Predictive Modelling via Simultaneous Model Fitting and Clustering

Bolsius, R.M.

Award date:
2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Statistics Research Group

Predictive Modelling via Simultaneous Model Fitting and Clustering

Master Thesis

Rex Bolsius

Supervisor:
dr. Rui M. Castro

Final Version

March 2020

Abstract

The goal of this master project is to develop a predictive modelling method that can be used for data for which there are multiple distinct groups within the overall population. This method should cluster the data and fit a predictive model to each of the resulting clusters. A statistical framework will be used that can construct a mixture model that is able to accomplish this task by solving a likelihood maximisation problem. The EM algorithm will be used to find a solution, which results in an iterative algorithm that can in turn cluster the data using soft labels, and fit a predictive model to each cluster. This method will also provide what will be called the cluster probability functions, which can assign any new set of features to the existing cluster. As concrete examples, linear regression will be looked at to model the individual clusters, and two different methods will be used to cluster the data, namely logistic regression and a kernel smoothing method. For this last method, a strategy will be constructed that can automatically find the optimal value of the bandwidth parameter that regularises the method.

The argument will be made that this method can be used as a general predictive model that can model non-linear relationships between target value and features, and that no assumption about different sub-populations in the data are necessary to use it. Two important problems related to the developed method, namely choosing how to initialise it and how to select the optimal number of clusters, will be looked at, and this will result in a strategy that provides a solution to both. To make the developed method able to use more general predictive modelling techniques as basis functions, the likelihood maximisation problem will be adjusted to instead allow a loss function to be minimised. The result of all this is a general algorithmic framework that can perform clusterwise predictive modelling on a given dataset. This framework is very flexible, as users can decide which methods to use to estimate the cluster probability functions, and to model the different clusters. Regression trees will be looked as an example of this last category. Furthermore, ensembling will be looked at as a way of improving the accuracy of the predictive models. Finally, the performance of these methods will be looked at by testing them on three open dataset. The conclusion of these experiments is that these methods are very powerful, and perform similarly or better than a number of popular regression techniques, such as random forests and support vector regression.

Contents

Chapter 1

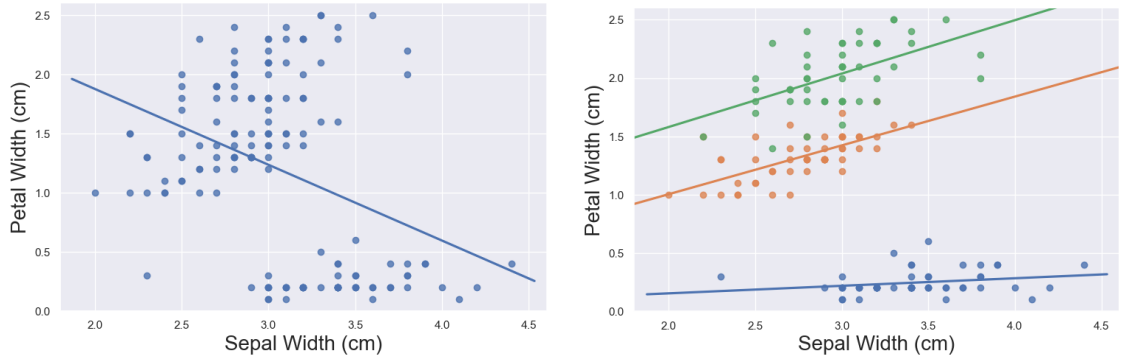
Introduction

Throughout their lives, people will regularly find themselves in situations where they are required to make decisions under uncertainty. A farmer might ask himself "What is the best day to start planting crops?"A stockbroker will have to decide "Which stock will provide the greatest return in investment?" And a doctor will often be faced with the question "Which treatment should a use for my patient?" Questions like these show that there is a great demand for methods that can make accurate predictions using the information that is available to us. Concurrently, the recent developments in technologies such as the internet and high speed computers make the amount of data available to us that could by used to make such predictions grow exponentially. This has led to an increased interest in predictive modelling, the study wherein one tries to make predictions of a certain outcome based on a mathematical model.

