Copenhagen Business School

Data Science: Data Driven Decision Making

Exam Number: S159569

Count of Standard Pages: 15

Submitted on: November 10th, 2022

Contents

Part 1

Question (i)

The "abalone" dataset contains 9 variables, 8 explanatory variables and 1 target variable y= "Age". Most of the variables are numeric, but the "Sex" variable is a factor variable with 3 levels. To avoid later complications, example with the K-NN classification, the "Sex" variable is converted into 3 dummies, where the dummy variables "Sex_F (Female), Sex_I (Infant) and Sex_M (Male)" can be either 0 or 1. Therefore, we will now have k=10 explanatory variables and 1 target variable. The same construction of the "abalone" data is applied to the "abaloneadditional". Furthermore, the data has been normalized, i.e.., we do not want the scale to impact our decision in K-NN.

Logistic Regression is used to model the posterior probabilities of the K classes using linear functions in x, while ensuring that the K classes sum to one and remain in [0,1]. The model for our data has the form:

$$\log \frac{\Pr(G=1)\,|X=x)}{\Pr(G=2)\,|X=x)} = \beta_{10} + \beta_1^T x$$

The data consists of K= 2 classes, "young" and "old", meaning we have a single linear function. We use the maximum likelihood to fit our model, using the conditional likelihood G given X. The log-likelihood for N=3000 observation is

$$l(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i;\theta),$$

where $p_k(x_i;\theta) = \Pr(G=k|\,X=x_i;\theta)$. Next, we can write the log-likelihood in the following two-class form, where $y_i = 1$ when $g_i = "old"$ and $y_i = 0$ when $g_i = "young"$. We let $p_1(x;\theta) = p(x;\theta)$, and $p_2(x;\theta) = 1 - p(x;\theta)$, then we have

$$l(\beta) = \sum_{i=1}^{N} \{y_i \log p(x_i;\theta) + (1 - y_i = \log(1 - p(x_i;\beta))\}$$

$$= \sum_{i=1}^{N} \{y_i \beta^T x_i - \log(1 + e^{\beta^T x_i})\},$$

where $\beta = \{\beta_{10}, \beta_1\}$ and assume that the vector of inputs x_i (explanatory variables) includes a constant term 1 to accommodate the intercept (*Hastie 2009, pp. 120-121*). When maximizing the log-likelihood, we set its derivatives to zero and the score equations become

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i(y_i - p(x_i;\beta)) = 0 \qquad 1)$$

which are p+1 equations nonlinear in β. Since the first component of $x_i\ is\ 1$, the above equation specifies that $\sum_{i=1}^{N} y_i = \sum_{i=1}^{N} p(x_i;\beta)$. We get that the expected number of class 1s' ("old") matches the observed number, which is also the case for class 2s' ("young").

To solve equation (1), one can use the Newton-Raphson algorithm, which requires the second derivative

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = -\sum_{i=1}^{N} x_i x_i^T\, p(x_i;\beta)(1 - p(x_i;\beta))$$

Using the Newton-Raphson algorithm, we would typically experience convergence since the log-likelihood is concave, but overshooting can occur. If the log-likelihood does decrease, step size halving can guarantee convergence.
The code for the logistic regression performed in R can be found in appendix 1. Here we have used "Age" as our target variable and the other 10 variables as explanatory variables. Notice, it is not needed to convert the "Sex" variable into dummies in order to perform a logistic regression. However, since it is preferred to have the same starting point for all the different models and classifications, the dummies and normalization are constructed in the beginning. Next, Generalized Linear Model (GLM) function is used to create the logistic regression with the family 'binomial', which uses the "logit" link. The "logit" is useful for the logistic regression since it models the probability of successes. Then a step function is performed on the regression. The step function chooses the model by Akaike information criterion (AIC) in a stepwise algorithm. AIC measures the relative amount of information lost by a given model. Therefore, the more information a model loses the lower the quality of the given model. Furthermore, AIC deals with the trade-off between a good fit of model and the simplicity of the model. Meaning, that one should be aware of the risk of overfitting and underfitting. AIC is given by

$$AIC = 2k - \ln(\hat{L})$$

Where k is the number of estimated parameters in the logistic regression and $\hat{L}$ is the maximized value of log-likelihood function.

The results from the 'step' function indicates, the variables "Sex_M" and Sex_I" are not statistically significant on a 5-pct. significance level, meaning they are excluded from the model. The rest of the variables seen in table 1 appendix 1 "Logistic Regression" are statistically significant on a 5-pct. significance level. The "Length" variable has an estimate of 2,68. It means that $e^{2,68} = 14,58$, which tells how much the odds of the outcome "age" will change for each 1 unit change in the predictor (length).

Question (ii)

Support vector machines (SVM) are linear discriminants, which classify instances based on a linear function of the features

$$f(x) = w_0 + w_1 x_1 + w_2 x_2$$

with the individual component features being x_i. SVM split the classes by first fitting the fattest line between the classes. Then the linear discriminant will be centred through the fattest line. There will be a margin around the linear discriminant, which is SVM's objective to maximize (*F. Provost and T. Fawcett (2013), Ch. 4).* Maximizing the margin will help the SVM to predict new features in the test data. Therefore, the boundary maximized by the margin gives the maximal leeway for classifying these new points that are beyond the training data. Furthermore, in our function that measures the fits of the point, we will penalize a training point for being in the wrong side of the given decision boundary. This means, when the data points are not perfectly separated, the best fit is somewhere between the fat margin and the lowest total error penalty. The hyperplane that separates the classes is $\omega^T x + b = 0$, where a correct classification can be written as $y_i(\omega^T x_i + b) > 0$. This is due to the following. If $y_i(\omega^T x_i + b) > 0$ then two possibilities can occur, $y_i = 1$, $(\omega^T x_i + b) > 0$ then $\hat{y} = 1$, or $y_i = -1$, $(\omega^T x_i + b) < 0$ then $\hat{y} = -1$. Meaning, in both cases the class is correctly classified. Where $\{-1, +1\}$, represents the two classes, "old" and "young" and ω being a linear function of the classifiers.

Maximizing the margin is equivalent to minimize $\sum_{j=1}^{k} \omega_j^2$. We can choose either a hard margin or a soft margin. The hard margin requires correct classification for all points and thus tends to overfit the training set, which gives a poor generalization. Instead, we will use the soft margin that do not require all individuals to be correctly classified and is defined as.

$$minimize_{\omega,b,\xi}\frac{1}{2}\sum_{j=1}^{k}\omega_j^2+\frac{C}{n}\sum_{i=1}^{n}\xi$$

s.t.

$$y_i(\omega^T x_i+b)\geq 1-\xi_i,\quad i=1,\ldots.,n$$

$$\omega\in R^k$$

$$b\in R$$

$$\xi_i\geq 0,\quad i=1,\ldots.,n$$

If we have that $\xi_i=0$, the individual 'i' is correctly classified. If $0<\xi_i\leq 1$, 'i' is inside the margin but at the correct side of the hyperplane, meaning that it is correctly classified. Else $\xi_i>1$, i is misclassified. C is the trade-off parameter between the margin and correct classification. This parameter will be tuned to represent the optimal trade-off and the larger the C, the more misclassification in the training set is penalized. However, we do not want to large of a C, because we still want a generalized model. We tune the parameter C in R giving it a range from $2\text{^}(-12{:}\,12)$ to be tuned for. When it comes to make prediction in the test data, we use that

$$\omega=\sum_{j=1}^{n}\alpha_j\,y_j\theta(x_j)$$

and the prediction $\hat{y}_{new}$ for x_{new} is equal to

$$\hat{y}_{new}=sign(\sum_{j=1}^{n}\alpha_j\,y_j\theta\left(x_j\right)^{\top}\phi(x_{new})+b)$$

It means the prediction depends on the scalar product $\theta\left(x_j\right)^{\top}\phi\theta\left(x_j\right)^{\top}\phi(x_{new})$ between the new individual x_{new} and the support vectors.

When estimating the SVM, the paper will consider only two kernels, the linear kernel and the Radial Basis Function (RBF) kernel. The kernels are used to improve SVM performance. We will have data that is transformed $\theta(x)$ with a kernelization of $K(x,z)=\theta(x)^{\top}\phi(z)$. The linear kernel implies no transformation of the data, $K(x,z)=x^{\top}z$. The RBF kernel tend to show good performance and is given by

$$K(x,z)=e^{-\frac{||x-z||^2}{\sigma^2}},$$

where σ is a parameter to be tuned. In our R code this parameter will be called gamma and will be given the same tuning range as the parameter, C.

The prediction with kernels will be $\hat{y}_{new}$ for x_{new} equal to

$$\hat{y}_{new}=sign\left(\sum_{j=1}^{n}\alpha_j\,y_jK\left(x_j,x_{new}\right)+b\right),$$

meaning that the prediction depends on the kernel $K\left(x_j,x_{new}\right)$ between the new individual x_{new} and the support vectors. The interpretation for the prediction $\hat{y}_{new}$ for x_{new} is for those support vectors x_j for which $K\left(x_j,x_{new}\right)$ is high, the prediction will be more influenced. This will occur when $\left|\left|x_j-x_{new}\right|\right|$ is small. For the RBF kernel,

we will have that the prediction $\hat{y}_{new}$ will be more influenced by those support vectors for which $\left|\left|x_j - x_{new}\right|\right|$ is small that can occur when x_j is close to x_{new} in the feature space.

Estimating the SVM with a linear kernel, we obtain C=32 as the best cost for the model with a 10-fold cross validation. Our best performance was 0.2013. The SVM with the RBF kernel has a best cost of 32 and a gamma of 0.0313. The best performance was 0.191 with a 10-fold cross validation. Cross validation resamples different portions of the data to test and train the model on different iterations.

