National Technical University of Athens
School of Applied Mathematical and Physical Sciences
Postgraduate Master's Programme in Applied Mathematical Sciences

University of Seville
Department of Statistics and Operational Research

Classical and Modern Approaches to Classification and Dimensionality Reduction Techniques

Dorothea Barmpakou

Master Thesis

Supervisors:
Professor C. Caroni (National Technical University of Athens, Greece)
Professor I. Barranco–Chamorro (University of Seville, Spain)

Examinor:
Jose María Fernández Ponce (Profesor Titular de Universidad de Sevilla)

This work was carried out with the support of the Erasmus Programme under an Inter–Institutional Agreement of Higher Education Student and Staff Mobility between the University of Seville and the National Technical University of Athens.

June 2019

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Εφαρμορμένων Μαθηματικών και Φυσικών Επιστημών
Δ.Π.Μ.Σ στις Εφαρμοσμένες Μαθηματικές Επιστήμες

Πανεπιστήμιο Σεβίλλης
Τμήμα Στατιστικής και Επιχειρησιακής Έρευνας

Κλασικές και Σύγχρονες Μέθοδοι για την Κατηγοριοποίηση και Μείωση Διαστάσεων

Δωροθέα Μπαρμπάκου

Διπλωματική εργασία

Επιβλέπουσες Καθηγήτριες:
Χ. Καρώνη (Εθνικό Μετσόβιο Πολυτεχνείο, Ελλάδα)
I. Barranco–Chamorro (Πανεπιστήμιο Σεβίλλης, Ισπανία)

Εξεταστής:
Jose María Fernández Ponce (Καθηγητής Πανεπιστημίου Σεβίλλης)

Η εργασία αυτή πραγματοποιήθηκε με την υποστήριξη του προγράμματος Erasmus στο πλαίσιο Κινητικότητας Φοιτητών και Προσωπικού μέσω της Διμερούς Συμφωνίας μεταξύ του Πανεπιστημίου της Σεβίλλης και του Εθνικού Μετσόβιου Πολυτεχνείου.

Ιούνιος 2019

Acknowledgements

Throughout the accomplishment of this master thesis I have received a great deal of support.
First of all, I would like to express my very great appreciation to my supervisors Professor C. Caroni of the School of Applied Mathematical and Physical Sciences at the National Technical University of Athens and Professor I. Barranco–Chamorro of the Department of Statistics and Operational Research at the University of Seville. Their valuable contribution encouraged our mutual and harmonious cooperation.
Then, I would like to acknowledge all of my professors from the Postgraduate Master's Programme, as well as those from my Undergraduate Degree that played an important role in my education.
Finally, I would like to thank my family for their support all of these years, financial and moral, and my friends for their help and encouragement.

Abstract

In this thesis, we focus on techniques for dimensionality reduction and classification problems, which facilitate the statistical analysis and interpretation of complex data.

In Chapter 1, we present Principal Components Analysis (PCA): a dimensionality reduction technique. We introduce its aim and the theoretical basis, we define the properties of Principal Components and their correlation structure. The loadings, component scores and correlation circle are analysed. Methods for extracting the appropriate number of Principal Components are included. Furthermore, we carry out a classical and a modern application of PCA to two different datasets. Specifically, we describe and inspect the Irish dataset, in which the number of variables is lower than the number of the individuals (classical application), and the Chicken dataset which includes far fewer individuals than variables (modern application).

In Chapter 2, Classification is introduced and some of the most important parametric classifiers are analysed. Firstly, we introduce Logistic Regression Analysis, the interpretation and estimation of its coefficients and the ROC Curve and we apply it to the Irish dataset. Then, Linear Discriminant Analysis is introduced, its method and application to the Irish data. Lastly, the theoretical basis of Quadratic Discriminant Analysis is presented and its application to the Irish dataset as well.

In Chapter 3, we introduce K–Nearest Neighbors non–parametric method for classification, its method and application to the Irish dataset and to a more complex one: Khan dataset. We extract important insights.

Chapter 4 is devoted to methods based on Trees. More precisely, Classification Trees and Regression Trees methods are analysed. Regarding the Classification Trees, we introduce the method, present the building procedure of a classification tree, the tree pruning and some advantages of Classification Trees method, and we apply it to the Irish dataset. Regarding the Regression Trees, we introduce the method and the pruning procedure, and we apply it to the Boston dataset.

Finally, Chapter 5 includes important remarks and conclusions, taking into account all of the methods applied to the Irish data.

Resumen

En este Trabajo Fin de Máster, nos centramos en técnicas para reducción de la dimensionalidad y clasificación, que facilitan el análisis estadístico e interpretación de datos más complejos.

En el primer capítulo del trabajo, se presenta el Análisis de Componentes Principales (ACP o PCA), una técnica útil para reducción de dimensionalidad. Introducimos sus objetivos y su base teórica, definimos las propiedades de las Componentes Principales y su estructura de correlaciones. Se analizan las cargas (loadings), scores y el círculo de correlación. Así mismo se incluyen métodos para escoger las Componentes Principales significativas. Además, realizamos una aplicación de ACP clásica y una moderna, a dos diferentes conjuntos de datos. Más concretamente, estudiamos el conjunto Irish data, en el que el número de variables es menor que el número de individuos (aplicación clásica), y el conjunto Chicken data, que incluye mucho menos individuos que variables (aplicación moderna). Se obtienen conclusiones para estos conjuntos de datos.

En el segundo capítulo, se introduce la clasificación y se analizan algunos de los clasificadores más importantes. Tratamos en primer lugar, el Análisis de Regresión Logística, la interpretación y estimación de sus coeficientes, la Curva ROC, y aplicamos este método al conjunto Irish data. En segundo lugar, se introduce el Análisis Discriminante Lineal (ADL o LDA), el método y aplicación al conjunto de datos Irish data. Finalmente, se analiza la base teórica del Análisis Discriminante Quadrático (ADQ o QDA) y su aplicación a Irish dataset también.

En el tercer capítulo, vemos el método de K vecinos más cercanos o K-Nearest Neighbors (KNN), un clasificador no paramétrico. Se detallan su método, su aplicación al conjunto de datos Irish data y además a un conjunto de datos más complejo: Khan dataset. Extraemos conclusiones importantes.

El cuarto capítulo trata sobre métodos basadas en Árboles. Más precisamente, se explican los Árboles de Clasificación y Regresión. En cuanto a los Árboles de Clasificación, introducimos el método, presentamos el proceso de construir un Árbol de Clasificación y de podarlo, citamos algunas ventajas de este clasificador y lo aplicamos al Irish dataset. En cuanto a los Árboles de Regresión, también introducimos el método, la poda y aplicamos un Árbol de Regresión al conjunto de datos Boston data.

Para finalizar, el quinto capítulo incluye algunas conclusiones y observaciones, teniendo en cuenta todos los métodos aplicados al conjunto de datos Irish dataset.

Περίληψη

Στο πλαίσιο της παρούσας διπλωματικής εργασίας, εστιάζουμε σε τεχνικές μείωσης διαστάσεων και σε προβλήματα κατηγοριοποίησης/ταξινόμησης (classification), που διευκολύνουν την στατιστική ανάλυση και τη γνώση και κατανόηση σύνθετων δεδομένων.

Στο Κεφάλαιο 1 παρουσιάζουμε τη Μέθοδο Κύριων Συνιστωσών (PCA), μια τεχνική μείωσης διαστάσεων. Εισάγουμε τον σκοπό της μεθόδου και το θεωρητικό της υπόβαθρο, ορίζουμε τις ιδιότητες των Κύριων Συνιστωσών και τη δομή συσχέτισης τους. Τα φορτία (loadings), οι τιμές (scores) των Κύριων Συνιστωσών και ο κύκλος συσχέτισης αναλύονται. Μέθοδοι για επιλογή του κατάλληλου αριθμού Κύριων Συνιστωσών που πρέπει να χρησιμοποιηθούν στην ανάλυση περιέχονται. Ακολούθως, πραγματοποιούμε μια κλασική και μια σύγχρονη εφαρμογή της Μεθόδου Κύριων Συνιστωσών σε δύο διαφορετικά σετ δεδομένων. Πιο συγκεκριμένα, εξετάζουμε το Irish dataset, κατά το οποίο το πλήθος των μεταβλητών είναι μικρότερο από αυτό των παρατηρήσεων (κλασική εφαρμογή) και το Chicken data, το οποίο περιέχει πολύ λιγότερες παρατηρήσεις σε σχέση με τις μεταβλητές (σύγχρονη εφαρμογή).

Στο δεύτερο κεφάλαιο, εισάγεται ο όρος της κατηγοριοποίησης/ταξινόμησης και κάποιοι από τους πιο σημαντικούς παραμετρικούς ταξινομητές αναλύονται. Πρώτα, εισάγουμε την Ανάλυση Λογιστικής Παλινδρόμησης (Logistic Regression), την ερμηνεία και εκτίμηση των παραμέτρων της, την Καμπύλη ROC και εφαρμόζουμε την τεχνική αυτή στο σύνολο δεδομένων Irish data. Έπειτα, εισάγεται η Γραμμική Διακριτική Ανάλυση (LDA), η μέθοδος της και η εφαρμογή της στο Irish dataset. Τέλος, αναλύεται το θεωρητικό/μαθηματικό υπόβαθρο της Τετραγωνικής Διακριτικής Ανάλυσης (QDA) και η εφαρμογή της στο Irish dataset, επίσης.

Στο τρίτο κεφάλαιο, εισάγουμε τη μέθοδο των Κ Κοντινότερων Γειτόνων (K Nearest Neighbors, KNN): έναν μη παραμετρικό ταξινομητή. Παραθέτουμε τη μέθοδο του KNN και την εφαρμογή αυτού στο Irish dataset, καθώς και σε ένα πιο σύνθετο: το σύνολο δεδομένων Khan. Αντλούμε σημαντικά συμπεράσματα.

Το Κεφάλαιο 4 εξειδικεύεται σε μεθόδους βασισμένες σε Δέντρα Αποφάσεων. Ειδικότερα, αναλύονται τα Δέντρα Κατηγοριοποίησης/Ταξινόμησης (Classification Trees) και τα Δέντρα Παλινδρόμησης (Regression Trees). Όσον αφορά τα Δέντρα Ταξινόμησης, εισάγουμε τη μέθοδο, παρουσιάζουμε τη διαδικασία κατασκευής ενός Δέντρου Ταξινόμησης, καθώς κι ενός Κλαδεμένου Δέντρου (Tree Pruning), παραθέτουμε κάποια βασικά πλεονεκτήματα της τεχνικής αυτής και την εφαρμόζουμε στο Irish dataset. Όσον αφορά τα Δέντρα Παλινδρόμησης, εισάγουμε τη μέθοδο, τη διαδικασία Κλαδέματος του Δέντρου και την εφαρμόζουμε στο σύνολο δεδομένων Boston data.

Κλείνοντας, το πέμπτο κεφάλαιο περιέχει σημαντικά συμπεράσματα και επισημάνσεις, λαμβάνοντας υπόψιν όλες τις εφαρμογές των μεθόδων που χρησιμοποιήθηκαν πάνω στο σύνολο δεδομένων Irish data.

Contents

Chapter 1

Principal Component Analysis

1.1 Introduction

PCA: An Unsupervised Learning Method

Principal Components Analysis (PCA) is a technique of *unsupervised learning*, which refers to a set of statistical tools intended for the setting in which we have only a set of features $x_1, x_2, \ldots, x_p$ measured on n observations. We are not interested in prediction, because we do not have an associated response variable $\mathbf{y}$. Rather, the goal is to discover interesting things about the measurements on $x_1, x_2, \ldots, x_p$. In unsupervised learning the exercise tends to be more subjective, and there is no simple goal for the analysis, such as prediction of a response. Unsupervised learning is often performed as part of an exploratory data analysis. Furthermore, it can be hard to assess the results obtained from unsupervised learning methods, since there is no universally accepted mechanism for performing cross validation or computing validating results on an independent data set. The reason for this difference is simple. If we fit a predictive model using a supervised learning technique, then it is possible to check our work by seeing how well our model predicts the response $\mathbf{y}$ on observations not used in fitting the model. However, in unsupervised learning, there is no way to check our work because we do not know the true answer–the problem is unsupervised. [14, 11]

Aim of PCA

Principal components analysis, which was developed by Hotelling in 1933 [12] after its origin by Karl Pearson in 1901 [31], is a technique for forming new variables which are linear composites of the original variables. If there are n observations with p variables, where $n > p$, then the maximum number of new variables that can be formed is equal to the number of original variables and the new variables are uncorrelated among themselves. In the case where $n < p$, this number is equal to $n - 1$. These few linear combinations can be used to summarize the data, losing in the process as little information as possible. This attempt to reduce dimensionality can be described as *parsimonious summarization* of the data. This greatly simplifies the task of understanding the structure of the data since it is much easier to interpret two or three uncorrelated variables than 20 or 30 that have a complicated pattern of interrelationships. The central idea is based on the concept of the proportion of the total variance (the sum of the variances of the p original variables) that is accounted for by each of the new variables. PCA transforms the set of correlated variables $x_1, x_2, \ldots, x_p$ to a set of uncorrelated variables $y_1, y_2, \ldots, y_p$ called principal components, in such a way that y_1 explains the maximum possible of the total variance, y_2 the maximum possible of the remaining variance, and so on. The full set of p principal components fully explains the total variance:

$$\sum_{i=1}^{p} var(y_i) = \sum_{i=1}^{p} var(x_i).$$

However, if it turns out that the first few principal components account for a large enough part of the total variance, most of the variation in the xs being explained by the first few ys, then the remaining principal components can be discarded without too great a loss of information. It is usual to standardize the xs to unit variance before carrying out PCA so that each x-variable makes the same contribution to the total variance, and thus:

$$\sum_{i=1}^{p} var(x_i) = p.$$

It is clear that there have to be some constraints on the coefficients/weights of each component. Otherwise, we could make the variance of any $\mathbf{y}$ as large as we pleased simply by making the components large enough. Hence, the PCs are defined in such a way that each succeeding principal component has the highest variance possible under the constraint that it must be orthogonal to all the preceding components. In this way, the resulting PCs form an uncorrelated orthogonal basis.

Additionally, another use of Principal Components Analysis is that it also serves as a tool for data visualization (visualization of the observations or visualization of the variables) and data preprocessing before supervised techniques are applied. [3, 35, 24, 14]

1.2 Definition and Properties of Principal Components

Definition 1.2.1. If $\mathbf{x}$ is a random vector with mean $\underline{\mu}$ and covariance matrix $\mathbf{\Sigma}$, then the principal component transformation is the transformation

$$\mathbf{x} \to \mathbf{y} = \mathbf{\Gamma}'(\mathbf{x} - \underline{\mu}),$$

where $\mathbf{\Gamma}$ is orthogonal, $\mathbf{\Gamma}'\mathbf{\Sigma}\mathbf{\Gamma} = \mathbf{\Lambda}$ is diagonal and $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$. The strict positivity of the eigenvalues λ_i is guaranteed if $\mathbf{\Sigma}$ is positive definite. This representation of $\mathbf{\Sigma}$ follows from the Spectral Decomposition Theorem (Theorem 1.2.1). The ith principal component of $\mathbf{x}$ may be defined as the ith element of the vector $\mathbf{y}$, namely as:

$$y_i = \underline{\gamma}_i{}'(\mathbf{x} - \underline{\mu})$$

Here, $\underline{\gamma}_{(i)}$ is the ith column of $\mathbf{\Gamma}$, and may be called the ith vector of *principal component loadings*. The function y_p may be called the *last principal component* of $\mathbf{x}$.

Theorem 1.2.1 (Spectral Decomposition Theorem)**.** *Any symmetric matrix $\mathbf{A}(p \times p)$ can be written as:*

$$\mathbf{A} = \mathbf{\Gamma}\mathbf{\Lambda}\mathbf{\Gamma}' = \sum \lambda_i \underline{\gamma}_{(i)} \underline{\gamma}_{(i)}{}',$$

where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues of $\mathbf{A}$, and $\mathbf{\Gamma}$ an orthogonal matrix whose columns are standardized eigenvectors of $\mathbf{A}$.

In our case, $\mathbf{\Sigma}$ is the symmetric matrix $\mathbf{A}$.

Theorem 1.2.2. *If $\mathbf{x} \sim (\underline{\mu}, \mathbf{\Sigma})$ and $\mathbf{y}$ is as defined in Definition 1.2.1 then:*

(a) $\mathrm{E}(y_i) = 0$

(b) $\mathrm{V}(y_i) = \lambda_i$

(c) $\mathrm{Cov}(y_i, y_j) = 0$, $i \neq j$.

(d) $\mathrm{V}(y_1) \geq \mathrm{V}(y_2) \geq \ldots \geq \mathrm{V}(y_p) \geq 0$

(e) $\sum_{i=1}^{p} \mathrm{V}(y_i) = tr\mathbf{\Sigma}$

(f) $\prod_{i=1}^{p} \mathrm{V}(y_i) = |\mathbf{\Sigma}|$

In practice, the PC technique should be applied by replacing the population features, described above, by the sample–based counterparts of them. That is, $\underline{\mu}$ should be replaced by the vector of the sample means, $\mathbf{\Sigma}$ by $\mathbf{S}$, which is the sample covariance matrix of $\mathbf{x}$, and so on.

Further Properties

I From part *(e)* of Theorem 1.2.2 we make the following remarks:

i The proportion of the variability in the data, explained by the kth principal component is:

$$\frac{\lambda_k}{\sum_{i=1}^{p} \lambda_i}$$

ii The proportion of the total variation in the data, explained by the first k principal components is:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$$

iii The proportion of the variability in the data, not explained by the first k principal components is:

$$\frac{\sum_{i=k+1}^{p} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$$

II The principal components of a random vector are not scale–invariant. Algebraically, the lack of scale invariance can be explained as follows. Let $\mathbf{S}$ be the sample covariance matrix. Then if the ith variable is divided by d_i, the covariance matrix of the new variables is $\mathbf{DSD}$, where $\mathbf{D} = \mathrm{diag}({d_i}^{-1})$. However, if $\mathbf{x}$ is an eigenvector of $\mathbf{S}$, then $\mathbf{D}^{-1}\mathbf{x}$ is not an eigenvector of $\mathbf{DSD}$. In other words, the eigenvectors are not scale invariant. The lack of scale–invariance illustrated above implies a certain sensitivity to the way scales are chosen. Two ways out of this dilemma are possible. First, one may seek so-called "natural units", by ensuring that all variables measured are of the same type (for instance, all heights or all weights). Alternatively, one can standardize all variables so that they have unit variance, and find the principal components of the correlation matrix rather than the covariance matrix. The second option is the one most commonly employed. Thus, we choose to work with the standardized data, where the matrix containing them, given a random sample of size n of $\mathbf{x}$ on p variables is the $n \times p$ matrix:

$$\mathbf{X}_s = \begin{pmatrix} \frac{x_{11}-\overline{x}_1}{s_1} & \cdots & \frac{x_{1p}-\overline{x}_p}{s_p} \\ \vdots & & \vdots \\ \frac{x_{n1}-\overline{x}_1}{s_1} & \cdots & \frac{x_{np}-\overline{x}_p}{s_p} \end{pmatrix}.$$

Correlation Structure

We now examine the correlations between $\mathbf{x}$ and its vector of principal components, $\mathbf{y}$. For simplicity we assume that $\mathbf{x}$ (and therefore $\mathbf{y}$) has mean zero. The covariance between $\mathbf{x}$ and $\mathbf{y}$ is then:

$$\mathrm{E}(\mathbf{xy}') = \mathrm{E}(\mathbf{xx}'\mathbf{\Gamma}) = \mathbf{\Sigma\Gamma} = \mathbf{\Gamma\Lambda\Gamma}'\mathbf{\Gamma} = \mathbf{\Gamma\Lambda}$$

Therefore the covariance between x_i and y_j is $\gamma_{ij}\lambda_j$. Now x_i and y_j, have variances σ_{ii} and λ_j respectively, so if their correlation is ρ_{ij}, then:

$$\rho_{ij} = \frac{\gamma_{ij}\lambda_j}{(\sigma_{ii}\lambda_j)^{\frac{1}{2}}} = \gamma_{ij}\frac{\sqrt{\lambda_j}}{\sqrt{\sigma_{ii}}}$$

When $\mathbf{\Sigma}$ is a correlation matrix $\sigma_{ii} = 1$, so:

$$\rho_{ij} = \gamma_{ij}\sqrt{\lambda_j}$$

The proportion of variation of x_i "explained" by y_j is ρ_{ij}^2. Then, since the elements of $\mathbf{y}$ are uncorrelated, any set G of components explains a proportion:

$$\rho_{iG}^2 = \sum_{i\in G}\rho_{ij}^2 = \frac{1}{\sigma_{ii}}\sum_{i\in G}\lambda_j\gamma_{ij}^2$$

of the variation of x_i.

If all the components are included, then:

$$\sum_{j=1}^{p}\rho_{ij}^2 = \frac{\sum_{j=1}^{p}\lambda_j\gamma_{ij}^2}{\sigma_{ii}} = \frac{\sigma_{ii}}{\sigma_{ii}} = 1$$

This is rational, as in the case where all the components are taken, the whole variability of the data is "explained". [24, 27, 3]

1.3 Analytical approach of PCA

Now that we presented the mathematical base of principal components, we can formally state the objective of principal components analysis. We highlight, here, that the solution to principal components analysis is obtained by computing the eigenvalues and eigenvectors of the covariance (or correlation) matrix. The eigenvectors give the weights/coefficients that can be used to form the new variables and the eigenvalues give the variances of the new variables. Thus, assuming that there are p variables, we are interested in forming the following p linear combinations:

$$\begin{aligned} y_1 &= w_{11}x_1 + w_{12}x_2 + \ldots + w_{1p}x_p \\ y_2 &= w_{21}x_1 + w_{22}x_2 + \cdots + w_{2p}x_p \\ &\vdots \\ y_p &= w_{p1}x_1 + w_{p2}x_2 + \ldots + w_{pp}x_p \end{aligned}$$

where $y_1, y_2, \ldots, y_p$ are the p principal components and w_{ij} is the weight of the jth variable for the ith principal component. Here, instead of γ_{ij}, which was used before, we use w_{ij}. The weights w_{ij} are estimated such that:

1. The first principal component, y_1, accounts for the maximum variance in the data, the second principal component, y_2, accounts for the maximum variance that has not been accounted for by the first principal component and so on.
2. $w_{i1}^2 + w_{i2}^2 + \ldots + w_{ip}^2 = 1$, $i = 1, 2, \ldots, p$.
3. $w_{i1}w_{j1} + w_{i2}w_{j2} + \ldots + w_{ip}w_{jp} = 0$, for all $i \neq j$.

Loadings
The simple correlations between the original and the new variables are called *loadings*. They give an indication of the extent to which the original variables are influential or important in forming new variables. That is, the higher the loading the more influential the variable is in forming the principal components score and vice versa. Furthermore, the loadings can be used to interpret the meaning of the principal components or the new variables. So, above, we had seen that the correlation between x_i and y_j is:

$$\rho_{ij} = \gamma_{ij}\frac{\sqrt{\lambda_j}}{\sqrt{\sigma_{ii}}}$$

As we defined the loading as the correlation between the two variables, where l_{ij} is the loading of the jth variable (x_j) for the ith principal component (y_i), we get:

$$l_{ij} = w_{ij}\frac{\sqrt{\lambda_i}}{s_j}$$

Here, s_j is the standard deviation of the jth variable.

Component scores
An individual's score on a particular component from the PCA of the data is the value of the new variable y_i and it is called *principal components score*.

Correlation circle
This plot is useful to visualize relationships between the original variables and the PCs. Since, if PCs are based on the correlation matrix, then the sample linear correlation coefficient between the jth original variable, x_j and the ith PC, y_i, r_{ij} is

$$r_{ij} = \gamma_{ij}\sqrt{\lambda_i}$$

This is the basis of the *correlation circle*, useful to identify the original variables more correlated to first and second PCs. The correlation circle is a plot of r_{1j} versus r_{2j}. This plot shows which of the original variables are most strongly correlated with the PCs, namely those that are close to the perimeter of the circle of radius 1. (It can be applied to any pair of PCs, not necessarily y_1 and y_2).

Number of PCs to extract
Once it has been decided that performing principal components analysis is appropriate the next obvious issue is determining the number of principal components that should be retained. Following are some of the suggested rules:

1. In the case of standardized data, retain only those components whose eigenvalues are greater than one. This is referred to as the eigenvalue–greater–than–one rule. The logic behind this rule of thumb is that a component with an eigenvalue of 1 explains the same amount of variation as one of the original xs. However, Jolliffe in 1972 [15], suggests that retaining components with eigenvalues greater than 0.7 is better than the cut–off at 1. In the case of unstardardized data, exclude those principal components whose eigenvalues are less than the average. (Kaiser)

2. Examine a scree plot. This is a plot of the eigenvalues versus the component number, or similarly a plot of the percent of variance accounted for by each principal component. The idea is to look for the "elbow" which corresponds to the point after which the eigenvalues decrease more slowly. Adding components after this point explains relatively little more of the variance.

3. Retain the first k components which explain a "large" proportion of the total variation, say $70 - 80\%$. [35, 24, 3]

Interpretation
The weight given to variable i on component j is w_{ij}. The relative sizes of the w_{ij}s reflect the relative contributions made by each variable to the component. To interpret a component, we examine the pattern in the w_{ij} values for that component. Since the principal components are linear combinations of the original variables, it is often necessary to interpret or provide a meaning to the linear combination. As mentioned earlier, one can use the loadings for interpreting the principal components. The higher the loading of a variable, the more influence it has in the formation of the principal component score and vice versa. Therefore, one can use the loadings to determine which variables are influential in the formation of principal components, and one can then assign a meaning or label to the principal component. But what do we mean by influential? How high should the loading be before we can say that a given variable is influential in the formation of a principal component score? Unfortunately, there are no guidelines to help us in establishing how high is high. Traditionally, researchers have used a loading of 0.5 or above as the cutoff point. In many instances the retained principal components cannot be meaningfully interpreted. In such cases researchers have typically resorted to a rotation of the principal components. [35, 17]

Rotation is one of the main ideas of factor analysis, "borrowed" for PCA, without any implication that a factor model is being assumed. Once PCA has been used to find an m–dimensional subspace which contains most of the variation in the original p variables, it is possible to redefine, by rotation, the axes (or variables) which form a basis for this subspace. The rotated variables will together account for the same amount of variation as the first few PCs, but will no longer successively account for the maximum possible variation. Furthermore, the rotated PCs, when expressed in terms of the original variables, may be easier to interpret than the PCs, because their coefficients will typically have a simpler structure. In addition, rotated PCs offer advantages compared to unrotated PCs in some types of analysis based on PC. Lastly, rotation can provide additional insight into the influence of individual (outlying) observations. [16, 18, 17]

The *scores* resulting from the principal components can also be used as input variables for further analyzing the data using other multivariate techniques such as cluster analysis, regression and discriminant analysis. The advantage of using principal components scores is that the new variables are not correlated and the problem of multicollinearity is avoided. It should be noted, however, that although we may have "solved" the multicollinearity problem, a new problem can arise due to the inability to meaningfully interpret the principal components.[35]

1.4 Irish Dataset

The problem with which we will deal in this section, has firstly been approached by a team from the University of Ireland [19], and the conducted survey was realized on Irish population. The Irish dataset is presented as detailed below.

1.4.1 Presentation

The Business Model Canvas (BMC)
Firstly, we have to define the widely known, among entrepreneurs, method BMC. The business model canvas (BMC) is a firm–level concept of business model [28, 30, 29]. It involves nine related elements of knowledge, which represent the content ("what") of doing business. These elements are represented in Table 1.1.

ELEMENTS	DESCRIPTIONS
Customer segments	A firm serves its value proposition(s) to one or more customer segments
Value propositions	A firm offers a mix of products/services to create value for each customer segment
Channels	A firm communicates and delivers its value proposition to each customer segment via various channels
Customer relationships	A firm establishes and maintains relationships with each customer segment
Revenue streams	A firm generates revenue streams from the delivery of value to each customer segment
Key resources	A firm requires resources (e.g., people) to create and deliver the business model elements
Key activities	A firm performs a set of activities to create and deliver the business model elements
Key partners	A firm may outsource some activities to its network of suppliers/partners
Cost structure	Each element of a firm's business model has a cost component

Table 1.1: The BMC elements and their descriptions

Two–dimensional tabular framework
In the sense described above, Bandura's guidelines [1] suggest that each of the nine BMC elements should be operationalised by a set of activities representing a range of difficulty. An interpretation of Krathwohl's approach [21] to describing objectives/activities implies that each element could be represented as a function of a number of cognitive processes, which could be ordered on a scale from simple (e.g., identify) to complex (e.g., create). Krathwohl notes that such a scale is a hierarchy of judged complexity. Notwithstanding this empirical question, the idea of adding a cognitive process dimension to the BMC is consistent with Bandura's assertion that self–efficacy is a mechanism by which knowledge and skills are turned into action, and is consistent with Zott et al.[40] in that business model research requires concurrent consideration of the content (know–what) and process (know–how) of doing business.

Thus, consistent with the idea that self–efficacy builds on a dual system of knowledge and cognitive skills (e.g., Bandura [1]), each of the nine BMC elements was represented as a function of six cognitive processes, and this two–dimensional tabular framework (see Figure 1.1) was used to generate a set of activities for each element.

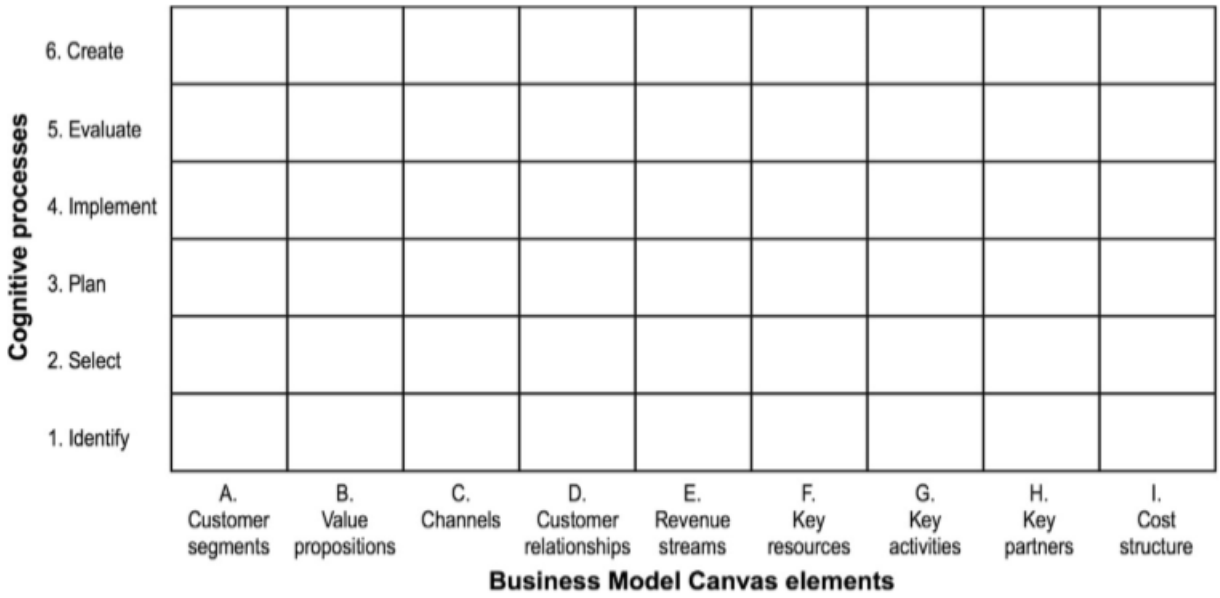

Figure 1.1: Two-dimensional tabular framework.

Scale Construction

A self–efficacy scale was constructed [19] to measure the 54 activities defined by Figure 1.1. Following Bandura's guidelines, each item was phrased as a judgement of capability. All items were scored on a 7–point Likert scale ($1 = \textit{strongly disagree}$, $7 = \textit{strongly agree}$). As a consequence, the nine subscale scores were created by calculating a total score from the six respective 7–point items. Each of these interval variables has a value from 6 to 42, and one can thus treat them as quantitative for data analysis purposes.

Data Description

Based on the above analysis, 108 entrepreneurs and 63 managers completed a survey. So, their scores according to the aforementioned scale construction on the nine variables of BMC are gathered on a data–frame, which consists of 171 rows and 10 columns. Additionally, apart from the 9 columns, which correspond to the 9 factors of BMC, one more column, indicating the status of the observations (entrepreneurs or managers), is included. More specifically, value 2 represents an entrepreneur and value 1 a manager.

Aim of our analysis

Our goal regarding the data is to find out whether the nine variables of BMC could be represented by a much smaller number of dimensions without much loss of information and also to examine if the content of these mental representations differ between entrepreneurs and managers.

By using self–efficacy to investigate how entrepreneurs and managers represent the nine business model elements, this study provides an empirical foundation for extending the reach of the BMC to the individual level, and it also extends the empirical evidence on self–efficacy differences between entrepreneurs and managers.

The dimensionality of the BMC is a key issue for both entrepreneurship and management research

on the business model [39, 23, 26, 36]. This is because while there is no one best business model for everyone, some type of business model is surfacing as a mechanism used by entrepreneurs and by managers [8, 38]. So when attempting to model the role of the BMC in either entrepreneurial or managerial processes, a researcher would generally like to replace the nine elements by a smaller number of independent variables. Indeed, researchers would typically prefer to work in lower dimensions for ease of interpretability, visualisation, understanding of the main underlying features, removing extraneous information, and so on.

The structure underlying the BMC is an important issue for those interested in the study of cognition, as it relates to how people represent nine content aspects of doing business – how they "connect the dots" so to speak [2, 9, 22, 37]. In entrepreneurial cognition research, it is usually assumed that such mental representations not only underlie thought (e.g., self–efficacy) and action (e.g., firm creation), but they also distinguish entrepreneurs from managers. For example, Brannback and Carsrud in 2009 posit that sense–making tools such as the BMC are a valid way of examining entrepreneurs' mental models and also of understanding differences in mental representations between entrepreneurs and managers [4]. However, they concluded that this area of research has yet to be fully explored. While it creates a cognitive map of nine elements of firm activities, the BMC has hitherto not been used to either study how entrepreneurs think or to compare how they differ from managers in their thinking.