Such predictive models are typically build by looking at historical data. A dataset that can be used to build such a model should contain a collection of past cases, where for each case the target variable you wish to predict is includes, as well as a number of features, certain measurable characteristics that are specified for each case. For example, an ice cream seller wishing to predict how many ice creams they are going to sell today can look at how many they sold during the past couple of moths. The target variable would then be the number of ice creams sold, and relevant features might be for example the temperature, the amount of rain and the day of the week. By seeing how in the past a certain target value was related to the value of the features, we can make a prediction how, based on the current values of those features, which outcome seems most likely.

Over the years, many different models that can be used for predictions have been developed, all with their own assumptions and constraints, strengths and weaknesses. The goal of this final project is to look into one particular family of predictive models that assumes that there might be some sort of heterogeneity in the data. In other words, we assume that there are a number of different "groups" or "clusters" in the data and it appears that for each cluster a different model leads to more accurate predictions. This can be caused, for example, when the data comes from different sources, but the source of each data point is unknown to us.

As an example of this, let us look a the famous Iris data set due to Fisher [14]. In this data set, the sepal length, sepal width, petal length and petal width, all in cm, are given for 150 irises, those colourful flowers with their characteristic petal structures. Let us say that we want to make a model that can predict the petal width of an iris given its sepal width. One of the simplest models we can use for this is a linear regression model, where we assume that the relation between these variables is linear. In Figue 1.1a, such a linear model is computed for the Iris data set. As you can see, this model does not capture the relationship between these variables very well, which means that any predictions we make using this model will likely not be very accurate. The samples in the Iris data set are actually from three different species of iris, namely of the "Iris



(a) *(b)*

Figure 1.1: *A plot of the data from the Iris data set, where the petal width is plotted as a function of the sepal width. Least-square linear regression has been applied to have been drawn trough the points. In (a), all the points are considered to be homogeneous, and therefore only one model is used for all points. In (b), the points are coloured according to which species of iris they belong to. Blue points are for "Iris setosa", red for "Iris versicolor" and green for "Iris virginica". For each species, a different model is build and drawn.*

setosa", "Iris virginica" and "Iris versicolor" types. If we made a separate linear model for each species separately, which is visible in Figure 1.1b, we capture the relationship between sepal and petal width much better. Consequently, if we knew the species of Iris for which we want to predict its petal width belongs to, we can make a much more accurate prediction.

The above example shows that recognising that there are multiple clusters in the data can greatly improve the accuracy of a predictive model. However, the cluster membership might not always be available in the data set. For example, the order list of a hardware store might not include whether the buyer is a professional builder or just a DIY enthusiast, while we expect these two groups to have very different consumer behaviour. In such cases it might still be possible to estimate for each data point which cluster it belongs to. An important question to ask then is: Is it possible for a a model that estimates the cluster membership to get a similar performance to a model that knows the true clusters? To answer this question, we will explore a number of models that are able to make such estimations. They will cluster the data and build a predictive model for each cluster as well as provide a way to assign a new data point to one of the clusters. We will look at the strengths and weaknesses of these models, and test their performance on a number of data sets. Furthermore, their performance will be compared to that of other popular predictive modelling techniques.

1.1 Formalisation

Before we can look at specific predictive models, we need to formalise this idea of having multiple clusters in the data. Furthermore, we need to introduce some important concepts and notations.

1.1.1 The Mixture Model

In order to construct a predictive model, we need to make some assumptions about how the data is generated. We will assume that the features X (generally a vector, i.e. $X = (X_1, X_2, \ldots, X_d)$) are elements of a space called the feature space $\mathcal{X}$, and the targets Y of a space called the target space $\mathcal{Y}$. In the standard statistical learning framework, we assume that the feature/target com-

binations (X, Y) are randomly drawn from $\mathcal{X} \times \mathcal{Y}$ according to some joint probability distribution $\mathbb{P}_{XY}$. This is likely a reasonable assumption, as we typically expect that some feature/label combinations are more common than others. Another way of looking at this is by saying that X is drawn randomly from $\mathcal{X}$ according to the marginal distribution $\mathbb{P}_X$, and we are interested in the conditional distribution of Y given X, denoted by $\mathbb{P}_{Y|X}$.

For the type of models we will consider in this thesis, we modify this framework slightly by assuming that there are k distinct clusters in the data. We can do this by making the assumption that, alongside the feature/target couple, our data is specified by another random variable denoted by Z with $Z \in \mathcal{Z} = \{1, 2, \ldots, k\}$. This random variable determines to which cluster that data point belongs to. The result of this is that our data is categorised by the feature/target/cluster combination (X, Y, Z), drawn randomly from $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$ according to some probability distribution $\mathbb{P}_{XYZ}$.