Question (iii)

Classification trees recursively partitions the feature space using "if" statements

$$\chi = \bigcup_{l} R^l$$

then a class will be signed to each region R^l that is assigned to class y^l, and new objects receive the class assigned to their regions. The classification rule is if $x_{new} \in R^l$, then $\hat{y}_{new} = y^l$. The branches are defined by the "if" statement and partition the observations in the parent node. Branches usually has the form $x \leq threshold$ & $x > threshold$. The main elements of the classification trees consist of a splitting rule, stopping criteria and a class assignment. The splitting rule is about the parent node and to choose whether the parent node should split into two nodes, which is called the children nodes. The splitting rule is based on an explanatory variable and a threshold. In the parent node, we will have a proportion of observations in "old" class and the "young" class. We say a node is pure if it contains observations from only one class. The way we choose our explanatory variable, and the threshold is to maximize gain in purity. The gain purity is defined as

$$gain = 1 - \frac{m_1}{m} impurity_1 - \frac{m_2}{m} impurity_2$$

were m being the number of observations in the parent node, m_1 is the number of observations in child 1 and m_2 being the number of observations in child 2. Thus, we have $m = m_1 + m_2$.

We need a stopping criteria to prevent the tree from the risk of overfitting the training sample. Each time we branch, we improve the purity of the children nodes. However, at the same time the amount of observation in the children nodes will decrease. The stopping criteria is a parameter of the classification tree and assures that the number of observations in each node is above a minimum. To set this stopping parameter the paper will take use of pruning. Pruning the classifications tree will help avoid overfitting. The goal of pruning is to minimize

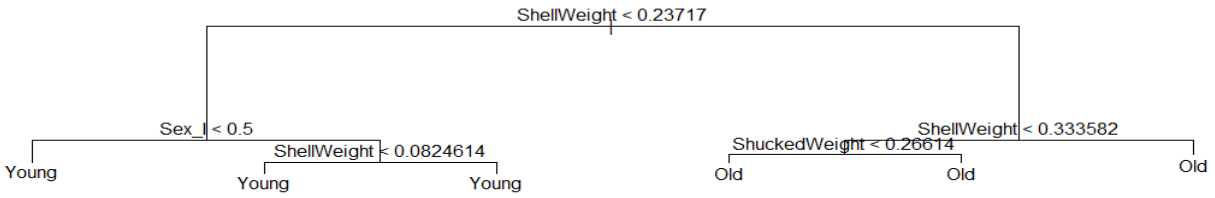
$$error(\mathrm{T}, \mathrm{S}) + \alpha \mathrm{size}(\mathrm{T})$$

where $error(\mathrm{T}, \mathrm{S})$: e.g., misclassification error and $\mathrm{size}(\mathrm{T})$ is the number of leaf nodes. α is the pruning parameter. The higher the α is, the smaller the pruned tree will be. However, pruning the parameter has a trade-off in the cost-complexity.

The class assignment is about signing a class, y^l to each leaf node l. It is used to classify future observations, and the usual class assignment is the majority calls in leaf node l.

Estimating the classification tree in R with our training data, we first observe whether we have an imbalance data set. We find that the target variable "Age" is evenly split by approximately 50 pct. Meaning, we do not have an imbalance data set. Last, we model our tree, prune it and obtain the tree seen in figure 1.

Figure 1: Classification tree.



Source: See code in appendix 1, Classification tree.

The figure shows that the classification tree only uses 3 explanatory variables. The tree contains 5 leaf nodes and has misclassified 712 observations out of 3000. We observe that the second node separates the two classes with a shell weight 0.23717 whereas 23.96 pct. is from the young class and 76.04 from the old class. This implies that a lot of old abalones have a shell weight of over 0.2317.

The minimum number of observations to branch a node is set to 10 as default. Since node 12 has the least number of observations of 289, the stop criteria will not be changed. As a stopping criterion above 289 will result in a too small classification tree. Furthermore, the code "cv.tree" runs a K-fold cross-validation experiment to find the deviance and misclassification as a function of the cost-complexity parameter 'k', the default is k=10. The trees that are collected in the cv.tree() function are found by cutting some branches from the original tree, constructed by the function tree(), and thus reducing the number of leaf nodes. We do this to construct a tree that will generalize better, example show better accuracy on the test data. However, since the original tree is similar to the pruned tree, we have not cut any branches. To obtain these results, we must first retrieve the best option for the pruning which is done using the code for the best size, and then prune the tree using this best value. The final tree is called 'myprunedtree' which we use for prediction.

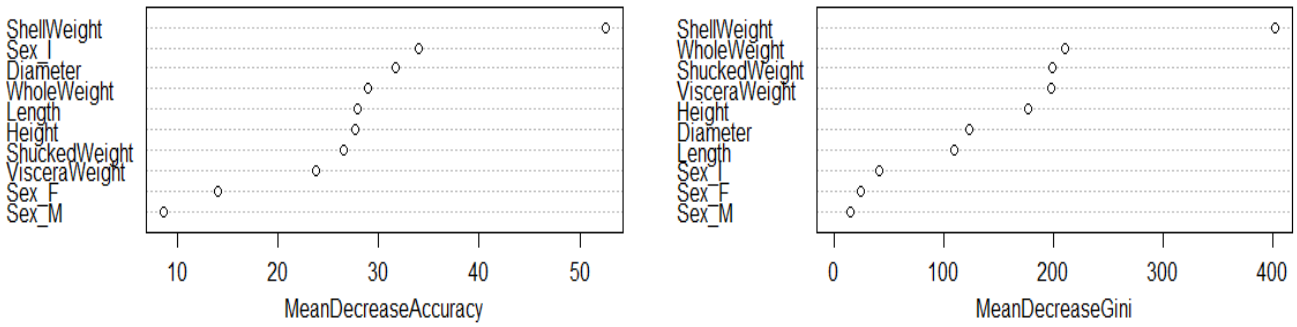
Question (iv)

Random forest builds a collection of trees, where each tree uses different training samples. The paper uses bootstrapping to construct each training sample, where each cut only considers a random sample of the explanatory variables. We can measure the explanatory variable importance of the classification task, which is $x_l(l = 1, \dots, k)$ and defined as the change in accuracy by reshuffling the values of x_l.

To estimate the random forest in R, we will use the function 'randomForest' with 'ntree' equal to500, meaning the random forest consists of 500 trees. The variable 'mtry'=4 is the squared root of the number of explanatory variables. Mtry controls how many of the explanatory variables we can use in each branch node. Once we have chosen a fixed mtry=4, the set of explanatory variables in each node is chosen randomly. We do this to avoid always having the same variable at the top of the tree. Instead, we want to give equal opportunity to all the variables. Figure 2 indicates the variable importance for the abalone data.

In figure 2 the variable ShellWeight seems to be most important. However, from the variable "length" and down the variables seems to have less of an importance. Furthermore, mtry=4 seems to be a good setting since the accuracy decrease when mtry>4.

Figure 2: Variable importance.



Source: See code in appendix 1, Random Forest.

Question (v)

K-Nearest Neighbours (K-NN) is based on similarity arguments, where similarities define the set of neighbours. For new individuals in our test data, the predicted class is defined by the K closest neighbours. The main elements for the K-NN are class assignment and training sample. Class assignment is about the following, given a new individual, calculate the $\delta_{new,i}$ for all 'i' in training sample. Second, order the dissimilarities in increasing order i.e.,

$$\delta_{new,(1)} \leq \delta_{new,(2)} \leq \cdots \delta_{new,(k)} \leq \delta_{new,(K+1)} \leq \cdots$$

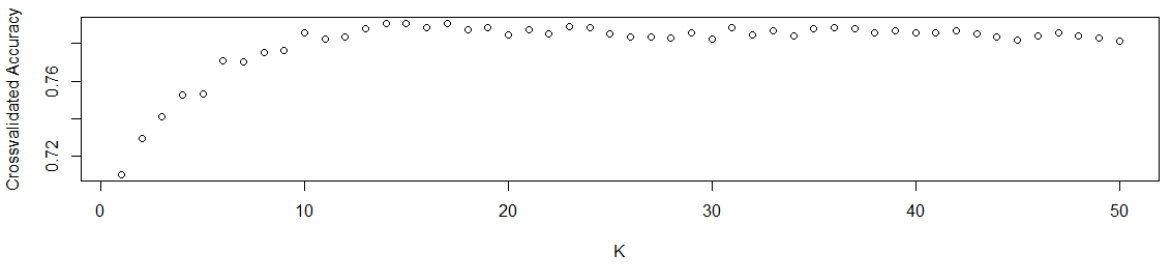
The nearest neighbours are called the K first individuals. The majority class among K is represented as y_{new}. To obtain a more optimal K parameter, we will tune the parameter in R. We use leave-one-out crossvalidation to tune K. Figure 2 shows the accuracy results from K-NN iterating over 50 values for K. The best K is 14.

In R we use the Euclidean distance to measure the dissimilarities, which is defined as:

$$d(x_i, x_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots (x_{ik} - x_{jk})^2}$$

Furthermore, when estimating K-NN, we will first need to split the data into 4 samples. Two training samples and 2 test samples.

Figure 2: K-NN visualizing the accuracy results.



Source: See R code in appendix 1, K-NN.

Question (vi)

It is not easy to determine the best approach for our given dataset, but in terms of interpretability, the logistic regression visualizes only a list of coefficients. However, this can be hard to interpret if we have many variables.