1.4.2 Statistical Analysis

Exploratory Analysis First of all, we carry out an exploratory analysis in which:

- "Status" column from the data–frame is excluded.
- The data is divided into two groups: entrepreneurs and managers.
- Calculate the descriptives statistics.
- Inspect correlation between the variables.
- Detect the outliers according to Rosner' s Test and replace them with the median.

The R–packages "dplyr", "Hmisc", "corrplot", "grDevices", "EnvStats", "naniar" and "imputeTS" are used for the above procedures, as it is next illustrated.

Firstly, we take a view of the data.

```
> class(keane)
[1] "data.frame"

> dim(keane)
[1] 171  10

> head(keane)
  Status C_Seg V_Prop Chan C_Rel Rev_Str Key_Res Key_Act Key_Part Cost_Str
1      2    33     34   33    34      35      34      34       34       32
2      2    30     33   35    35      30      30      30       30       32
3      2    30     31   30    32      33      31      33       32       36
4      2    31     35   37    40      29      33      39       39       35
5      2    31     29   30    36      35      35      36       36       33
6      2    37     27   33    35      33      33      33       32       36
```

As we see, our data are structured in a data–frame, which consists of 171 observations and 10 variables. As we said in Section 1.4.1, we want to examine separately the two groups: entrepreneurs and managers. Furthermore, we do not need the "Status" column in our analysis. For this reason, using the following command in R:

```
 > entrepreneurs <- keane %>% filter(Status==2)%>% select(C_Seg:Cost_Str)
```

we extract only the entrepreneurs from the whole of our data, with the nine variables of BMC (For this, "dplyr" R–package was loaded, which facilitates the data manipulation). To check this out:

```
> dim(entrepreneurs)
[1] 108   9
```

Likewise, for the managers we get:

```
> managers<- keane %>% filter(Status==1)%>% select(C_Seg:Cost_Str)

> dim(managers)
[1] 63  9
```

Now that we have our data, we present some descriptive statistics for these two groups. In Table 1.2, the mean and the standard deviation of each of the nine variables on the two groups are provided.

	Entrepreneurs		**Managers**	
Variable	Mean	St.Dev	Mean	St.Dev
Customer Segments	35.27	3. 81	30.62	4. 45
Value Propositions	35. 9	3. 56	31.75	4. 17
Channels	34.04	4. 5	31.32	5. 22
Customer Relationships	36.89	3. 15	34.75	4. 23
Revenue Streams	34.05	4. 71	28.54	5. 86
Key Resources	33.55	4. 92	29.98	5. 05
Key activities	34.18	4. 27	30.41	5. 58
Key Partners	35.31	4. 98	32.22	5. 47
Cost Structure	35.05	5. 15	28.79	7. 36

Table 1.2: Descriptive statistics for the variables.

As a first notice, we observe from the Table 1.2 that entrepreneurs' scale scores are higher than these of managers. That means that a different mental representation between the two groups is quite reasonable.

Let us, now, inspect the correlation coefficients and the p–value of the correlation for all possible pairs of columns in each of the datasets: both entrepreneurs and managers, entrepreneurs and managers, separately. For this purpose, the R–package "Hmisc" is needed.

	C_Seg	V_Prop	Chan	C_Rel	Rev_Str	Key_Res	Key_Act	Key_Part	Cost_Str
C_Seg	1.00	0.60	0.53	0.57	0.62	0.55	0.44	0.47	0.52
V_Prop	0.60	1.00	0.55	0.47	0.44	0.54	0.52	0.45	0.30
Chan	0.53	0.55	1.00	0.51	0.46	0.50	0.58	0.44	0.33
C_Rel	0.57	0.47	0.51	1.00	0.55	0.57	0.47	0.51	0.35
Rev_Str	0.62	0.44	0.46	0.55	1.00	0.68	0.58	0.46	0.65
Key_Res	0.55	0.54	0.50	0.57	0.68	1.00	0.69	0.61	0.54
Key_Act	0.44	0.52	0.58	0.47	0.58	0.69	1.00	0.59	0.45
Key_Part	0.47	0.45	0.44	0.51	0.46	0.61	0.59	1.00	0.49
Cost_Str	0.52	0.30	0.33	0.35	0.65	0.54	0.45	0.49	1.00

n= 171

P

	C_Seg	V_Prop	Chan	C_Rel	Rev_Str	Key_Res	Key_Act	Key_Part	Cost_Str
C_Seg		0	0	0	0	0	0	0	0
V_Prop	0		0	0	0	0	0	0	0
Chan	0	0		0	0	0	0	0	0
C_Rel	0	0	0		0	0	0	0	0
Rev_Str	0	0	0	0		0	0	0	0
Key_Res	0	0	0	0	0		0	0	0
Key_Act	0	0	0	0	0	0		0	0
Key_Part	0	0	0	0	0	0	0		0
Cost_Str	0	0	0	0	0	0	0	0	

Figure 1.2: Correlation Matrix for both entrepreneurs and managers.

	C_Seg	V_Prop	Chan	C_Rel	Rev_Str	Key_Res	Key_Act	Key_Part	Cost_Str
C_Seg	1.00	0.58	0.59	0.52	0.50	0.50	0.46	0.41	0.45
V_Prop	0.58	1.00	0.47	0.49	0.31	0.41	0.46	0.41	0.20
Chan	0.59	0.47	1.00	0.46	0.33	0.31	0.47	0.31	0.26
C_Rel	0.52	0.49	0.46	1.00	0.50	0.46	0.46	0.46	0.28
Rev_Str	0.50	0.31	0.33	0.50	1.00	0.68	0.63	0.42	0.52
Key_Res	0.50	0.41	0.31	0.46	0.68	1.00	0.61	0.50	0.56
Key_Act	0.46	0.46	0.47	0.46	0.63	0.61	1.00	0.48	0.40
Key_Part	0.41	0.41	0.31	0.46	0.42	0.50	0.48	1.00	0.43
Cost_Str	0.45	0.20	0.26	0.28	0.52	0.56	0.40	0.43	1.00

n= 108

P

	C_Seg	V_Prop	Chan	C_Rel	Rev_Str	Key_Res	Key_Act	Key_Part	Cost_Str
C_Seg		0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
V_Prop	0.0000		0.0000	0.0000	0.0011	0.0000	0.0000	0.0000	0.0363
Chan	0.0000	0.0000		0.0000	0.0004	0.0011	0.0000	0.0009	0.0072
C_Rel	0.0000	0.0000	0.0000		0.0000	0.0000	0.0000	0.0000	0.0029
Rev_Str	0.0000	0.0011	0.0004	0.0000		0.0000	0.0000	0.0000	0.0000
Key_Res	0.0000	0.0000	0.0011	0.0000	0.0000		0.0000	0.0000	0.0000
Key_Act	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000		0.0000	0.0000
Key_Part	0.0000	0.0000	0.0009	0.0000	0.0000	0.0000	0.0000		0.0000
Cost_Str	0.0000	0.0363	0.0072	0.0029	0.0000	0.0000	0.0000	0.0000	

Figure 1.3: Correlation Matrix for entrepreneurs.

	C_Seg	V_Prop	Chan	C_Rel	Rev_Str	Key_Res	Key_Act	Key_Part	Cost_Str
C_Seg	1.00	0.35	0.33	0.51	0.52	0.45	0.17	0.38	0.32
V_Prop	0.35	1.00	0.54	0.31	0.26	0.55	0.40	0.35	0.02
Chan	0.33	0.54	1.00	0.49	0.48	0.67	0.62	0.51	0.25
C_Rel	0.51	0.31	0.49	1.00	0.48	0.63	0.36	0.48	0.25
Rev_Str	0.52	0.26	0.48	0.48	1.00	0.57	0.38	0.34	0.60
Key_Res	0.45	0.55	0.67	0.63	0.57	1.00	0.71	0.68	0.37
Key_Act	0.17	0.40	0.62	0.36	0.38	0.71	1.00	0.62	0.30
Key_Part	0.38	0.35	0.51	0.48	0.34	0.68	0.62	1.00	0.41
Cost_Str	0.32	0.02	0.25	0.25	0.60	0.37	0.30	0.41	1.00

n= 63

P

	C_Seg	V_Prop	Chan	C_Rel	Rev_Str	Key_Res	Key_Act	Key_Part	Cost_Str
C_Seg		0.0048	0.0090	0.0000	0.0000	0.0002	0.1791	0.0019	0.0097
V_Prop	0.0048		0.0000	0.0131	0.0375	0.0000	0.0013	0.0049	0.8520
Chan	0.0090	0.0000		0.0000	0.0000	0.0000	0.0000	0.0000	0.0523
C_Rel	0.0000	0.0131	0.0000		0.0000	0.0000	0.0036	0.0000	0.0456
Rev_Str	0.0000	0.0375	0.0000	0.0000		0.0000	0.0023	0.0064	0.0000
Key_Res	0.0002	0.0000	0.0000	0.0000	0.0000		0.0000	0.0000	0.0027
Key_Act	0.1791	0.0013	0.0000	0.0036	0.0023	0.0000		0.0000	0.0167
Key_Part	0.0019	0.0049	0.0000	0.0000	0.0064	0.0000	0.0000		0.0008
Cost_Str	0.0097	0.8520	0.0523	0.0456	0.0000	0.0027	0.0167	0.0008	

Figure 1.4: Correlation Matrix for managers.

Then, using the "corrplot" package in R, we visualize the correlation matrix between the variables on both entrepreneurs and managers (Figure 1.5).

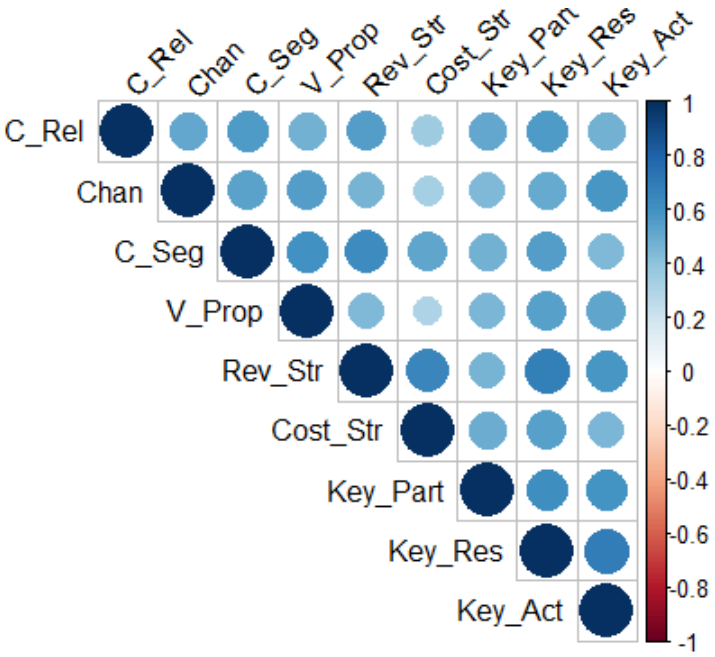

Figure 1.5: Correlation between the variables for the whole of the data.

A similar image is taken, if we look separately for the correlations between the variables in the two groups. Figure 1.6 visualizes the correlations regarding the entrepreneurs' scores, whereas Figure 1.7 visualizes the correlations regarding the managers' scores.

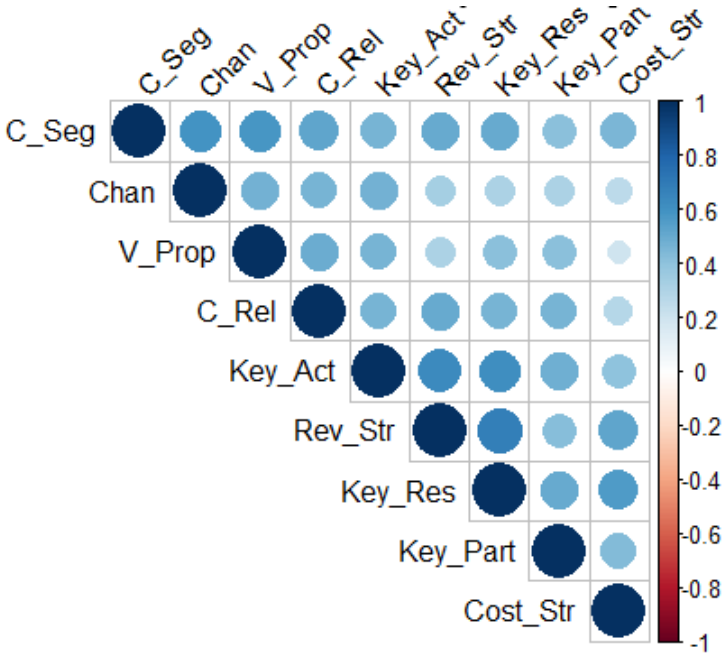

Figure 1.6: Correlation between the variables for the entrepreneurs.

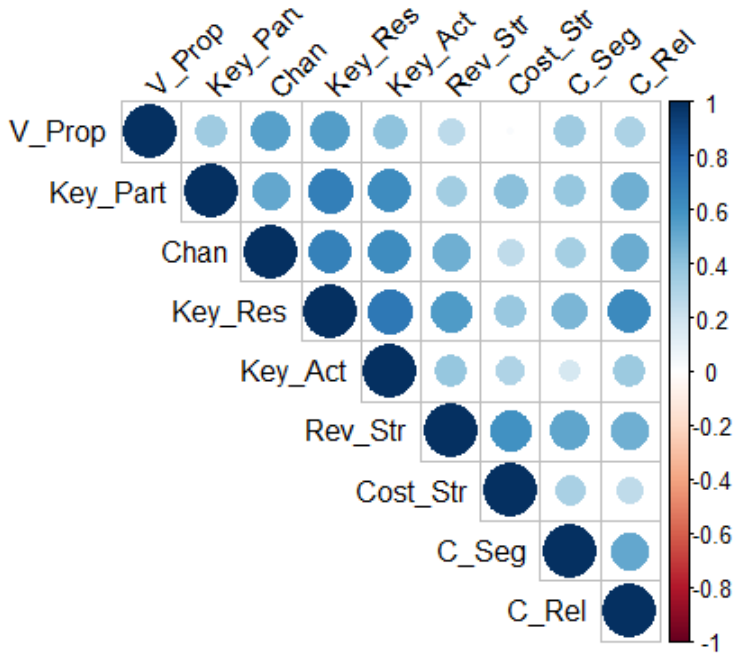

Figure 1.7: Correlation between the variables for the managers.

From all of the three figures (Figure 1.5, Figure 1.6, Figure 1.7), what we understand is that there is a strong correlation between the 9 variables. For this reason, Principal Components Analysis seems indispensable, so that uncorrelated variables would be created.

Continuing, we look for outliers. Firstly, let us explain how we define the outliers: Suppose we have a sample of observations. We first determine the first quartile (Q_1) and the third quartile (Q_3) and the *inter–quartile range* ($IQR = Q_3 - Q_1$) based on the values of the sample observations. Now the values outside the $[(Q_1 - 1.5 * IQR), (Q_3 + 1.5 * IQR)]$ are considered as *outliers*. The values outside the range $[(Q_1 - 3 * IQR), (Q_3 + 3 * IQR)]$ are known as *extreme outliers* and the values outside the range $[(Q_1 - 1.5 * IQR), (Q_3 + 1.5 * IQR)]$ but inside the range $[(Q_1 - 3 * IQR), (Q_3 + 3 * IQR)]$ are called *mild outliers*. This procedure is equivalent to the *boxplot* construction. In order to detect the outliers, we use the R–packages "grDevices" and the "EnvStats", where the latter enables us to do a Rosner's Test. Via the $boxplot.stats(x)\$out$ function of the "grDevices" package we get the outliers from the boxplot. Moving on, we examine which of these possible outliers are truly outliers, according to *Rosner's Test*. So, following the above procedure in R for each of the 9 variables separately on the entrepreneurs and managers, we get that:

In the entrepreneurs' dataset there are 4 outliers.

- For the variable "Channels", there is one outlier in the 102 position with value 18.
- For the variable "Revenue Streams", there is one outlier in the 95 position with value 17.
- For the variable "Key Partners", there are two outliers: one in the 87 position with value 16 and another one in the 97 position with value 17.

In the managers' dataset there are 4 outliers, too:

- For the variable "Customer Relationships", there is one outlier in the 12 position with value 21.

- For the variable "Revenue Streams", there is one outlier in the 54 position with value 6.
- For the variable "Key Resources", there is one outlier in the 17 position with value 12.
- For the variable "Key Activities", there is one outlier in the 17 position with value 12.

We cite the R code for the case where we examine the outliers for the entrepreneurs on the "Channel" variable. The procedure is exactly the same for the rest of them.

```
> entrepreneurs$Chan[which(entrepreneurs$Chan %in% boxplot.stats(entrepreneurs$Chan)$out)]
[1] 23 18
> rosnerTest(entrepreneurs$Chan, k = 4, warn = F)

Results of Outlier Test
-------------------------

Test Method:                     Rosner's Test for Outliers

Hypothesized Distribution:       Normal

Data:                            entrepreneurs$Chan

Sample Size:                     108

Test Statistics:                 R.1 = 3.566172
R.2 = 2.639368
R.3 = 2.501242
R.4 = 2.592180

Test Statistic Parameter:        k = 4

Alternative Hypothesis:          Up to 4 observations are not
from the same Distribution.

Type I Error:                    5%

Number of Outliers Detected:     1

i   Mean.i     SD.i Value Obs.Num    R.i+1 lambda.i+1 Outlier
1 0 34.03704 4.496989    18     102 3.566172   3.410133    TRUE
2 1 34.18692 4.238482    23      62 2.639368   3.407006   FALSE
3 2 34.29245 4.114937    24      44 2.501242   3.403844   FALSE
4 3 34.39048 4.008393    24      81 2.592180   3.400645   FALSE
```

Then, dealing with the outliers, we chose not to omit these observations, but replace them with the median in each case. This is because median is robust to outliers, contrary to mean and in this way we do not risk to lose information. (At this point, we needed "naniar" and "imputeTS" packages from R). The R code is as follows:

- For entrepreneurs:

```
> entrepreneurs[102, 3] <- NA
> entrepreneurs[95, 5] <- NA
> entrepreneurs[c(87, 97), 8] <- NA
> sum(is.na(entrepreneurs))
```

```
[1] 4
> entrepreneurs <- na.mean(entrepreneurs,option = "median")
```

- For managers:

```
> managers[12, 4] <- NA
> managers[54, 5] <- NA
> managers[17,6] <- NA
> managers[17, 7] <- NA
> sum(is.na(managers))
[1] 4
> managers <- na.mean(managers,option = "median")
```

PCA Application After this procedure, we go on applying Principal Components Analysis with the contribution of the R–package "FactoMineR" for our analysis and "factoextra" package for the interpretation of PCA and visualization. Furthermore, the "psych" R–package was used.

Applying PCA to entrepreneurs dataset

[13, 6]

The code is as follows:

```
> pca1 <- PCA(entrepreneurs, scale.unit = T, graph = F)
```

After applying PCA, we can extract information for the data.

- **Variances of the principal components**
  As we mentioned in Section 1.3, eigenvalues of the correlation matrix correspond to variances of the principal components. So, computing the eigenvalues in R as it is shown below,

```
> eig1 <- get_eigenvalue(pca1)
```

we get the results:

```
> eig1
eigenvalue variance.percent cumulative.variance.percent
Dim.1  4.6168831        51.298701                    51.29870
Dim.2  1.1683793        12.981992                    64.28069
Dim.3  0.6694271         7.438079                    71.71877
Dim.4  0.6383342         7.092602                    78.81137
Dim.5  0.5464243         6.071381                    84.88275
Dim.6  0.5059367         5.621519                    90.50427
Dim.7  0.3384333         3.760370                    94.26464
Dim.8  0.2649976         2.944417                    97.20906
Dim.9  0.2511846         2.790940                   100.00000
```

What we see from these results is that the first principal component explains approximately 51.3% of the variance in the data, the second one explains almost 13% and so on. Together, the first two principal components explain nearly the 64.3% of the data, which is an acceptably large percentage of the variation. We note here that after the third principal component, a significant improvement in this proportion is not achieved. This, will be discussed better when we will present the scree plot.

Another criterion for choosing which principal components to retain, is to check which PCs have an eigenvalue greater than 1. Such a thing indicates that this PC accounts for more variance than accounted by one of the original variables in standardized data. Thus, in our case we see that only the first two principal components are greater than one, whereas the rest ones are quite smaller than one, so it is probable that keeping the first two principal components is an appropriate choice.

An alternative method to determine the number of principal components is to look at the scree plot. We get the scree plot (Figure 1.8), which is the plot of the percent of variance accounted for by each principal component, through the command:

```
> fviz_eig(pca1, addlabels = TRUE, ylim = c(0, 60), ggtheme = theme_bw())
```

Eyeballing the scree plot, and looking for a point at which the proportion of variance explained by each subsequent principal component drops off, make us choose the smallest number of principal components that are required in order to explain a sizable amount of the variation in the data. This is often referred to as an *elbow* in the scree plot. Inspecting Figure 1.8 we conclude that a fair amount of variance is explained by the first two principal components, and that there is an elbow after the second component. After all, the third principal component explains less than 7.5% of the variance in the data and so is essentially worthless, and also beyond the second eigenvalue, the remaining eigenvalues are all relatively small and of comparable size.

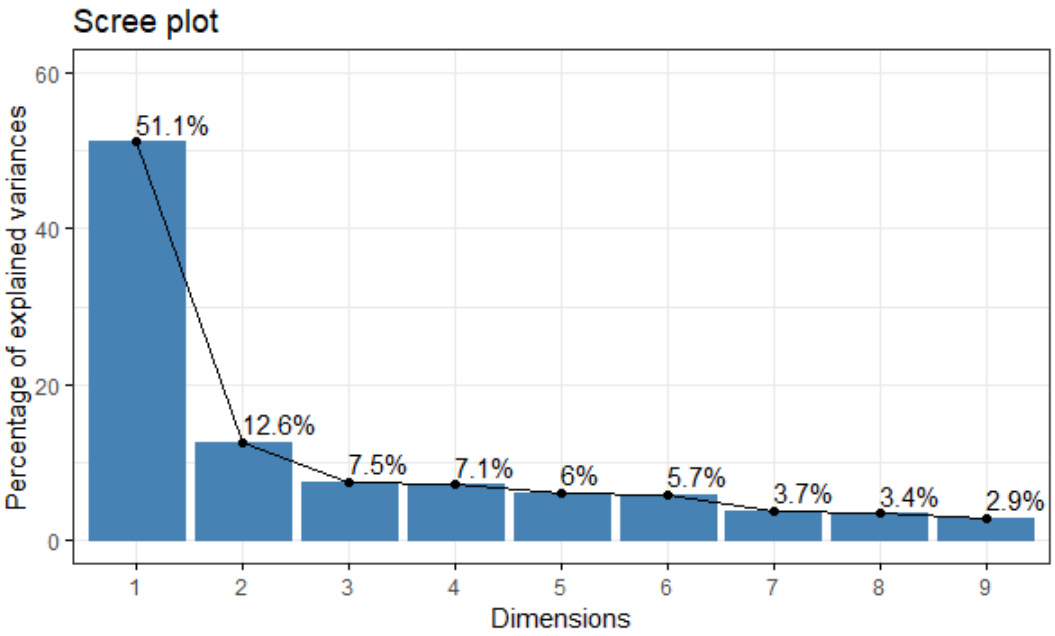

Figure 1.8: Scree plot: entrepreneurs data.

- **Variables**
  Using the following commands, we get the coordinates/loadings of the variables:

```
> variable<- get_pca_var(pca1)
> variable$coord
Dim.1       Dim.2       Dim.3       Dim.4       Dim.5
C_Seg    0.7773302  0.25127380 -0.24748822  0.25823439 -0.10234605
V_Prop   0.6516228  0.49599232  0.14076101  0.03509079 -0.49450972
Chan     0.6301532  0.49294574 -0.27779827  0.16826052  0.33363642
C_Rel    0.7209100  0.26158534  0.18473314 -0.25899954  0.31287556
Rev_Str  0.7628346 -0.35378872 -0.19599877 -0.28723757  0.06635240
Key_Res  0.7850108 -0.37822316 -0.06557736 -0.10890732 -0.24022851
Key_Act  0.7991856 -0.07526561 -0.03077001 -0.33669279 -0.02246202
Key_Part 0.6685162 -0.10766647  0.64968455  0.20076168  0.12639445
Cost_Str 0.6224581 -0.51218534 -0.10635839  0.47643669  0.06010682
```

As we are interested in keeping only the first two principal components, we look at Dim.1 and Dim.2. Because of the difficulty we encountered to interpret the two components, *varimax rotation* is a usual method in order to improve interpretability.

Let us move on, then, by applying varimax rotation in our data. For this, we needed "psych" package from R. The code follows:

```
> pca_rotated <- principal(entrepreneurs, rotate="varimax", nfactors=2, scores=TRUE)
> pca_rotated
Principal Components Analysis
Call: principal(r = entrepreneurs, nfactors = 2, rotate = "varimax",
scores = TRUE)
Standardized loadings (pattern matrix) based upon correlation matrix
RC1  RC2   h2   u2 com
C_Seg    0.40 0.71 0.67 0.33 1.6
V_Prop   0.14 0.81 0.67 0.33 1.1
Chan     0.12 0.79 0.64 0.36 1.0
C_Rel    0.35 0.68 0.59 0.41 1.5
Rev_Str  0.80 0.26 0.71 0.29 1.2
Key_Res  0.83 0.26 0.76 0.24 1.2
Key_Act  0.64 0.49 0.64 0.36 1.9
Key_Part 0.56 0.38 0.46 0.54 1.8
Cost_Str 0.80 0.05 0.65 0.35 1.0

RC1  RC2
SS loadings           3.01 2.78
Proportion Var        0.33 0.31
Cumulative Var        0.33 0.64
Proportion Explained  0.52 0.48
Cumulative Proportion 0.52 1.00

Mean item complexity =  1.4
Test of the hypothesis that 2 components are sufficient.

The root mean square of the residuals (RMSR) is  0.08
with the empirical chi square  47.18  with prob <  0.00034

Fit based upon off diagonal values = 0.97
```

In Table 1.3 the coordinates of the unrotated and rotated principal components 1 and 2 are presented. It is obvious that in the rotated case, we can come to some conclusions much easier than in the unrotated case.

	Unrotated		**Rotated**	
Variable	PC1	PC2	PC1	PC2
Customer Segments	0.77733	0 .25127	0. 4	0.71
Value Propositions	0.65162	0 .49599	0.14	0.81
Channels	0.63015	0 .49295	0.12	0.79
Customer Relationships	0.72091	0 .26159	0.35	0.68
Revenue Streams	0.76283	-0. 3538	0. 8	0.26
Key Resources	0.78501	-0. 3782	0.83	0.26
Key activities	0.79919	-0. 0753	0.64	0.49
Key Partners	0.66852	-0. 1077	0.56	0.38
Cost Structure	0.62246	-0. 5122	0. 8	0.05

Table 1.3: Coefficients of components 1 and 2: entrepreneurs data

More specifically, rotated PC1 gave relatively high weights to the variables Revenue streams, Key resources, Key Activities, Key Partners and Cost Structure. On the other hand, PC2 had some high weights associated with Customer Segments, Value Propositions, Channels, Customer Relationships and relatively high weight to Key Activities.

It is noticeable that all of the 9 variables seem to play an important role to our analysis, and each of them is significant to one and only one of the two Principal Components, except for the variable Key Activities, which is highly–weighted in both PCs.
Inspecting the content of these variables, we note that the first Principal Component has to do with the company's finances and operations, while the second Principal Component is related to serving products to new and existing customers.

In other words, through the rotation method, we managed to make some inference about our data, which was impossible if we looked our initial results on PCs.

Now, we will carry on examining our data via some graphs. During this procedure, we will compare our conclusions with the ones described above, through the rotation method.

First of all, let us plot the variables. The variable plot (Figure 1.9) shows the relationships between the variables. Since positively correlated variables are grouped together, it seems that Channels, Value Propositions, Customer Segments and Customer Relationships consist one group and Cost Structure, Key Resources, Revenue Streams, Key Partners and Key Activities consist another one.

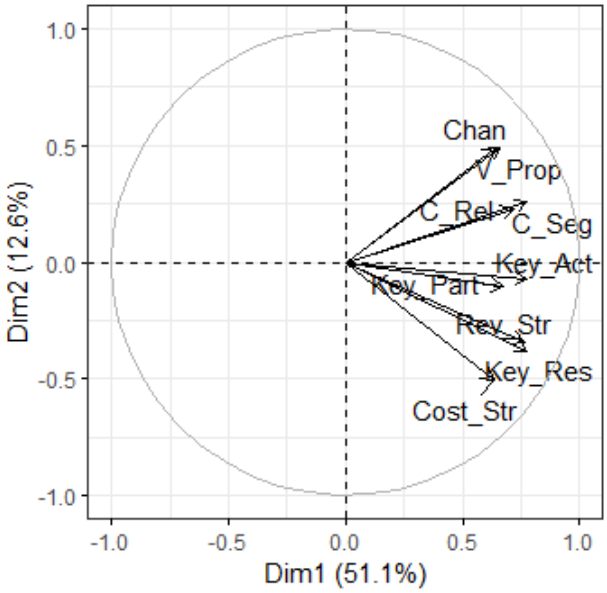

Figure 1.9: Variable Plot: entrepreneurs data.

The distance between variables and the origin measures the quality of the variables on the factor map ($var.cos2 = var.coord * var.coord$). Variables that are away from the origin are well represented on the factor map. Hence, Channels, Value Propositions, Customer Segments, Cost Structure, Key Resources and Revenue Streams seem to be well represented, contrary to the rest of them.

Next, we will examine the quality of representation of the variables (cos2) on the factor map. In Figure 1.10, we visualize the cos2 of the variables in the first five dimensions. In Figure 1.11, an alternative way is applied, creating a bar plot of variables cos2.
Generally, high cos2 indicates a good representation of the variable on the principal component. In this case the variable is positioned close to the circumference of the correlation circle (Figure 1.9). On the other hand, a low cos2 indicates that the variable is not perfectly represented by the PCs. In this case the variable is close to the center of the circle.
Observing Figure 1.10 and Figure 1.11, our conclusions coincide with those which we came to, in Figure 1.9.

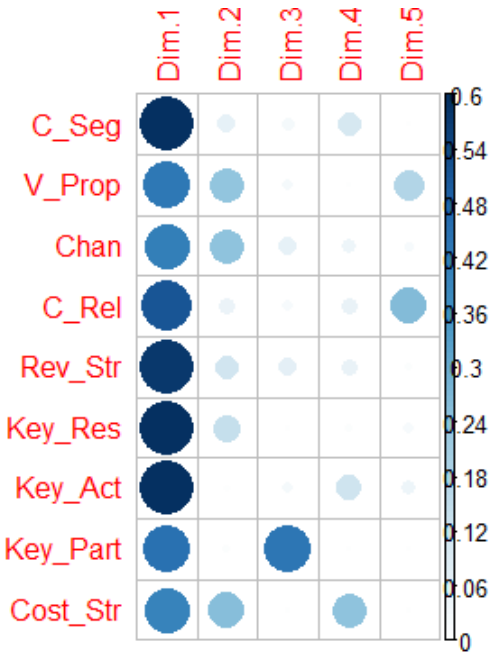

Figure 1.10: Cos2 of the variables: entrepreneurs data

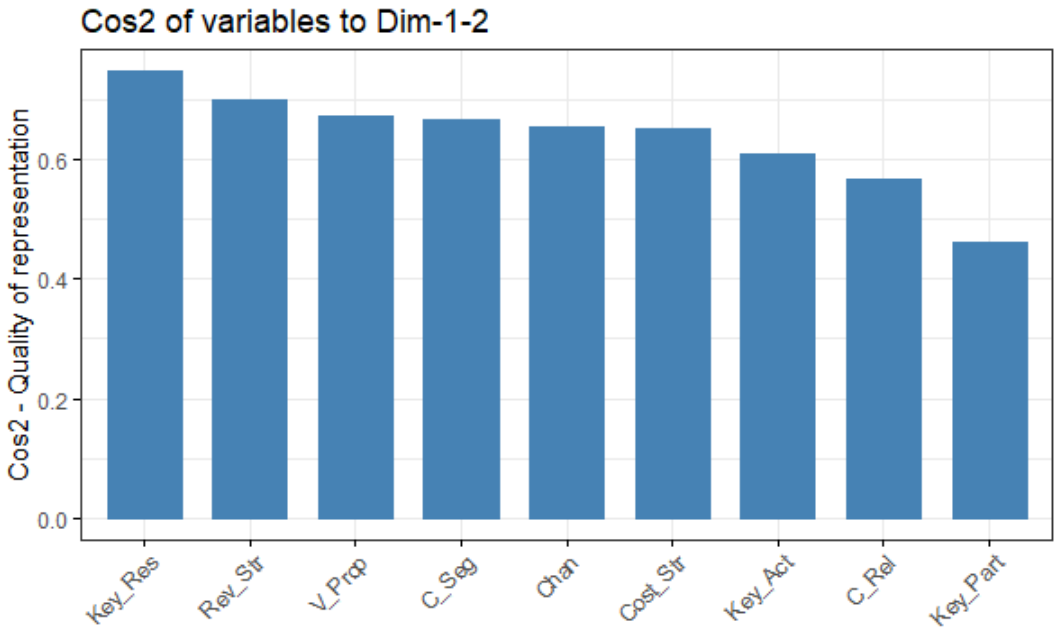

Figure 1.11: Barplot of the cos2 of the variables: entrepreneurs data

For a more clear view, Figure 1.12 helps us understand better all the above. For example, Key Resources and Revenue Streams are high, so are important to be included in the representation of the components, whereas Key Partners' cos2 is low, which means that its quality of representation on the factor map is less important.