For each possible value of Z we assume that the probability distribution of (X, Y) is different. For $j \in \{1, 2, \ldots, k\}$, let $v_j : \mathcal{X} \to \mathcal{Y}$ denote the probability density function (PDF) of Y, given $X = x$ and $Z = j$. Furthermore, we assume that $Z \,|\, X$ is distributed categorically and we define the function $h : \mathcal{X} \to [0, 1]^k$, so that $h(x)$ is a k-dimensional vector which represents this categorical distribution. It has components given by

$$h_j(x) = \mathbb{P}(Z = j \,|\, X = x), \tag{1.1}$$

for $j \in \{1, 2, \ldots, k\}$. From here on out, we will call $h_j(x)$ the cluster probability functions. The assumption made here is that the probability of a data point belonging to a specific cluster is dependent on the features. Consequently, there are regions within the feature space where it is more likely to find data belonging to a particular cluster than to the other clusters. This assumption will be quite important when making predictions, because it allows us to appoint new data points to clusters based on the values of their features.

We can now define the PDF of Y given $X = x$ as

$$w(y \,|\, x) = \sum_{j=1}^{k} h_j(x) v_j(y \,|\, x), \tag{1.2}$$

Which gives us the statistical model of our data. Such models that combine multiple different models are typically called mixture models.

1.1.2 The Predictive Model

As stated before, our goal is to make predictions based on the features. Formally, we to construct a prediction rule $f : \mathcal{X} \to \mathcal{Y}$ that takes as input a feature vector X and returns a prediction $\hat{Y}$. We say that $\hat{Y}$ is an estimate of the true target Y, and we want to construct $f(X)$ in such a way that $\hat{Y}$ is a "good" estimate of Y. But how do we define what is a "good" estimate? To this end we introduce the loss function $L(\hat{Y}, Y)$, which gives a measurement to how "different" the estimate $\hat{Y}$ is from the true outcome Y. If, for example, the target space consist of two distinct outcomes, i.e. $\mathcal{Y} = \{0, 1\}$ we can use the 0/1 loss function

$$L(\hat{y}, y) = \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{if } \hat{y} = y \end{cases},$$

where $\hat{y}$ and y are realisations of $\hat{Y}$ and Y respectively. In this thesis, we will focus mainly on

regression problems, which is when the target is (part of) the real line. The most commonly used loss functions for such problems is the squared error loss function

$$L(\hat{y}, y) = (\hat{y} - y)^2. \tag{1.3}$$

We want to construct a predictive model that has a low loss on average, to this end we can define the statistical risk

$$R(f) \equiv \mathbb{E}[L(f(X), Y], \tag{1.4}$$

where $(X, Y) \sim \mathbb{P}_{XY}$. We can now define the learning problem as finding a prediction rule that has a small risk.

The function that minimises the squared error risk is

$$f^*(x) = \mathbb{E}[Y \mid X = x], \tag{1.5}$$

and is called the regression function. The proof that this expression minimises the squared error risk is included in Appendix A. If we know that our data is generated by a mixture model with a known PDF, we get the the prediction rule

$$f^*(x) = \sum_{j=1}^{k} h_j(x)\mathbb{E}[Y \mid X = x, Z = j], \tag{1.6}$$

where $Y \mid X = x, Z = j$ has PDF $v_j(y \mid x)$. Furthermore, we define $g_j(x) = \mathbb{E}[Y \mid X = x, Z = j]$, the regression function of cluster j.

The problem is that we generally do not know the precise distribution of our data. We we will need to estimate this using a set of historical data called the training data. Let $D_n \equiv \{x_i, y_i\}_{i=1}^n$ denote the available training data, where we make the assumption that x_i and y_i are realisations of X_i and Y_i respectively, which in turn are i.i.d. copies of (X, Y). Note that the cluster membership is not given in this training data. We will be focusing our attention on situations where we expect there to be distinct clusters in the data, but where the cluster membership is not given in the available data.