Instead, classification trees can give a better visualization, if you have more variables. However, too many variables can still create too much of a messy tree. So, the interpretability is somewhat fair. Furthermore, the trees are good at handling mixed data types, and the computational scalability is good, but the predictive power is not good compared to the logistic regression, (*F. Provost and T. Fawcett (2013), pp.103).*

Using "Table 10.1" *Hastie (2009),* we observe that SVM is poor at handling mixed data types. Example, categorical variables with numeric variables is hard to estimate in an SVM without any data manipulation. Furthermore, the interpretability is poor, and the computational scalability is also poor compared to the classifications trees. However, the ability to extract linear combinations of features is good and the predictive power is good as well.

Random Forest (RF) is good at handling mixed data. There is little pre-processing that needs to be done when estimating a random forest, example the data do not need to be scaled, which is opposite for the K-NN, where scaling is important. Furthermore, random forest can handle high dimensional data. The drawbacks are that the RF can be difficult to interpret, which is seen in very large datasets, where the trees can take up a lot of memory. Also, it can tend to overfit the data, which is why tuning the parameters is important.

K-NN is good at handling missing values, and its predictive power is good. On the other hand, the computational scalability is poor which is the same with handling different data types. This too was the reason to construct the "Sex" variable into dummies, so we would not be dealing with mixed data. Table 2 below shows accuracy for the different models.

Random Forest has the highest accuracy, and the logistic regression has the lowest. However, this does not imply that RF is a better model, example we must also take interpretability into account. RFs' accuracy is not substantially higher compared to LR with a difference of 1.61 percentage points. Furthermore, there is a difference between the two SVM models with 0,69 percentage point. However, since the accuracy for the RBF is not substantially greater compared to the SVM with a linear kernel, it is preferred to choose the linear kernel. This is due to the linear kernel being more intuitive and easier to interpret.

Table 2: Accuracy for the different models

	LR	SVM (linear)	SVM (RBF)	Tree	RF	K-NN
ACC:	77,40%	77,82%	78,51%	74,77%	79,01%	78,17%

Source: See appendix 1.

Question (vii)

See appendix 1 "Question(vii)" for data construction using the "blood" data with an implemented outlier. The implemented point is 0 for the variable blood, 100 for the variable "smoke" and 150 for the variable "alcohol". This is clearly an outlier, since no other observation for the blood 0 has nearly as high smoking- nor alcohol consumption.

We measure the log-likelihood to test the fit of the model and to see if the outlier has an impact. When determine the best fit, we will look at the smallest absolute value. The log-likelihood with the outlier is 67.45 and 39 without the outlier. The fit becomes better when removing the outlier. Next, we will look at the prediction. To do this, we will first need to convert the blood variable into factor variables, and then split the dataset into a training and test dataset. The results are, prediction rate with the outlier is 80.65 pct., and 83.33 pct. without the outlier. We find that removing the outlier results in better accuracy.

As we observe, logistic regression is sensitive to outliers and to make Supervised Learning methodologies more robust, we can implement following approaches. If the outliers seem to be a typo, one could simply remove the point. This technique was done in the above example as the point clearly was not meant to be in the dataset. If the

outlier is a valid observation, it is not fair to simply remove the observation. Instead, one could implement the capping scheme *(B. Baesens, 2014 (pp.23))*. It is about implementing an upper- and lower limit on the explanatory variables. Any values above or below will be brought back to these limits. The limits can be constructed using the interquartile range (IQR) given as

$$\frac{upperlimit}{lowerlimit} = M \pm 3s$$

where M is the median and $s = IQR(2 \times 0.6745)^3$.

Question (viii)

The reasons for which a non-binary tree can be more suitable, is if you have a dataset where the root note cannot be divided into two nodes. Example, looking at the hierarchical structure of an organization with a president and several vice presidents, where each vice president has several subordinates and so on. To model the company with a data structure, it will be more optimal to use a non-binary tree instead of a binary tree. Because it is likely that the organization will consist of more than two vice presidents, the root node cannot be easily split into two nodes. Instead, it is more useful to use a non-binary tree, whose nodes have an arbitrary number of children. However, when implementing a tree with an arbitrary number of children, it becomes more difficult to implement compared to binary trees *(Shaffer 2010, Ch. 6)*.

The first methodology one can use to build a non-binary tree is the "list of children". It stores with each node a linked list of its children with an order from left to right *(Shaffer 2010, pp. 217)*. The tree nodes will be stored in an array where each node contains a value, a pointer to its parent and a pointer to a linked list of the node's children. This is all stored from left to right. The left most child is easy to find, since it is the first element in the list. However, to find the right most child is more challenging. Example, M is a node and P is its parent. To find M's right most child, one must move down the child list of P until we have found the linked list storing the pointer to M. Then, going one step further will lead us to the linked list element that stores a pointer to M's right sibling.

Another approach that also uses an array to store the collection of nodes is called "Left-Child/Right-Sibling". Here each node stores its value and pointers to its parent, left most child, and right sibling. Each of the abstract data type (ADT) operations can be used by reading a value directly from the node. Then, if two trees are stored in the same node array, adding one as the subtree requires setting 3 pointers. The reason for this build is because, it is more space efficient than the "list of children", and in the node array each node needs a fixed amount of space *(Shaffer 2010, pp. 218)*.

Question (ix)

In variable importance there are two goals, i) find the contribution of each explanatory variable to the target variable, and ii) measure dependency of the explanatory variable and the output one. If correlation is present between the explanatory variables, the two objectives might be contradictory. Example, if two or more variables are strongly correlated, one of the variables may be discarded without degrading the performance of the model, but still being related to the output variable *(Marie Chavent et. al. (2021)*. This means that correlated explanatory variables can impact the RF's ability to identify strong predictors.

To resolve this problem, one can cluster the input variables according to their correlation. Second, a synthetic variable summarises each cluster of the input variables, and the first principal component is constructed within the cluster. Third, we train a RF algorithm on the synthetic variables, and for each of them the associated Mean Decreased Accuracy (MDA) is constructed. Fourth, we determine the importance of each of the original variables using the MDA of the synthetic variables and the correlation between synthetic and original variables. The clustering of the variables will reduce the correlation in the data. However, there is a trade-off between interpretability and performance, which can be adjusted by the number of clusters used to fit the RF. Due to

Synthetic MDA, the importance between the explanatory variables and the target variable should be better since it is not being diluted by correlation *(Marie Chavent et. al. (2021)*.

Part 2

Question (i)

The paper has standardized the data to prevent any predominant features of the variables. Further description of the code and the data is provided in appendix 2, data.

Clustering is about grouping individuals that are similar given a dissimilarity between individuals. Each group is defined as a cluster which is intra homogeneity, meaning that all individuals within a given cluster should be similar. Intra heterogeneity is a collection of clusters, clustering, in which clusters should be dissimilar to other clusters. To measure the distance between clusters, the paper uses the linkage 'complete' given as

$$L(S_1, S_2) = \max_{x_1 \in S_1; x_2 \in S_2} d(x_1, x_2)$$

Hierarchical clusters require us to specify a measure of dissimilarity between groups of observations. This form of cluster will produce a hierarchical representation, where the clusters at each level are created by merging clusters at the next lower level.

To create the hierarchical cluster, we use the function 'hclust' combined with the 'dist()' function. The 'dist()' will as default return the Euclidean distance to measure the dissimilarity. However, the paper also examines the Manhattan and Maximum distance. The plot for the Euclidean and Maximum distance can be found in appendix 2. The Manhattan distance is defined as

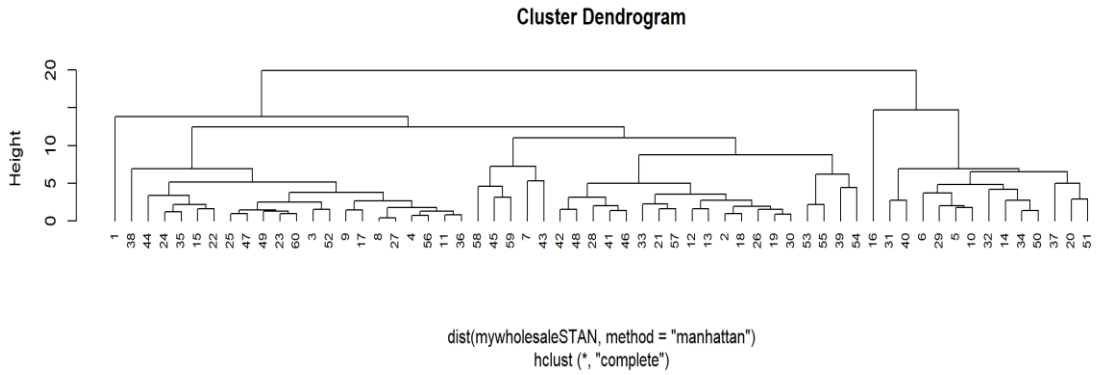
$$d(x_i, x_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ik} - x_{jk}|$$

and the maximum distance as

$$d(x_i, x_j) = \max \{|x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ik} - x_{jk}|\}$$

This paper proceeds with the Manhattan distance due to its preferred visualization. The 'dist()' function creates a 32x32 matrix i.e., nxn where n is the number of objects. To work with the matrix, we use the function 'as.matrix'. Figure 3 is a plot of the dendrogram using the Manhattan distance.