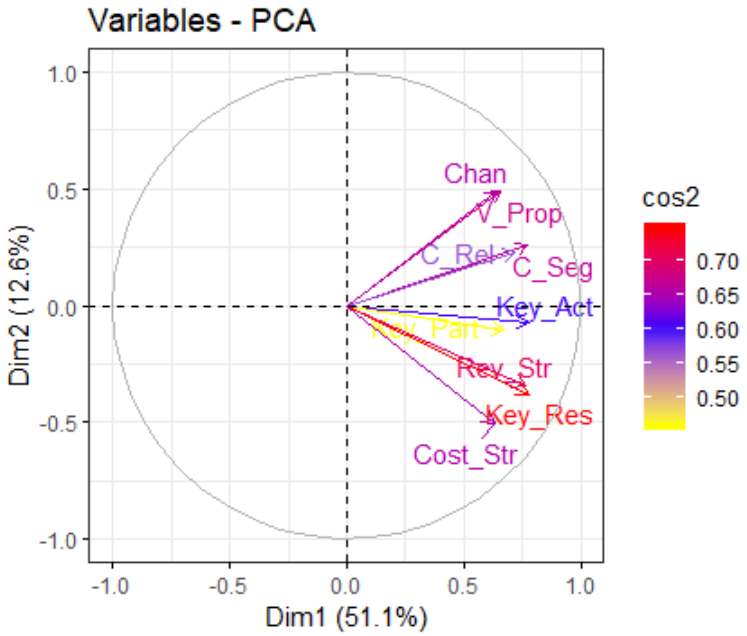

Figure 1.12: Variable Plot over the cos2: entrepreneurs data

Then, we analyse the variables based on their contribution in each dimension. Figure 1.13 presents these contributions.

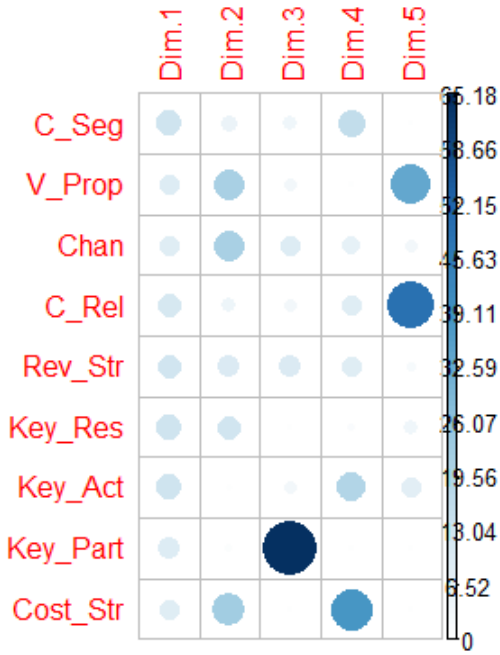

Figure 1.13: Contribution of the variables: entrepreneurs data

Unfortunately this figure does not help us understand the importancy of the variables. For this reason, Figure 1.14 , Figure 1.15 and Figure 1.16 are made, which help us in our inference.

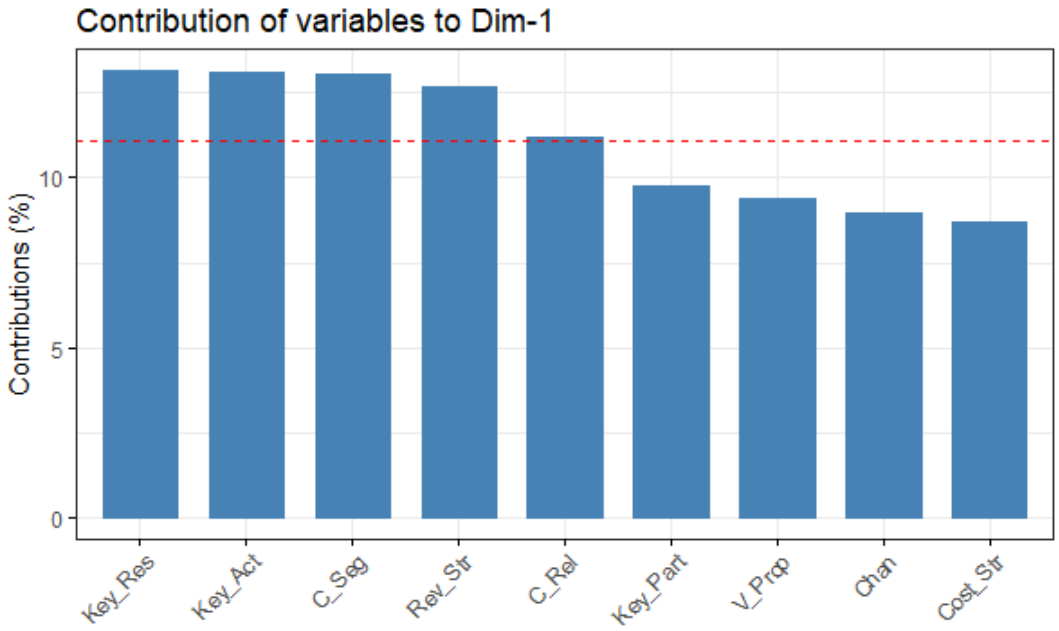

Figure 1.14: Barplot of the contribution of the variables on the first dimension: entrepreneurs data

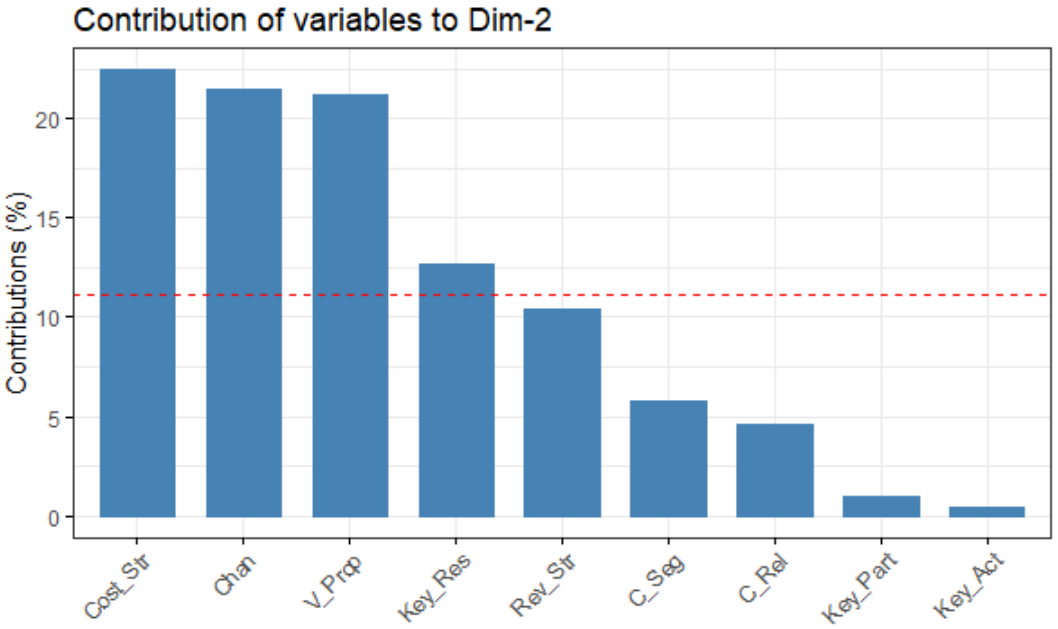

Figure 1.15: Barplot of the contribution of the variables on the second dimension: entrepreneurs data

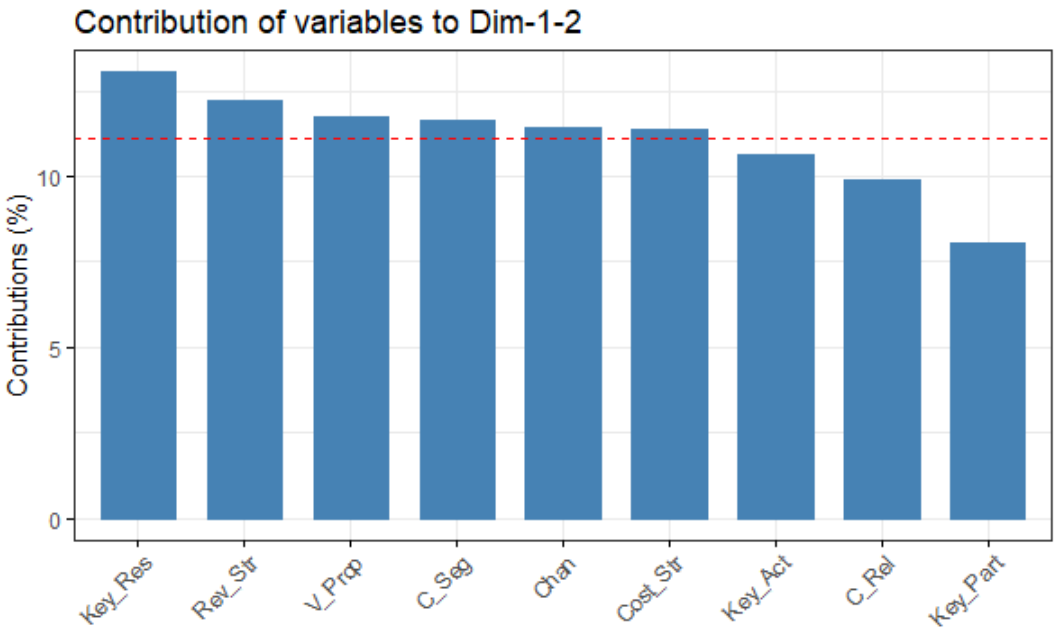

Figure 1.16: Barplot of the contribution of the variables on the first 2 dimensions: entrepreneurs data

The red dashed line on Figure 1.14, Figure 1.15 and Figure 1.16 indicates the expected average contribution. If the contribution of the variables were uniform, the expected value would be 10%. For a given component, a variable with a contribution larger than this cutoff could be considered as important in contributing to the component.
In other words, from Figure 1.14, we can see that the variables contributing most to the first component are: Key Resources, Key Activities, Customer Segments, Revenue Streams and Customer Relationships. Figure 1.15 shows us that the variables contributing most in the second principal component are: Cost Structure, Channels, Value Propositions, Key Resources and Revenue Streams. Summing up, we conclude that all the variables except for Key Activities, Customer Relationships and Key Partners contribute to both Principal Components.

Now, highlighting the variables contributing most on the correlation plot we get Figure 1.17, which gives the same results as described above. Namely, Key Resources and Revenue Streams are contributing highly, so they are important in explaining the variability in the data, while Key Partners is not contributing, which means that we may remove it, so that the overall analysis would get simpler. We have to notice here that the same variables were significant regarding the cos2.

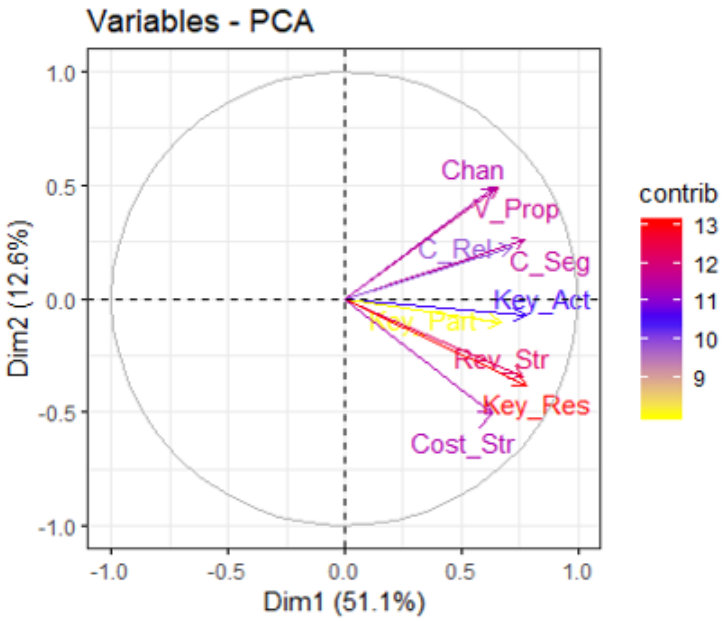

Figure 1.17: Variable Plot over the contribution: entrepreneurs data

- **Individuals**

  Using the following commands, we get the coordinates of the first individuals, just to take a look of their structure:

```
> ind <- get_pca_ind(pca1)
> head(ind$coord)
Dim.1      Dim.2       Dim.3       Dim.4      Dim.5
1 -0.9318841 -0.4754595 -0.36163089 -0.43637967 -0.3682712
2 -2.3371652  0.2614375 -0.41844837  0.06540382  0.3640769
3 -2.2506998 -1.4633712 -0.39664180  0.06734092 -0.3083011
4  0.2610744  0.3810159  0.89492205 -0.67707924  0.5147132
5 -1.1131951 -1.6667657  0.02663858 -1.02062965  0.6028385
6 -1.2583783 -1.1671340 -1.03782881  0.71719053  1.0624936
```

  Plotting the individuals in the two first Principal Components, we get Figure 1.18.

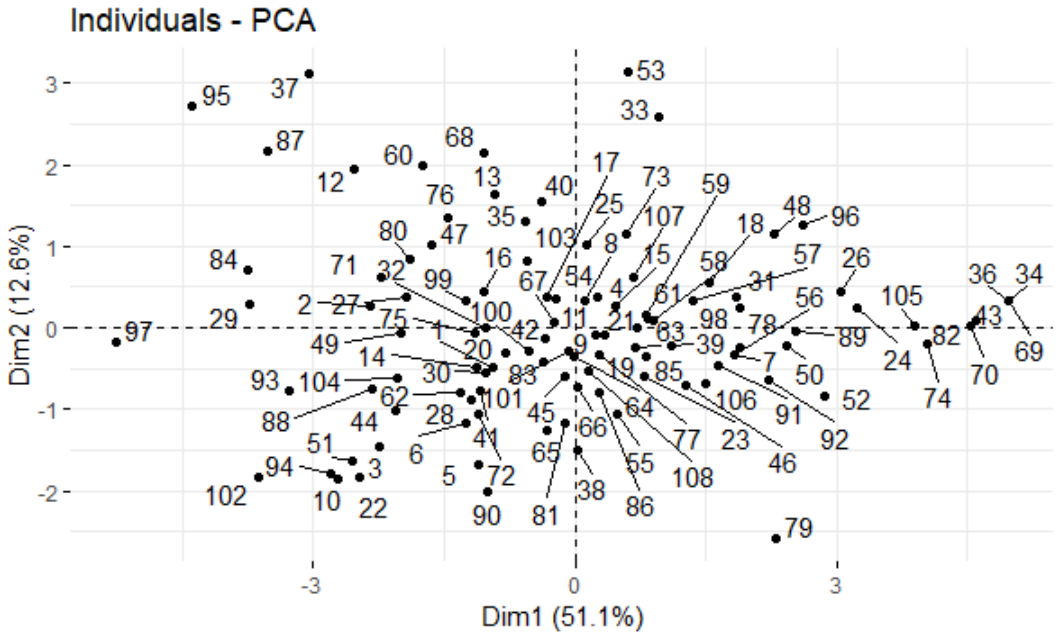

Figure 1.18: Individuals Plot: entrepreneurs data

In a similar way with this of the variables, we can plot the individuals with respect to their quality of representation, cos2 (see Figure 1.19).

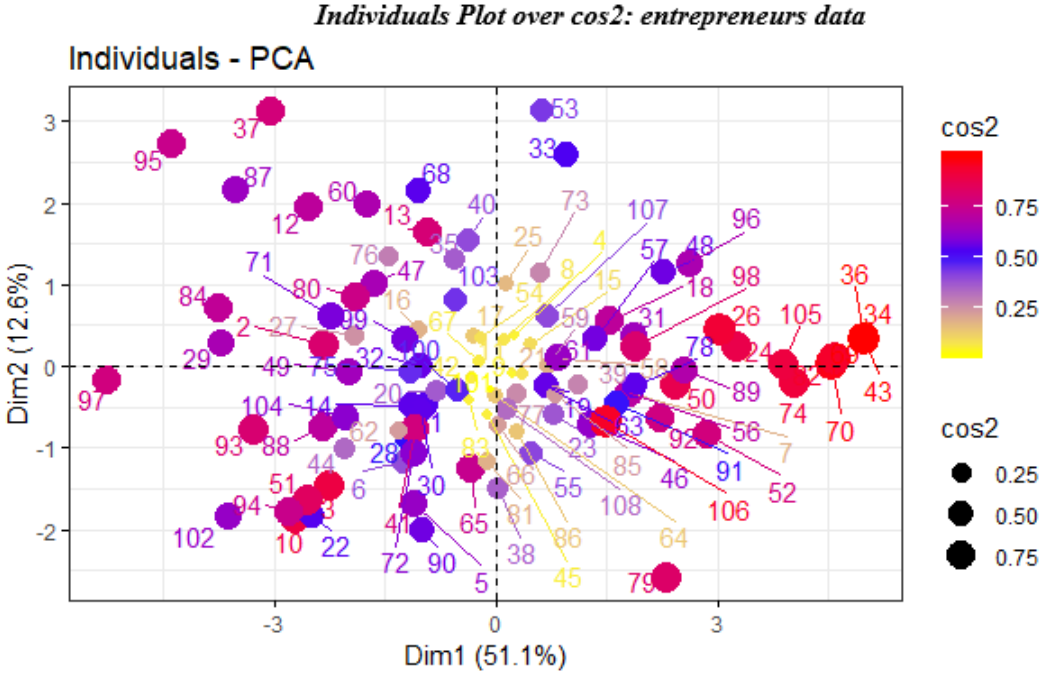

Figure 1.19: Individuals Plot: entrepreneurs data

In Figure 1.19, the individuals, are depicted according to their cos2. The higher the cos2, the

bigger is the point and its color ranges from red (when we have a high cos2) to yellow (when we have low cos2), as is shown it Figure 1.19.

Lastly, we represent both the principal component scores and the loading vectors in a single *biplot* display (Figure 1.20). This figure shows us the "behaviour" of each of the entrepreneurs regarding the first Principal Component, which has to do with the company's finances and operations, and the second Principal Component, which is related to serving products to new and existing customers.

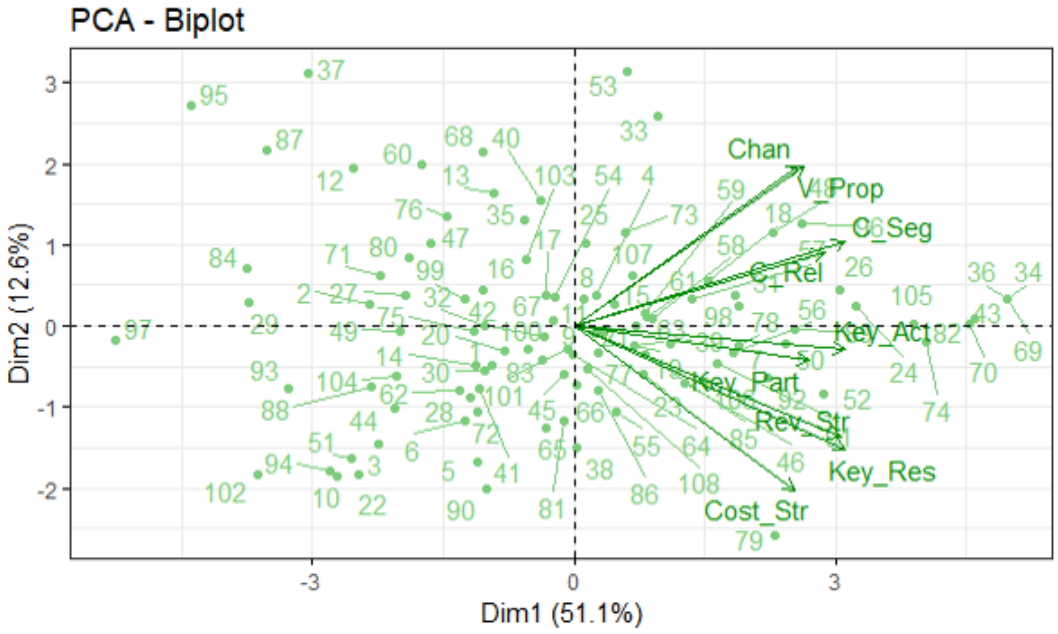

Figure 1.20: Biplot: entrepreneurs data

Applying PCA to managers dataset

The code is as follows:

```
> pca2 <- PCA(managers,scale.unit = T, graph = F)
```

After applying PCA to managers dataset, we keep on with the same procedure that we followed in entrepreneurs dataset.

- **Variances of the principal components**

  The eigenvalues are given in R as it is shown below,

  ```
  > eig2 <- get_eigenvalue(pca2)
  ```

And we get the results:

```
eigenvalue variance.percent cumulative.variance.percent
Dim.1  4.4353217        49.281352                    49.28135
Dim.2  1.3667369        15.185965                    64.46732
Dim.3  0.9059678        10.066309                    74.53363
Dim.4  0.7113650         7.904055                    82.43768
Dim.5  0.5167366         5.741518                    88.17920
Dim.6  0.4107330         4.563700                    92.74290
Dim.7  0.2984126         3.315695                    96.05859
Dim.8  0.1904989         2.116655                    98.17525
Dim.9  0.1642275         1.824751                   100.00000
```

Here, we see that the first principal component explains about 49.3% of the variance in the data, the second one explains 15.2% and so on. Together, the first two principal components explain nearly 64.5% of the data, which is an acceptably large percentage of the variation.
Now looking at the eigenvalues, we see that, indeed, the eigenvalues which correspond to the first two principal components are the only ones that are greater than 1. However, we need to mention that the third eigenvalue is very close to 1 (0.91), which raises doubts about the number of components that is appropriate to be used. For reasons of interpretability and in order to compare the results of entrepreneurs and managers, we choose to keep the first two Principal Components.

Finally, the scree plot (Figure 1.21) verifies the uncertainty about the number of components to be retained, which was commented previously. In this scree plot, it is not so clear which is the point in which we determine an elbow. Additionally, we see that the third principal component explains more than 10% of the variation in the data, not so trivial percentage to omit. Nevertheless, as we analysed above, we will keep the first two Principal Components in further investigation of the data.

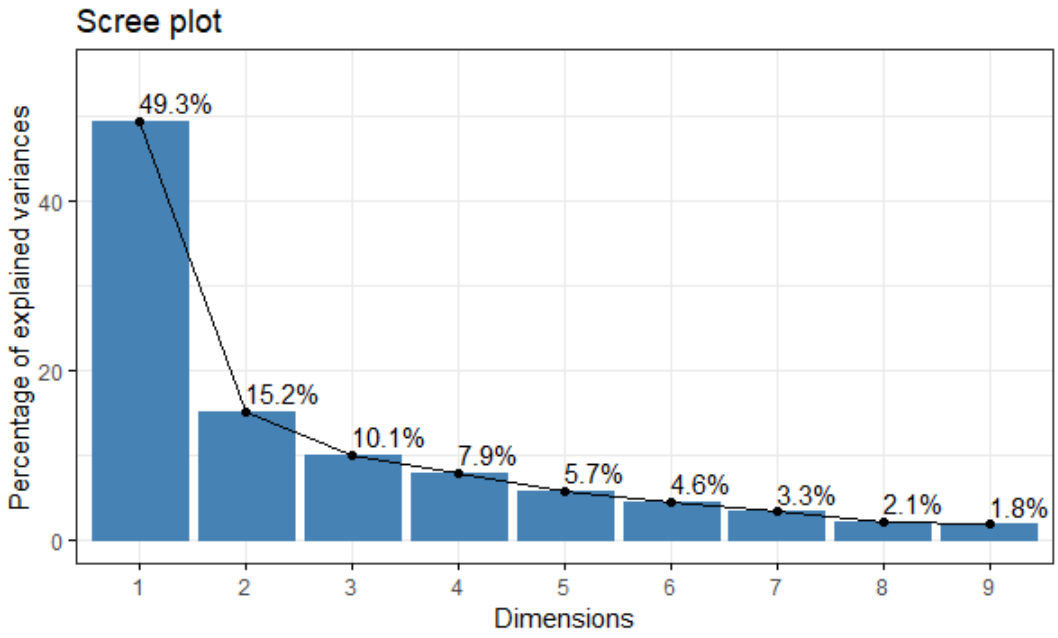

Figure 1.21: Scree plot: managers data

- **Variables**
  The coordinates of the variables are given below:

```
            Dim.1      Dim.2        Dim.3       Dim.4       Dim.5
C_Seg     0.6115452  0.3723525  0.532820749 -0.04772280  0.26713258
V_Prop    0.6245799 -0.4173899  0.368690809  0.39357559  0.15934628
Chan      0.7788908 -0.3821339  0.002031817  0.21071742 -0.08640856
C_Rel     0.6667110  0.2264419  0.302099065 -0.47581380 -0.31476516
Rev_Str   0.6851754  0.4956123 -0.048918481  0.32416090 -0.30863213
Key_Res   0.9034951 -0.1283870 -0.054360635 -0.06435301 -0.06422116
Key_Act   0.7336309 -0.3622181 -0.402388262 -0.06511751 -0.14646275
Key_Part  0.7640635 -0.1143080 -0.256413545 -0.35821682  0.38528938
Cost_Str  0.4607424  0.6709626 -0.402314167  0.20397049  0.21015382
```

Now, we will apply varimax rotation, for the same reasons we discussed in enterpreneurs' case.

```
 Principal Components Analysis
Call: principal(r = managers, nfactors = 2, rotate = "varimax", scores = TRUE)
Standardized loadings (pattern matrix) based upon correlation matrix
RC1  RC2   h2   u2 com
C_Seg     0.26 0.67 0.51 0.49 1.3
V_Prop    0.75 0.05 0.56 0.44 1.0
Chan      0.85 0.17 0.75 0.25 1.1
C_Rel     0.39 0.58 0.50 0.50 1.8
Rev_Str   0.25 0.81 0.72 0.28 1.2
Key_Res   0.80 0.44 0.83 0.17 1.6
Key_Act   0.80 0.16 0.67 0.33 1.1
Key_Part  0.68 0.37 0.60 0.40 1.6
Cost_Str -0.04 0.81 0.66 0.34 1.0

RC1  RC2
SS loadings           3.31 2.49
Proportion Var        0.37 0.28
Cumulative Var        0.37 0.64
Proportion Explained  0.57 0.43
Cumulative Proportion 0.57 1.00

Mean item complexity =  1.3
Test of the hypothesis that 2 components are sufficient.

The root mean square of the residuals (RMSR) is  0.1
with the empirical chi square  42.67  with prob <  0.0014

Fit based upon off diagonal values = 0.95
```

In Table 1.4 the coordinates of the unrotated and rotated first Principal Components are presented.

	Unrotated		Rotated	
Variable	PC1	PC2	PC1	PC2
Customer Segments	0.6115452	0 .3723525	0 .26	0.67
Value Propositions	0.6245799	-0.4173899	0 .75	0.05
Channels	0.7788908	-0.3821339	0 .85	0.17
Customer Relationships	0.6667110	0 .2264419	0 .39	0.58
Revenue Streams	0.6851754	0 .4956123	0 .25	0.81
Key Resources	0.9034951	-0.1283870	0 .80	0.44
Key activities	0.7336309	-0.3622181	0 .80	0.16
Key Partners	0.7640635	-0.1143080	0 .68	0.37
Cost Structure	0.4607424	0 .6709626	-0.04	0.81

Table 1.4: Coefficients of components 1 and 2: managers data

As we know, the higher the loading of a variable, the more influence it has in the formation of the principal component score and vice versa. Thus, inspecting the Table 1.4, we understand that in the formation of PC1, influential are the following variables: Value Propositions, Channels, Key Resources, Key Activities and Key Partners. For PC2, influential variables are: Customer Segments, Revenue Streams and Cost Structure. Therefore, we observe that PC1 is associated with making products and serving them to existing customers, where on the other hand, PC2 seems to be associated with a firm's finances. In the managers' case, we see that Customer Relationships does not play an influential role.

Continuing with the graphical examination of our data we will check if this coincide with the above conclusions.

Firstly, we plot the variables. The variable plot (Figure 1.22) shows that Customer Segments, Revenue Streams, Cost Structure and Customer Relationships form one group and Value Propositions, Channels, Key Resources, Key Activities and Key Partners constitute another one.
Furthermore, it is observed that Channels, Value Propositions, Cost Structure, Key Resources and Revenue Streams seem to be well represented, in contrast to the rest of them.

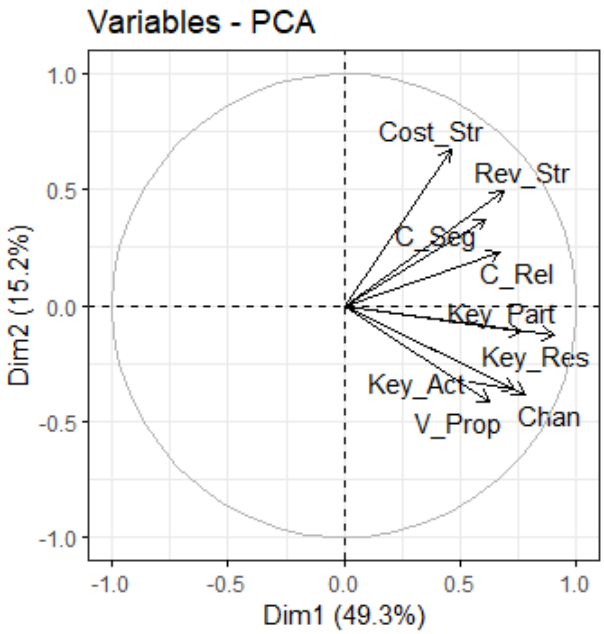

Figure 1.22: Variable Plot: managers data

Next, examining the quality of representation of the variables (cos2) on the factor map (Figure 1.23), we visualize the cos2 of the variables in all the dimensions. In Figure 1.24, an alternative way is applied, creating a bar plot of variables cos2.
Observing the Figure 1.23 and Figure 1.24, we come to the same conclusions as in variable plot (Figure 1.22).

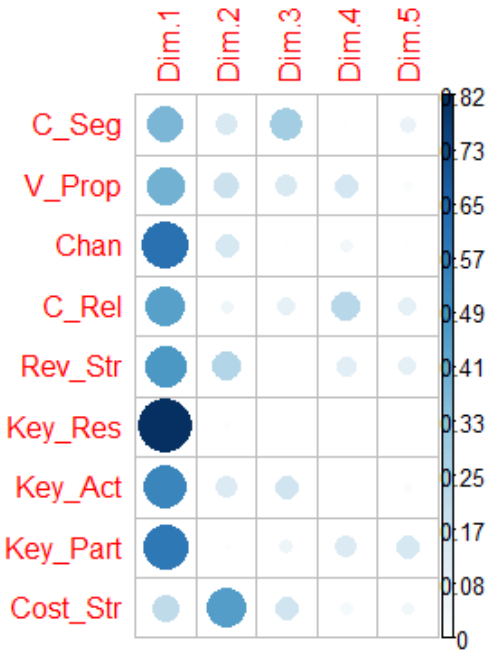

Figure 1.23: Cos2 of the variables: managers data

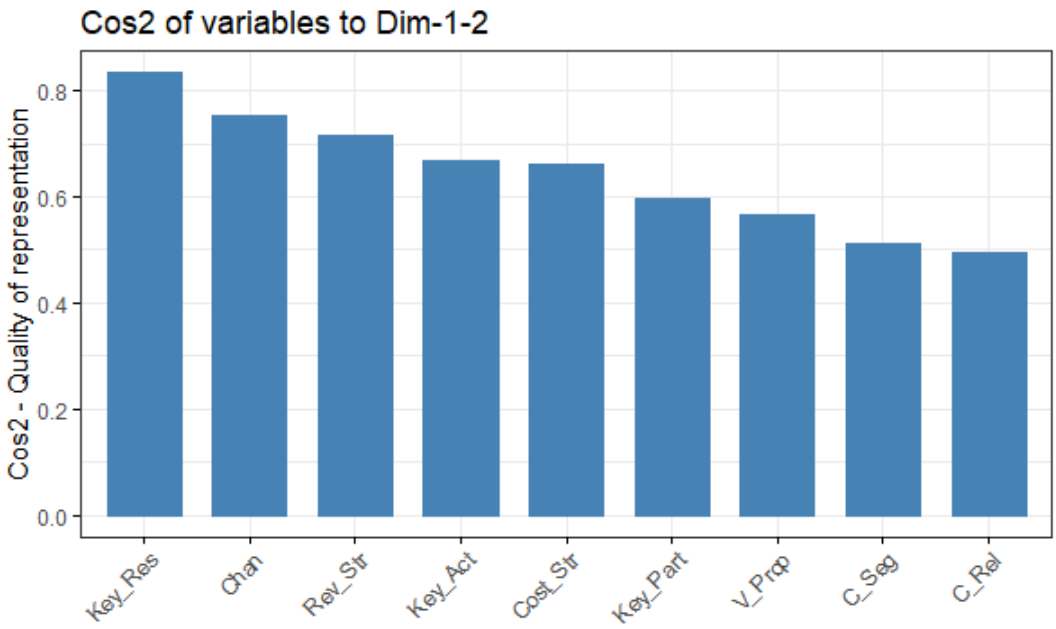

Figure 1.24: Barplot of the cos2 of the variables: managers data

Next, Figure 1.25, visualises in an alternative way all we just discussed. For example, Key Resources, Channels and Revenue Streams have a high quality of representation, so are important to be included in it, whereas Customer Relationships' cos2 is low, which means that it is not so important for the representation of the components in the factor map.

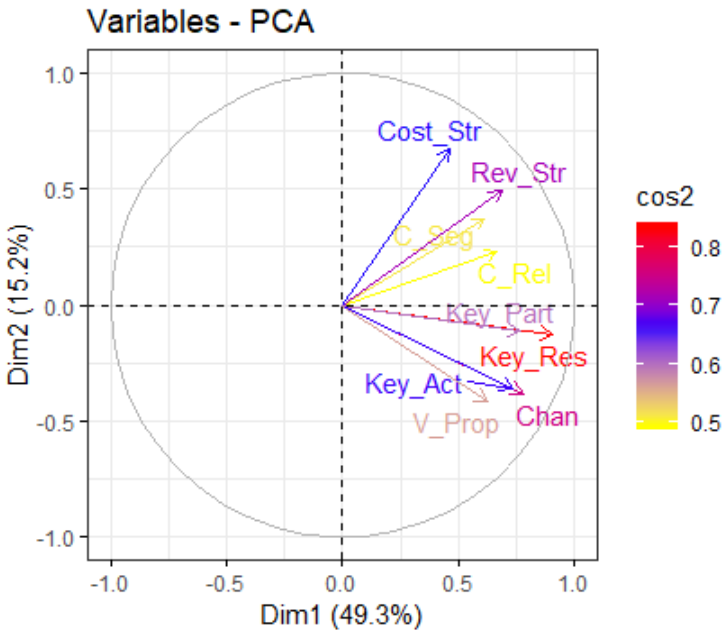

Figure 1.25: Variable Plot over cos2: managers data

We carry on, analysing the variables based on their contribution in each dimension.

Figure 1.26 presents these contributions, but it is not so informative contrary to Figure 1.27, Figure 1.28 and Figure 1.29, which are quite helpful for our analysis.