We see that estimating the distribution of a mixture model consists of two parts. We will need to estimate the cluster probability functions $h_j(x)$, as well the regression functions of the individual clusters $g_j(x)$. Let $\hat{h}_j(x; D_n)$ and $\hat{g}_j(x; D_n)$ denote these estimates, where the extra variable D_n is added to emphasise that these estimates are dependent on our training data. From now on this variable will be dropped from these functions, and the convention will be used that a hat ($\wedge$) over a function will denote that it is an estimate constructed using training data. Using these estimates we can construct the prediction rule

$$\hat{f}(x) = \sum_{j=1}^{k} \hat{h}_j(x)\hat{g}_j(x). \tag{1.7}$$

Since it is a combination of multiple model, this prediction rule is also a type of mixture model. Prediction rules of this form will henceforth be called clusterwise predictive models. In the following chapters we will develop a general method for constructing $\hat{h}_j$ and $\hat{g}_j$, but before that we will take a look at existing literature that deals with such mixture models.

1.2 Related Literature

There already exists an extensive amount of literature related to the construction of mixture models on a given dataset for a wide variety of contexts and goals. In the statistical community, most research has been conducted towards estimating the densities of the individual clusters for the model in Equation (1.2), often considering unconditional density functions only (see e.g. McLachlan and Peel [27]). Mixture models of this type have found considerable use as a form of density estimation, as the set of all normal mixture densities is dense in the set of all density functions under the L1 metric [23], and as a popular tool for clustering and classification [29]. This has made it an important concept for data analysis and inference as well as discriminant analysis, image analysis, and survival analysis in a ton of different fields [26]. As they do not consider data that is distributed conditional on any features, they cannot be used for prediction purposes.

An extension of these type of models are clusterwise linear regression models [10], which assume that the PDFs of the individual clusters $v_j(y \mid x)$ in Equation (1.2) are normal density functions with a conditional mean equal to a linear combinations of features, i.e. the models that describe each cluster are considered to be linear regression model. These models are typically used for data analysis purposes, and since they, in their general form, do not provide a way of assigning any new data point to an existing cluster, their usefulness for predicting is limited.

A number of methods for constructing mixture models for the specific purpose of prediction have been developed. Perhaps the most commonly seen one is the decision tree [18, Section 9.2], which can be used for both regression and classification purposes. They work by repeatedly splitting the feature space in rectangular regions, and in each regions fitting a constant function that minimises some loss function. This procedure can be represented using a tree structure, where internal nodes correspond to a splitting of a region in the feature space, and the end nodes each correspond to one of the clusters found in the data. Decision trees are popular because they are very flexible and easy to interpret, and often require little data preparation. However, the shape of clusters they can find is limited to box-shaped regions. Furthermore, the final model it fits to each of the found clusters is a constant function, meaning that more complicated relationships between target and features within a cluster cannot be modelled well. This also means that the final predictive model is not continuous, having non-smooth transitions between each cluster.

The method of Hinging Hyperplanes [4] works by approximating a function using a number of hinges. A hinge is a combination of two hyperplanes that are continuously joined together. The method works by first fitting a single hinge to the data, and computing the residual error. Then, another hinge is added to the model by fitting a hinge to the errors. This procedure is repeated until some convergence criterion, such as a maximum number of hinges, is reached. The clusters this method implicitly finds can be viewed as the regions in the feature space where a single hyperplane is fitted to. This method can be viewed as a more refined versions of decision trees, since it produces a continuous regression function where data in each resulting cluster is modelled by a linear function. Nevertheless, it can still only model clusters of limited amount of shapes, namely ones that are polyhedric in shape.

A method that allows for more flexible cluster shapes is k-plane regression [24], which is similar to the k-means clustering method. Just like in that method, data is repeatedly clustered according to which of the current k models it is closest to, and then models are fitted to each cluster. Wherein k-means clustering these models are the average positions of each point in the cluster, in k-plane regression they are the linear models fitted to the cluster members. The modified version of this method clusters the data according to both the distance to each linear model as well as to each cluster centre. This procedure results in a piecewise linear function that is not necessarily continuous. A problem with this method, however, is that it clusters data according to absolute cluster membership, in other words when this model is given in the form of Equation (1.7), $\hat{h}_j(x) \in \{0, 1\}$. It uses *hard* clustering rather than *soft* clustering. We know, however, that the rule that minimises the mean squared error risk is a weighted average of the predictions given

by the individual models, using the probability of belonging to each cluster as the weights. A method that uses soft clustering might therefore be preferred for making predictions

