c("RF", "RT", "NN")

  all.mses <- data.frame(mse.rf, mse.ct, mse.nn)
  colnames(all.mses) <- c("RF", "RT", "NN")

  list(mean.mses, all.mses, data.frame(Chosen.mtry = chosen.mtry,
                                       Chosen.Size = chosen.size, Chosen.cp = chosen.cp,
                                         Chosen.nnsize = chosen.nnsize))
}

### 6.2 Comparasion of the methods

analysis.output.regression <- compare.regression.methods(shellfish[,1:7], shellfish[, "Rings"],
                                                          indices)

### 6.2 Plot of the mean square errors and their standard deviations for the three methods

pdf(file = "rcomparison.pdf", width = 6, height = 6)

plot(sort(analysis.output.regression[[1]]), axes = FALSE, xlab = "Method",
     ylab = "Mean Squared Error",
     col = "darkred", pch = 16, xlim = c(0.5,3.5), ylim = c(3, 7), pty = "s",
     main = "Generalisation Errors")

box();axis(2, at = 3:7, las = 1)
axis(1,at = 1:3,labels = toupper(names(sort(analysis.output.regression[[1]]))))

error.bar(1:3, sort(analysis.output.regression[[1]]),
       apply(analysis.output.regression[[2]], 2,sd)[order(analysis.output.regression[[1]])],
          col = "cornflowerblue")
points(1:3, sort(analysis.output.regression[[1]]), col = "darkred",pch = 16)
dev.off()
```