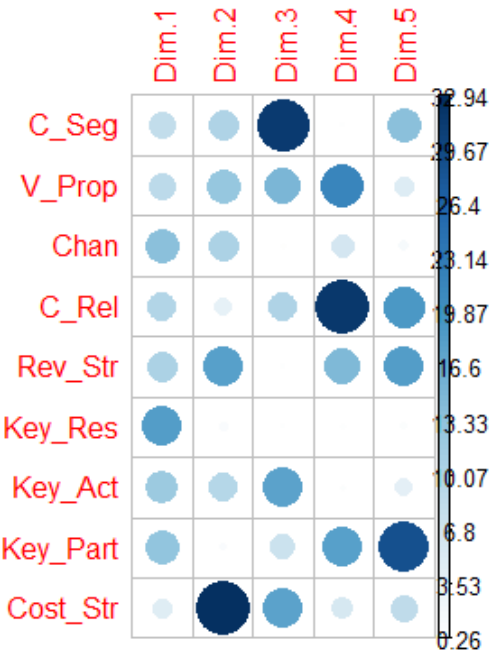

Figure 1.26: Contribution of the variables: managers data

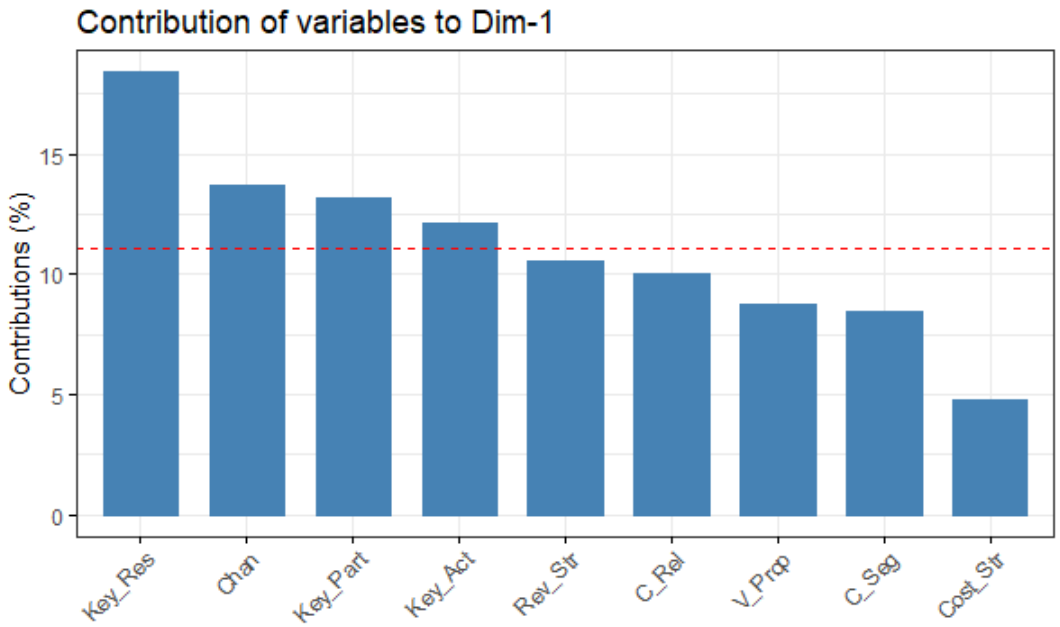

Figure 1.27: Barplot of the contribution of the variables on the first dimension: managers data

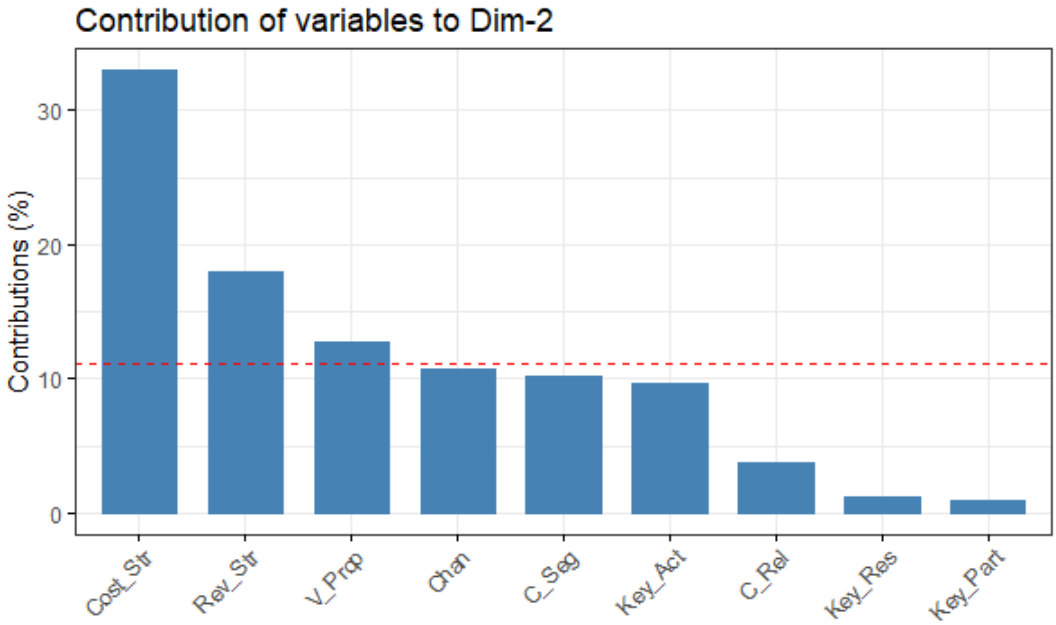

Figure 1.28: Barplot of the contribution of the variables on the second dimension: managers data

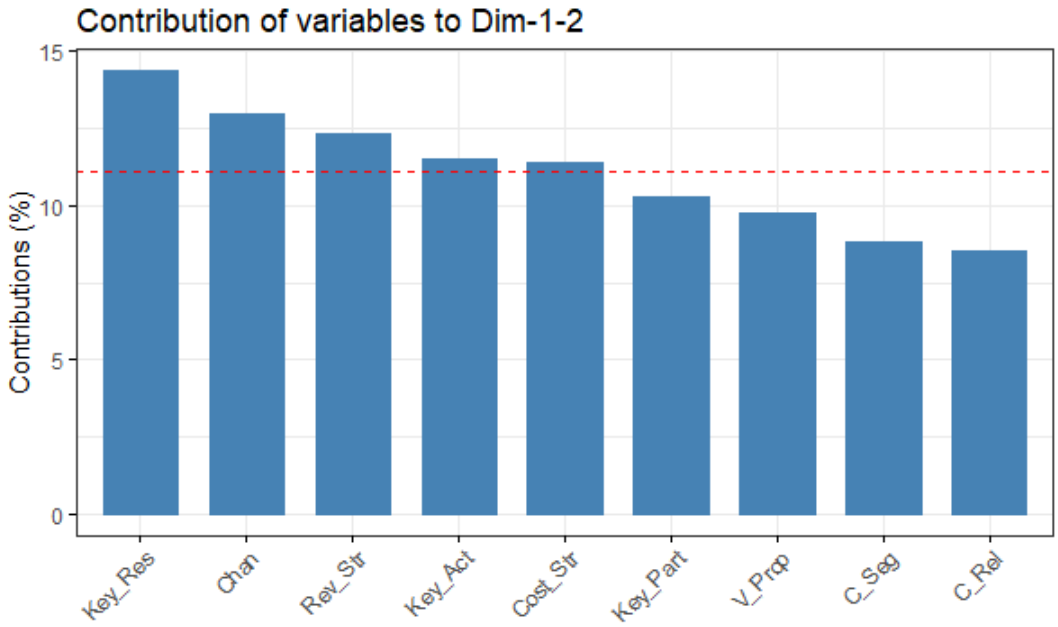

Figure 1.29: Barplot of the contribution of the variables on the first 2 dimensions: managers data

From Figure 1.27, we can see that the variables contributing most to the first component are: Key Resources, Channels, Key Partners and Key Activities. Figure 1.28 shows us that the variables contributing most in the second principal component are: Cost Structure, Revenue Streams and Value Propositions. For both Principal Components, we conclude that Key Resources, Channels, Revenue Streams, Key Activities and Cost Structure are contributing the most.

Alternatively, we graph the variables contributing most on the correlation plot (Figure 1.30). In the same way, we get that Key Resources, Channels and Revenue Streams are highly con-

tributing, so they are important in explaining the variability in the data, whereas Customer Relationships and Customer Segments are not contributing, which means that we may remove them.

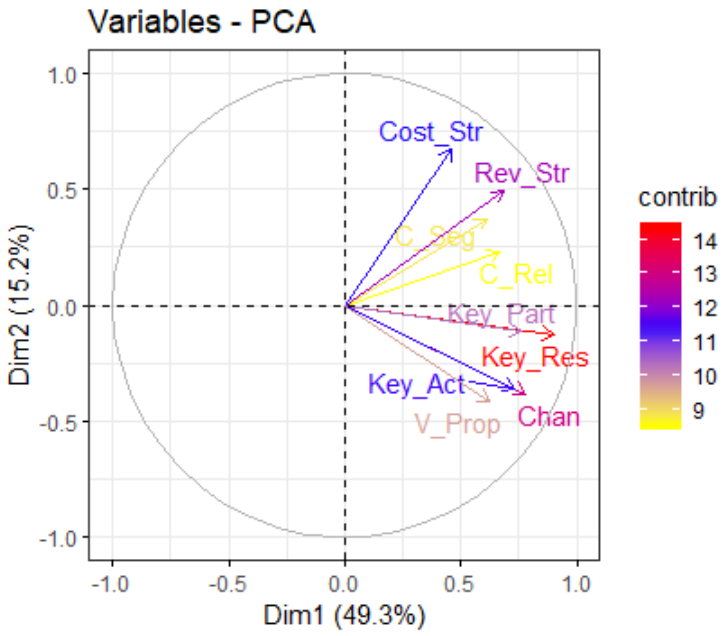

Figure 1.30: Variable Plot over contribution: managers data

- **Individuals**

Plotting the individuals in the first two Principal Components, we get Figure 1.31.

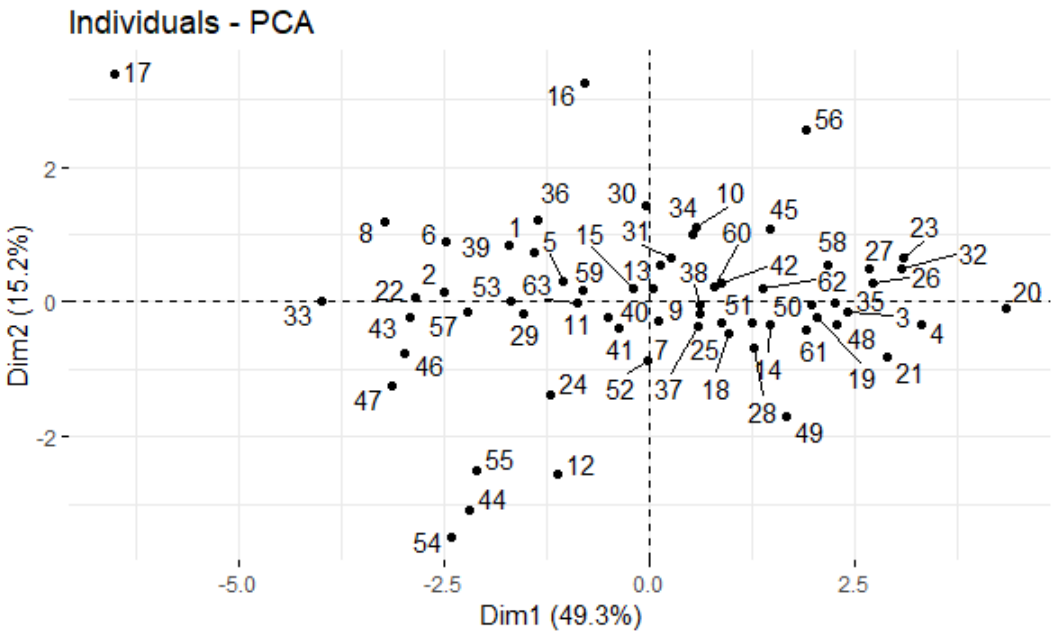

Figure 1.31: Individuals Plot: managers data

In a similar way with this of the variables, we can plot the individuals with respect to their quality of representation, cos2 (see Figure 1.32).

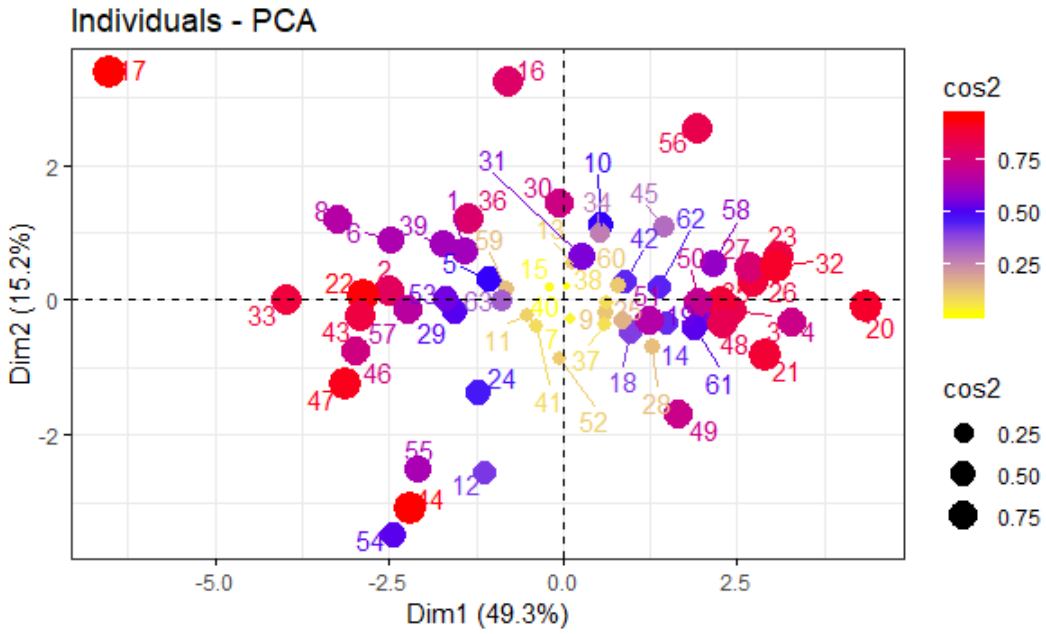

Figure 1.32: Individuals Plot over cos2: managers data

Lastly, Figure 1.33 shows the biplot, in which both scores and loadings are represented. More specifically, each of the managers appears in the graph according to his attitude towards making

products and serving them to existing customers (first Principal Component), and his attitude on firm's finances (second Principal Component).

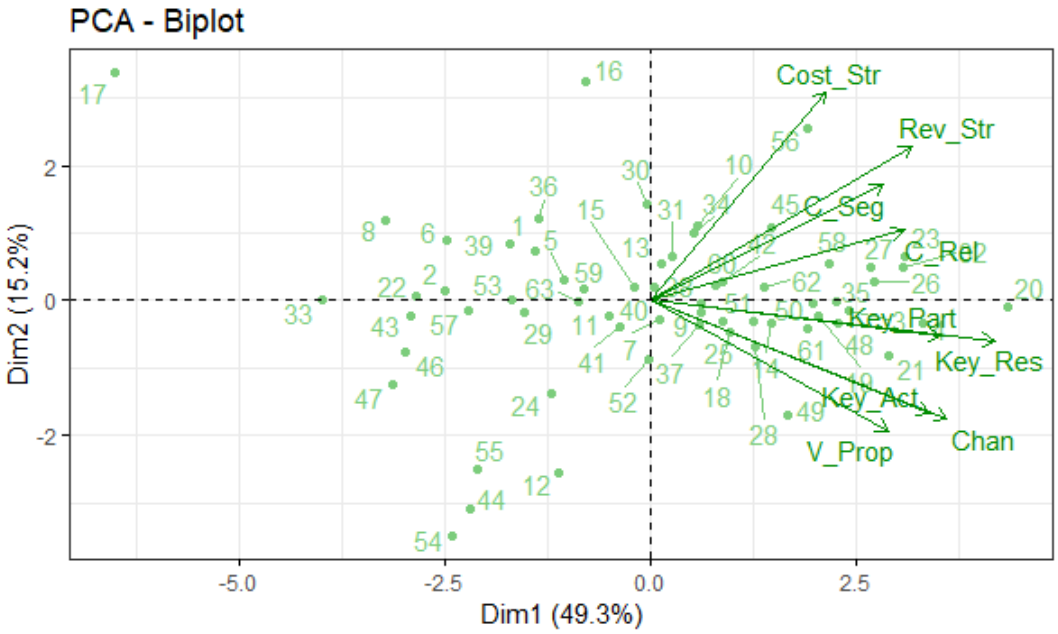

Figure 1.33: Biplot: managers data

Conclusions After all this discussion, it seems that a two–dimensional representation of Irish data is achievable. More specifically, we concluded that both entrepreneurs and managers may represent the nine BMC elements by two factors.

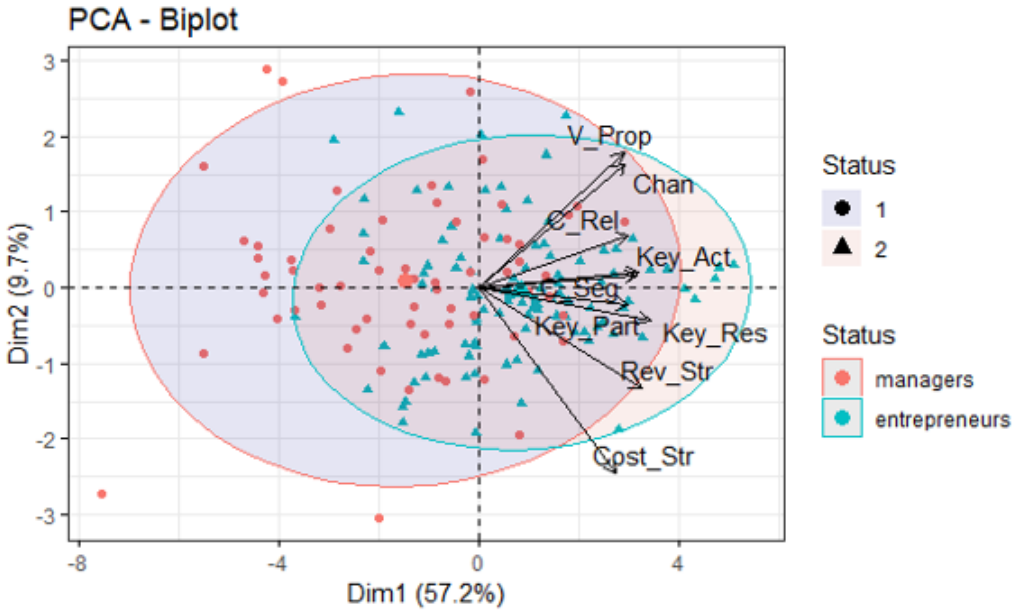

Figure 1.34: Biplot for both entrepreneurs and managers

However, the two groups represent the elements in a different way, since, as far as entrepreneurs are concerned, the first PC corresponds to Finance and Operations and the second one to Serving Products to New and Existing Customers, while regarding the managers, first PC has to do with Making Products and Serving them to Existing Customers and the second one refers to Costs and Revenues. This difference between the two groups is visible in Figure 1.34, where both entrepreneurs and managers are included but they are distinguished by different colors. [6]

1.5 Chicken Dataset

In this section, we will examine the modern application of Principal Components Analysis, in which the number of the observations is much smaller than the number of the variables. In other words, we have the case $n \ll p$.

1.5.1 Presentation

Due to the outstanding technological development in our days, and more specifically in the field of Biology and Medicine, huge amounts of data are being generated and the consequential need to analyse and explore such datasets is more than essential. For this purpose, the application of multivariate projection techniques to reduce dimensionality is vital [25].
Such a dataset is the one we will present in this subsection. It is formed by the observations on 43 chickens and 4306 genes. These chickens have an additional factor that distinguish them: their diet. Thus, our factors are 4307 if we consider this information, where the last variable is a qualitative one, which includes 6 different diet types [13].

Data Description
The chicken data pertains to the gene expression levels of chickens and is gathered in a data–frame consisting of 4306 rows and 43 columns. The rows correspond to the genes, while the columns correspond to the chickens. As we have already mentioned, we will include in our analysis the qualitative variable, diet, which the chickens followed. This categorical variable contains 6 diets based on its distinct conditions: normal diet (N), fasting for 16 hours (F16), fasting for 16 hours then refed for 5 hours (F16R5), fasting for 16 hours then refed for 16 hours (F16R16), fasting for 48 hours (F48) and fasting for 48 hours then refed for 24 hours (F16R5).

Aim of our analysis
The purpose of our analysis on this dataset is to detect whether the difference between the 6 types of diets, that the 43 chickens follow, affects the gene expression. More precisely, it may be interesting to see how long the chicken needs to be refed after fasting before it returns to a normal state, i.e., a state comparable to a state of a chicken with a normal diet [13]. This problem, which is known as batch effect problem is among the most popular when dealing with this kind of data and PCA is often used as an exploratory tool to visualize data and carry out tests [25].

1.5.2 Statistical Analysis

Exploratory Analysis First of all, we carry out an exploratory analysis in which:

- The matrix is transposed.
- The supplementary variable "Diet" is created.

The R–package "dplyr" is loaded.

Firstly, as we have mentioned earlier and as we see above, the data is structured in a data–frame of 7406 rows and 43 columns.

```
> class(chicken)
[1] "data.frame"

> dim(chicken)
[1] 7406   43
```

However, given that in a data–frame rows correspond to the observations and the columns to the variables, we have to transpose the matrix, as follows:

```
> chicken <- as.data.frame(t(chicken))
> dim(chicken)
[1]   43 7406
```

Next, we need to create the supplementary variable "Diet", which includes the 6 different diets of the chickens. For this, we used the R–package "dplyr", which is really useful in such manipulation matters. As we see below, we concluded in the desired matrix.

```
> chicken <- mutate(chicken,
+                   Diet = as.factor(c(rep("N",6),rep("J16",5),rep("J16R5",8),
+                                      rep("J16R16",9),rep("J48",6),rep("J48R24",9))))
> dim(chicken)
[1]   43 7407
```

Applying PCA Now that we fixed the form of our data–frame, we move on applying Principal Components Analysis on the chicken data. For this procedure we used the R–package "FactoMineR". The command is given below:

```
> pca <- PCA(chicken,quali.sup=7407,scale.unit = T, graph = F)
```

We have to point out that the command needs the qualitative/supplementary variable to be mentioned.
Continuing, using the "factoextra" R–package, we get the eigenvalues, which correspond to the variances of the principal components. We notice, here, that the number of Principal Components is 42, in other words $n-1$, which was expected as we are in the case where $n < p$.
Here we present just the first 6 principal components in order to be concise.

```
> eig <- get_eigenvalue(pca)
> head(eig)
eigenvalue variance.percent cumulative.variance.percent
Dim.1  1453.5724        19.626957                    19.62696
Dim.2   692.7879         9.354413                    28.98137
Dim.3   536.2080         7.240183                    36.22155
Dim.4   434.4534         5.866235                    42.08779
Dim.5   374.6216         5.058352                    47.14614
Dim.6   324.0825         4.375945                    51.52209
```

From the above results we take the percentages of the variance explained by each Principal Component, and the cumulative percentages of them. For example, the first PC explains approximately 19.63% of the variance, the second PC 9.35% and both of them together explain 29% of the variation.

Carrying on, we look at the individuals.
Firstly, we calculate the distances of the first individuals from their categories.

```
> head(ind$dist)
1        2        3        4        5        6
72.88762 69.63425 70.81357 80.13850 71.00282 76.83545
```

Then, we take their contributions in the determination of the first five PCs.

```
> head(round(ind$contrib,3))
Dim.1 Dim.2 Dim.3 Dim.4 Dim.5
1 0.061 1.011 9.590 0.957 0.003
2 0.383 0.692 5.951 0.677 0.034
3 0.460 0.709 7.619 0.273 0.223
4 0.371 1.191 5.514 0.010 0.002
5 0.203 0.006 3.118 0.148 0.743
6 0.617 0.635 1.172 0.068 0.035
```

Also, we can see their quality of representation in the first five PCs.

```
> head(round(ind$cos2,3))
Dim.1 Dim.2 Dim.3 Dim.4 Dim.5
1 0.007 0.057 0.416 0.034 0.000
2 0.049 0.042 0.283 0.026 0.001
3 0.057 0.042 0.350 0.010 0.007
4 0.036 0.055 0.198 0.000 0.000
5 0.025 0.000 0.143 0.005 0.024
6 0.065 0.032 0.046 0.002 0.001
```

As far as the variables are concerned, now, we take a look at the first ones, as the inspection of all of them is difficult and meaningless because of their large number.
Firstly, we calculate the coordinates/loadings of the first variables for the first five PCs:

```
> variable <-get_pca_var(pca)
> head(round(variable$coord, 3))
Dim.1  Dim.2  Dim.3  Dim.4  Dim.5
A4GALT  -0.526 -0.060 -0.139 -0.082 -0.098
```

```
A4GNT   -0.124  0.217  0.228 -0.183  0.274
AACS     0.332  0.036  0.019  0.127  0.042
AADACL1  0.253  0.284 -0.339  0.141  0.059
AADACL2  0.510 -0.486  0.178 -0.314  0.100
AADACL3  0.314  0.237 -0.268  0.157  0.164
```

Then, we see their contribution in the formation of the first five PCs:

```
> head(round(variable$contrib, 3))
Dim.1 Dim.2 Dim.3 Dim.4 Dim.5
A4GALT  0.019 0.001 0.004 0.002 0.003
A4GNT   0.001 0.007 0.010 0.008 0.020
AACS    0.008 0.000 0.000 0.004 0.000
AADACL1 0.004 0.012 0.021 0.005 0.001
AADACL2 0.018 0.034 0.006 0.023 0.003
AADACL3 0.007 0.008 0.013 0.006 0.007
```

At last, we take the quality of representation of these variables on the factor map:

```
> head(round(variable$cos2, 3))

Dim.1 Dim.2 Dim.3 Dim.4 Dim.5
A4GALT  0.276 0.004 0.019 0.007 0.010
A4GNT   0.015 0.047 0.052 0.034 0.075
AACS    0.110 0.001 0.000 0.016 0.002
AADACL1 0.064 0.081 0.115 0.020 0.003
AADACL2 0.260 0.236 0.032 0.098 0.010
AADACL3 0.098 0.056 0.072 0.025 0.027
```

Moving on by displaying the variable plot for those variables whose correlation is greater than 0.8 (this is because of the large number of variables in the dataset), we will get an image for the relationships between them. This variable plot, which is given in Figure 1.35, shows which of the variables are highly, positively correlated between them (variables that appear close to each other), which are significantly, negatively correlated (variables that are on opposite sides of the graph), and we see that there are no uncorrelated variables (those which are orthogonal in the graph).

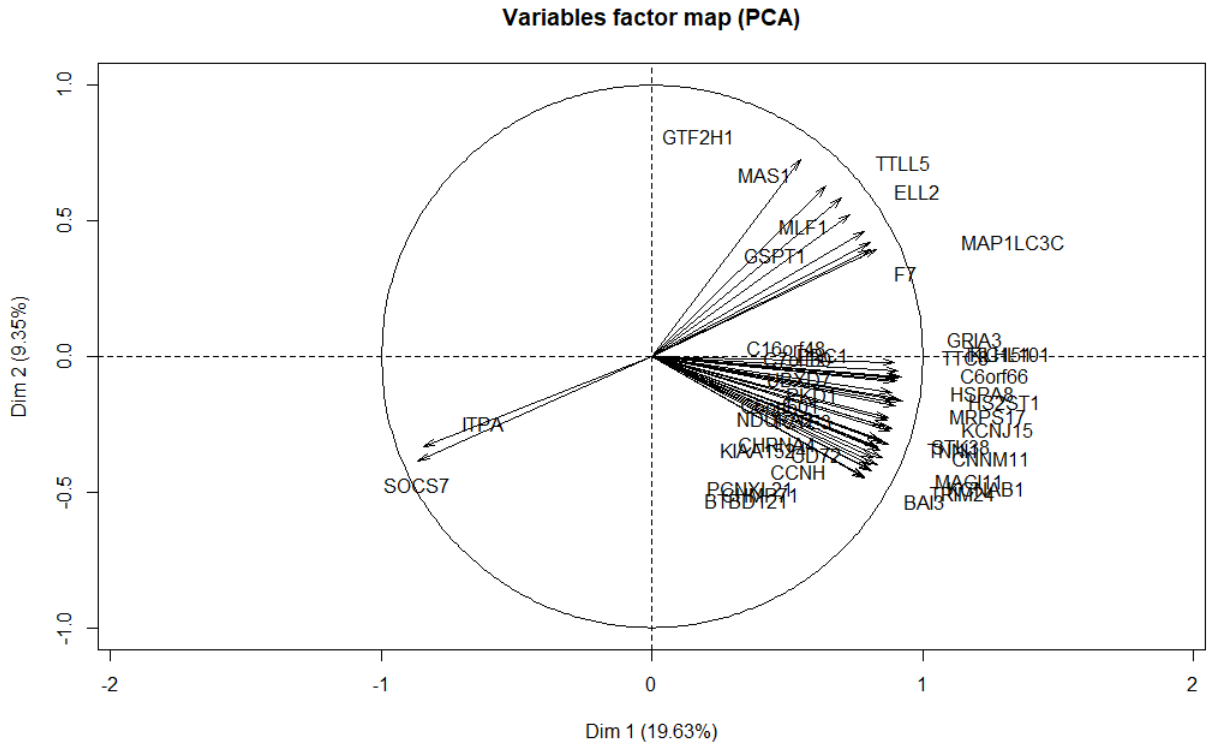

Figure 1.35: Variable Plot: chicken data

Next, we inspect the supplementary category "Diet".
From the results below, which are taken from the summary command, we see:

- The distances of each of the six diet–categories from the center of the circle.
- Their coordinates/loadings on the three first PCs.
- Their quality of representation (cos2) on the factor map.

```
Supplementary categories
Dist     Dim.1    cos2  v.test
J16          |  55.795 |  26.069   0.218   1.607
J16R16       |  42.365 |  34.124   0.649   2.984
J16R5        |  41.933 |  18.534   0.195   1.506
J48          |  79.310 | -73.356   0.855  -5.021
J48R24       |  48.622 | -25.591   0.277  -2.238
N            |  46.001 |  14.122   0.094   0.967



Dim.2    cos2  v.test       Dim.3    cos2  v.test

J16       7.307    0.017   0.653 |  35.312   0.401   3.585 |
J16R16   -7.524    0.032  -0.953 | -10.099   0.057  -1.454 |
J16R5   -13.613    0.105  -1.602 |  17.219   0.169   2.304 |
J48     -22.706    0.082  -2.251 |   7.912   0.010   0.892 |
J48R24   36.262    0.556   4.594 |  -7.333   0.023  -1.056 |
N        -8.340    0.033  -0.827 | -34.148   0.551  -3.849 |
```

Because, as we have already underlined, our goal in the whole analysis is to find out if there exist differences in gene expression between the distinct types of diet, the presentation of some graphs corresponding to the categorical variable "Diet" is necessary.

Thus, in Figure 1.36 the 43 individuals are represented on the factor map according to the category of the variable "Diet", to which they belong. The two–dimensional space is formed by the two first principal components. From Figure 1.36, we observe that the first PC distinguishes the individuals which are grouped within the diet–types J48R24 and J48 from the remaining four categories, while the second PC distinguishes these individuals that come under J48R24 and J16 from the rest. On the other hand, the individuals which followed the diets J16R16, J16R5 and N lie together on the factor map, so no outstanding difference among them is noticed.

For a better visualization, we plot the confidence ellipses around the categories of the variable "Diet", as shown in Figure 1.37. Figure 1.37 helps us understand if an observation comes or not from a determined population, which has a normal bivariate distribution, detect abnormal points and, of course, compare the different six categories. More specifically, we see that J16R16, J16R5, J16 and N overlap, contrary to the J48 and J48R24.