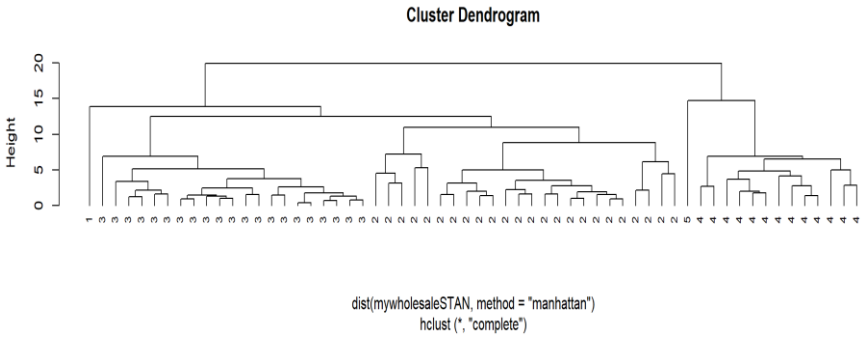
Figure 3: Dendrogram with Manhattan distance.



Source: See code in appendix 2, Question (i).

The plot shows that customer 1 and 16 are behaving differently. Furthermore, at the top of the cluster, we have all individual together, and at the bottom each individual is clustered by itself, *(Hastie, T. 2009 (p. 521))*.

Figure 4: Dendrogram with Manhattan distance with 5 clusters.



Source: See code in appendix 2, Question(i).

Figure 4 shows how the different costumers will be grouped using 5 clusters. We find that costumer 1 and 16 are being grouped differently due to their dissimilarities. The reason for customer 1 and 16 being treated differently is due to their consumption history, which is different.

Question (ii)

In partitioning-based clustering the goal is to partition Ω into clusters

$$\Omega = \bigcup_{l=1}^{K} C_l$$

$$C_l \cap C_{l'} = \theta$$

where Ω is a population of individuals and for each individual $i \in \Omega$ we have $x_i = (x_{i1}, x_{i2}, \dots, x_{ik}) \in \chi$: vector of explanatory variables. This is done to achieve intra homogeneity and inter heterogeneity. To find the partitions. the paper will make use of iterative partitioning algorithms. Using the distance d, we make use of prototypes, meaning that each cluster will have a prototype i.e., a representative of the cluster, where p_l is the prototype to cluster l. Prototypes will be used to measure intra-homogeneity. We measure the distance from the points in the cluster to its prototype to have an estimation of the intra-homogeneity of the cluster

$$\sum_{l=1} \sum_{i \in C_l} d(x_i, p_l)$$

which aims to improve the overall intra-homogeneity. The iterative partitioning algorithm chooses K initial prototypes, $p_1, p_2, \dots, p_k$. Then it clusters around the prototypes i.e., each observation is assigned to the closest prototype

$$i \in C_l \; if \; and \; only \; if \; d(x_i, p_l) = \min_{l'=1,2\dots,k} d(x_i, p_l)$$

and if the clusters have not changed then stop. Otherwise, re-calculate the prototypes and perform the step described above again.

There are variants of partitioning-based clustering method, but the paper uses K-means since the data consists of only continuous variables. K-means uses a top-down procedure and have initial K prototypes which are chosen randomly with the use of multistart. In R we do this using the 'nstart' function, where we run it 5000 times. Multistart will ensure that we do not get trapped in a local optimum.

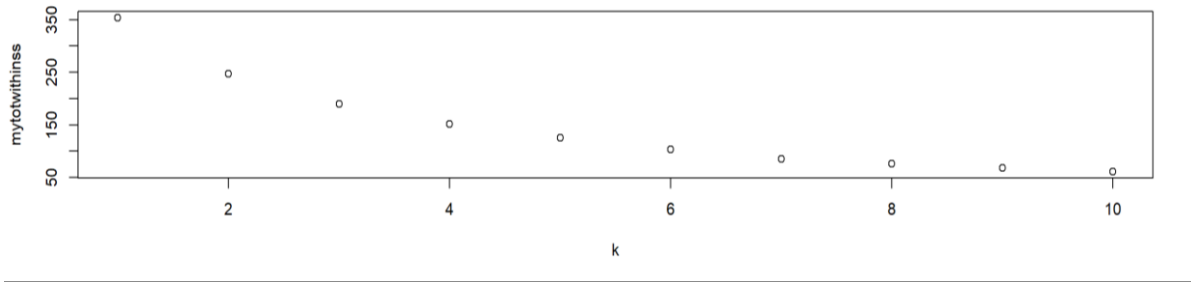
The paper will use the Squared Euclidean distance to define closeness and is given by

$$\left(d(x_i, m_l)\right)^2 = (x_{i1} - m_{l1})^2 + (x_{i2} - m_{l2})^2 + \cdots + (x_{ik} - m_{lk})^2.$$

When estimating k-means in R, we will first need to experiment with different values of K, which is shown in figure 5.

Figure 5 illustrates that the total within sum of squares decreases as the number of clusters K increases. However, more clusters are not necessarily good, since it might not be manageable for the given user to use a high K. Instead, we use the elbow rule to choose a good K. In figure 5, the rule leads to a K=3. The goodness of the classifications k-means is (between_SS / total_SS = 46.4 %). In other words, 46.4 pct., is the measure of the total variance in the dataset, which is explained by the clustering.

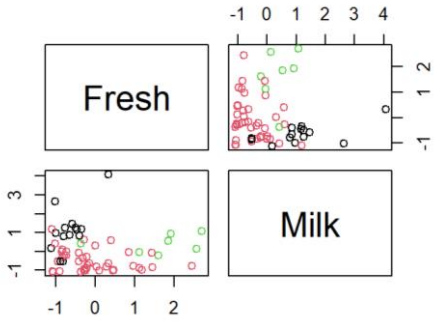
Figure 5: Values of K for K-means.



Source: See code in appendix 2, Question (ii).

Next, we will build the clustering with K=3, which is shown in figure 6 below. The figure illustrates three different clusters using the variables 'Fresh' and 'Milk'. A more detailed plot of all the variables is shown in appendix 2, Question (ii). Furthermore, the figure explains how the costumers can be split into three different groups according to their purchase history. Example, green group have a different purchase history compared to the black and red group, when looking at 'Fresh' and 'Milk'.

Figure 6: Fresh & Milk with K=3.



Source: See code in appendix 2, Question (ii).

Question (iii)

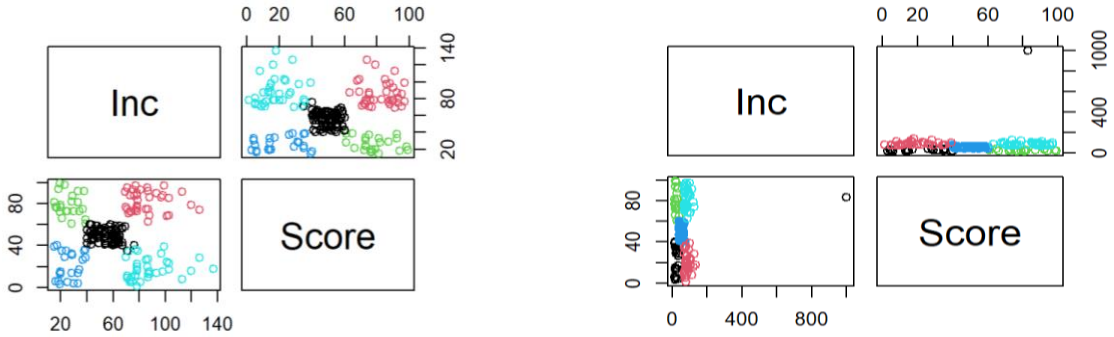
We use a customer dataset containing 5 variables. The dataset contains 200 client's information about Gender, Age, Annual income (in thousands of dollars) and a spending score. Client number 200 have an annual income of 1000 whereas the second and third highest has an annual income of 137 and 126. Client number 200 is clearly an outlier. We can identify outliers in a clustering approach by looking at instances that do not belong to clusters or if their clusters are significantly smaller than the other clusters. Figure 7 illustrates the negative impact of an outlier. The right plot in figure 7 is not fit for interpretation due to the outlier. The outlier affects the scaling of the plot, meaning that it becomes difficult to see the clusters. Furthermore, since k-means is taking the mean, one would get many outlier-sensitive calculations due to the mean being disrupted by outliers.

To make Clustering methodologies more robust against outliers, one can use the 'Trimming' technique which removes a proportion $\alpha/2$ of the largest observations and a proportion $\alpha/2$ of the smallest variables before constructing the univariate sample mean. However, it is not always easy to identify the largest and smallest observations in the multivariate setting $(p \geq 2)$, where there does not exist a natural geometrical order to be applied for identifying the largest and smallest observations. To overcome this challenge, one can let the data itself define which observations should be trimmed. Therefore, one can use Trimmed k-means, which is defined through the search of k centres $\{m_1, \ldots, m_k\} \subset R^P$ solving the double minimization problem

$$\arg\min_{Y} \min_{m_1,\ldots,m_k} \sum_{x_i \in Y} \min_{j=1,\ldots,k} \left|\left|x_i - m_j\right|\right|^2,$$

where Y will be in the range of the class of subsets of size $[n(1-\alpha)]$ within the sample $\{x_1, \ldots, x_n\}$. Using the trimmed k-means, one is not forced to classify all the observations because it allows for proportion α of observations, hopefully all the outliers, to be left unassigned. This means that the trimmed k-means can be viewed as a robust version of k-means, *(Luis Angel García-Escudero, et.al. 2010).*

Figure 7: Income and Spending score, K=5.



Source: See code in appendix 2, Question (iii).

Note: Left plot no outlier, right plot with outlier.