The method that approaches mixture models used for predictive modelling most closely to the way they have been introduced in Section 1.1 is perhaps that of the Mixture of Experts [20]. In this method, the feature space is split in a number of regions by functions known as the gates. These gates are originally given by the softmax functions, which provide a soft partitioning of the data. In each of these regions, a predictive model, known as the experts, is fit to the data. Classically, these experts are linear models, but a number of different models that have also shown favourable results, such as Gaussian Processes [40]. Mixture of Experts can also be used for classification problems, in which the experts are, for example, logistic regression models or support vector machines [40]. An extension of this method successively applies gates to split the feature space in smaller regions is known as the Hierarchical Mixture of Experts method [21]. This method can be seen as a generalisation of the decision tree, which allows for splits not parallel to a coordinate axis, fits a more general model to each cluster, and provides a smooth transition between the models of the different clusters.

Gitman et. al. developed a method called Predictive CLR [16] that modifies clusterwise linear regression by incorporating a classification method that can assign a new data point, for which the target value is unknown, to an existing cluster, making this method usable for prediction purposes. In that same paper, they also propose a different adjustment to clusterwise linear regression where data points are assigned to clusters based on the value of a categorical feature that is known at test time. They named this second approach Constrained CLR. Furthermore, they found out that building an ensemble of different models that all use different initial values can greatly improve the performance of these models.

Chapter 2

Mixture Models in the Statistical Context

To find a way of constructing a clusterwise predictive model, we will take a look at how the PDF of a mixture distribution is typically learned from data. In most research conducted towards such distributions, a slightly simpler form is given as their definition. Primarily, the data is assumed to be generated according to an unconditional PDF. In other words, there is no assumed relationship between the target Y and some number of features X. Consequently, the probabilities describing the categorical distribution $\mathbb{P}_Z$ are assumed to be constants. Let ζ_j for $j \in \{1, 2, \ldots, k\}$ with $\sum_{j=1}^{k} \zeta_j = 1$ denote these probabilities. For practical consideration, we assume that the densities of the individual clusters $v_{\theta_j}(y)$ are known up to a vector of parameters θ_j. We can now express the PDF of this simplified form of a mixture model as

$$w_{\mathbf{\Psi}}(y) = \sum_{j=1}^{k} \zeta_j v_{\theta_j}(y), \tag{2.1}$$

where $\mathbf{\Psi} = (\zeta_1, \ldots, \zeta_k, \theta_1, \ldots, \theta_k)$ is the parameter vector that characterises the mixture density, with $\mathbf{\Psi} \in \mathbf{\Omega}$, i.e. $\mathbf{\Omega}$ is the parameter space. We see that finding an estimate for this density comes down to finding estimates for the components in $\mathbf{\Psi}$.

We will look at how the parameter estimates are typically found for this model, and use this method as an inspiration to find parameter estimates for the more general model given by Expression (1.2). One of the earliest attempts to compute a mixture of densities dates back to a paper from 1894 by Karl Pearson [31]. However, as noted by McLachlan and Peel ([27] section 1.1.3), it was not until more efficient ways of computing the parameter estimates using maximum likelihood methods were developed in the 1960s and 1970s that mixture models became popular. Monumental was the paper by Dempster et. al. [9] from 1977, which described a method known as the Expectation Maximisation (EM) algorithm which can be used as a very general and efficient way to compute the maximum likelihood of a mixture density model. We will now look at how this method works.

2.1 The Maximum Likelihood Solution

Maximum likelihood estimation (MLE) is a popular method of estimating the unknown parameter values of a probability distribution. It works by, given a set of data, maximising the likelihood function, a measure of how probable that data is under the given model. Given a data set $\{y_i\}_{i=1}^{n}$,

we can express the likelihood function of the model in Expression (2.1) as

$$\mathcal{L}(\mathbf{\Psi}) = \prod_{i=1}^{n} \sum_{j=1}^{k} \zeta_j v_{\theta_j}(y_i), \tag{2.2}$$