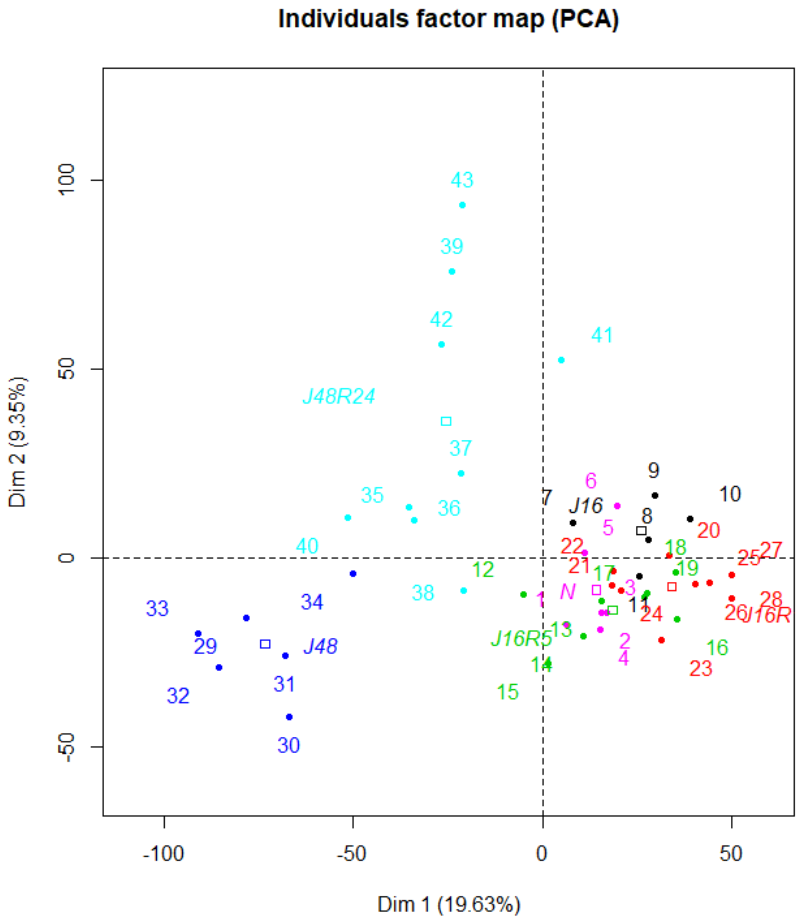

Figure 1.36: Individual Plot: chicken data

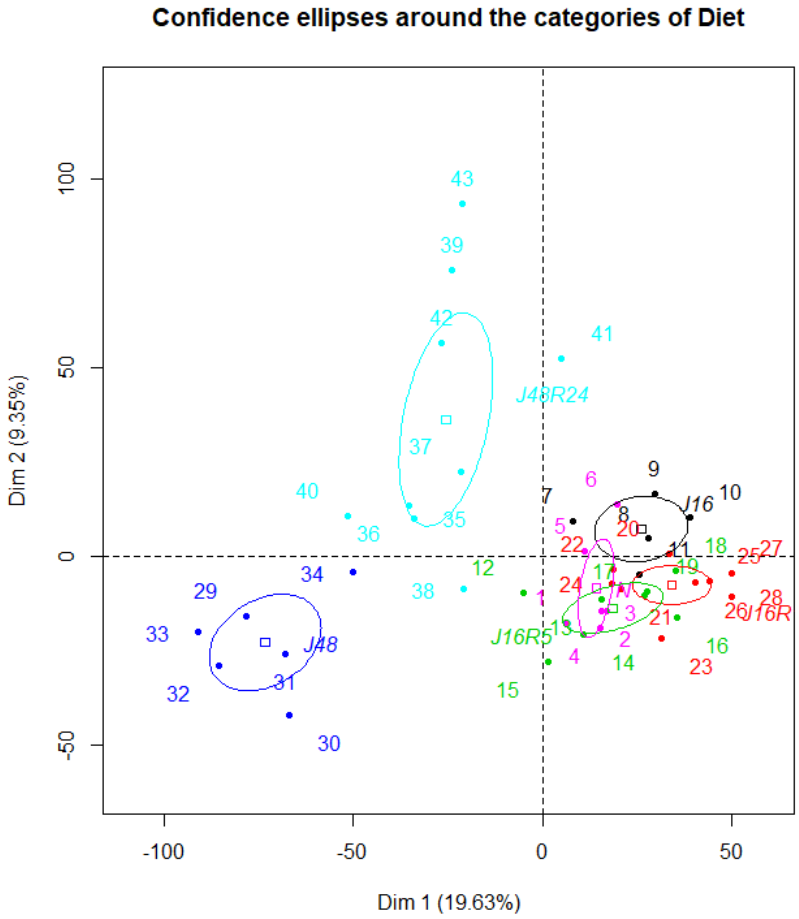

Figure 1.37: Confidence Ellipses around the categories of Diet

Conclusions In other words, from both Figure 1.36 and Figure 1.37 a significant difference between the diet–types J48, J48R24 (diets under excessive stress) and the rest is concluded. That means that the chickens that follow the diets J16R16, J16R5, J16 and N, have a similar gene expression, whereas those that follow J48 and those that follow J48R24 appear a uniquely different gene expression.

Chapter 2

Classification: Parametric Techniques

2.1 Introduction to Classification

Examples of Classification Problems

- A medical researcher is interested in determining whether the probability of a heart attack can be predicted given the patient's blood pressure, cholesterol level, calorie intake, gender, and lifestyle.
- An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the user's IP address, past transaction history, and so forth.
- A criminologist is interested in determining differences between on–parole prisoners who have and who have not violated their parole, then using this information for making future parole decisions.
- On the basis of DNA sequence data for a number of patients with and without a given disease, a biologist would like to figure out which DNA mutations are deleterious (disease–causing) and which are not. [35, 14]

Presenting the problem of classification, we will first need to introduce some definitions.

Discriminant Analysis
Discriminant analysis is one of the available techniques for achieving the following objectives:

1. Identify the variables that discriminate "best" between the two groups.
2. Use the identified variables or factors to develop an equation or function for computing a new variable or index that will parsimoniously represent the differences between the two groups.
3. Use the identified variables or the computed index to develop a rule to classify future observations into one of the two groups.

The third objective of discriminant analysis is to classify future observations into one of the two groups. *Classification* can be considered as an independent procedure unrelated to discriminant analysis. However, it can also be treated as a part of the discriminant analysis procedure. [35]

Training Data
We will always assume that we have observed a set of n different data points. Let x_{ij} represent

the value of the jth predictor, or input, for observation i, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$. Correspondingly, let y_i represent the response variable for the ith observation. These observations are called the *training data* because we will use these observations to train, or teach, our method how to estimate f. Then our training data consist of $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})'$. Our goal is to apply a statistical learning method to the training data in order to estimate the unknown function f (f is some fixed but unknown function of $X_1, \ldots, X_p$ and represents the systematic information that X provides about Y) . In other words, we want to find a function f such that $\hat{Y} \approx f(X)$ for any observation (X, Y). When we have to handle a statistical problem, we explore linear and non–linear approaches for estimating f [14].

Test Data
Test data is referred to the unseen or held–out observations that we use to evaluate the performance of a statistical learning method [14].

Parametric methods
Parametric methods involve a two–step model–based approach.

1. First, we make an assumption about the functional form, or shape, of f. For example, one very simple assumption is that f is linear in X:

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p. \tag{2.1}$$

   Once we have assumed that f is linear, the problem of estimating f is greatly simplified. Instead of having to estimate an entirely arbitrary p–dimensional function $f(X)$, one only needs to estimate the $p + 1$ coefficients $\beta_0, \beta_1, \ldots, \beta_p$.

2. After a model has been selected, we need a procedure that uses the training data to fit or train the model. In case of the linear model given in (2.1), we need to estimate the parameters $\beta_0, \beta_1, \ldots, \beta_p$. That is, we want to find values of these parameters such that $Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$. The most common approach to fitting the model (2.1) is referred to as (ordinary) least squares. However, least squares is one of many possible ways to fit the linear model.

The model–based approach just described is referred to as *parametric* because it reduces the problem of estimating f down to one of estimating a set of parameters. Assuming a parametric form for f simplifies the problem of estimating f because it is generally much easier to estimate a set of parameters, such as $\beta_0, \beta_1, \ldots, \beta_p$ in the linear model (2.1), than it is to fit an entirely arbitrary function f [14].

Non–Parametric methods
Non–parametric methods do not make explicit assumptions about the functional form of f. Instead, they seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly. Such approaches can have a major advantage over parametric approaches: by avoiding the assumption of a particular functional form for f, they have the potential to accurately fit a wider range of possible shapes for f. Any parametric approach brings with it the possibility that the functional form used to estimate f is very different from the true f, in which case the resulting model will not fit the data well. In contrast, non–parametric approaches completely avoid this danger, since essentially no assumption about the form of f is made. But non–parametric approaches do suffer from a major disadvantage: since they do not reduce the problem of estimating f to a small number of parameters, a very large number of observations (far more than is typically needed for a parametric approach) is required in order to obtain an accurate estimate for f [14] .

Supervised Learning
For each observation of the predictor measurement x_i, $i = 1, \ldots, n$ there is an associated response

measurement y_i. We wish to fit a model that relates the response to the predictors, with the aim of accurately predicting the response for future observations (prediction) or better understanding the relationship between the response and the predictors (inference) [14].

Unsupervised Learning
Unsupervised learning describes the somewhat more challenging situation in which for every observation $i = 1, \ldots, n$, we observe a vector of measurements x_i but no associated response y_i. It is not possible to fit a regression model, since there is no response variable to predict. In this setting, we are in some sense working blind [14].

Quantitative Variables
Quantitative variables are the variables that take on numerical values. Examples include a person's age, height, or income, the value of a house, and the price of a stock [14].

Qualitative Variables
Qualitative/Categorical variables are the variables that take on values in one of K different classes, or categories. Examples of qualitative variables include class a person's gender (male or female), the brand of product purchased (brand A, B, or C), whether a person defaults on a debt (yes or no), or a cancer diagnosis (Acute Myelogenous Leukemia, Acute Lymphoblastic Leukemia, or No Leukemia) [14].

Regression Problems
The problems with a quantitative response are defined as *regression problems* [14].

Classification Problems
The problems that involve a qualitative response are referred to as *classification problems*. Predicting a qualitative response for an observation can be referred to as classifying that observation, since it involves assigning the observation to a category, or class [14].

Training Error Rate
Suppose that we seek to estimate f on the basis of training observations $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where $y_1, \ldots, y_n$ are qualitative. The most common approach for quantifying the accuracy of our estimate $\hat{f}$ is the *training error rate* , the proportion of mistakes that are made if we apply our estimate $\hat{f}$ to the training observations:

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i) \tag{2.2}$$

Here $\hat{y}_i$ is the predicted class label for the ith observation using $\hat{f}(x_i)$. And $I(y_i \neq \hat{y}_i)$ is an indicator variable that equals 1 if $y_i \neq \hat{y}_i$ and zero if $y_i = \hat{y}_i$. If $I(y_i \neq \hat{y}_i) = 0$, then the ith observation was classified properly by our classification method. Otherwise it was misclassified. Hence Equation (2.2) computes the fraction of incorrect classifications. Equation (2.2) is referred to as the training error rate because it is computed based on the data that was used to train our classifier [14].

Test Error Rate
The *test error rate*–for which we are most interested among the error rates,when applying our classifier– is the error rate applied to test observations that were not used in training set. So, the test error rate, associated with a set of test observations of the form (x_0, y_0) is given by:

$$\text{Ave}(I(y_0 \neq \hat{y}_0)), \tag{2.3}$$

where $\hat{y}_0$ is the predicted class label that results from applying the classifier to the test observation with predictor x_0. A *good classifier* is one for which the test error introduced in (2.3) is smallest [14].

Classifier
The methods used for classification are known as *classifiers* [14].

In this chapter we will present the widely parametric classifiers: logistic regression, linear discriminant analysis and quadratic discriminant analysis. The non–parametric classifiers K–nearest neighbors and Classification Trees are presented in Chapter 3 and Chapter 4, respectively.

k–fold Cross Validation on Classification Problems
This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining $k-1$ folds. The number of misclassified observations, is then computed on the observations in the held–out fold. This procedure is repeated k times; each time, a different group of observations is treated as a validation set. This process results in k estimates of the test error, $\mathrm{Err}_1, \mathrm{Err}_2, \ldots, \mathrm{Err}_k$. The k–fold CV estimate is computed by averaging these values,

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{Err}_i, \tag{2.4}$$

where $\mathrm{Err}_i = I(y_i \neq \hat{y}_i)$. [14]

2.2 Logistic Regression

2.2.1 Introduction

Logistic regression models the probability that a variable Y belongs to a particular category. Here we consider the simplest case of a binary response coded 0 or 1. Let us denote the conditional probability that $Y = 1$ for given values of multiple predictors $X = (X_1, \ldots, X_p)$ by $p(X) = p(X_1, \ldots, X_p) = \Pr(Y = 1|X)$. Next we have to find an appropriate way to express the dependence of $p(X)$ on the regressor variables. We cannot simply write

$$p(X_1, \ldots, X_k) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

because the right–hand side of this expression is not, in general, contained in the interval $[0, 1]$. Probabilities cannot be negative or greater than 1. However there is merit in trying to retain some linearity and this is done by using the *logistic regression model* with the *systematic component*

$$p(X_1, \ldots, X_p) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_p X_p}}, \tag{2.5}$$

and the *random component*

$$Y|X \sim Bernoulli(p(X)) \tag{2.6}$$

The systematic component shows how p depends on the Xs and the random component shows how Y varies about p. [3, 14]
The logistic function (2.5) will always produce an *S–shaped curve* of this form (see Figure 2.1), and so regardless of the value of X, we will obtain a sensible prediction.

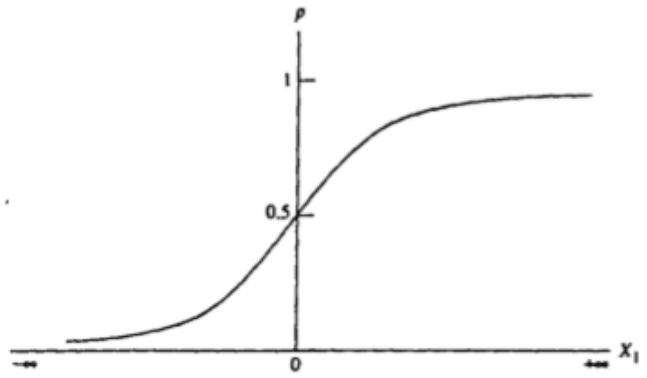

Figure 2.1: The logistic curve.

Figure 2.1 gives the relationship between probability p, and each independent variable, X_i. It can be seen that the relationship between probability and the independent variable is represented by a logistic curve that asymptotically approaches one as X_i approaches positive infinity and zero as X_i approaches negative infinity. The function that gives the relationship between probability and the independent variables is known as the *link function*, which is logit for the above model. Other link functions such as *normit* or *probit* (i.e., the inverse of the cumulative standard normal distribution function) and *complementary log-log function* (i.e., inverse of the Gompertz function) can also be used [35].

The Bernoulli distribution is a special case of the Binomial distribution and simply specifies that a single random variable, Y, say, will take the value 1 with probability p and the value 0 with probability

$(1-p)$. Thus the model in equations (2.5) and (2.6) specifies that the probability that $Y = 1$ is the given function of the X values. After a bit of manipulation of (2.5), we find that:

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_p X_p} \tag{2.7}$$

The quantity $p(X)/[1-p(X)]$ is called the *odds*, and can take on any value between 0 and ∞. Values of the odds close to 0 and ∞ indicate very low and very high probabilities, respectively.
By taking the logarithm of both sides of (2.7), we arrive at

$$\text{logit}(p(X)) = \log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p \tag{2.8}$$

The left–hand side of (2.8) is called the *log–odds* or *logit*. We see that the logistic regression model (2.5) has a logit that is linear in X. [3, 14]

2.2.2 Interpretation and Estimation of the Coefficients

Interpretation of the coefficients
In a multiple linear regression model, β_i gives the average change in Y associated with a one–unit increase in X_i. In contrast, in a multiple logistic regression model, increasing X_i by one unit changes the log–odds by β_i, (2.8), or equivalently it multiplies the odds by e^{β_i}, (2.7). However, because the relationship between $p(X)$ and the X_i in (2.5) is not a straight line, β_i does not correspond to the change in $p(X)$ associated with a one-unit increase in X_i. The amount that $p(X)$ changes due to a one–unit change in X_i will depend on the current value of X_i. But regardless of the value of X_i, if β_i is positive then increasing X_i will be associated with increasing $p(X)$, and if β_i is negative then increasing X_i will be associated with decreasing $p(X)$. The fact that there is not a straight–line relationship between $p(X)$ and X_i, and the fact that the rate of change in $p(X)$ per unit change in X_i depends on the current value of X_i, can also be concluded by the *S–shaped curve*, see Figure 2.1, that the logistic regression produces [14].

As a summary, under the logistic model:

- β_i is the change in the log–odds associated with a unit increase in X_i. The odds are multiplied by e^{β_i} for each unit increase in X_i.
- β_0 is log–odds at $X = \underline{0}$; e^{β_0} is the odds of a favorable response at this X–value (which may not have a reasonable interpretation if $X = \underline{0}$ is far from the range of the data) [7].

Estimation of the coefficients
The coefficients β_i in (2.5) are unknown, and must be estimated based on the available training data. While in the case of linear regression the *least squares* approach is used to estimate the unknown coefficients, in the case of logistic regression the more general method of *maximum likelihood* is preferred, since it has better statistical properties. The basic intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek estimates for β_is such that the predicted probability $\hat{p}(x_i)$ for each observation, using (2.5), corresponds as closely as possible to its observed value. In other words, we try to find $\hat{\beta}_i$s such that plugging these estimates into the model for $p(X)$, given in (2.5), yields a number close to one for the observations where $Y = 1$, and a number close to zero for

those where $Y = 0$. This intuition can be formalized using a mathematical equation called a likelihood function

$$\ell(\beta_0, \beta_1, \ldots, \beta_p) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})). \tag{2.9}$$

The estimates $\hat{\beta}_i$s are chosen to maximize this likelihood function. Maximum likelihood is a very general approach that is used to fit many of the non–linear models [14].

2.2.3 ROC Curve

From Equation (2.5) we have that the estimated/predicted probability of success is:

$$\hat{p} = \widehat{\Pr}(Y = 1) = \frac{e^{\hat{\beta}_0 X_0 + \hat{\beta}_1 X_1 + \ldots + \hat{\beta}_p X_p}}{1 + e^{\hat{\beta}_0 X_0 + \hat{\beta}_1 X_1 + \ldots + \hat{\beta}_p X_p}} \tag{2.10}$$

Now we suppose a threshold p_0 for which:

- if $\hat{p} > p_0$, then it is predicted $Y = 1$
- if $\hat{p} \leq p_0$, then it is predicted $Y = 0$

Comparing the predictions with the true values of the binary variable Y, gives the *confusion matrix* (see Table 2.1). For example, a is the number of the observations with a true status of $Y = 1$, for which their predicted status is also 1. [5]

		True Status		
		$Y = 1$	$Y = 0$	
Predicted Status	$Y = 1$	a	b	$a + b$
	$Y = 0$	c	d	$c + d$
		$a + c$	$b + d$	n

Table 2.1: Confusion Matrix

Based on the Table 2.1, the following terms are defined:

- Sensitivity $= a/(a+c)$, the percentage of the observations, where $Y = 1$, that are identified (*true positive rate*).
- Specificity $= d/(b + d)$, the percentage of the observations, where $Y = 0$, that are correctly identified (*true negative rate*).

In the confusion matrix the elements on its diagonal represent individuals whose statuses were correctly predicted, while off–diagonal elements represent individuals that were misclassified. [10]

The *ROC* curve is a popular graphic for simultaneously displaying the two types of errors: false positive $(1 - Specificity)$ and false negative rate for all possible thresholds (See Figure 2.2).

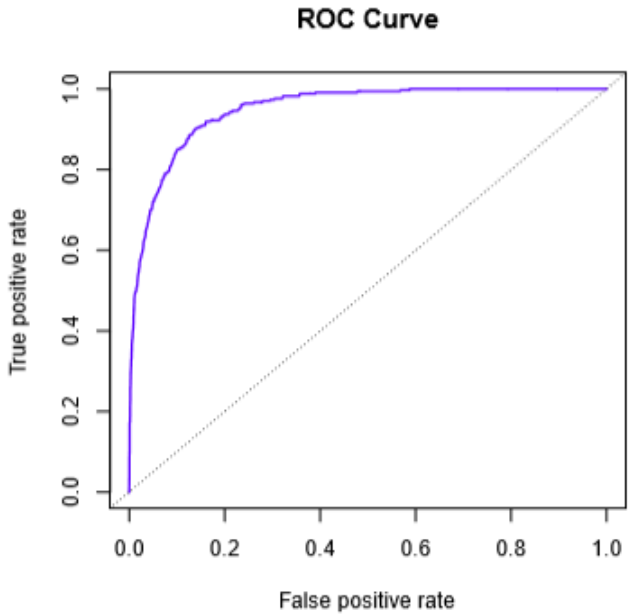

Figure 2.2: A ROC curve.

The overall performance of a classifier, summarized over all possible thresholds, is given by the *area under the (ROC) curve (AUC)*. An ideal ROC curve will hug the top left corner, so the larger area under the (ROC) curve the better the classifier. If the AUC is close to the maximum of one, it will be considered very good. We expect a classifier that performs no better than chance to have an AUC of 0.5 (when evaluated on an independent test set not used in model training). ROC curves are useful for comparing different classifiers, since they take into account all possible thresholds [14].

2.2.4 Application to Irish data

In this subsection we will examine the Irish data via logistic regression. This dataset was introduced in Section 1.4.

Exploratory Analysis First of all, we carry out an exploratory analysis in which outliers are detected and the dataset is prepared to apply logistic regression. The R–packages "naniar" and "imputeTS" are used, as it is next illustrated.

Thus, first we will replace the outliers of our dataset, which had been detected in the Subsection 1.4.2, with the median of the observations, as follows:

```
> irish[102,4] <- NA
> irish[95,6] <- NA
> irish[c(87, 97), 9] <- NA
> irish[120,5] <- NA
> irish[162,6] <- NA
> irish[125,7] <- NA
> irish[125,8] <- NA
> irish <- na.mean(irish, option = "median")
```

For this procedure, the R–packages "naniar" and "imputeTS" were loaded.

Moving on, we rename some of the columns of the data in order to gain a better view of them in our results. This is done as follows:

```
> colnames(keane)[c(1,2,3,5)] <- c("Status","C_Seg","V_Prop","C_Rel")
```

Next, we create the variable "status", with the following command, in which we give the value "ent" (entrepreneur) to the observations which have a "Status" equal to 2 and the value "mgr" (manager) to the rest of them, in other words, to these that have a "Status" equal to 1.

```
> status = as.factor(ifelse(keane$Status==2, "ent", "mgr"))
```

In order to find out how the categorical variable "status" is encoded in R, we use the *contrasts*() function:

```
> contrasts(status)
mgr
ent   0
mgr   1
```

We see that *contrasts*() function indicates that R has created a dummy variable with a 1 for managers. To be coherent with the whole analysis, we will change this encoding as follows:

```
> status <- factor (as.character(status), levels = c("mgr", "ent"))
```

In this way we have managed to make entrepreneurs correspond to the value 1 and managers to the value 0 of the dummy variable. We check this as follows:

```
> contrasts(status)
ent
mgr   0
ent   1
```

Applying Logistic Regression Now, we are ready to realize our whole analysis upon the fact that an observation is an entrepreneur.

Firstly, we will fit a logistic regression model in order to predict status using the nine variables: Customer Segments, Value Propositions, Channels, Customer Relationships, Revenue Streams, Key Resources, Key Activities, Key Partners and Cost Structure. We apply the $glm()$ function in which we pass the argument $family = binomial$, in order to run a logistic regression rather than some other type of generalized linear model.

```
> glm.fit = glm(status ~ C_Seg+V_Prop+Chan+C_Rel+Rev_Str+Key_Res+Key_Act+Key_Part+Cost_Str,
+                       data=irish,
+                       family=binomial)
> summary(glm.fit)

Call:
glm(formula = status ~ C_Seg + V_Prop + Chan + C_Rel + Rev_Str +
Key_Res + Key_Act + Key_Part + Cost_Str, family = binomial,
data = irish)

Deviance Residuals:
Min       1Q   Median       3Q      Max
-2.5334  -0.5562   0.2329   0.6704   2.3479

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -14.30728    3.19174  -4.483 7.37e-06 ***
C_Seg         0.15446    0.06775   2.280  0.02260 *
V_Prop        0.29564    0.07444   3.971 7.15e-05 ***
Chan         -0.05435    0.06003  -0.905  0.36532
C_Rel        -0.11635    0.08721  -1.334  0.18213
Rev_Str       0.14363    0.06399   2.244  0.02480 *
Key_Res      -0.14105    0.07498  -1.881  0.05996 .
Key_Act       0.03175    0.06658   0.477  0.63344
Key_Part     -0.02416    0.06067  -0.398  0.69047
Cost_Str      0.17099    0.05473   3.124  0.00178 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 225.07  on 170  degrees of freedom
Residual deviance: 139.24  on 161  degrees of freedom
AIC: 159.24

Number of Fisher Scoring iterations: 5
```

Here, we see that the most statistically significant predictors are: Customer Segments, Value Propositions, Revenue Streams, Key Resources and Cost Structure (For a level of statistical significance 5%). Examining their coefficients, we observe that all of them are positive except for Key Resources, which means that if high values of these predictors are noticed, it is more likely for the individual/observation to be an entrepreneur. However, in the case of Key Resources, we have the opposite result: high values of the variable Key Resources indicate an individual to be more likely a manager. That means that, high scores of the above variables predict the individual as an entrepreneur, apart from the Key Resources variable, whose high scores forecast the individual as a manager.

For a less cumbersome interpretation, we will find the odds ratios and their confidence intervals, using the *exp*() and *confint*() functions. The code is the following:

```
> exp(cbind(OddsRatio = coef(glm.fit),confint(glm.fit)))
Waiting for profiling to be done...
OddsRatio        2.5 %       97.5 %
(Intercept) 6.115438e-07 6.110882e-10 0.0001834451
C_Seg       1.167032e+00 1.024764e+00 1.3397285648
V_Prop      1.343984e+00 1.167533e+00 1.5672468497
Chan        9.471036e-01 8.393961e-01 1.0647334200
C_Rel       8.901613e-01 7.482377e-01 1.0558293732
Rev_Str     1.154458e+00 1.022385e+00 1.3166412224
Key_Res     8.684492e-01 7.420023e-01 0.9976975744
Key_Act     1.032261e+00 9.065822e-01 1.1790869574
Key_Part    9.761306e-01 8.651556e-01 1.0987969356
Cost_Str    1.186480e+00 1.072202e+00 1.3308615706
```

Regarding the significant predictors, as mentioned above we can come to the following conclusions:

- **Customer Segments**: If the value of the variable Customer Segments is increased by one unit, while the rest of the variables have a fixed value, it is more likely for the individual to be an entrepreneur by about 17% (the odds ratio is 1.17).
- **Value Propositions**: It is about 1.34 times more likely for an individual to be an entrepreneur than a manager, if the variable Value Propositions increases by 1 unit and the rest of the variables are fixed.
- **Revenue Streams**: If the variable Revenue Streams is increased by one unit, while the rest variables remain fixed, it is more likely for the individual to be an entrepreneur than a manager, with an odds ratio of 1.15.
- **Key Resources**: Increase of the value of the variable Key Resources by one unit, makes the probability of an individual to be an entrepreneur 0.87 times as likely to be a manager. Or, managers are more likely than entrepreneurs with an odds ration of 1.15.
- **Cost Structure**: Similarly to the above, here the odds ratio for this variable is approximately 1.19, which means that it is by 19% more likely for an individual to be an entrepreneur than a manager.

The *predict*() function can be used to predict the probability that the observation is an entrepreneur, given values of the predictors. The *type="response"* option tells R to output probabilities of the form $\Pr(Y = 1|X)$, as opposed to other information such as the logit. If no data set is supplied to the *predict*() function, then the probabilities are computed for the training data that was used to fit the logistic regression model. We know that these values correspond to the probability of the individual

to be an entrepreneur, rather than a manager, because we have made the *contrasts*() function indicate that R has created a dummy variable with a 1 for entrepreneur instead of manager.

```
> glm.probab=predict(glm.fit ,type="response")
```

Here we have printed the first 10 probabilities:

```
> glm.probab [1:10]
1         2         3         4         5         6         7         8         9
0.7414775 0.4713426 0.7181912 0.4887675 0.3190825 0.5100011 0.9784613 0.9382211 0.8800505
10
0.4064996
```

Additionally, we calculate the last 10 probabilities:

```
> glm.probab [161:171]
161          162          163          164          165          166
0.0822730267 0.0007846501 0.0241564444 0.9475589620 0.0827757677 0.1935759826
167          168          169          170          171
0.6197401126 0.1978726475 0.5050419361 0.1899609389 0.2789557278
```

We notice, then, that the first 10 probabilities are–on the whole–significantly greater than the last 10 probabilities.

In order to make a prediction as to whether we have an entrepreneur or not, we must convert these predicted probabilities into class labels: mgr and ent. The following two commands create a vector of class predictions based on whether the predicted probability of an individual to be an entrepreneur is greater than or less than 0.5.

```
glm.predict = rep("mgr",171)
glm.predict[glm.probab > 0.5]="ent"
```

The first command creates a vector of 171 elements labelled as manager (mgr). The second line transforms to entrepreneur (ent) all of the elements for which the predicted probability of an individual to be an entrepreneur exceeds 0.5.

Performance metrics Given the predictions discussed previously, the *table*() function can be used to produce a confusion matrix in order to determine how many observations were correctly or incorrectly classified.

```
> table(glm.predict ,status)
status
glm.predict mgr ent
ent  20  94
mgr  43  14
> accuracy = mean(glm.predict == status)
> accuracy
[1] 0.8011696
```

Normally the diagonal elements of the confusion matrix indicate correct predictions, while the off–diagonals represent incorrect predictions. Here we have the opposite result because of the encoding we have applied. Hence, our model correctly predicted that the individual is an entrepreneur in 94 out of 108 of the cases, and that the individual is a manager in 43 out of 63 of the them, for a total of $94 + 43 = 137$ correct predictions. The *mean*() function can be used to compute the fraction of observations for which the prediction was correct. In this case, logistic regression correctly predicted the different categories with an accuracy of about 80.12%.

As an additional measure for assessing the performance of the model, we use the ROC curve. The Area Under the Curve (AUC) summarizes the overall performance of the classifier, over all possible probability cutoffs. As we have mentioned in Subsection 2.2.3 the larger the AUC, the better the classifier.

For the construction of the ROC we used the R–package "pROC". The code in R is:

```
> roc(status, fitted.values(glm.fit),smooth=TRUE,plot=TRUE,asp=NA)

Call:
roc.default(response=status,predictor=fitted.values(glm.fit),smooth=TRUE,plot=TRUE,asp=NA)

Data: fitted.values(glm.fit) in 108 controls (status ent) < 63 cases (status mgr).
Smoothing: binormal
Area under the curve: 0.8809
```

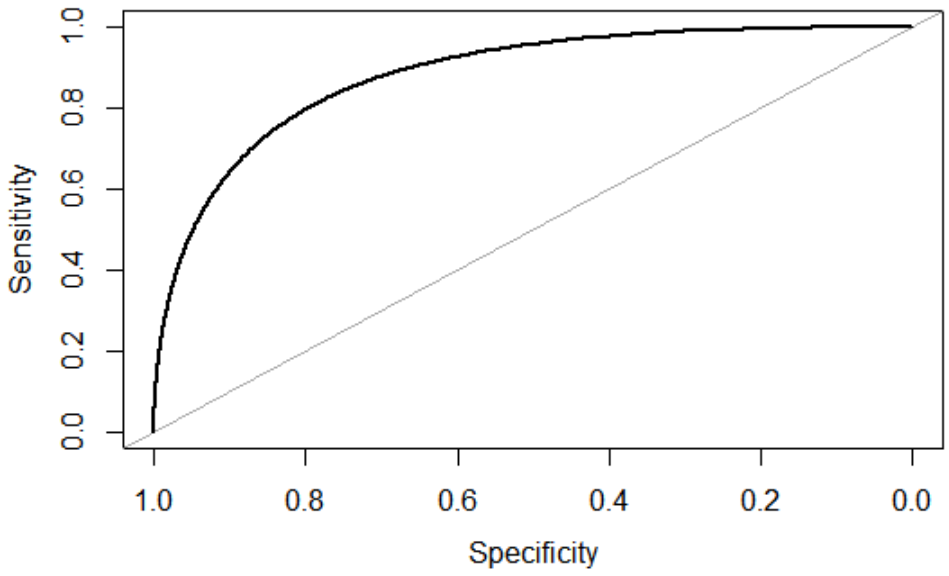

Figure 2.3: ROC for the glm.fit model.

We see that the AUC is approximately 0.881, which is quite near the "ideal" 1.

Remarks Concerning the accuracy of the model, this is really high. However, this result is misleading because we trained and tested the model on the same set of 171 observations. In other words, $100\% - 80.12\% = 19.88\%$ is the training error rate. The training error rate is often overly optimistic–it tends to underestimate the test error rate. In order to better assess the accuracy of the logistic regression model in this setting, we can fit the model using part of the data, and then examine how well it predicts

the held out data. This will yield a more realistic error rate, in the sense that in practice we will be interested in our model's performance not on the data that we used to fit the model, but rather on the observations that we set apart.

Splitting the data To implement this strategy, we will randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model). We make sure to set seed for reproducibility.

```
> set.seed(123)
> train <- sample(1:nrow(irish), nrow(irish) * 0.80)
> test <- irish[-train, ]
> status.test <- status[-train]
> dim(test)
[1] 35 10
```

As we see, the number of observations in the test dataset is 35, which we will use later.

Applying Logistic Regression after splitting the data We, now, fit a logistic regression model using only the subset of the observations that correspond to the training set, using the subset argument. We then obtain predicted probabilities of the individual to be an entrepreneur in our test set.

```
> glm.fit1 = glm(status ~ C_Seg+V_Prop+Chan+C_Rel+Rev_Str+Key_Res+Key_Act+Key_Part+Cost_Str,
+                          data=irish ,
+                          family = binomial,
+                          subset=train)
> summary(glm.fit1)

Call:
glm(formula = status ~ C_Seg + V_Prop + Chan + C_Rel + Rev_Str +
Key_Res + Key_Act + Key_Part + Cost_Str, family = binomial,
data = irish, subset = train)

Deviance Residuals:
Min       1Q   Median       3Q      Max
-2.4080  -0.5726   0.2516   0.6165   2.3218

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -12.73639    3.15671  -4.035 5.47e-05 ***
C_Seg         0.10177    0.07807   1.304 0.192369
V_Prop        0.33016    0.09286   3.556 0.000377 ***
Chan         -0.06886    0.08014  -0.859 0.390232
C_Rel        -0.11896    0.10299  -1.155 0.248080
Rev_Str       0.14259    0.08323   1.713 0.086678 .
Key_Res      -0.15417    0.09327  -1.653 0.098366 .
Key_Act       0.01027    0.08197   0.125 0.900244
Key_Part     -0.00725    0.07125  -0.102 0.918950
Cost_Str      0.17513    0.06288   2.785 0.005351 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 177.78  on 135  degrees of freedom
```

```
Residual deviance: 106.60  on 126  degrees of freedom
AIC: 126.6

Number of Fisher Scoring iterations: 5
```

What we see from the above output, is that the most statistically significant predictors here, for a significance level of 5%, are: Value Propositions and Cost Structure. We notice an outstanding difference from our previous model, where all of the data was included. More specifically, in the initial model we considered five of the predictors as significant, whereas in the glm.fit1 where the data is split, only two of them are significant. Regarding the inference for the coefficients of these variables, this is similar to that of the previous model. That is to say that, it is more likely for the individual/observation to be an entrepreneur, if high values of the previously referred predictors are given. In this case we forecast the individual as entrepreneur.

However, we should also consider the significant predictors for a significance level of 10%, in order to be more meticulous in our analysis. These are: Value Propositions, Revenue Streams, Key Resources and Cost Structure.