Question (iv)

In practical, costumers take relevant attributes into consideration when selecting a product or service. These attribute assessments of a product are often presented by linguistic data sequence. Clustering is a handy tool to partition linguistic data sequences of costumers' assessments on a product. One methodology to handle linguistic data with clustering, is to present the linguistic data sequences by fuzzy data sequences. Then a fuzzy compatible relation is built to present the binary relation among the two data sequences. The fuzzy binary relation satisfies compatible relations by being rooted in the similarity of two data sequences. Next, derive a fuzzy equivalence relation by max-min transitive closure from the fuzzy compatible relation. The max-min transitive closure solve the overlap problem, which can occur since the elements of different clusters might overlap. They might overlap due to the binary relations merely satisfies compatible relation for partitioning. Due to the fuzzy equivalence relation, the linguistic data sequences are classified into clusters. Costumers' selection preferences of products and services will be presented in clusters, *(Yu-Jie Wang, 2010).*

Question (v)

The goal of Semi-Supervised Learning (SSL) is to classify the unlabelled data using the labelled dataset, meaning that we would like to predict a target variable with given input data. A real-world application is speech analysis. SSL can help to improve traditional speech analytic models, since labelling audio files with human resources is very resourceful *(Jesus Rodriguez, 2017).*

SSL works by building a model that contains few labelled patterns as training data and treats the rest of the data as test data. Then SSL is divided into two types i) Semi-Supervised Classification (SSC) and ii) Semi-Supervised Clustering. SSC will allow one to reduce the usage of training data. In Semi-supervised Clustering, we use both labelled and unlabelled data with side information that functions as pair wise constraints, which helps to create clusters. The generic Semi-Supervised Learning algorithm is defined as, input: N training samples, all labelled input. Using dataset: $\{X \to Y\}, \{< x_1, y_1 >, < x_2, y_2 >, \dots, < x_n, y_p \}. cv \leftarrow 10$, where cv is cross validation. Output: M - training models based on probabilistic approach. $i \leftarrow 0$

do

for each i in cv **do**

//iterates cv-times

dataset _samples ← dataset / k

$training_i \leftarrow dataset - datasetsamples[i]$

$M_i \leftarrow Classifier\ (training_1, k)$

$M \leftarrow M_j$

while till assign labels to all training examples return M, *(International Journal of Engineering &Technology, 2018)*. However, SSL still has it challenges with datasets containing many dimensions or attributes. This is due to the SSL algorithms where the labelling task will become very complex.

Part 3

The wine dataset has been normalized, and the class variable has been converted into a factor variable.

Question (i)

In principal component analysis (PCA), the goal is to look for the best linear representation of X in dimension, *l*, *l*<<*k* and is computed in R with the 'pcromp' function. In PCA there is a population Ω and for each individual $i \in \Omega$, we have $x_i = (x_{i1}, x_{i2}, \dots, x_{ik}) \in R^k$. We can write this as an $n \times k$ matrix X. Next, we will define the V-space as following. Consider $v_1, v_2, \dots, v_l \in R^l$, orthogonal vector, where V is the matrix of these vectors. The linear space generated by V is the collection of $z(\lambda_1, \lambda_2, \dots, \lambda_l) = \lambda_1 v_1 + \lambda_2 v_2, \dots, \lambda_l v_l$ for all $(\lambda_1, \lambda_2, \dots, \lambda_l)$.

PCA assigns the closest z in the V-space to each x_i, $z_i(V)$, where $z_i(V)$ is the orthogonal projection of x_i onto the V-space. To determine the best V-space, i.e.,

$$minimize_v \sum_{i=1}^{n} \left(d\left(x_i, z_i(V)\right)\right)^2$$

meaning that we are searching for that V-space, which minimizes the total of all Euclidean distances between a point and its projection. Second, to find the first and the second principal component we use

$$minimize_{v_1 \in R^l} \sum_{i=1}^{n} \left(d\left(x_i, z_i(v_1)\right)\right)^2$$

$$minimize_{v_2 \in R^l, v_2\ is\ orthogonal\ to\ v_1} \sum_{i=1}^{n} \left(d\left(x_i, z_i(v_1, v_2)\right)\right)^2.$$

The best principal component (PC) is the one that explains the largest proportion of the total variance, which would be the first PC. The second PC will be the one explaining the second largest proportion of the total variance. Table 2 below shows the different PC's standard deviation, proportion of variance and cumulative proportion. The table indicates that PC1 explains 47 pct. of the variance while PC2 explains 12 pct., and together they explain 59 pct. of the total variance. Furthermore, notice that PC11 explains 0 pct. of the total variance, meaning that it has little to no importance.

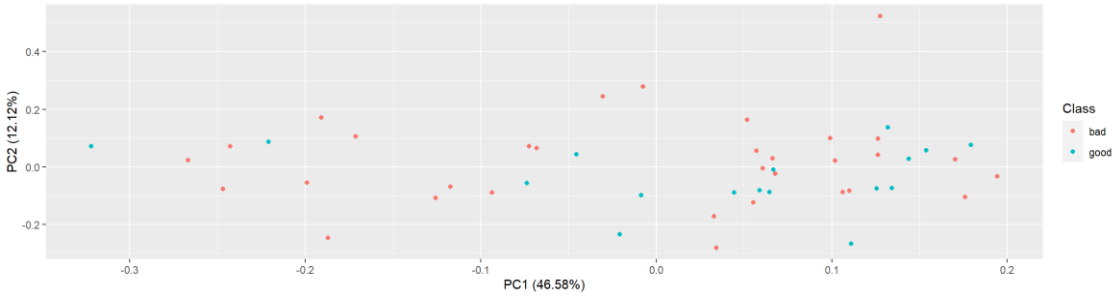
Table 2: Importance of components.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
SD	0.51	0.26	0.23	0.21	0.19	0.18	0.17	0.14	0.10	0.09	0.02
Var	0.47	0.12	0.09	0.08	0.06	0.06	0.05	0.04	0.02	0.02	0.00
CP	0.47	0.59	0.68	0.76	0.82	0.88	0.93	0.97	0.98	1.00	1.00

Source: See code in appendix part 3, Question (i).

Figure 8 below shows a plot with PC1 on the first axis and PC2 on the second, which is computed with the 'autoplot' function. Here we see how the two different classes "god" and "bad" wine behave. It indicates that there is not a clear separation between the two classes, however, one could argue that the good wine is most to the right while the bad is most to the left. But it could also mean that the explanatory variables do not clearly separate the good wine from the bad wine.

Figure 8: PC1 and PC2 plot.



Source: See code in appendix 3, Question (i).

Question (ii)

In Multidimensional Scaling (MDS) the goal is to find representatives for each individual $i \in \Omega$ and is computed in R with the 'cmdscale' function. In the code we set k=2, which is the dimension of the space that the data is represented in and last, we would like to return the eigenvalues. For each pair of individuals $i, i \in \Omega$, we have a dissimilarity $\delta_{ii'}$. The representative of i is characterized by $z_i \in R^l$, with $z_i = (z_{i1}, z_{i2}, \dots, z_{il})$. For all i, i', one would make sure that $d(z_i, z_{i'}) \approx \delta_{ii'}$.

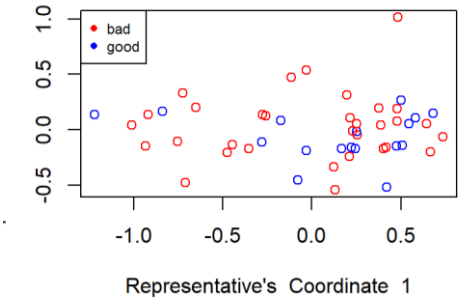
To construct the MDS, the paper will make use of the Euclidean distance, figure 8, since it makes the clusters clearer. To see how the Manhattan and Maximum distance have affected the MDS plot, see appendix 3, Question (ii). The MDS will minimize the Euclidean distance defined as,

$$minimize_{z_i \in R^l} \sum_{i \neq i'}^{n} \left(d(z_i, z_{i'}) - d_{ii'}\right)^2$$

where $d(\cdot,\cdot)$ is the Euclidean distance. The paper scales the dataset to $l = 2$ dimension just as in the PCA.

Figure 9 is nearly identical to the figure 7, and is constructed in R using the 'plot' function. This is due to the same scaling and using the same distance. Therefore, the interpretation about the plot will be the same. However, the scales on the x and y-axis differ.

Figure 9: MDS with Euclidean distance.



Source: See code in appendix 3, Question (iii).

Question (iii)

The similarities between PCA and MDS is the fact that they both are two alternative grouping techniques which can be classified as dimensioning techniques. These techniques produce two- or three-dimensional plots in which the entries are spread according to their relatedness. PCA and MDS do not provide clusters.

PCA visualizes the relationship among entries and unlike MDS, PCA will use the dataset itself instead of the similarity matrix to measure how similar the points are. The principal components are a maximization of a variation among entries along the first two or three dimensions. MDS will optimize a three-dimensional representation of the similarity matrix. In question (ii) we use the Euclidean distance, which measures the similarity between the points. The main difference between the PCA and MDS, will be the fact that the MDS needs a similarity matrix before it can be calculated *(Applied Maths NV, 2018).*

Question (iv)

Data that are collected and aggregated by geographical areas can be visualized with choropleth maps. It uses a colour encoding of the geographic area. Even though it is commonly used it needs care. A typical error is to encode raw data values, example population rather than first normalizing the data to produce a density map. Another challenge is one's interpretation of the shaded value that can be affected by the underlying area of the geographic region *(J. Heer, M. Bostock, and V. Ogievetsky, 2010).*

Instead of using a choropleth map, one can instead use the graduated symbol maps. This visualization tool avoids confounding geographic area with data values and allows for more dimensions to be visualized, example shape and colour. However, the challenges can be to calculate the actual value if it is not present, and it can be time-consuming to construct. *(J. Heer, M. Bostock, and V. Ogievetsky, 2010).*

A third visualization tool is Force-directed layouts. This is one way to visualize relationship given a network, example social networks, who is friend with whom? It models the graph as a physical system. It is constructed by nodes that are charged particles which repel each other. Links are dampened springs that pull similar nodes together *(J. Heer, M. Bostock, and V. Ogievetsky, 2010).* There will be a physical simulation of these forces that determines the node position. A positive point is the fact that interactivity will allow the user to direct the layout and sort the nodes to links. Force-directed layout will provide an easy understanding of the structure of a general undirected graph. The node colours in the visualization depict cluster memberships computed by a community-detection algorithm. The challenges are high running cost, meaning that it needs to do lots of iterations, because all pairs of nodes need to be visited, and their mutual repulsive forces needs to be calculated.