Again, for a better interpretation, we will find the odds ratios and their confidence intervals, for these four predictors as follows:

```
> exp(cbind(OddsRatio = coef(glm.fit1),confint(glm.fit1)))
Waiting for profiling to be done...
OddsRatio        2.5 %       97.5 %
(Intercept) 2.942105e-06 3.061592e-09 0.0008341132
C_Seg       1.107125e+00 9.513840e-01 1.2962507249
V_Prop      1.391198e+00 1.171557e+00 1.6921147082
Chan        9.334602e-01 7.935409e-01 1.0887482245
C_Rel       8.878442e-01 7.231863e-01 1.0874267081
Rev_Str     1.153261e+00 9.854192e-01 1.3684656410
Key_Res     8.571283e-01 7.059337e-01 1.0188377471
Key_Act     1.010328e+00 8.600232e-01 1.1880948573
Key_Part    9.927763e-01 8.626233e-01 1.1423108507
Cost_Str    1.191398e+00 1.061800e+00 1.3606579141
```

From the above output, the conclusions to which we can come, as far as the significant predictors are concerned, are:

- **Value Propositions**: The odds ratio of this variable is approximately 1.39, which means that an increase of 39% of the probability of an individual to be an entrepreneur is noted if the variable Value Propositions is increased by one unit.
- **Revenue Streams**: If the value of the variable Revenue Streams is increased by one unit, while the rest of the variables are fixed, it is more likely for an individual to be an entrepreneur with an odds ration of 1.15.
- **Key Resources**: For this variable we have the opposite inference, compared to the rest. That is, here, if the value of the variable Key Resources is increased by one unit, the probability for an individual to be an entrepreneur decreases by more than 14%.
- **Cost Structure**: For this variable, we have almost the same results as those we got before the splitting of the dataset. That is, if the variable Cost Structure is increased by one unit, it is more likely for the individual to be an entrepreneur than a manager, with an odds ratio of 1.19, which means that it is by 19% more likely for an individual to be an entrepreneur than a manager.

Notice that we have trained and tested our model on two completely separate data sets.

Performance metrics after splitting the data Finally, we compute the predictions for the test set and compare them to the actual status of the observations.

```
> glm.probab1 = predict(glm.fit1 ,test, type="response")
> glm.predict1 = rep("mgr", 35)
> glm.predict1[glm.probab1 > 0.5] = "ent"
> table(glm.predict1, status.test)
status.test
glm.predict1 mgr ent
ent   6  19
mgr   8   2
> accuracy1 = mean(glm.predict1 == status.test)
> accuracy1
[1] 0.7714286
```

The accuracy of this model is approximately 77.14%. One can think that this is a worse result, compared to our initial model, which had an accuracy of 80.12%. However, here we have a typical overfitting problem, in which the model is being trained really well in a specific dataset, but is unable to predict correctly new data. Thus, we should not take into account the accuracy of the model which has not been split into training and test set, as such an accuracy is meaningless.

Examining the ROC curve now, we get the following:

```
> roc(status[train],fitted.values(glm.fit1),smooth=TRUE,plot=TRUE,asp=NA)

Call:
roc.default(response=status[train],predictor=fitted.values(glm.fit1),smooth=TRUE,plot=TRUE,asp=NA)

Data: fitted.values(glm.fit1) in 49 controls (status[train] mgr) < 87 cases (status[train] ent).
Smoothing: binormal
Area under the curve: 0.889
```

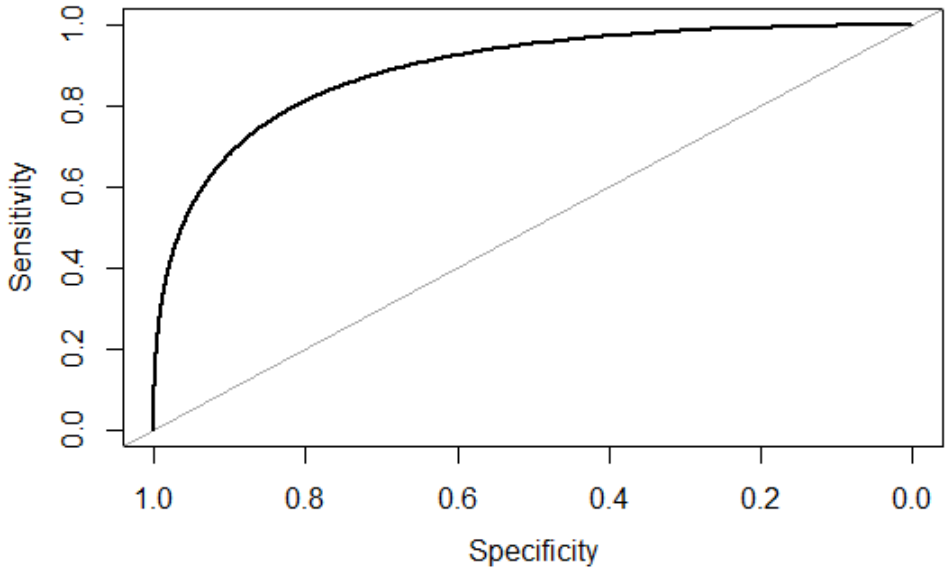

Figure 2.4: ROC for the glm.fit1 model.

Here, the AUC is 0.889, that means higher than the initial model. In other words, the ability of classification in this model, is greater than the initial, where the whole dataset was included in the logistic regression analysis.

Applying Logistic Regression with two variables Next, we examine if after removing the variables that appear not to be helpful in predicting the status, we can obtain a more effective model. After all, using predictors that have no relationship with the response tends to cause a deterioration in the test error rate (since such predictors cause an increase in variance without a corresponding decrease in bias), and so removing such predictors may in turn yield an improvement. From the output of the previous logistic regression model, we found out that the most statistically significant predictors for a significance level of 5% were: Value Propositions and Cost Structure. Thus, refitting the logistic regression using these two predictors, instead of all of them, we get the results below:

```
> glm.fit2 = glm(status ~ V_Prop+Cost_Str, data=irish,
+                family = binomial,
+                subset=train )
> summary(glm.fit2)

Call:
glm(formula = status ~ V_Prop + Cost_Str, family = binomial,
data = irish, subset = train)

Deviance Residuals:
Min       1Q   Median       3Q      Max
-2.2161  -0.5687   0.3225   0.6994   2.0669

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -13.63065    2.55012  -5.345 9.04e-08 ***
V_Prop        0.27059    0.06005   4.506 6.60e-06 ***
```

```
Cost_Str      0.15553    0.04179   3.722 0.000198 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 177.78  on 135  degrees of freedom
Residual deviance: 119.57  on 133  degrees of freedom
AIC: 125.57

Number of Fisher Scoring iterations: 5
```

What we notice here is that, regarding the AIC, in the second model we achieve a lower value of it (125.57), compared to the previous one (126.6), which means that indeed the second model with the two predictors is better.

Performance metrics for the two–variable model Moving on, we follow the same procedure as previously computing the predictions for the test set and compare them to the actual status of the observations.

```
> glm.probab2 = predict(glm.fit2 ,test, type="response")
> glm.predict2 = rep("mgr", 35)
> glm.predict2[glm.probab2 > 0.5] = "ent"
> table(glm.predict2, status.test)
status.test
glm.predict2 mgr ent
ent   7  19
mgr   7   2
> accuracy2 = mean(glm.predict2 == status.test)
> accuracy2
[1] 0.7428571
```

In this case, we see that the model predicts correctly the status of the individuals with a percentage of 74.29%. In other words, the accuracy of the model falls from 77.14%, which had been achieved in the first model where all of the predictors were included, to 74.29%. Generally, a decrease in the accuracy of the model is not the desirable, so one could conclude that the first model is the most appropriate. Nevertheless, we should consider the fact that in the second model, the status depends on just two predictors out of nine we had in the beginning. Under this point of view, we understand the advantage of the second model in the predicting process. Additionally, the difference between the accuracies of the two models is not so important. We prefer to sacrifice hardly 3% of the accuracy of the model, than to keep the complex one in our analysis.
Another point worth noting here is that the glm.fit2 model correctly predicts 19 out of 21 entrepreneurs, which is a really high true positive rate (sensitivity), while for the case of managers, things are not so good, as half of them are classified correctly and half of them incorrectly.

To sum up, from both the AIC criterion and the satisfying accuracy of the second model, where only the significant predictors are included, we choose it as the most appropriate logistic regression model for predicting the status of the individuals. Thus, in order to forecast if an individual is a manager or an entrepreneur, we only need their values related to: Value Propositions and Cost Structure. High values of them are more likely to predict an individual as an entrepreneur, whereas low values of

them classify him as a manager.

Finally, we examine the ROC graphical method to detect the total proportion of correctly classified observations via the second model.

```
> roc(status[train],fitted.values(glm.fit2),smooth=TRUE,plot=TRUE,asp=NA)

Call:
roc.default(response=status[train],predictor=fitted.values(glm.fit2),smooth=TRUE,plot=TRUE,asp=NA)

Data: fitted.values(glm.fit2) in 49 controls (status[train] mgr) < 87 cases (status[train] ent).
Smoothing: binormal
Area under the curve: 0.8508
```

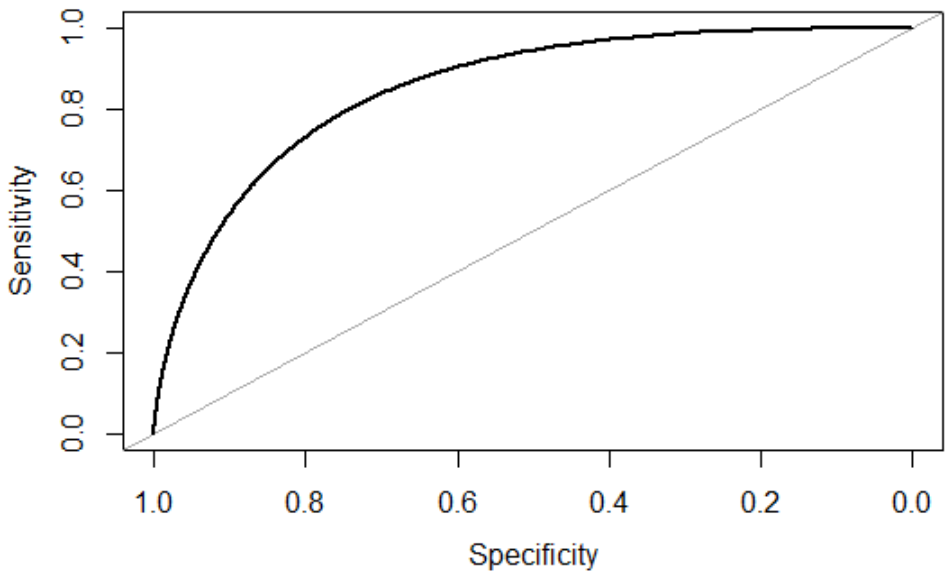

Figure 2.5: ROC for the glm.fit2 model.

As we see, the AUC drops from 0.889 to 0.8508 which does not deter us from selecting the second model with the two predictors in it, for all the reasons we mentioned previously.

To be careful with our analysis, we will also examine the corresponding results for the case where we fit a logistic regression model to the Irish data with the most significant predictors for a significance level of 10%, instead of 5%. This is because, by excluding some of the predictors, the performance of the prediction model may be decreased significantly.

Applying Logistic Regression with four variables Thus, based on the summary of the model glm.fit1 where all of the predictors were included after the splitting of the data had been applied, we had seen that significant variables in a significant level of 10% were the following: Value Propositions, Revenue Streams, Key Resources, Cost Structure. Then, the logistic regression with these four predictors gives us the following results:

```
> glm.fit3 = glm(status ~ V_Prop+Rev_Str+Key_Res+Cost_Str ,
+               data=irish,
+               family = binomial,
+               subset=train)
> summary(glm.fit3)

Call:
glm(formula = status ~ V_Prop + Rev_Str + Key_Res + Cost_Str,
family = binomial, data = irish, subset = train)

Deviance Residuals:
Min       1Q   Median       3Q      Max
-2.2955  -0.5407   0.2927   0.6309   1.8520

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -14.45769    2.73649  -5.283 1.27e-07 ***
V_Prop        0.30726    0.07703   3.989 6.65e-05 ***
Rev_Str       0.15667    0.07410   2.114  0.03450 *
Key_Res      -0.19159    0.08195  -2.338  0.01938 *
Cost_Str      0.17767    0.05689   3.123  0.00179 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 177.78  on 135  degrees of freedom
Residual deviance: 110.65  on 131  degrees of freedom
AIC: 120.65

Number of Fisher Scoring iterations: 5
```

For this model, we see that the value of AIC is the lowest of all that have been fitted. Here AIC is equal to 120.65, whereas in the glm.fit2 with the two predictors the AIC was 125.57. This is a quite noticeable difference. However, we should compare the accuracy of the two models, as well as their ROC curves.

Performance metrics for the four–variable model In this way, to get the accuracy of the model glm.fit3, we follow the same procedure as above:

```
> glm.probab3 = predict(glm.fit3 ,test, type="response")
> glm.predict3 = rep("mgr", 35)
> glm.predict3[glm.probab3 > 0.5] = "ent"
> table(glm.predict3, status.test)
status.test
glm.predict3 mgr ent
ent   7  18
mgr   7   3
> accuracy3 = mean(glm.predict3 == status.test)
> accuracy3
[1] 0.7142857
```

As we see, the accuracy of this model is approximately 0.71, while this of the glm.fit2 model was 0.74. In other words, the model with the two most significant predictors achieves a higher accuracy of about 3%, compared to the model with the four more significant predictors.
Regarding the confusion matrix, we have a similar image, as the model glm.fit3 correctly predicts 18 out of 21 entrepreneurs but only 7 out of 14 managers.

Continuing, regarding the ROC curve, we have:

```
> roc(status[train],fitted.values(glm.fit3),smooth=TRUE,plot=TRUE,asp=NA)

Call:
roc.default(response=status[train],predictor=fitted.values(glm.fit3),smooth=TRUE,plot=TRUE,asp=NA)

Data: fitted.values(glm.fit3) in 49 controls (status[train] mgr) < 87 cases (status[train] ent).
Smoothing: binormal
Area under the curve: 0.8747
```

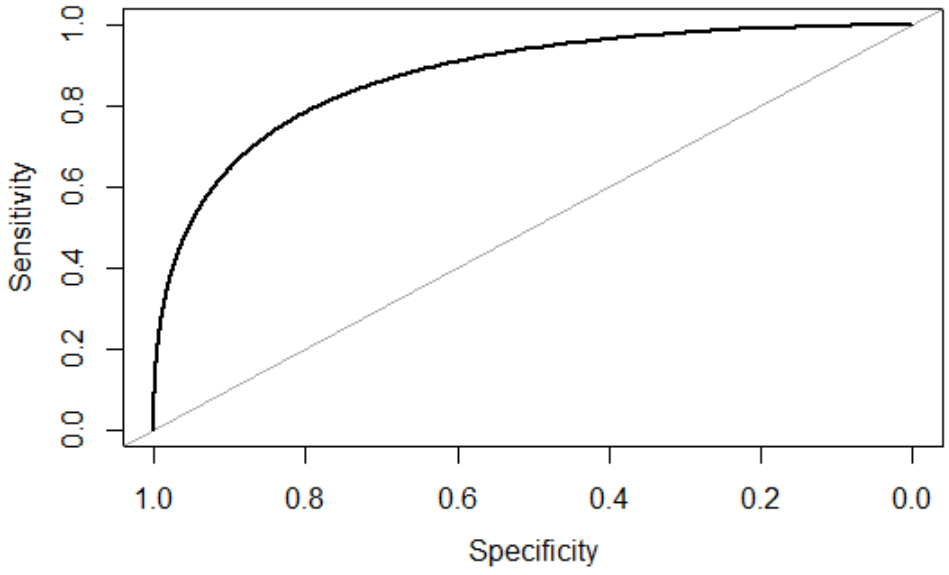

Figure 2.6: ROC for the glm.fit3 model.

The AUC of the ROC here is 0.8747, while the corresponding for the glm.fit2 was 0.8508. This means that the model with the four variables has a better AUC of the ROC by nearly 2.5%.

Model selection All in all, comparing the two models, we have the following results:

	Model	
	glm.fit2	glm.fit3
accuracy	0.74	0.71
ROC	85 .08%	87 .47%
AIC	125. 57	120. 65

Table 2.2: Comparing the two models with the two and four most significant predictors, respectively.

Conclusions As we see from the Table 2.2, the model with the four predictors (glm.fit3), has a higher AUC of ROC and a lower value for AIC, which is desirable. On the other hand, the model with the two predictors (glm.fit2) gets a higher accuracy and also has the advantage that includes the fewer variables. As a consequence of the above results, the selection of the most appropriate model is a matter of choice. If we are more interested in fitting the less complex model with the higher accuracy, then the glm.fit2 should be selected. Otherwise, if we do not give such an importance to the number of predictors to be included, but we prefer a model with a better value for ROC and AIC, then glm.fit3 is the most adecuate.

2.3 Linear Discriminant Analysis

2.3.1 Introduction

The *linear discriminant analysis* is preferred to logistic regression for the following reasons:

- When the classes are well–separated, the parameter estimates for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem.
- If n is small and the distribution of the predictors X is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.
- Linear discriminant analysis is popular when we have more than two response classes.

Let us, now, present some definitions.

Prior
Suppose that we wish to classify an observation into one of K classes, where $K \geq 2$. In other words, the qualitative response variable Y can take on K possible distinct and unordered values. Let π_k represent the *overall* or *prior* probability that a randomly chosen observation comes from the kth class; this is the probability that a given observation is associated with the kth category of the response variable Y.

Density Function
Let $f_k(X) = \Pr(X = x|Y = k)$ denote the *density function* of X for an observation that comes from the kth class. In other words, $f_k(x)$ is relatively large if there is a high probability that an observation in the kth class has $X \approx x$, and $f_k(x)$ is small if it is very unlikely that an observation in the kth class has $X \approx x$.

Bayes' Theorem
Bayes' theorem states that:

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)} \tag{2.11}$$

We will use the abbreviation $p_k(X) = Pr(Y = k|X)$. This suggests that instead of directly computing $p_k(X)$ as in Subsection 2.2.1, we can simply plug in estimates of π_k and $f_k(X)$ into (2.11).

Estimation of π_ks
In general, estimating π_k is easy if we have a random sample of Ys from the population: we simply compute the fraction of the training observations that belong to the kth class.

Estimation of $f_k(X)$
Estimating $f_k(X)$ tends to be quite challenging, unless we assume some simple forms for these densities.

Posterior
We refer to $p_k(x)$ as the *posterior* probability that an observation $X = x$ belongs to the kth class (see 2.11). That is, it is the probability that the observation belongs to the kth class, given the predictor value for that observation.

Bayes classifier has the lowest possible error rate out of all classifiers. Therefore, if we can find a way to estimate $f_k(X)$, then we can develop a classifier that approximates the Bayes classifier. [14]

2.3.2 Method

Linear Discriminant Analysis for p = 1

We assume that $p = 1$, that is we have only one predictor. We would like to obtain an estimate for $f_k(x)$ that we can plug into (2.11) in order to estimate $p_k(x)$. We will then classify an observation to the class for which $p_k(x)$ is greatest. In order to estimate f_k, we will first make some assumptions about its form.
Suppose we assume that f_k is *normal* or *Gaussian*. In the one–dimensional setting, the normal density takes the form:

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\Big(- \frac{1}{2\sigma_k^2}(x - \mu_k)^2 \Big), \tag{2.12}$$

where μ_k and σ_k^2 are the mean and variance parameters for the kth class. Let us further assume that $\sigma_1^2 = \ldots = \sigma_K^2$: that is, there is a shared variance term across all K classes, which for simplicity we can denote by σ^2. Plugging (2.12) into (2.11), we find that:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\big(- \frac{1}{2\sigma^2}(x - \mu_k)^2 \big)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\big(- \frac{1}{2\sigma^2}(x - \mu_l)^2 \big)} \tag{2.13}$$

The *Bayes classifier* involves assigning an observation $X = x$ to the class for which (2.13) is largest. Taking the log of (2.13) and rearranging the terms, it is not hard to show that this is equivalent to assigning the observation to the class for which:

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \tag{2.14}$$

is largest.

The *linear discriminant analysis (LDA)* method approximates the Bayes classifier by plugging estimates for π_k, μ_k, and σ^2 into (2.14). In particular, the following estimates are used:

$$\begin{aligned} \hat{\mu}_k &= \frac{1}{n_k} \sum_{i:y_i=k} x_i \\ \hat{\sigma}^2 &= \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2, \end{aligned} \tag{2.15}$$

where n is the total number of training observations, and n_k is the number of training observations in the kth class. The estimate for μ_k is simply the average of all the training observations from the kth class, while $\hat{\sigma}^2$ can be seen as a weighted average of the sample variances for each of the K classes. Sometimes we have knowledge of the class membership probabilities $\pi_1, \ldots, \pi_K$, which can be used directly. In the absence of any additional information, LDA estimates π_k using the proportion of the training observations that belong to the kth class. In other words,

$$\hat{\pi}_k = n_k / n. \tag{2.16}$$

The LDA classifier plugs the estimates given in (2.15) and (2.16) into (2.14), and assigns an observation $X = x$ to the class for which

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k) \tag{2.17}$$

is largest. The word linear in the classifier's name stems from the fact that the *discriminant functions* $\hat{\delta}_k(x)$ in (2.17) are linear functions of x (as opposed to a more complex function of x).

To reiterate, the LDA classifier results from assuming that the observations within each class come from a normal distribution with a class–specific mean vector and a common variance σ^2, and plugging estimates for these parameters into the Bayes classifier.

Linear Discriminant Analysis for p > 1

We now extend the LDA classifier to the case of multiple predictors. To do this, we will assume that $X = (X_1, X_2, \ldots, X_p)$ is drawn from a *multivariate Gaussian* (or *multivariate normal*) distribution, with a class–specific multivariate Gaussian mean vector and a common covariance matrix.
The multivariate Gaussian distribution assumes that each individual predictor follows a one–dimensional normal distribution, as in (2.12), with some correlation between each pair of predictors.
To indicate that a p–dimensional random variable X has a multivariate Gaussian distribution, we write $X \sim \mathcal{N}(\mu, \mathbf{\Sigma})$. Here $\mathrm{E}(X) = \mu$ is the mean of X (a vector with p components), and $\mathrm{Cov}(X) = \mathbf{\Sigma}$ is the $p \times p$ covariance matrix of X. Formally, the multivariate Gaussian density is defined as

$$f(x) = \frac{1}{(2\pi)^{\frac{p}{2}} |\mathbf{\Sigma}|^{\frac{1}{2}}} \exp\Big(-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)\Big). \tag{2.18}$$

In the case of $p > 1$ predictors, the LDA classifier assumes that the observations in the kth class are drawn from a multivariate Gaussian distribution $\mathcal{N}(\mu_k, \mathbf{\Sigma})$, where μ_k is a class–specific mean vector, and $\mathbf{\Sigma}$ is a covariance matrix that is common to all K classes. Plugging the density function for the kth class, $f_k(X = x)$, into (2.11) and performing a little bit of algebra reveals that the Bayes classifier assigns an observation $X = x$ to the class for which

$$\delta_k(x) = x^T \mathbf{\Sigma}^{-1} \mu_k - \frac{1}{2} \mu_k^T \mathbf{\Sigma}^{-1} \mu_k + \log \pi_k \tag{2.19}$$

is largest. This is the vector/matrix version of (2.14).

Once again, we need to estimate the unknown parameters $\mu_1, \ldots, \mu_K$, $\pi_1, \ldots, \pi_K$, and $\mathbf{\Sigma}$; the formulas are similar to those used in the one–dimensional case, given in (2.15). To assign a new observation $X = x$, LDA plugs these estimates into (2.19) and classifies to the class for which $\hat{\delta}_k(x)$ is largest. Note that in (2.19) $\delta_k(x)$ is a linear function of x; that is, the LDA decision rule depends on x only through a linear combination of its elements. Once again, this is the reason for the word linear in LDA. [14]

2.3.3 Application to Irish data

In this subsection we will perform Linear Discriminant Analysis (LDA) on the Irish data, introduced in Section 1.4.

Firstly, we fit a LDA model using the *lda*() function, whose syntax is identical to that of *glm*(). We fit the model using the training set that we had created in the Subsection 2.2.4.
We will apply the LDA to the two models we had concluded to, in the Subsection 2.2.4: the one which includes Value Propositions and Cost Structure and the other which includes Value Propositions, Revenue Streams, Key Resources and Cost Structure. For the application of LDA in R, the library "MASS" has to be loaded.

Examining the model with the two most significant predictors.

Fitting the linear discriminant model with the two most significant predictors we get:

```
> lda.fit2 <- lda(status ~ V_Prop+Cost_Str,
+                 data=irish,
+                 subset=train)
> lda.fit2
Call:
lda(status ~ V_Prop + Cost_Str, data = irish, subset = train)

Prior probabilities of groups:
mgr       ent
0.3602941 0.6397059

Group means:
V_Prop Cost_Str
mgr 31.20408 28.97959
ent 36.13793 35.66667

Coefficients of linear discriminants:
LD1
V_Prop   0.18307987
Cost_Str 0.09538782
```

The LDA output indicates that $\hat{\pi}_1 = 0.36$ and $\hat{\pi}_2 = 0.64$. In other words, 36% of the training observations correspond to managers. It also provides the group means. These are the average of each predictor within each class, and are used by LDA as estimates of μ_k. What we see is that, when the observation is an entrepreneur, both variables get a higher value compared to the case when the observation is a manager. The coefficients of linear discriminants output provides the linear combination of Value Proposition and Cost Structure that are used to form the LDA decision rule. In other words, if $0.183 \times V\_Prop + 0.095 \times Cost\_Str$ is large, then the LDA classifier will predict an individual as an entrepreneur, while if it is small, then the LDA classifier will predict an individual as a manager.

The *predict*() function returns a list with three elements, as it is shown below:

```
> lda.pred2 = predict(lda.fit2, test)
> names(lda.pred2)
[1] "class"     "posterior" "x"
```

For these elements we have that:

- The first element, class, contains LDA's predictions about the status of the individual.
- The second element, posterior, is a matrix whose kth column contains the posterior probability that the corresponding observation belongs to the kth class.
- The third element, x contains the linear discriminants, described earlier.

Performance metrics for the two–variable model Moving on, we compute the accuracy of this model, as it is shown below:

```
> lda.class2 = lda.pred2$class
> table(lda.class2, status.test)
status.test
lda.class2 mgr ent
mgr   7   2
ent   7  19
> acc2 <- mean(lda.class2 == status.test)
> acc2
[1] 0.7428571
```

From the above result, we observe that the accuracy that is being achieved through the application of LDA is exactly the same with the accuracy we got through the application of the logistic regression.

Applying a 50% threshold to the posterior probabilities allows us to recreate the predictions contained in lda.pred2$class.

```
> sum(lda.pred2$posterior[ ,1] >= 0.5)
[1] 9
> sum(lda.pred2$posterior[ ,1] < 0.5)
[1] 26
```

Here, the posterior probability output by the model corresponds to the probability that the individual will be a manager. We will compute the first ten posterior probabilities for the individual to be a manager, as well as the prediction of the status of each individual. The code in R is as follows:

```
> lda.pred2$posterior[1:10, 1]
2          3         12         13         14         40         41         44         49
0.41647566 0.41072263 0.29117043 0.35532999 0.28388177 0.38404715 0.23015402 0.52295971 0.62739678
56
0.05040484
> lda.class2[1:10]
[1] ent ent ent ent ent ent ent mgr mgr ent
Levels: mgr ent
```

If we wanted to use a posterior probability threshold other than 50% in order to make predictions, then we could easily do so. For instance, suppose that we wish to predict a manager only if we are very certain that the individual is indeed a manager – say, if the posterior probability is at least 90%.

```
> sum(lda.pred2$posterior[ ,1] >= 0.9)
[1] 0
```

What we get, after considering a threshold of 90%, is that none of the individuals is classified as a manager.

Examining the model with the four most significant predictors.

The linear discriminant model for the four most significant predictors is as follows:

```
> lda.fit3 <- lda(status ~ V_Prop+Rev_Str+Key_Res+Cost_Str,
+                 data=irish,
+                 subset=train)
> lda.fit3
Call:
lda(status ~ V_Prop + Rev_Str + Key_Res + Cost_Str, data = irish,
subset = train)

Prior probabilities of groups:
mgr       ent
0.3602941 0.6397059

Group means:
V_Prop  Rev_Str  Key_Res Cost_Str
mgr 31.20408 29.38776 30.42857 28.97959
ent 36.13793 34.70115 33.93103 35.66667

Coefficients of linear discriminants:
LD1
V_Prop    0.17372709
Rev_Str   0.10040681
Key_Res  -0.10086966
Cost_Str  0.09784925
```

The LDA output, here, indicates that $\hat{\pi}_1 = 0.36$ and $\hat{\pi}_2 = 0.64$. We notice that the results in this case, regarding the prior probabilities and the group means of the variables Value Propositions and Cost Structure–which are the two most significant predictors,included in the previous model–are identical with those in the lda.fit2. What changes here is the LDA rule. More specifically, we have $LD1 = 0.174 \times V\_Prop + 0.1 \times Rev\_Str - 0.101 \times Key\_Res + 0.098 \times Cost\_Str$. As we know, if LD1 is large, then the LDA classifier will predict an individual as an entrepreneur, while if it is small, then the LDA classifier will predict an individual as a manager.

Performance metrics for the four–variable model Let us now compute the accuracy of this model:

```
> lda.class3 = lda.pred3$class
> table(lda.class3, status.test)
status.test
lda.class3 mgr ent
mgr   7   3
ent   7  18
> acc3 <- mean(lda.class3 == status.test)
> acc3
[1] 0.7142857
```

As we had observed in the lda.fit2, the accuracy of the model lda.fit3 is exactly the same with the accuracy we got through the application of the logistic regression in these four predictors.

Next, considering a 50% threshold to the posterior probabilities, we get the predictions contained in lda.pred3$class. The code is the following:

```
> sum(lda.pred3$posterior[ ,1] >= 0.5)
[1] 10
> sum(lda.pred3$posterior[ ,1] < 0.5)
[1] 25
```

Here, we see that this model predicts 10 of the 35 individuals to be a manager and 25 to be an entrepreneur, while the lda.fit2 model predicted 9 and 26, respectively.

Continuing, we will, again, compute the first ten posterior probabilities for the individual to be a manager, as well as the prediction of the status of each individual. For this, we have:

```
> lda.pred3$posterior[1:10, 1]
2          3         12         13         14         40         41         44         49
0.41384442 0.31835083 0.57149064 0.36159888 0.37794124 0.38345950 0.21400168 0.10895106 0.63893481
56
0.03408866
> lda.class3[1:10]
[1] ent ent mgr ent ent ent ent ent mgr ent
Levels: mgr ent
```

Let us, in this case too, take a look at the number of individuals to be classified as managers, if the posterior probability is at least 90%.

```
> sum(lda.pred3$posterior[ ,1] >= 0.9)
[1] 2
```

Remark Here, we resulted in a quite noticeable difference in output: after considering a threshold of 90%, 2 out of the 35 individuals were predicted as managers, whereas in the case of the lda.fit2 we had seen that for such a threshold none of the individuals was classified as a manager.

2.4 Quadratic Discriminant Analysis

2.4.1 Method

Quadratic discriminant analysis (QDA) provides an alternative approach to LDA, which assumes that the observations within each class are drawn from a multivariate Gaussian distribution with a class specific mean vector and a covariance matrix that is common to all K classes. Like LDA, the QDA classifier results from assuming that the observations from each class are drawn from a Gaussian distribution, and plugging estimates for the parameters into Bayes' theorem in order to perform prediction. However, unlike LDA, QDA assumes that each class has its own covariance matrix. That is, it assumes that an observation from the kth class is of the form $X \sim \mathcal{N}(\mu_k, \mathbf{\Sigma}_k)$, where $\mathbf{\Sigma}_k$ is a covariance matrix for the kth class. Under this assumption, the Bayes classifier assigns an observation $X = x$ to the class for which

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}_k^{-1}(x-\mu_k) - \frac{1}{2}\log|\mathbf{\Sigma}_k| + \log\pi_k \\ &= -\frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}x + x^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\log|\mathbf{\Sigma}_k| + \log\pi_k \qquad (2.20)\end{aligned}$$

is largest. So the QDA classifier involves plugging estimates for $\mathbf{\Sigma}_k$, μ_k, and π_k into (2.20), and then assigning an observation $X = x$ to the class for which this quantity is largest. Unlike in (2.19), the quantity x appears as a quadratic function in (2.20). This is where QDA gets its name. [14]

2.4.2 Application to Irish data

In this subsection, we will fit a QDA model to the Irish data. QDA is implemented in R using the *qda*() function, whose syntax is identical to that of *lda*(). Similarly to the Subsection 2.3.3, we will fit the model using the training set that we had created in the Subsection 2.2.4, and apply this method to the two models: the one that contains the variables Value Propositions and Cost Structure and the other which includes Value Propositions, Revenue Streams, Key Resources and Cost Structure. For the application of QDA in R, the library "MASS" has to be loaded, too.

Examining the model with the two most significant predictors.

Fitting the quadratic discriminant model with the two most significant predictors we get:

```
> qda.fit2 <- qda(status ~ V_Prop+Cost_Str,
+                 data=irish,
+                 subset=train)
> qda.fit2
Call:
qda(status ~ V_Prop + Cost_Str, data = irish, subset = train)

Prior probabilities of groups:
mgr       ent
0.3602941 0.6397059

Group means:
V_Prop Cost_Str
mgr 31.20408 28.97959
ent 36.13793 35.66667
```

The output contains the group means, which are the same with the corresponding means of the linear discriminant model, as well as the prior probabilities of the groups, which are also the same with those of the LDA. However, this output does not contain the coefficients of the linear discriminants, because the QDA classifier involves a quadratic, rather than a linear, function of the predictors.

The *predict*() function works in exactly the same fashion as for LDA, as is shown below:

```
> qda.pred2 = predict(qda.fit2, test)
> names(qda.pred2)
[1] "class"     "posterior"
```

The difference of the *predict*() function between the two methods is that in the case of the QDA, it returns a list with two elements, instead of three we had in the case of LDA. This is due to the fact that, as we have already commented above, QDA does not contain linear discriminants.

Performance metrics to two–variable model Moving on, we calculate the accuracy of the model:

```
> qda.class2 = qda.pred2$class
> table(qda.class2, status.test)
status.test
qda.class2 mgr ent
```

```
mgr   7   2
ent   7  19
> accur2 <- mean(qda.class2 == status.test)
> accur2
[1] 0.7428571
```

The accuracy of this model is exactly the same with the accuracy of the corresponding linear discriminant model. This means that the quadratic form assumed by QDA does not improve at all the accuracy of the linear discriminant and logistic regression models. Thus, we do not have any reason to apply this method in the Irish data, taking into account its complexity as a model.