Appendices

Appendix 1

First, we collect the data we will use:

```
myabalone <- read.table(file.choose(),header=TRUE, stringsAsFactors=TRUE)
myadditionalabalone <- read.table(file.choose(),header=TRUE, stringsAsFactors=TRUE)
```

Next, we inspect the nature of the variables:

```
str(myabalone)
str(myadditionalablone)
```

We observe that the variables “Sex” and “Age” is factor variables, and the rest is numeric. To prevent complications in the future we transform the variable “Sex” into dummies. However, we want our target variable “Age” to stay as a factor variable. Therefore, we use the following code to convert these variables:

```
install.packages('fastDummies')
library(fastDummies)
myX <- myabalone [,-9]
myXD<-dummy_cols(myX, select_columns=c("Sex"),remove_selected_columns = TRUE)
Age <- as.factor(myabalone $Age)
myabaloneD<-  cbind(myXD, Age)
myXadditional <- myadditionalabalone [,-9]
myXadditionalD<-dummy_cols(myXadditional,     select_columns=c("Sex"),remove_selected_columns     =
TRUE)

Age <- as.factor(myadditionalabalone $Age)
myadditionalabaloneD<-  cbind(myXadditionalD, Age)
```

Again, we inspect the new nature of the variables:

```
dim(myabaloneD)
```

We see that we now have 11 explanatory variables because we converted the “Sex” variables into dummies.

```
str(myabaloneD)
```

Now we normalize the data using the following code:

```
myabaloneNORM<- myabaloneD
myadditionalabaloneNORM<- myadditionalabaloneD
for(i in 1:length(colnames(myabaloneD))-1) {
      if(class(myabaloneD[,i]) == "numeric" || class(myabaloneD[,i]) == "integer") {
         minimum<-min(myabaloneD[,i])
         maximum<-max(myabaloneD[,i])
         myabaloneNORM[,i]<-as.vector(scale(myabaloneD[,i],center=minimum,scale=maximum-
minimum))
         myadditionalabaloneNORM[,i]<-
as.vector(scale(myadditionalabaloneD[,i],center=minimum,scale=maximum-minimum))
}
      }
summary(myabaloneNORM)
summary(myadditionalabaloneNORM)
```

Logistic Regression

Table 1: Logistic Regression (Coefficients).

	Estimate	Std. Error	z-value
(Intercept)	2.6789	0.4105	6.527
Length	3.6371	1.8838	1.931
Diameter	-3.4119	1.8738	-1.821
Height	-3.5111	2.0965	-1.675
WholeWeight	-23.3015	3.9009	-5.973
ShuckedWeight	24.872	2.3059	10.786
VisceraWeight	3.0439	1.6062	1.895
ShellWeight	-11.4937	2.1072	-5.455
Sex_I	0.7654	0.122	6.274

Source: see code in appendix 1 "logistic regression".

Next, we will run a logistic regression using the following code:

```
LRmodel <- glm(Age~ ., family='binomial', myabaloneNORM)
```

Using following code to eliminate irrelevant variables:

```
LRmodelR <- step(LRmodel)
```

Now we predict on the test sample, and we will use the type "response" to use GLM for regression: Next, we will find the probabilities and find the predicted class. In our data we found that the class "Age" is classified as "young" and "old".

```
probabilitiesLR <- predict(LRmodelR, myadditionalabaloneNORM [,-11],type= "response")
predictionLR <- ifelse(probabilitiesLR > 0.5, "young", "old")
classificationtable <- table(pred= predictionLR, myadditionalabaloneNORM [,11])
acctestLR <- sum(diag(classificationtable))/sum(classificationtable)
acctestLR
```

ACC: 0.7740

We see that our accuracy is 77,40 pct.

Support Vector Machine

First, we install the packages to model the SVM

```
install.packages('e1071',dependencies=TRUE)
library('e1071')
```

We will first tune the SVM model to obtain the best cost for our model, then train the SVM again with the best values parameters. Node, we first we perform an SVM with a linear kernel, and afterwards using an RBF kernel to see which model performs best.

We set the seed to 1 so we can replicate our results in the future.

```
set.seed(1)
```

Tune the parameter C with a range of -12 to 12 using first a linear kernel.

```
tunedmodellinear <- tune.svm(Age~.,data = myabaloneNORM, cost=2^(-12:12), kernel= "linear")
summary(tunedmodellinear)
bestcost <- tunedmodellinear$best.parameters[[1]]
```

We assemble our final model:

```
finalmodellinear <- svm(Age~.,data= myabaloneNORM,cost= bestcost, kernel="linear")
```

And at last, we will predict our trained model against our test data and visualize the data in a table to get the accuracy.

```
predictionlinear <- predict(finalmodellinear, myadditionalabaloneNORM  [,-11])
classificationtable <- table(pred= predictionlinear, myadditionalabaloneNORM  [,11])
acctestfinalmodellinear <- sum(diag(classificationtable))/sum(classificationtable)
acctestfinalmodellinear
```

ACC: 0,7783
We obtain an accuracy of 77,82 pct.

Now we will perform an SVM with the RBF kernel. We start tuning our model, and this time we will also need to find the optimal gamma.

```
set.seed(1)
tunedmodelRBF <- tune.svm(Age~.,data = myabaloneNORM, gamma=2^(-12:12), cost=2^(-12:12))
summary(tunedmodelRBF)
```

```
bestgamma <- tunedmodelRBF$best.parameters[[1]]
bestcost <- tunedmodelRBF$best.parameters[[2]]
finalmodelRBF <- svm(Age~.,data= myabaloneNORM,gamma=bestgamma,cost=bestcost)
predictionRBF <- predict(finalmodelRBF, myadditionalabaloneNORM   [,-11])
classificationtable<-table(pred= predictionRBF, myadditionalabaloneNORM   [,11])
acctestfinalmodelRBF <- sum(diag(classificationtable))/sum(classificationtable)
```

ACC: 0.7850467

We obtain an accuracy of 78,51 pct.

Classifications trees

First, we install the required packages to perform a classification tree.

```
install.packages('tree',dependencies=TRUE)
library('tree')
```

We will first split the data to see how many abalones are young and old:

```
Agetable <- table(myabaloneNORM$Age)
old <- Agetable [1]/nrow(myabaloneNORM)
young <- Agetable [2]/nrow(myabaloneNORM)
```

We find that the split between young and old is roughly 50 pct.

We build a tree using our target variable “Age” and using our testing data.

```
set.seed(1)

mytree <- tree(Age~., myabaloneNORM)
plot(mytree)
text(mytree)
summary(mytree)

set.seed(1)
```

We prune the tree using the following code.

```
mycrossval <- cv.tree(mytree,FUN=prune.tree)
mybestsize <- mycrossval$size[which(mycrossval$dev==min(mycrossval$dev))]
myprunedtree <- prune.tree(mytree,best =mybestsize)
plot(myprunedtree)
text(myprunedtree)
summary(myprunedtree)
```

Next, we will make a prediction.

```
myprediction <- predict(myprunedtree, myadditionalabaloneNORM   , type='class')
classificationtable <- table(myprediction, myadditionalabaloneNORM    [,11])
acctesttree <- sum(diag(classificationtable))/sum(classificationtable)
```

ACC: 0.7477

Random Forest

Install the packages.

```
install.packages('randomForest',dependencies=TRUE)
library('randomForest')
set.seed(1)
```

We build the random forest with ntree=500 and mtry =4.

```
myrf <- randomForest(Age~., myabaloneNORM,ntree=500,mtry=4,importance=TRUE)
```

We get the importance of the variables.

```
importance(myrf)
varImpPlot(myrf)
```

Make predicitons.

```
myprediction <- predict(myrf, myadditionalabaloneNORM    [,-11], type='class')
classificationtable <- table(myprediction, myadditionalabaloneNORM    [,11])
acctestrandomforest <- sum(diag(classificationtable))/sum(classificationtable)
acctestrandomforest
```

ACC: 0.79099

K-NN

```
install.packages('FNN',dependencies=TRUE)
library('FNN')
```

We split the dataset into 4 different samples.

```
myxtrainNORM <- myabaloneNORM[,-11]
myytrainNORM <- myabaloneNORM[,11]
myxtestingNORM <- myadditionalabaloneNORM    [,-11]
myytestingNORM <- myadditionalabaloneNORM    [,11]
```

We find the best k for the best accuracy with k being 1 to 50.

```
bestk=0
bestaccuracy=0
accuracy <- NULL
for(auxk in  1:50){
   mycv <- knn.cv(train= myxtrainNORM, cl= myytrainNORM, k=auxk)
   mytable <- table (mycv, myytrainNORM)
   accuracy[auxk] <- sum(diag(mytable))/sum(mytable)
   if(bestaccuracy< accuracy[auxk]) bestk=auxk
   if(bestaccuracy< accuracy[auxk]) bestaccuracy = accuracy[auxk]}
plot(accuracy, xlab="K", ylab="Crossvalidated Accuracy")
```

We build the final model with the best parameters and make predicitons.

```
mybestknn <- knn(train= myxtrainNORM, test= myxtestingNORM, cl= myytrainNORM, k=bestk)
mytestingaccuracytablebestknn <- table (mybestknn,myytestingNORM)
mytestingaccuracybestknn<-
sum(diag(mytestingaccuracytablebestknn))/sum(mytestingaccuracytablebestknn)
```

ACC: 0.7817

Question (vii)

The paper uses the “blood” data set used in the course. Because we want to illustrate the negative effect with outliers, we have implanted an outlier, which simplifies the example. The dataset consists of two explanatory variables, “smoke” and “alcohol” and one target variable “blood”. The target variable is converted into a factor variable to predict the blood condition. The blood condition can be either “0” or “1”, where “0” relates to good habits and “1” otherwise.

```
myblood <- read.table(file.choose(),header=TRUE)
```

Code for the log-likelihood estimation with outlier:

```
LRblood <- glm(Blood ~ ., family='binomial', myblood)
logLik(LRblood)
```

Code for the log-likelihood estimation without outlier.

```
LRbloodOUT <- glm(Blood ~ ., family='binomial', myblood[-1,])
logLik(LRbloodOUT)
```

Now we want to make prediction using the blood dataset, first with the outlier, and then without the outlier.

We make our blood variable as a factor variable.

```
myblood <- read.table(file.choose(),header=TRUE)
Blood <- as.factor(myblood$Blood)
mybloodnew <- cbind(Blood, myblood[-1])

set.seed(2)
```

We split the dataset into a training and test set to make predictions.

```
reshuffleblood <- mybloodnew [sample(nrow(mybloodnew)),]
miniblood <- reshuffleblood[1:70,]
testblood <- reshuffleblood[71: nrow(reshuffleblood),]

LRmodelblood <- glm(Blood ~ ., family='binomial', miniblood)
```

We make predictions using the following code.

```
probabilitiesLRblood <- predict(LRmodelblood, testblood [,-1],type= "response")
predictionLRblood <- ifelse(probabilitiesLRblood > 0.5, "1", "0")
classificationtableblood <- table(pred= predictionLRblood, testblood [,1])
acctestLRblood <- sum(diag(classificationtableblood))/sum(classificationtableblood)
acctestLRblood
```

ACC:0.8064

After eliminating the outlier

```
mybloodOUT <- mybloodnew[-1,]
```

We reshuffle our datasets and then create a logistic regression model. Last, we make predictions without the outlier.

```
set.seed(2)
reshuffleblood <- mybloodOUT [sample(nrow(mybloodOUT)),]
miniblood <- reshuffleblood[1:70,]
testblood <- reshuffleblood[71: nrow(reshuffleblood),]

LRmodelblood <- glm(Blood ~ ., family='binomial', miniblood)

probabilitiesLRblood <- predict(LRmodelblood, testblood [,-1],type= "response")
predictionLRblood <- ifelse(probabilitiesLRblood > 0.5, "1", "0")
classificationtableblood <- table(pred= predictionLRblood, testblood [,1])
acctestLRblood <- sum(diag(classificationtableblood))/sum(classificationtableblood)
acctestLRblood
```

ACC:0.8333

Appendix 2

Data

```
mywholesale <- read.table(file.choose(),header=TRUE)
```

The wholesale data have predominant features. Example, one can observe that “Grocery” has a maximum of 39694 whereas Delicatessen has a maximum of 7844. Therefore, we standardize using the following code.

```
means <- apply(mywholesale,2,mean)
standarddeviations <- apply(mywholesale,2,sd)
mywholesaleSTAN <- scale(mywholesale,center=means,scale=standarddeviations)
mydistmatrixSTAN <- dist(mywholesaleSTAN)
print(mydistmatrixSTAN)
mydistsubmatrixSTAN <- as.matrix(mydistmatrixSTAN)
mydistsubmatrixSTAN <- mydistsubmatrixSTAN[1:10,1:10]
print(mydistsubmatrixSTAN)
```

Question (i)

Following code will run the dist function to return the Euclidean distance. However, other distances are used later, example Manhattan and maximum. The "dist" function create a structure where the distance matrix is saved and to work with the matrix, we use the code "as.matrix".

```
myahclustSTAN <-hclust(dist(mywholesaleSTAN))
plot(myahclust,hang=-1,cex=0.75)
myahclustManhattan <-hclust(dist(mywholesaleSTAN,method="manhattan"))
plot(myahclustManhattan,hang=-1,cex=0.75)
myahclustMaximum <-hclust(dist(mywholesaleSTAN,method="maximum"))
plot(myahclustMaximum,hang=-1,cex=0.75)
```

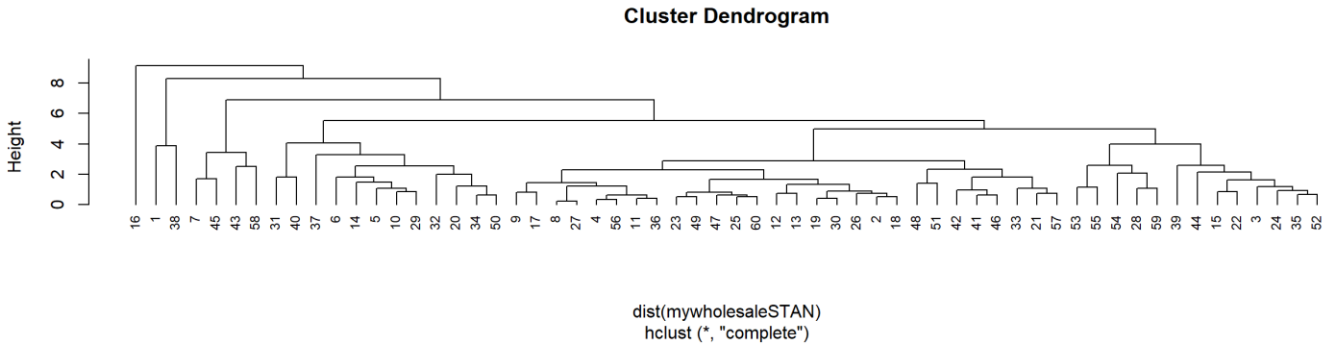
The "hang" function will allow one to adjust the position of the labels of the objects.

After finding the preferred distance we continue with splitting the dataset into different clusters:

```
par(mfrow=c(1,1))
labels2 <-cutree(myahclustManhattan,k=2)
labels3 <-cutree(myahclustSTAN,k=3)
labels4 <-cutree(myahclustSTAN,k=4)
labels5 <-cutree(myahclustManhattan,k=5)
plot(myahclustManhattan,hang=-1,labels5, ,cex=0.75)
```

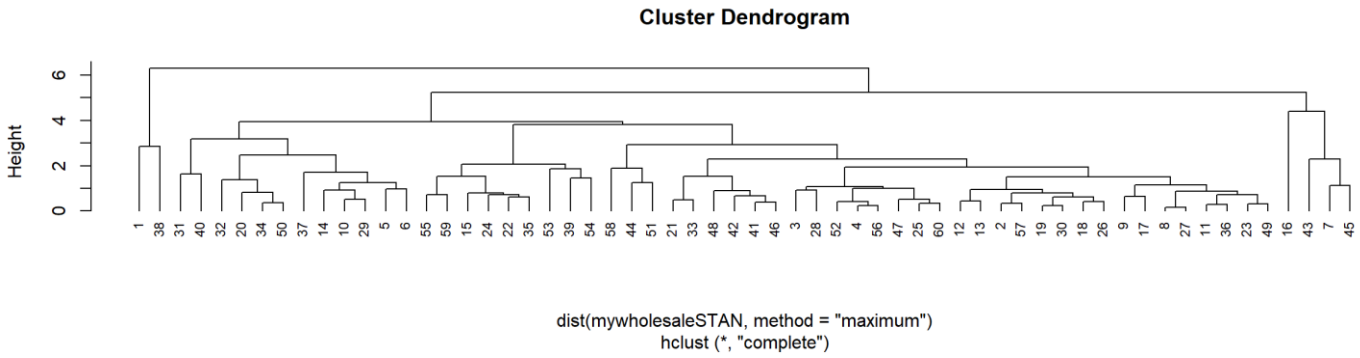
The preferred distance is the Manhattan with the use of 5 clusters.

Figure 10: Dendrogram using Euclidean distance.



Source: See code in appendix 2, Question (i).

Figure 11: Dendrogram using Maximum distance.



Source: See code in appendix 2, Question (i).