Examining the model with the four most significant predictors.

Let us follow the same procedure, fitting the quadratic discriminant model with the four most significant predictors to check if the inference will be the same:

```
> qda.fit3 <- qda(status ~ V_Prop+Rev_Str+Key_Res+Cost_Str,
+                 data=irish,
+                 subset=train)
> qda.fit3
Call:
qda(status ~ V_Prop + Rev_Str + Key_Res + Cost_Str, data = irish,
subset = train)

Prior probabilities of groups:
mgr       ent
0.3602941 0.6397059

Group means:
V_Prop  Rev_Str  Key_Res Cost_Str
mgr 31.20408 29.38776 30.42857 28.97959
ent 36.13793 34.70115 33.93103 35.66667
```

The output, here, regarding the prior probabilities and the group means of the predictors, is the same as in the LDA.

Performance metrics for the four–variable model Continuing, we calculate the accuracy of the model:

```
> qda.pred3 = predict(qda.fit3, test)
> names(qda.pred3)
[1] "class"     "posterior"
> qda.class3 = qda.pred3$class
> table(qda.class3, status.test)
status.test
qda.class3 mgr ent
mgr   8   3
ent   6  18
> accur3 <- mean(qda.class3 == status.test)
> accur3
[1] 0.7428571
```

Conclusions What we notice, here, is that the accuracy of the quadratic discrimination model, applied to the four most significant predictors (qda.fit3), is greater than the accuracy of the linear discrimination model applied to the four most significant predictors, too (lda.fit3). Especially, we see that the value of the accuracy of qda.fit3 is exactly the same with the accuracy of the model applied to the two most significant predictors with both LDA and QDA (lda.fit2 and qda.fit2, respectively). Additionally, we have to notice that through the model qda.fit3 is the first time we achieve a higher sensitivity for the managers' case, as 8 out of 14 are correctly classified, whereas in every other case 7 of them where correctly classified.
Concluding, if we decide to include the four most significant predictors in our analysis, then this suggests that the quadratic form assumed by QDA may capture the true relationship more accurately than the linear form assumed by LDA and logistic regression.

Chapter 3

K-Nearest Neighbors: A Non–Parametric Classifier

3.1 Introduction

As we have discussed in Subsection 2.3.1 the Bayes classifier has the lowest possible error rate out of all classifiers. In theory we would always like to predict qualitative responses using the Bayes classifier. But for real data, we do not know the conditional distribution of Y given X, and so computing the Bayes classifier is impossible. Therefore, the Bayes classifier serves as an unattainable gold standard against which to compare other methods. Many approaches attempt to estimate the conditional distribution of Y given X, and then classify a given observation to the class with highest estimated probability. One such method is the *K-nearest neighbors (KNN)* classifier [14].
KNN is a memory–based classifier, and require no model to be fit [11]
.

3.2 Method

Given a positive integer K and a test observation x_0, the KNN classifier first identifies the K points in the training data that are closest to x_0, represented by $\mathcal{N}_0$ [14].
For simplicity we will assume that the features are real–valued, and we use Euclidean distance in feature space:

$$d_{(i)} = \|x_{(i)} - x_0\|.$$

[11]
It then estimates the conditional probability for class j as the fraction of points in $\mathcal{N}_0$ whose response values equal j:

$$\Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j). \tag{3.1}$$

Finally, KNN applies Bayes rule and classifies the test observation x_0 to the class with the largest probability [14].

Remark

- Other distances–apart from Euclidean distance–that are used for the case of real-valued features are: *Manhattan distance* or *city–clock* and *Minkowski distance*.
- For the case of the qualitative features the *Hamming distance* is used, according to which the number of features where there is a discrepancy between the two cases is calculated.

- Other distances can be defined for the case where both qualitative and quantitative variables are included. Such a distance is *Gower distance*.
- *Weighted distances* can be used, where weights are assigned to each attribute. [32]

The choice of K has a drastic effect on the KNN classifier obtained. When $K = 1$, the decision boundary is overly flexible and finds patterns in the data that don't correspond to the Bayes decision boundary. This corresponds to a classifier that has low bias but very high variance. Thus, in the case where K is small an overfitting is quite probable. As K grows, the method becomes less flexible and produces a decision boundary that is close to linear. This corresponds to a low–variance but high–bias classifier.
The most appropriate K is that one that minimizes the classification error. [14]

Generally, for much more complicated decision boundaries (in the case of KNN no assumptions are made about the shape of the decision boundary), a non–parametric approach such as KNN can be superior to parametric approaches such as those analysed in Sections 2.2, 2.3, 2.4. However, KNN does not tell us which predictors are important [14].

3.3 Application to Irish data

In this section we will perform KNN in the Irish data.
We will use the *knn*() function. For this purpose "class" library is needed. This function works rather differently from the other model fitting functions that we have used thus far in our analyses. Rather than a two–step approach in which we first fit the model and then we use the model to make predictions, *knn*() forms predictions using a single command. The function requires four inputs:

1. A matrix containing the predictors associated with the training data, labeled train.X below.
2. A matrix containing the predictors associated with the data for which we wish to make predictions, labeled test.X below.
3. A vector containing the class labels for the training observations, labeled train.status below.
4. A value for k, the number of nearest neighbors to be used by the classifier.

Similarly to the previous sections of Chapter 2, we will apply the KNN method to the two models: the one that contains the variables Value Propositions and Cost Structure and the other which includes Value Propositions, Revenue Streams, Key Resources and Cost Structure.

Examining the model with the two most significant predictors.

We use the *cbind*() function, short for column bind, to bind the Value Propositions and Cost Structure variables together into two matrices, one for the training set and the other for the test set.

```
> train.X2 = cbind(irish$V_Prop, irish$Cost_Str)[train, ]
> test.X2 = cbind(irish$V_Prop, irish$Cost_Str)[-train, ]
> train.status = status[train]
> status.test = status[-train]
```

Now the *knn*() function can be used to predict the status of the individuals that belong to the test set. We set a random seed before we apply *knn*() because if several observations are tied as nearest neighbors, then R will randomly break the tie. Therefore, a seed must be set in order to ensure reproducibility of results.

```
> set.seed(2)
> knn.pred2_1 = knn(train.X2, test.X2, train.status, k = 1)
> table(knn.pred2_1 , status.test)
status.test
knn.pred2_1 mgr ent
mgr   6   6
ent   8  15
> acc.knn2_1 <- mean(knn.pred2_1 == status.test)
> acc.knn2_1
[1] 0.6
```

The results using $k = 1$ are worse than the previous methods, since only 60% of the observations are correctly predicted. Of course, it may be that $k = 1$ results in an overly flexible fit to the data.

In order to find the best tuning parameter k that maximizes the model accuracy, avoiding the several trials, we will create the following function in R:

```
> calc_acc = function(actual, predicted) {
+   mean(actual == predicted)
+ }
```

Then, using the *calc_ acc* function that we created, we apply the following code:

```
> set.seed(2)
> k_to_try = 1:100
> acc_k = rep(x = 0, times = length(k_to_try))
>
> for (i in seq_along(k_to_try)) {
+   pred = knn(train.X2, test.X2, train.status, k = k_to_try[i])
+   acc_k[i] = calc_acc(status.test, pred)
+ }
```

Continuing, we plot the k–nearest neighbors results as it is shown in Figure 3.1. The code for the specific plot is as follows:

```
> plot(acc_k, type = "b", col = "dodgerblue", cex = 1, pch = 20,
+      xlab = "k, number of neighbors", ylab = "accuracy",
+      main = "Accuracy vs Neighbors")
> abline(h = max(acc_k), col = "darkorange", lty = 3)
```

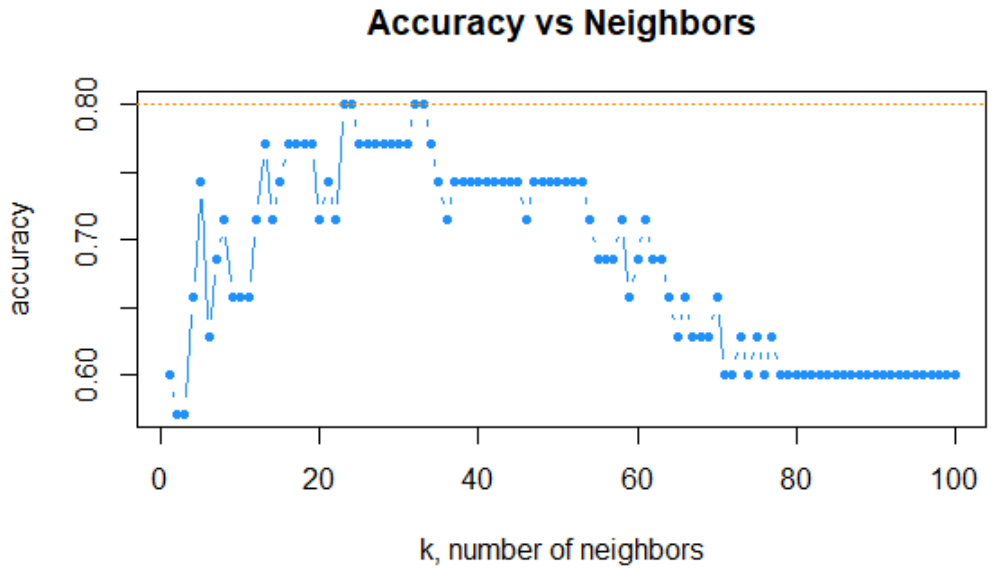

Figure 3.1: Plot Accuracy vs k: model with 2 variables.

The dotted orange line represents the greatest accuracy.

Moving on, we calculate the highest value of the accuracy, as well as the corresponding values of k, in which the highest accuracy is achieved. The code and its results are the following:

```
> max(acc_k)
[1] 0.8
> which(acc_k == max(acc_k))
[1] 23 24 32 33
```

We see, here, that the highest accuracy (80%) can be achieved for four values of k. We select the largest, as it is the least variable, and has the least chance of overfitting. In other words, we choose $k = 33$ for our model.

So now that we concluded that the best tuning parameter k that maximizes the model accuracy is equal to 33, we repeat the analysis for $k = 33$ and we get:

```
> knn.pred2_2 = knn(train.X2, test.X2, train.status , k = 33)
> table(knn.pred2_2 , status.test)
status.test
knn.pred2_2 mgr ent
mgr   7   0
ent   7  21
> acc.knn2_2 <- mean(knn.pred2_2 == status.test)
> acc.knn2_2
[1] 0.8
```

The results have improved a lot. Additionally, we notice that with this model, we got the highest accuracy of all the classifiers that have been examined so far.

Examining the model with the four most significant predictors.

Now, we bind the Value Propositions, Revenue Streams, Key Resources and Cost Structure variables together into two matrices, one for the training set and the other for the test set. The code is as follows:

```
> train.X3 = cbind(irish$V_Prop,irish$Rev_Str,irish$Key_Res, irish$Cost_Str)[train, ]
> test.X3 = cbind(irish$V_Prop,irish$Rev_Str,irish$Key_Res,irish$Cost_Str)[-train, ]
> train.status = status[train]
> status.test = status[-train]
```

Next, we compute the accuracy of this model as follows:

```
> set.seed(2)
> knn.pred3_1 = knn(train.X3,test.X3, train.status, k=1)
> table(knn.pred3_1 , status.test)
status.test
knn.pred3_1 mgr ent
mgr   5   8
ent   9  13
> acc.knn3_1 <- mean(knn.pred3_1 == status.test)
> acc.knn3_1
[1] 0.5142857
```

The results using $k = 1$ are–in this case too–worse than the previous methods, since only 51.43% of the observations are correctly predicted. We also observe that the four–variable model achieves lower accuracy than the two–variable model.

Then, following the same procedure as in the case of the model with the two variables, we use the *calc_ acc* function and we apply the following code:

```
> set.seed(2)
> k_to_try = 1:100
> acc_k = rep(x = 0, times = length(k_to_try))
>
> for (i in seq_along(k_to_try)) {
+   pred = knn(train.X3, test.X3, train.status, k = k_to_try[i])
+   acc_k[i] = calc_acc(status.test, pred)
+ }
```

We then plot the accuracy over the k neighbors and the result is shown in Figure 3.2. The code is the following:

```
> plot(acc_k, type = "b", col = "dodgerblue", cex = 1, pch = 20,
+      xlab = "k, number of neighbors", ylab = "accuracy",
+      main = "Accuracy vs Neighbors")
> abline(h = max(acc_k), col = "darkorange", lty = 3)
```

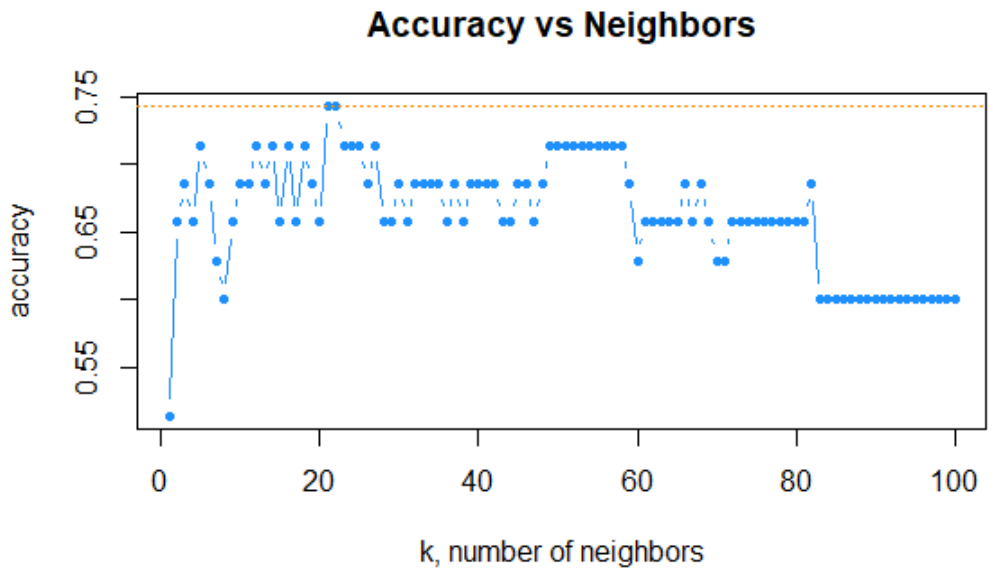

Figure 3.2: Plot Accuracy vs k: model with 4 variables.

```
> max(acc_k)
[1] 0.7428571
> which(acc_k == max(acc_k))
[1] 21 22
```

From the above results, we conclude that the maximum accuracy for the model with the four variables, is approximately 74.29% and is achieved at the k–values 21 and 22. As we have explained,

we choose the higher k.
Thus, applying the model for $k = 22$, the confusion matrix is given below:

```
> knn.pred3_2 = knn(train.X3, test.X3, train.status , k = 22)
> table(knn.pred3_2 , status.test)
status.test
knn.pred3_2 mgr ent
mgr   7   2
ent   7  19
> acc.knn3_2 <- mean(knn.pred3_2 == status.test)
> acc.knn3_2
[1] 0.7428571
> max(acc_k)
[1] 0.7428571
```

In this case, the results have also improved a lot. However, the accuracy of this model is by 6% lower than the model with the two variables.

Conclusions To sum up, we see that the KNN method achieves a better accuracy for both models. More specifically, for the model with the two most significant predictors included (Cost Structure and Value Propositions) the accuracy reaches 80% for the first time and for the model with the four most significant predictors included (Customer Segments,Value Propositions, Revenue Streams and Cost Structure) achieves the higher encountered, so far: approximately 74%. As a consequence, the contribution of the KNN method to the Irish data seems important.

3.4 Application to Khan data

In this section, we will examine the KNN method in a more complex dataset, where the number of the variables is much greater than the number of the observations, and the number of the categories for the observations to be classified is four.

3.4.1 Description of the Khan data

Khan et al. [20] realized a study, whose purpose was to develop a method of classifying cancers to specific diagnostic categories based on their gene expression signatures using artificial neural networks (ANNs).
They used cDNA microarrays containing 6567 clones of which 3789 were known genes and 2778 were ESTs (expressed sequence tags) to study the expression of genes of four types of small round blue–cell tumors of childhood (SRBCTs). These were neuroblastoma (NB), rhabdomyosarcoma (RMS), Burkitt lymphoma, a subset of non–Hodgkin lymphoma (BL), and the Ewing family of tumors (EWS).
Gene expression profiles from both tumor biopsy and cell line samples were obtained and are contained in this dataset. This dataset contains the filtered dataset of 2308 gene expression profiles as described by Khan et al. [20].
Khan is a dataset containing the following four components: xtrain, xtest, ytrain, and ytest. In our analysis we will use the xtrain data–frame, which consists of 64 arrays and 2308 gene expression values.

3.4.2 Statistical Analysis of the Khan data

Exploratory Analysis First we will carry out an exploratory analysis in which we:

- take the first 306 rows.
- transpose the matrix.
- create the variable "tumor".

For these procedures the "dplyr" R–package is used.

Khan dataset was available in the "MADE4" package, containing only 306 genes instead of 2308, but it is no longer. Thus, to be coherent with the "MADE4" package, we will take the first 306 rows of the xtrain data–frame with the following command in R:

```
khan <- read.csv("khan_train.csv", header = T, nrows = 306)
```

Moving on, we transform the data and inspect the dimension of the data–frame as follows:

```
> khan <- khan[ ,-1]
> dim(khan)
[1] 306  64
```

We see that our data–frame has 306 rows and 64 columns. In order to make rows correspond to the observations and the columns to the variables, we have to transpose the matrix, as follows:

```
> khan <- as.data.frame(t(khan))
> dim(khan)
[1]  64 306
```

We check, here, that the dimensions we get are the desirable ones.

Moving on, using the R–package "dplyr" we create a variable called "tumor", which includes the 4 different types of cancer. Below, the code to achieve this transformation and the final dimensions of the matrix are given:

```
> khan <- mutate(khan, tumor = as.factor(c(rep("EWS",23),rep("BL",8),rep("NB",12),rep("RMS",21))))
> dim(khan)
[1]  64 307
```

Applying KNN In this KNN application, where we treat a more complex dataset, we will take advantage of the "caret" package. This is because, "caret" automatically tests different possible values of k, then chooses the optimal k that minimizes the cross–validation ("cv") error, and fits the final best KNN model that explains the best our data.
Additionally "caret" can automatically *preprocess* the data in order to normalize the predictor variables.

So first, through the *createDataPartition* function of the "caret" package, we randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model).

```
> set.seed(3033)
> intrain <- createDataPartition(y = khan$tumor, p= 0.8, list = FALSE)
> training <- khan[intrain,]
> testing <- khan[-intrain,]
```

Now we are just checking the dimensions of the training and test datasets:

```
> dim(training); dim(testing);
[1]  53 307
[1]  11 307
```

Continuing, we fit the model on the training dataset. For this purpose, the *train*() function of the "caret" package is being used. The code is the following:

```
> knn_fit <- train(tumor ~.,
+                  data = training,
+                  method = "knn",
+                  trControl= trainControl("cv", number = 10),
+                  preProcess = c("center", "scale"),
+                  tuneLength = 10)
```

Let us explain the arguments we used in the function *train*():

- *trControl*, to set up ten–fold cross validation.
- *preProcess*, to normalize the data.

- *tuneLength*, to specify the number of possible k values to evaluate.

Then, we inspect the output of the KNN model that we fit:

```
> knn_fit
k-Nearest Neighbors

53 samples
306 predictors
4 classes: 'BL', 'EWS', 'NB', 'RMS'

Pre-processing: centered (306), scaled (306)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 48, 48, 47, 47, 47, 48, ...
Resampling results across tuning parameters:

k   Accuracy   Kappa
5  0.7300000  0.5930124
7  0.6766667  0.5500702
9  0.7333333  0.6166543
11  0.7216667  0.6040481
13  0.6900000  0.5361293
15  0.6166667  0.4502424
17  0.5666667  0.3792232
19  0.5550000  0.3721280
21  0.6316667  0.4869555
23  0.5383333  0.3404726

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

What we can find from the above output, is that the k parameter in which we take the highest accuracy is equal to 9. The accuracy of this model is approximately 73.33%, quite high if we take into account the complexity of the problem we are dealing with in this section.

We can, also, visualize the above output, plotting the accuracy of the KNN model with respect to the k (see Figure 3.3). The command for this plot is the following:

```
> plot(knn_fit)
```

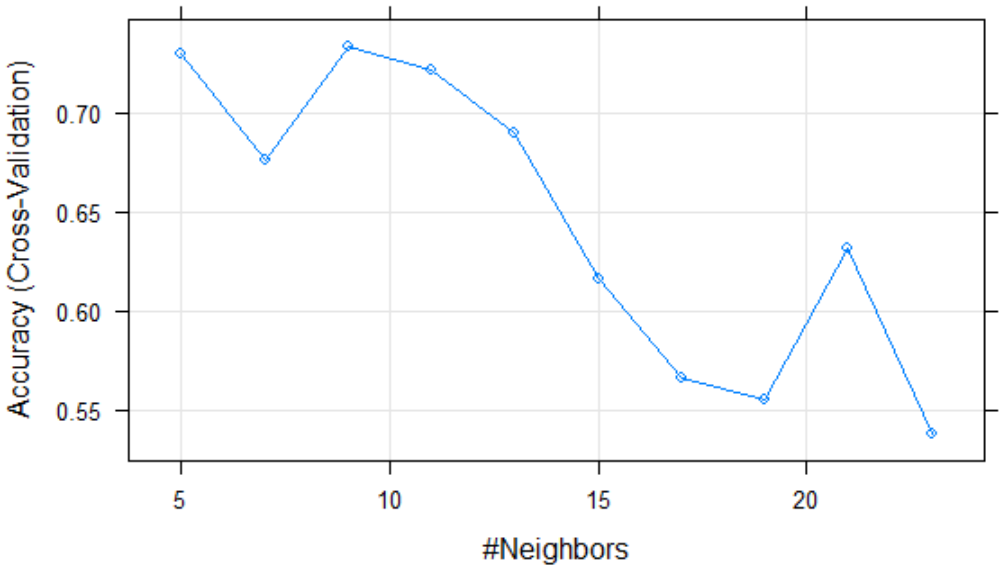

Figure 3.3: Plot Accuracy vs k: Khan data.

Furthermore we can check that, indeed, the best value for the tuning parameter k is 9:

```
> print(knn_fit$bestTune)
k
3 9
```

Additionally, using this KNN classifier we take the predictions for the observations on the test set by using the *predict*() function. The code is the following:

```
> test_pred <- predict(knn_fit, newdata = testing)
> test_pred
[1] EWS EWS EWS EWS BL  NB  NB  NB  RMS EWS EWS
Levels: BL EWS NB RMS
```

Performance metrics Lastly, we can evaluate the performance of our model on the data, inspecting the confusion matrix and the statistics measures. Below the code and its results for this, are shown:

```
> confusionMatrix(test_pred, testing$tumor)
Confusion Matrix and Statistics

Reference
Prediction BL EWS NB RMS
BL   1   0  0   0
EWS  0   4  0   2
NB   0   0  2   1
RMS  0   0  0   1

Overall Statistics

Accuracy : 0.7273
```

```
95% CI : (0.3903, 0.9398)
No Information Rate : 0.3636
P-Value [Acc > NIR] : 0.01577

Kappa : 0.6163
Mcnemar's Test P-Value : NA

Statistics by Class:

Class: BL Class: EWS Class: NB Class: RMS
Sensitivity            1.00000     1.0000    1.0000    0.25000
Specificity            1.00000     0.7143    0.8889    1.00000
Pos Pred Value         1.00000     0.6667    0.6667    1.00000
Neg Pred Value         1.00000     1.0000    1.0000    0.70000
Prevalence             0.09091     0.3636    0.1818    0.36364
Detection Rate         0.09091     0.3636    0.1818    0.09091
Detection Prevalence   0.09091     0.5455    0.2727    0.09091
Balanced Accuracy      1.00000     0.8571    0.9444    0.62500
```

Conclusions From the confusion matrix we can see that the model correctly predicts the true type of tumor for the "BL", "EWS" and "NB" categories, but for the "RMS" type of tumor, the model correctly predicts one out of the four observations that are truly contained in this category, whereas two of them are classified as "EWS" and the fourth as "NB".
Moreover, we can inspect more specifically the sensitivity and specificity of the model. As we expected, the "BL" type of tumor achieves the ideal 1 for both of these measures, while "EWS" and "NB" get quite high values of sensitivity and specificity. However, for the "RMS" tumor the sensitivity is really low, as we explained previously, but the specificity is equal to the ideal 1, as for the observations that do not belong in the "RMS" category the model correctly does not classify them to the "RMS" type of tumor.
Concluding, we check that the accuracy of the model is 72.73%, a quite high accuracy, which enables us to use this KNN model for classification of the different types of tumor.

All in all, the results of this KNN application to the Khan data are really important, as we managed easily to fit a model for accurately distinguishing cancers belonging to several diagnostic categories.

Chapter 4

Methods Based on Trees

4.1 Classification Trees

4.1.1 Introduction

Classification Trees: Part of Tree–Based Methods
Tree–based methods partition the feature/predictor space into a set of rectangles/regions, and then fit a simple model (like a constant) in each one. They are conceptually simple yet powerful [11].
In classification trees method, we predict that a given observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs. Since the set of splitting rules used to segment the feature/predictor space can be summarized in a tree, these types of approaches are known as *decision tree methods* [14].
In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the *class proportions* among the training observations that fall into that region [14].

Let's consider a classification problem with a binary response Y and inputs X_1 and X_2, each taking values in the unit interval. To simplify matters, we restrict attention to recursive binary partitions like that in the Figure 4.1. We first split the space into two regions, and model the response by the most commonly occurring class of training observations in each region of Y. We choose the variable and split–point to achieve the best fit. Then one or both of these regions are split into two more regions, and this process is continued, until some stopping rule is applied. For example, in Figure 4.1 we first split at $X_1 = t_1$. Then the region $X_1 \leq t_1$ is split at $X_2 = t_2$ and the region $X_1 > t_1$ is split at $X_1 = t_3$. Finally, the region $X_1 > t_3$ is split at $X_2 = t_4$. The result of this process is a partition into the five regions $R_1, R_2, \ldots, R_5$ shown in the Figure 4.1.

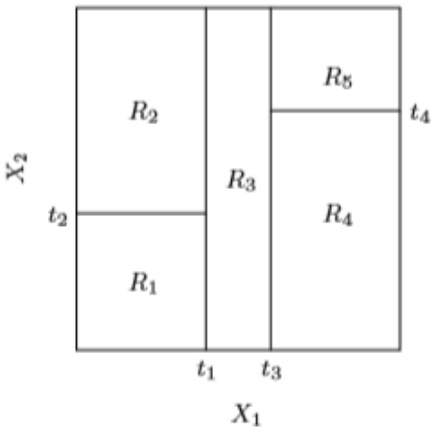

Figure 4.1: Partition of a two–dimensional feature space by recursive binary splitting.

This same model can be represented by the binary tree in the Figure 4.2. The full dataset sits at the top of the tree. Observations satisfying the condition at each junction are assigned to the left branch, and the others to the right branch. The *terminal nodes* or *leaves* of the tree correspond to the regions $R_1, R_2, \ldots, R_5$.
Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree. The points along the tree where the predictor space is split are referred to as internal nodes . In Figure 4.2, the four internal nodes are indicated by the text $X_1 \leq t_1$, $X_2 \leq t_2$, $X_1 \leq t_3$ and $X_2 \leq t_4$. We refer to the segments of the trees that connect the nodes as *branches*.
A key advantage of the recursive binary tree is its interpretability. The feature space partition is fully described by a single tree. [11]

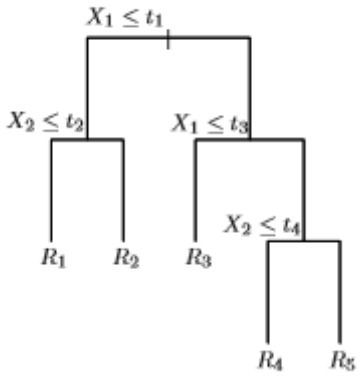

Figure 4.2: Tree corresponding to the partition of Figure 4.1.

4.1.2 Building Classification Trees

The process of building a classification tree includes two steps:

1. We divide the predictor space–that is, the set of possible values for $X_1, X_2, \ldots, X_p$–into J distinct and non–overlapping regions, $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, we make the same prediction, which is simply the most commonly occurring class of training observations in R_j.

We now elaborate on Step 1 above. How do we construct the regions $R_1, \ldots, R_J$? In theory, the regions could have any shape. However, we choose to divide the predictor space into high–dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model. The goal is to find boxes $R_1, \ldots, R_J$ that minimize the classification error rate.
Let us define the *classification error rate*. Since we plan to assign an observation in a given region to the most commonly occurring class of training observations in that region, the classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}). \tag{4.1}$$

Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.

Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes. For this reason, we take a *top–down*, *greedy* approach that is known as *recursive binary splitting*.
The approach is *top–down* because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
It is *greedy* because at each step of the tree–building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

In order to perform recursive binary splitting, we first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in the classification error rate. (The notation $\{X|X_j < s\}$ means the region of predictor space in which X_j takes on a value less than s). That is, we consider all predictors $X_1, \ldots, X_p$, and all possible values of the cutpoint s for each of the predictors, and then choose the predictor and cutpoint such that the resulting tree has the lowest classification error rate. In greater detail, for any j and s, we define the pair of half–planes:

$$R_1(j, s) = \{X|X_j < s\} \text{ and } R_2(j, s) = \{X|X_j \geq s\} \tag{4.2}$$

and we seek the value of j and s that minimize the equation

$$1 - \max_k\{\hat{p}_{R_1k}, \hat{p}_{R_2k}\}, \tag{4.3}$$

where $\hat{p}_{R_1k}$ is the proportion for the training observations in $R_1(j, s)$ region, and $\hat{p}_{R_2k}$ is the proportion for the training observations in $R_2(j, s)$. Finding the values of j and s that minimize (4.3) can be done quite quickly, especially when the number of features p is not too large.
Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the classification error rate within each of the resulting regions. However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions. Again, we look to split one of these three regions further, so as to minimize the classification error rate. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.
Once the regions $R_1, \ldots, R_J$ have been created, we predict the response for a given test observation using the most commonly occurring class of training observations in the region to which that test observation belongs. [14]

Regarding the node purity, there are two more different measures, apart from the classification error rate. Indeed, it turns out that classification error is not sufficiently sensitive for tree–growing, and in practice these two measures are preferable: the *Gini index* and the *cross–entropy*.

- The *Gini index* is defined by:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{4.4}$$

  a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node purity–a small value indicates that a node contains predominantly observations from a single class.

- The cross–entropy is given by:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}. \tag{4.5}$$

  Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$. The cross–entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one. Therefore, like the Gini index, the cross–entropy will take on a small value if the mth node is pure.

In fact, it turns out that the Gini index and the cross–entropy are quite similar numerically. When building a classification tree, either the Gini index or the cross–entropy are typically used to evaluate the quality of a particular split, since these two approaches are more sensitive to node purity than is the classification error rate. Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal. [14]

4.1.3 Tree Pruning

The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. This is because the resulting tree might be too complex. A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
One possible alternative to the process described above is to build the tree only so long as the decrease in the classification error rate due to each split exceeds some (high) threshold. This strategy will result in smaller trees, but is too short–sighted since a seemingly worthless split early on in the tree might be followed by a very good split– that is, a split that leads to a large reduction in the classification error rate later on. Therefore, a better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*.

Determination of the pruning method.
Intuitively, our goal is to select a subtree that leads to the lowest test error rate. Given a subtree, we can estimate its test error using cross–validation or the validation set approach. However, estimating the cross-validation error for every possible subtree would be too cumbersome, since there is an extremely large number of possible subtrees. Instead, we need a way to select a small set of subtrees for consideration. *Cost complexity pruning*–also known as *weakest link pruning*–gives us a way to do

just this. Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that:

$$E = 1 - \max_k(\hat{p}_{mk}) + \alpha|T| \tag{4.6}$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, $\hat{p}_{mk}$ represents the proportion of training observations in the mth terminal node that are from the kth class.
The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. When $\alpha = 0$, then the subtree T will simply equal T_0, because then (4.6) just measures the misclassification error. However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity (4.6) will tend to be minimized for a smaller subtree. It turns out that as we increase α from zero in (4.6), branches get pruned from the tree in a nested and predictable fashion, so obtaining the whole sequence of subtrees as a function of α is easy. We can select a value of α using a validation set or using cross–validation. We then return to the full data set and obtain the subtree corresponding to α. This process is summarized in Algorithm 1. [14]

Algorithm 1 *Building a Classification Pruned Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K–fold cross–validation to choose α. For each $k = 1, \ldots, K$:
   (a) Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   (b) Evaluate the mean squared prediction error on the data in the left–out kth fold, as a function of α.
   Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

4.1.4 Advantages of the Classification Trees Method

- Making predictions is fast. (no complicated calculations, just looking up constants in the tree)
- It's easy to understand what variables are important in making the prediction. (just look at the tree)
- If some data is missing, we might not be able to go all the way down the tree to a leaf, but we can still make a prediction by averaging all the leaves in the sub–tree we do reach.
- The model gives a jagged response, so it can work when the true regression surface is not smooth. If it is smooth, though, the piecewise-constant surface can approximate it arbitrarily closely (with enough leaves)
- There are fast, reliable algorithms to learn these trees. [33]

4.1.5 Application to Irish data

In this subsection we will apply the method of classification trees onto the Irish data.

Exploratory Analysis Firstly, we recode the variable that we want to examine: Status. This variable is already a binary variable. However, for a better interpretation, we take as "ent" (entrepreneur) if the Status variable has the value 2 and "mgr" (manager) if the Status gets the value 1.

The code in R is as follows:

```
> Status = ifelse(irish$Ent_Mgr_Status==2,"ent","mgr")
```

Moving on, we use the $data.frame()$ function to merge the variable Status with the rest of the Irish data.

```
> irish = data.frame(irish ,Status)
```

Classification Tree application Now, we are ready to fit a classification tree in order to predict the Status using all variables but Status. For this purpose, we use the $tree()$ function, whose syntax is quite similar to that of the $lm()$ function. In this procedure, the R–package "tree" was loaded, which enables us to construct classification and regression trees.

```
> tree.irish = tree(Status ~.-Ent_Mgr_Status ,irish )
```

Next, we apply the $summary()$ function, which lists the variables that are used as internal nodes in the tree, the number of terminal nodes, and the (training) error rate.