Question (ii)

The following code allows one to experiment with several values of K, and try to choose a good K. Here we will experiment with k=1, … ,10.

```
mytotwithinss <- NULL
for(auxk in  1:10){
set.seed(1)
myKmeansauxkmultistart <- kmeans(mywholesaleSTAN, auxk, nstart=5000)
mytotwithinss[auxk] = myKmeansauxkmultistart$tot.withinss
plot(mytotwithinss, xlab='k')}
```

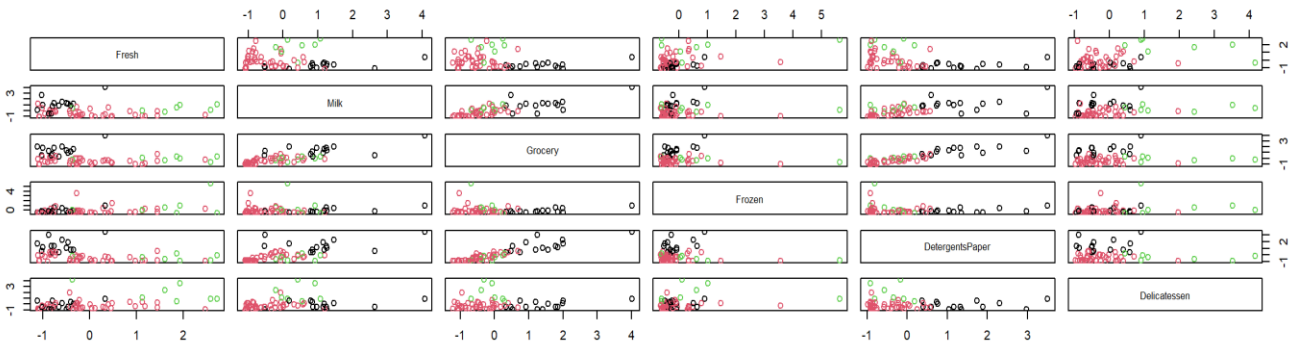
Create k-means using 3 clusters.

```
set.seed(1)
myKmeans3 <- kmeans(mywholesaleSTAN, 3)
```

And to plot the data we use the code:

```
with(as.data.frame(mywholesaleSTAN),pairs(as.data.frame(mywholesaleSTAN),
col=c(1:11)[myKmeans3$cluster]))
with(as.data.frame(mywholesaleSTAN),pairs(as.data.frame(mywholesaleSTAN)[,1:2],
col=c(1:11)[myKmeans3$cluster]))
```

Figure 12: K-means of all variables with K=3.



Source: Appendix 2, Question (ii).

Question (iii)

We test for K=1, …, 10 and find the best with a multistart of 5000.

```
customer <- read.table(file.choose(),header=TRUE)
mytotwithinss <- NULL
for(auxk in  1:10){
set.seed(1)
myKmeansauxkmultistart <- kmeans(customer[-2], auxk, nstart=5000)
mytotwithinss[auxk] = myKmeansauxkmultistart$tot.withinss
plot(mytotwithinss, xlab='k')}
```

First, we test the data without the outlier using 5 clusters. We do not include the ID of the customer as it is not relevant for our clustering process.

```
set.seed(3)
myKmeans3customer <- kmeans(customer[-200,-1:-2], 5)
with(as.data.frame(customer[-200,-1:-2]), pairs(as.data.frame(customer[-200,-1:-2],),
col=c(1:11)[myKmeans3customer $cluster]))
myKmeans3customer <- kmeans(customer[-200,-1:-2], 5)
with(as.data.frame(customer[-200,-1:-2]), pairs(as.data.frame(customer[-200,-2],)[,3:4], col=c(1:11)[
myKmeans3customer$cluster]))
```

With outliers

```
set.seed(3)
myKmeans3customer <- kmeans(customer[-1:-2], 5)
with(as.data.frame(customer[-1:-2]), pairs(as.data.frame(customer[-2],)[,3:4], col=c(1:11)[
myKmeans3customer$cluster]))
```

Appendix Part 3

We will first read the file and then convert our class variable into a factor variable and last, we normalize the data.

```
mywine <- read.table(file.choose(),header=TRUE)

myX <- mywine [,-12]
Class <- as.factor(mywine$Class)
mywineF<-  cbind(myX, Class)
mywineNORM<- mywineF
for(i in 1:length(colnames(mywineF))-1) {
      if(class(mywineF[,i]) == "numeric" || class(mywineF[,i]) == "integer") {
         minimum<-min(mywineF[,i])
         maximum<-max(mywineF[,i])
         mywineNORM[,i] <- as.vector(scale(mywineF[,i],center=minimum,scale=maximum-minimum))

}
      }
summary(mywineNORM)
```

Question (i)

To plot in 3D, we will be using the following R package;

```
install.packages('scatterplot3d',dependencies=TRUE)
library('scatterplot3d')
```

Package for autoplot

```
install.packages('ggfortify')
library('ggfortify')
```

We will construct the covariance matrix and next, construct the PCA using the function 'prcomp'. The function 'autoplot' is used to create the plot with colour coding according to the classes.

```
covmatmywine<- cov(mywineNORM[-12])
print(round(covmatmywine,2))

mywinewithoutlabel <- mywineNORM[,-12]
myPCA <- prcomp(mywinewithoutlabel)
autoplot(myPCA, data = mywineNORM, colour = 'Class')
summary(myPCA)
```

Question (ii)

First, we derive a Multidimensional Scaling with *l=2* and the Euclidean distance. Furthermore, one will might need to run the MDS several times with different seeds to make sure we do not get trapped in a local optimum, as opposed to the global optimum.

```
set.seed(23)
```

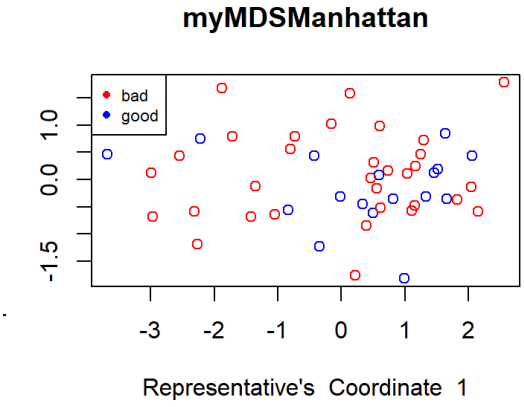
With Euclidean distance:

```
myMDS <- cmdscale(dist(mywineNORM), 2, eig=TRUE)
x <- myMDS$points[,1]
y <- myMDS$points[,2]
plot(x, y, xlab="Representative's Coordinate 1", ylab="Representative's Coordinate 2", main="
myMDS ", col =cols[mywineNORM $Class])
legend('topleft', col=cols, legend=levels(mywineNORM $Class), pch = 16, cex = 0.7)
```

With Manhattan distance:

```
myMDSManhattan <- cmdscale(dist(mywineNORM,method= "manhattan"), 2, eig=TRUE)
x <- myMDSManhattan$points[,1]
y <- myMDSManhattan$points[,2]
plot(x, y, xlab="Representative's Coordinate 1", ylab="Representative's Coordinate 2", main="
myMDSManhattan ", col =cols[mywineNORM $Class])
legend('topleft', col=cols, legend=levels(mywineNORM $Class), pch = 16, cex = 0.7)
```

Figure 13: MDS, Manhattan distance.

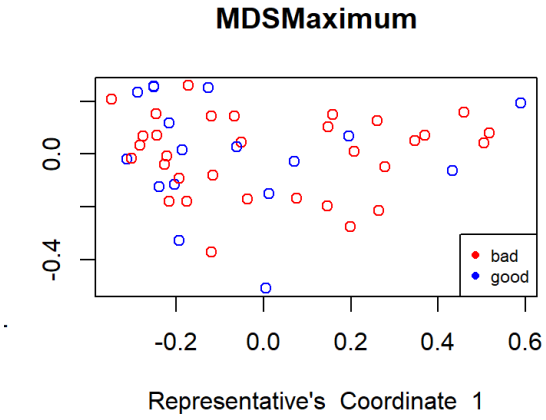


Source: See appendix 3, Question (ii).

With Maximum distance:

```
myMDSMaximum <- cmdscale(dist(mywineNORM,method= "maximum"), 2, eig=TRUE)
x <- myMDSMaximum$points[,1]
y <- myMDSMaximum$points[,2]
plot(x, y, xlab="Representative's Coordinate 1", ylab="Representative's Coordinate 2",
main="MDSMaximum", col =cols[mywineNORM $Class])
legend('bottomright', col=cols, legend=levels(mywineNORM $Class), pch = 16, cex = 0.7)
```

Figure 13: MDS, Manhattan distance.



Source: See appendix 3, Question (ii

References

- Applied Maths NV (2018), BioNumerics Tutorial Calculating a PCA and a MDS on a character data set.
- Chavent M. et. al (2021), Handling Correlations in Random Forests: which Impacts on Variable Importance and Model Interpretability?
- Customer dataset, https://www.kaggle.com/datasets/shrutimechlearn/customer-data, accessed (04/11-2022)
- Darst BF, Malecki KC, Engelman CD, September (2018). Using recursive feature elimination in random forest to account for correlated variables in high dimensional data. BMC Genet.
- F. Provost and T. Fawcett (2013), Data Science for Business: What You Need to Know About Data Mining and Data-Analytic Thinking. O'Reilly
- Hastie, T. Tibshirani, R. Friedman, J. (2009).” The Elements of Statistical Learning: Data Mining, Inference and Prediction”, second edition.
- International Journal of Engineering &Technology, 7 (1.8) (2018), “Semi-supervised learning: a brief review”, 81-85.
- Jesus Rodriguez, 2017, Understanding Semi-supervised Learning, https://jrodthoughts.medium.com/understanding-semi-supervised-learning-a6437c070c87, accessed (3/11-2022).
- Luis Angel García-Escudero · Alfonso Gordaliza ·Carlos Matrán · Agustín Mayo-Iscar, (2010). “A review of robust clustering methods”
- Shaffer A. Clifford, January 19, (2010). A Practical Introduction to Data Structures and Algorithm Analysis Third Edition (C++ Version)
- Yu-Jie Wang (2010), A clustering method based on fuzzy equivalence relation for customer relationship management. Expert Systems with Applications, Volume 37, Issue 9, Pages 6421-6428.