```
> summary(tree.irish)

Classification tree:
tree(formula = Status ~ . - Ent_Mgr_Status, data = irish)
Variables actually used in tree construction:
[1] "Val_Prop" "Rev_Str"  "Cust_Seg" "Chan"     "Cust_Rel" "Key_Res"  "Cost_Str"
[8] "Key_Part"
Number of terminal nodes:  16
Residual mean deviance:  0.5353 = 82.97 / 155
Misclassification error rate: 0.1228 = 21 / 171
```

We see that the training error rate is approximately 12.3%. The deviance is 53.53%. A small deviance indicates a tree that provides a good fit to the training data. The residual mean deviance reported is simply the deviance divided by $n- \mid T_o \mid$, which in this case is $171 - 16 = 155$. One of the most attractive properties of trees is that they can be graphically displayed. We use the $plot()$ function to display the tree structure, and the $text()$ function to display the node labels. The argument $pretty = 0$ instructs R to include the category names for any qualitative predictors, rather than simply displaying a letter for each category.

Thus, we have:

```
> plot(tree.irish)
> text(tree.irish ,pretty =0)
```

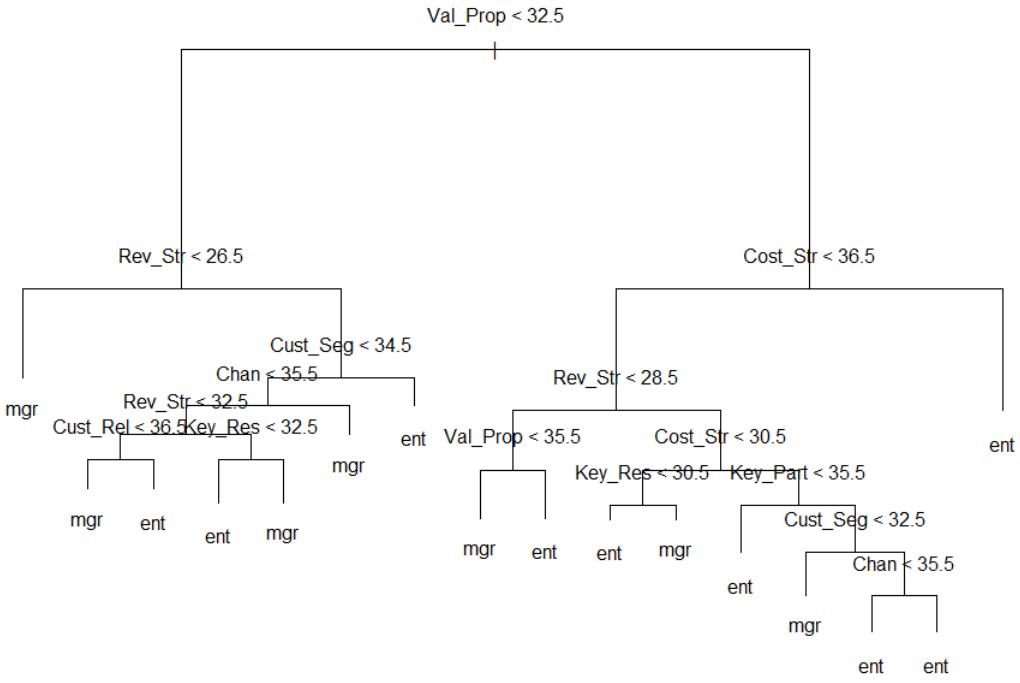

Figure 4.3: Fully Grown Classification Tree for the whole of the data.

The plot shows the different possible splitting rules that can be used to effectively predict the Status. The most important variable that determines the Status appears to be Value Propositions, since the first branch differentiates individuals that have a score lower than 32.5 with those who exceed 32.5.

If we just type the name of the tree object, R prints output corresponding to each branch of the tree. R displays the split criterion (e.g.$Val\_Prop < 32.5$), the number of observations in that branch, the deviance, the overall prediction for the branch (ent or mgr), and the fraction of observations in that branch that take on values of ent and mgr. Branches that lead to terminal nodes are indicated using asterisks. For example, line 4) indicates a terminal node, where the individual is predicted as a manager 100%.

```
> tree.irish
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 171 225.100 ent ( 0.63158 0.36842 )
2) Val_Prop < 32.5 56  68.750 mgr ( 0.30357 0.69643 )
4) Rev_Str < 26.5 16   0.000 mgr ( 0.00000 1.00000 ) *
5) Rev_Str > 26.5 40  54.550 mgr ( 0.42500 0.57500 )
10) Cust_Seg < 34.5 32  41.180 mgr ( 0.34375 0.65625 )
20) Chan < 35.5 27  36.500 mgr ( 0.40741 0.59259 )
```

```
40) Rev_Str < 32.5 16  17.990 mgr ( 0.25000 0.75000 )
80) Cust_Rel < 36.5 11   6.702 mgr ( 0.09091 0.90909 ) *
81) Cust_Rel > 36.5 5   6.730 ent ( 0.60000 0.40000 ) *
41) Rev_Str > 32.5 11  14.420 ent ( 0.63636 0.36364 )
82) Key_Res < 32.5 5   0.000 ent ( 1.00000 0.00000 ) *
83) Key_Res > 32.5 6   7.638 mgr ( 0.33333 0.66667 ) *
21) Chan > 35.5 5   0.000 mgr ( 0.00000 1.00000 ) *
11) Cust_Seg > 34.5 8   8.997 ent ( 0.75000 0.25000 ) *
3) Val_Prop > 32.5 115 117.800 ent ( 0.79130 0.20870 )
6) Cost_Str < 36.5 81  98.450 ent ( 0.70370 0.29630 )
12) Rev_Str < 28.5 16  21.170 mgr ( 0.37500 0.62500 )
24) Val_Prop < 35.5 6   0.000 mgr ( 0.00000 1.00000 ) *
25) Val_Prop > 35.5 10  13.460 ent ( 0.60000 0.40000 ) *
13) Rev_Str > 28.5 65  67.730 ent ( 0.78462 0.21538 )
26) Cost_Str < 30.5 16  21.930 ent ( 0.56250 0.43750 )
52) Key_Res < 30.5 8   8.997 ent ( 0.75000 0.25000 ) *
53) Key_Res > 30.5 8  10.590 mgr ( 0.37500 0.62500 ) *
27) Cost_Str > 30.5 49  40.190 ent ( 0.85714 0.14286 )
54) Key_Part < 35.5 19   0.000 ent ( 1.00000 0.00000 ) *
55) Key_Part > 35.5 30  32.600 ent ( 0.76667 0.23333 )
110) Cust_Seg < 32.5 6   7.638 mgr ( 0.33333 0.66667 ) *
111) Cust_Seg > 32.5 24  18.080 ent ( 0.87500 0.12500 )
222) Chan < 35.5 14   0.000 ent ( 1.00000 0.00000 ) *
223) Chan > 35.5 10  12.220 ent ( 0.70000 0.30000 ) *
7) Cost_Str > 36.5 34   0.000 ent ( 1.00000 0.00000 ) *
```

Splitting the data In order to properly evaluate the performance of a classification tree on these data, we must estimate the test error rather than simply computing the training error. We split the observations into a training set and a test set, build the tree using the training set, and evaluate its performance on the test data. The *predict*() function can be used for this purpose. In the case of a classification tree, the argument *type="class"* instructs R to return the actual class prediction. This approach leads to correct predictions for around 64% of the Status in the test data set.

```
> set.seed(2)
> train=sample(1:nrow(irish), nrow(irish)/2)
> irish.test=irish[-train, ]
> Status.test=Status[-train]
> tree.irish =tree(Status~.-Ent_Mgr_Status ,irish ,subset=train)
> tree.pred=predict(tree.irish ,irish.test ,type="class")
> table(tree.pred ,Status.test)
Status.test
tree.pred ent mgr
ent  36  10
mgr  21  19
> acc = mean(tree.pred == Status.test)
> acc
[1] 0.6395349
```

Next, we consider whether pruning the tree might lead to improved results. The function *cv.tree*() performs cross–validation in order to determine the optimal level of tree complexity: cost complexity pruning is used in order to select a sequence of trees for consideration. We use the argument $FUN = prune.misclass$ in order to indicate that we want the classification error rate to guide the cross-validation and pruning process, rather than the default for the *cv.tree*() function, which is deviance.

The *cv.tree*() function reports the number of terminal nodes of each tree considered (size) as well as the corresponding error rate and the value of the cost–complexity parameter used (k).

```
> set.seed(3)
> cv.irish =cv.tree(tree.irish ,FUN=prune.misclass )
> names(cv.irish)
[1] "size"   "dev"    "k"      "method"
> cv.irish
$size
[1] 10  7  5  4  2  1

$dev
[1] 38 38 38 36 30 37

$k
[1] -Inf  0.0  0.5  1.0  2.0 15.0

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

Note that, despite the name, dev corresponds to the cross–validation error rate in this instance. The tree with two terminal nodes results in the lowest cross–validation error rate, with 30 cross–validation errors. We plot the error rate as a function of both size and k. The code in R is the following:

```
> par(mfrow=c(1,2))
> plot(cv.irish$size ,cv.irish$dev ,type="b")
> plot(cv.irish$k ,cv.irish$dev ,type="b")
```

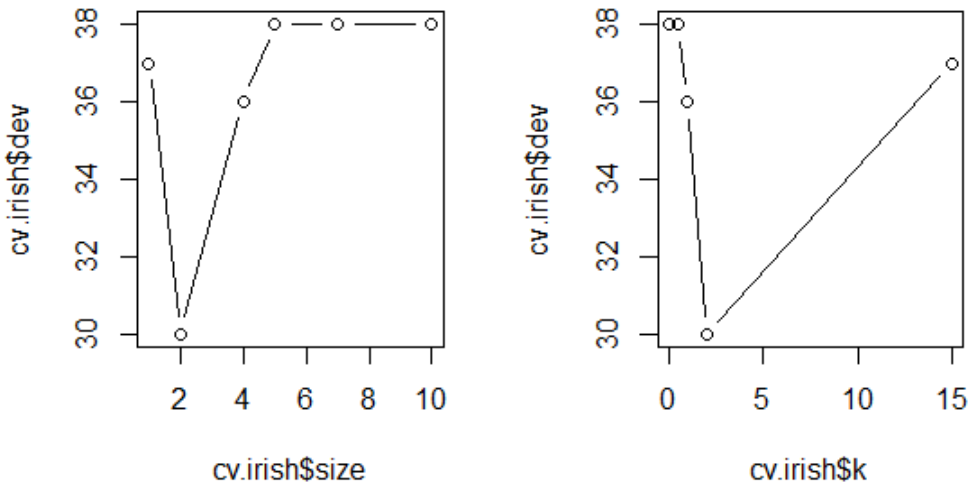

Figure 4.4: Error Rate for size and k.

We now apply the *prune.misclass*() function in order to prune the tree to obtain the two–node tree.

```
prune.irish =prune.misclass (tree.irish ,best=2)
plot(prune.irish)
text(prune.irish ,pretty =0)
```

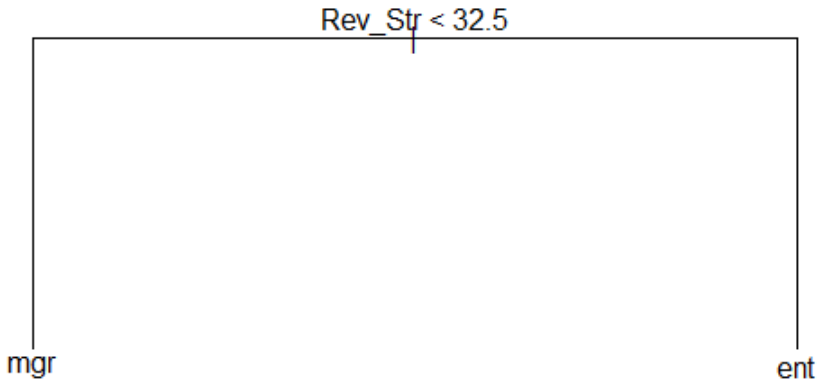

Figure 4.5: Best Classification Pruned Tree for the training data.

Performance metrics of the pruned tree Now, we will check how well this pruned tree performs on the test dataset.

From R we get:

```
> tree.pred=predict(prune.irish ,irish.test ,type="class")
> table(tree.pred ,Status.test)
Status.test
tree.pred ent mgr
ent  39   9
mgr  18  20
> accuracy = mean(tree.pred == Status.test)
> accuracy
[1] 0.6860465
```

As we see, 68.6% of the test observations are correctly classified, so not only has the pruning process produced a more interpretable tree, but it has also improved the classification accuracy. If we increase the value of best, we generally obtain a larger pruned tree with lower classification accuracy.

Inspecting a random tree Let us inspect such a case. Arbitrarily we take for best the value five. Thus, we take:

```
prune.irish =prune.misclass (tree.irish ,best=5)
plot(prune.irish)
text(prune.irish ,pretty =0)
```

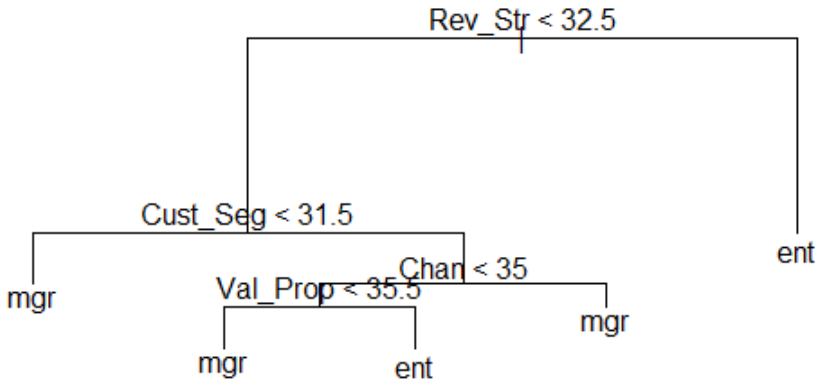

Figure 4.6: Classification Pruned Tree with 5 terminal nodes.

Conclusions In this case, we notice that the accuracy of this pruned tree with five terminal nodes, is exactly the same with this of the pruned tree with two terminal nodes. However, we choose the one with the two terminal nodes, as it is the simplest one.

All in all, we come to the conclusion that for the prediction of the two groups based on the method of classification trees we only need to examine the variable Revenue Streams. If the individual gets a score lower than 32.5 for this variable, we classify him as a manager, whereas if his score is greater than 32.5 regarding the Revenue Streams we predict him as an entrepreneur.

4.2 Regression Trees

4.2.1 Introduction

Regression Trees is part of the tree–based methods, and all the definitions we presented in Section 4.1.1 for the classification trees are the same for the regression trees, as well.

4.2.2 Method

Growing a Regression Tree
Suppose that our data consists of p inputs and a response, for each of N observations: that is, (x_i, y_i) for $i = 1, 2, \ldots, N$, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have. Suppose first that we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m). \tag{4.7}$$

If we adopt as our criterion minimization of sum-of-squares $\sum((y_i - f(x_i))^2$, it is easy to see that the best $\hat{c}_m$ is just the average of y_i in region R_m:

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m). \tag{4.8}$$

Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a greedy algorithm. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half–planes

$$R_1(j, s) = \{X | X_j < s\} \text{ and } R_2(j, s) = \{X | X_j \geq s\} \tag{4.9}$$

Then we seek the splitting variable j and split point s that solve

$$S = \min_{j,s}[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2]. \tag{4.10}$$

For any choice j and s, the inner minimization is solved by

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1) \text{ and } \hat{c}_2 = \text{ave}(y_i | x_i \in R_2) \tag{4.11}$$

For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j, s) is feasible.
Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. Then this process is repeated on all of the resulting regions. [11]

How large should we grow the tree?
A typical stopping criterion is to stop growing the tree when further splits gives less than some minimal amount of extra information, or when they would result in nodes containing less than, say, five percent of the total data.
Clearly a very large tree might overfit the data, while a small tree might not capture the important structure.

The basic *regression–tree–growing algorithm* then is as follows:

1. Start with a single node containing all points. Calculate $\hat{c}_m$ and sum-of-squares.

2. If all the points in the node have the same value for all the independent variables, stop. Otherwise, search over all binary splits of all variables for the one which will reduce sum-of-squares as much as possible. If the largest decrease in sum-of-squares would be less than some threshold δ, or one of the resulting nodes would contain less than q points, stop. Otherwise, take that split, creating two new nodes.

3. In each new node, go back to step 1. [34]

4.2.3 Pruning the Regression Tree

After a large tree T_0 has been grown, then it is pruned using *cost–complexity pruning* with the following way: We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal (non–terminal) nodes. We index terminal nodes by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T. Letting:

$$\begin{aligned} N_m &= \#\{x_i \in R_m\}, \\ \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \end{aligned} \tag{4.12}$$

we define the cost complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|. \tag{4.13}$$

The idea is to find, for each α, the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$. The tuning parameter $\alpha \neq 0$ governs the tradeoff between tree size and its goodness of fit to the data. Large values of α result in smaller trees T_α, and conversely for smaller values of α. As the notation suggests, with $\alpha = 0$ the solution is the full tree T_0.

How to choose α?
For each α one can show that there is a unique smallest subtree T_α that minimizes $C_\alpha(T)$. To find T_α we use *weakest link pruning*: we successively collapse the internal node that produces the smallest per–node increase in $\sum_m N_m Q_m(T)$, and continue until we produce the single–node (root) tree. This gives a (finite) sequence of subtrees, and one can show this sequence must contain T_α. Estimation of α is achieved by five– or tenfold cross–validation: we choose the value $\hat{\alpha}$ to minimize the cross-validated sum of squares. Our final tree is $T_{\hat{\alpha}}$. [11]

4.2.4 Application to Boston Data

In this section we will apply the Regression Trees method to a dataset available in R, and more specifically in the "MASS" library: Boston data.

Presentation of the Boston Data

The Boston data–frame has 506 rows and 14 columns.
Boston dataset records *medv* (median house value) for 506 neighborhoods around Boston. We will seek to predict *medv* using 13 different predictors.
The columns of the Boston data–frame are the following:

- *crim*, per capita crime rate by town.
- *zn*, proportion of residential land zoned for lots over $25,000$ sq.ft.
- *indus*, proportion of non–retail business acres per town.
- *chas*, Charles River dummy variable ($= 1$ if tract bounds river; 0 otherwise).
- *nox*, nitrogen oxides concentration (parts per 10 million).
- *rm*, average number of rooms per dwelling.
- *age*, proportion of owner–occupied units built prior to 1940.
- *dis*, weighted mean of distances to five Boston employment centres.
- *rad*, index of accessibility to radial highways.
- *tax*, full–value property–tax rate per USD $10,000$\$,
- *ptratio*, pupil–teacher ratio by town.
- *black*, $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town.
- *lstat*, percentage of lower status of the population.
- *medv*, median value of owner–occupied homes in $1000s$\$.

Fitting the Regression Tree

In this decision tree application, we will use the R–package "caret" for easier machine learning workflow and the "tidyverse" for easy data manipulation and visualization.

First, we create a training and test set. The code is as follows:

```
> set.seed(123)
> portion <- Boston$medv %>%
+   createDataPartition(p = 0.8, list = FALSE)
> train.data  <- Boston[portion, ]
> test.data <- Boston[-portion, ]
```

Then, we fit the model on the training data. For this procedure we use the *train*() function from "caret" package in which we apply the *rpart* method for automatically testing different possible values of cp (complexity parameter). Here we use the arguments:

- *trControl*, to set up ten–fold cross validation.
- *tuneLength*, to specify the number of possible cp values to evaluate. Here we'll use 10.

Thus the code is the following:

```
> set.seed(123)
> model <- train(
+   medv ~., data = train.data, method = "rpart",
+   trControl = trainControl("cv", number = 10),
+   tuneLength = 10
+ )
```

Continuing, the best cp value is the one that minimize the square root of the MSE (RMSE). The lower the RMSE, the better the model.
So, we choose the cp with the following commands in R:

```
> plot(model)
```

```
> model$bestTune
cp
1 0.007165585
```

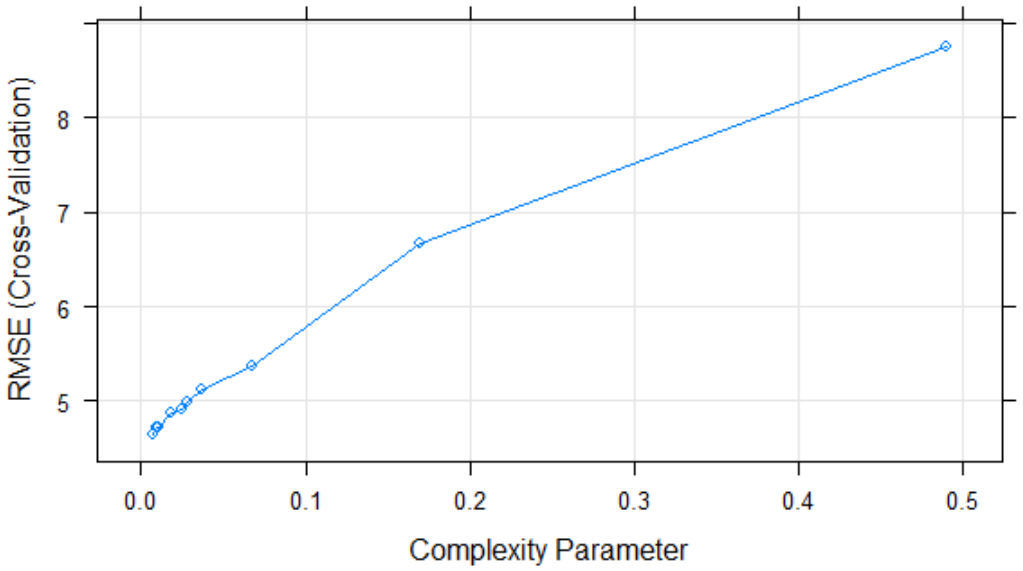

Figure 4.7: Model Error vs cp

In this case, as we notice from Figure 4.7 the most complex tree is selected by cross–validation. Next, we plot the final tree model. For a prettier plot we will use the "rattle" package. The code is the following and the plot is shown in Figure 4.8:

```
> fancyRpartPlot(model$finalModel, sub = NA)
```

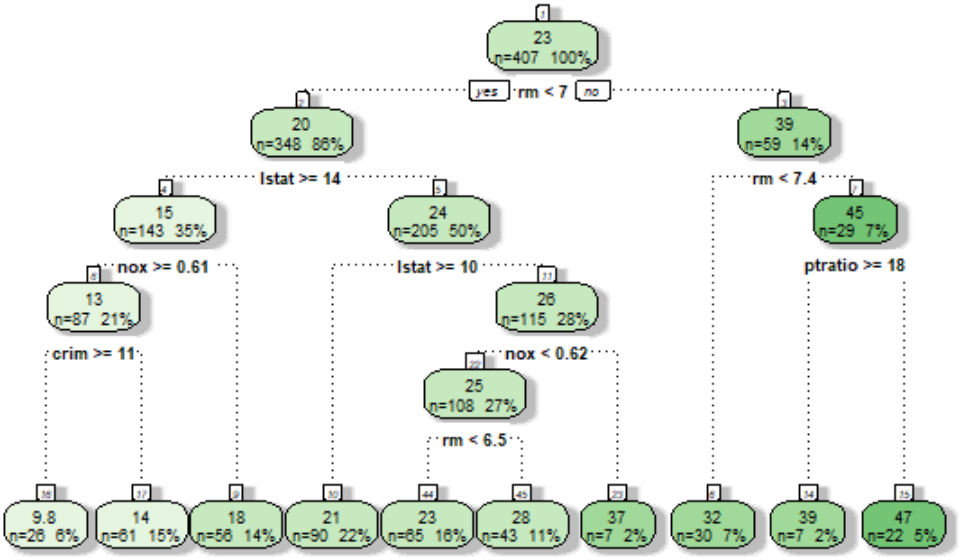

Figure 4.8: Final Tree Model

The output of the decision rules in the final model, as its code too, is the following:

```
> model$finalModel
n= 407

node), split, n, deviance, yval
* denotes terminal node

1) root 407 35292.0500 22.739310
2) rm< 6.974 348 13999.9200 20.056320
4) lstat>=14.4 143  2641.2730 15.093010
8) nox>=0.607 87   965.8699 12.998850
16) crim>=11.36915 26   144.7465  9.811538 *
17) crim< 11.36915 61   444.4092 14.357380 *
9) nox< 0.607 56   701.1193 18.346430 *
5) lstat< 14.4 205  5378.5900 23.518540
10) lstat>=9.95 90   492.0000 20.666670 *
11) lstat< 9.95 115  3581.7470 25.750430
22) nox< 0.618 108  1736.0900 25.050000
44) rm< 6.543 65   494.5222 23.052310 *
45) rm>=6.543 43   590.0507 28.069770 *
23) nox>=0.618 7   975.1771 36.557140 *
3) rm>=6.974 59  4011.4950 38.564410
6) rm< 7.437 30   522.6897 32.303330 *
7) rm>=7.437 29  1096.1900 45.041380
14) ptratio>=17.6 7   465.9686 38.885710 *
15) ptratio< 17.6 22   280.5800 47.000000 *
```

According to the final model we got, we make the predictions on the test dataset. The code for

this and the predictions of the first observations are as follows:

```
> predictions <- model %>% predict(test.data)
> head(predictions)
3        5       11       12       14       15
32.30333 32.30333 18.34643 20.66667 23.05231 20.66667
```

Lastly, we will compute the square root of the MSE with the following code:

```
> RMSE(predictions, test.data$medv)
[1] 4.545207
```

Conclusions What we see is that RMSE is around 4.55, indicating that this model leads to test predictions that are within around $4.55 of the true median home value for the suburb on average.

Applying the Regression Trees method to such a complex dataset facilitates the inference and provides reliable predictions about the true median home value.

Chapter 5

A Discussion of the Results of the Irish Data

First of all, let us compare the different classification methods we applied in the Irish data.
Regarding the model with the two most significant variables: Cost Structure and Value Propositions, we have the following remarks:

- Logistic, LDA and QDA models result in the same accuracy: approximately 74%.
  The fact that logistic and LDA model give the same results is generally expected, since logistic regression and LDA both produce linear decision boundary and differ only in their fitting procedures.
  The fact that QDA gives exactly the same accuracy with the linear models, make us wonder because with the quadratic decision boundary assumption the accuracy of the model does not fall but neither increases.

- KNN model results in a remarkable increase in the accuracy of our model, as it reaches 80%. In this way, we can understand that in our case we have a much more complicated decision boundary for our model, that is why the quadratic decision boundary did not deteriorate but neither improved the performance of the model.

Regarding, now, the model with the four most significant variables: Customer Segments, Value Propositions, Revenue Streams and Cost Structure, we notice the following:

- Logistic and LDA model perform exactly in the same way, as they both achieve an accuracy of about 71%.

- QDA and KNN model outperform LDA and logistic, as they both increased the accuracy to 74%. This means that assuming a quadratic decision boundary is quite logical. Choosing between the two classification methods, we would be for the QDA, as the KNN does not give us any information about the importance of the predictors.

Regarding the Classification Trees method, we see that it has the lowest accuracy–69%–but is the easiest method for interpretability.

Moreover, if we take a look at the results in which we had resulted in Subsection 1.4.2, we will come to some interesting conclusions.

- *Cost Structure*–which is one of the two most significant predictors for the classification among the entrepreneurs and the managers–was contributing to the first principal component of the entrepreneurs dataset and the second principal component of the managers dataset. This variable has to do with the firm's finances.

- *Value Propositions*–which is second of the two most significant predictors for the classification among the entrepreneurs and the managers–was contributing to the second principal component of the entrepreneurs dataset and the first principal component of the managers dataset. This variable has to do with serving products to new and existing customers.

- *Revenue Streams* is the variable that–according to Classification Trees method–predicts an individual as a manager if he gets a value lower than 32.5 and as an entrepreneur otherwise. Revenue Streams is also one of the four more significant variables for the models analysed in Subsections 2.2.4, 2.3.3 and 2.4.2. It is also contributing to the forming of the principal components of the two groups of individuals.

- *Key Resources* and *Channels* are the two variables contributing most to the representation of the principal components in the factor map, for both entrepreneurs and managers dataset. However, through the classification methods we see that these two variables do not play an important role for predicting an individual. This may mean that variables that are really important for both entrepreneurs and managers, do not make any difference between them and all the individuals behave in a similar way regarding these variables.

To conclude, we notice that the aspects of venturing that enable us to distinguish the managers from the entrepreneurs are: "Serving Products to New and Existing Customers" and "Cost/Finances".

Bibliography

[1] A. Bandura. "Self-Efficacy Beliefs of Adolescents". In: *Guide for constructing self-efficacy scales*. Vol. 5. IAP, 2006, pp. 307–337.

[2] R.A. Baron. "Opportunity Recognition as Pattern Recognition: How Entrepreneurs "Connect the Dots" to Identify New Business Opportunities". In: *Academy of Management Perspectives* 20.1 (2006), pp. 104–119.

[3] D.J. Bartholomew et al. *Analysis of Multivariate Social Science Data*. 2nd. CRC Press, 2011.

[4] M. Brannback and A.L. Carsrud. "Understanding the Entrepreneurial Mind". In: Springer-Verlag New York, 2009. Chap. Cognitive Maps in Entrepreneurship: Researching Sense Making and Action, pp. 75–96.

[5] C. Caroni and P. Economou. *Statistical Regression Models with MINITAB and R*. 2nd. in Greek. Symeon Publications, 2017.

[6] B.S. Everitt and T. Hothorn. *A Handbook of Statistical Analyses Using R*. 2nd. Chapman and Hall/CRC, 2010.

[7] M. Friendly and D. Meyer. *Discrete Data Analysis with R Visualization and Modeling Techniques for Categorical and Count Data*. Chapman and Hall/CRC, 2011.

[8] G. George and A.J. Bock. "The Business Model in Practice and its Implications for Entrepreneurship Research". In: *Entrepreneurship: Theory and Practice* 35.1 (2011), pp. 83–111.

[9] D.A. Gregoire, A.C. Corbett, and J.S. McMullen. "The Cognitive Perspective in Entrepreneurship: An Agenda for Future Research". In: *Journal of Management Studies* 48.6 (2011), pp. 1443–1477.

[10] D.J. Hand. *Statistics: a brief insight*. A Brief Insight. Sterling Pub, 2010.

[11] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. Springer Series in Statistics. Springer, 2009.

[12] H. Hotelling. "Analysis of a complex of statistical variables into principal components". In: *Journal of Educational Psychology* 24.6 (1933), pp. 417–441.

[13] F. Husson, S. Lé, and J. Pagès. *Exploratory Multivariate Analysis by Example Using R*. 2nd. Computer Science and Data Analysis. Chapman and Hall/CRC, 2017.

[14] G. James et al. *An Introduction to Statistical Learning with Applications in R*. Vol. 103. Springer, 2013.

[15] I.T. Jolliffe. "Discarding Variables in a Principal Component Analysis. I: Artificial Data". In: *Applied Statistics* 21.2 (1972), pp. 160–173.

[16] I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer New York, 1986.

[17] I.T. Jolliffe. "Rotation of III-Defined Principal Components". In: *Applied Statistics* 38.1 (1989), pp. 139–147.

[18] I.T. Jolliffe and M. Uddin. "The Simplified Component Technique: An Alternative to Rotated Principal Components". In: *Journal of Computational and Graphical Statistics* 9.4 (2000), pp. 689–710.

[19] S.F. Keane, K.T. Cormican, and J.N. Sheahan. "Comparing how entrepreneurs and managers represent the elements of the business model canvas". In: *Journal of Business Venturing Insights* 9 (2018), pp. 65–74.

[20] J. Khan et al. "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks". In: *Natural Medicine* 7 (2001), pp. 673–679.

[21] D.R. Krathwohl. "A Revision of Bloom's Taxonomy: An Overview". In: *Theory into Practice* 41.4 (2002), pp. 212–218.

[22] N.F. Krueger. "What lies beneath? The experiential essence of entrepreneurial thinking". In: *Entrepreneurship Theory and Practice* 31.1 (2007), pp. 123–138.

[23] J. Magretta. "Why Business Models Matter". In: *Harvard Business Review* 80.5 (2002), pp. 86–92.

[24] K.V. Mardia, J.T. Kent, and J.M. Bibby. *Multivariate Analysis*. Academic Press, 1979.

[25] C. Meng, B. Kuster, and A.C Culhane. "A multivariate approach to the integration of multi-omics datasets". In: *BMC Bioinformatics* 15.162 (2014).

[26] M. Morris, M. Schindehutte, and J. Allen. "The entrepreneur's business model: toward a unified perspective". In: *Journal of Business Research* 58.6 (2005), pp. 726–735.

[27] D.F. Morrison. *Multivariate Statistical Methods*. 2nd. McGraw-Hillbook Company, 1976.

[28] A. Osterwalder. "The Business Model Ontology-A Proposition in A Design Science Approach". MA thesis. Institut d' Informatique et Organisation, 2004.

[29] A. Osterwalder and Y. Pigneur. *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. John Wiley and Sons, 2010.

[30] A. Osterwalder, Y. Pigneur, and C.L. Tucci. "Clarifying business models: Origins, present, and future of the concept". In: *Communications of the Association for Information Systems* 16 (2005), pp. 1–25.

[31] K. Pearson. "On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine* 2.11 (1901), pp. 559–572.

[32] J.M. Muñoz Pichardo. *Regresión y clasificación mediante KNN*. University Lecture.

[33] C. Shalizi. *Classification and Regression Trees*. University Lecture. 2009.

[34] C. Shalizi. *Regression Trees*. University Lecture. 2006.

[35] S. Sharma. *Applied Multivariate Techniques*. Wiley, 1996.

[36] H. Tikkanen et al. "Managerial cognition, action and the business model of the firm". In: *Management Decision* 43.6 (2005), pp. 789–809.

[37] J.P. Walsh. "Managerial and Organizational Cognition: Notes from a Trip Down Memory Lane". In: *Organization Science* 6.3 (1995), pp. 280–321.

[38] C. Zott and R. Amit. "Business Model Design: An Activity System Perspective". In: *Long Range Planning* 43.2–3 (2010), pp. 216–226.

[39] C. Zott and R. Amit. "The Business Model: Recent Developments and Future Research". In: *Strategic Management Journal* 22.6–7 (2001), pp. 493–520.

[40] C. Zott, R.H. Amit, and L. Massa. "The Business Model: Recent Developments and Future Research". In: *Journal of Management* 37.4 (2011), pp. 1019–1042.