As an example, we look at the case where $k = 2$ and $v_{\theta_j}(y)$ are the PDFs of the univariate normal distribution with unknown parameters. We can express its likelihood function as

$$\mathcal{L}(\mathbf{\Psi}) = \prod_{i=1}^{n} \left(\zeta_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(y_i - \mu_1)^2}{2\sigma_1^2} \right) + (1 - \zeta_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{(y_i - \mu_2)^2}{2\sigma_2^2} \right) \right), \tag{2.3}$$

where, since we know that $\zeta_1 + \zeta_2 = 1$ and we therefore need to estimate only one of these parameters, we replace ζ_2 by $\zeta_1 - 1$. We see that there are five unknown parameters that we need to find, i.e. $\mathbf{\Psi} = (\zeta_1, \mu_1, \mu_2, \sigma_1, \sigma_2)$. Maximising this likelihood function is, however, very hard because it is not a convex function of the parameters. If the clustering of the data were known, maximising this expression would be quite easy. This is precisely the type of problem the EM algorithm is suitable for.

2.1.1 The Expectation Maximisation Algorithm

As defined by McLachlan and Krishnan, the EM algorithm is a procedure to find the MLE in situations where we have incomplete data [25, Section 1.5.1]. This includes cases where there is missing data, but also situations where one or more variables are not observed, such as the cluster membership in the case of mixture models. The EM algorithm is appealing to be used in situations where finding the MLE using only the incomplete data is very difficult, but would be straightforward if we had the complete data.

If we go back to the mixture of normal distributions, we can say that the data set $\{y_i\}_{i=1}^n$ is the incomplete data, and the set $\{(y_i, z_i)\}_{i=1}^n$ is the complete data set, with z_i realisations of Z_i, which are i.i.d. copies of Z. Using this complete data, we can define the complete data likelihood function

$$\begin{aligned} \mathcal{L}_c(\mathbf{\Psi}) = \prod_{i=1}^{n} \Bigg(& \mathbb{1}\{Z_i = 1\} \zeta_1 \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(x_i - \mu_1)^2}{2\sigma_1^2} \right) \\ & + \mathbb{1}\{Z_i = 2\} (1 - \zeta_1) \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{(x_i - \mu_2)^2}{2\sigma_2^2} \right) \Bigg), \end{aligned} \tag{2.4}$$

which has the property that within the product, one term will always be zero. It is often easier to maximise the log of the likelihood function. Due to the non decreasing nature of the log-function, the parameter values that maximise the log likelihood function are the same maximum as those that maximise the original likelihood function. For our example, the log likelihood function is given by

$$\ell_c(\mathbf{\Psi}) \equiv \log \mathcal{L}_c(\mathbf{\Psi}) = \sum_{i=1}^{n} \left(\mathbb{1}\{Z_i = 1\} \left(\log \zeta_1 - \frac{1}{2}\log(2\pi) - \log \sigma_1 - \frac{(y_i - \mu_1)^2}{2\sigma_1^2} \right) + \mathbb{1}\{Z_i = 2\} \left(\log(1-\zeta_1) - \frac{1}{2}\log(2\pi) - \log \sigma_2 - \frac{(y_i - \mu_2)^2}{2\sigma_2^2} \right) \right).$$

If we had access to the cluster memberships $\{z_i\}_{i=1}^n$, maximising this function would be straightforward. Since we do not have it available to us, we need to come up with a strategy to deal with this missingness. This is where the EM algorithm comes into play. The central idea of this algorithm is that, although we cannot compute the log likelihood, we can still compute its expected value conditioned on the observed data. This also requires some initial guess of the parameters $\mathbf{\Psi}^{(0)}$. We define

$$Q^{(0)}(\mathbf{\Psi}) \equiv \mathbb{E}_{\mathbf{\Psi}^{(0)}}[\ell_c(\mathbf{\Psi}) \mid \{y_i\}_{i=1}^n].$$

We can then maximise this $Q^{(0)}(\mathbf{\Psi})$ with respect to $\mathbf{\Psi}$ to get a new estimate $\mathbf{\Psi}^{(1)}$. Using this new estimate, we can calculate a new expected value of the log likelihood $Q^{(1)}(\mathbf{\Psi})$ which we are then able to maximise, repeating the process.

We can see that at iteration $t+1$ of the algorithm, we perform two steps. The E-step where we find an expression for $Q^{(t)}(\mathbf{\Psi})$ given by

$$Q^{(t)}(\mathbf{\Psi}) = \mathbb{E}_{\mathbf{\Psi}^{(t)}}[\ell_c(\mathbf{\Psi}) \mid \{y_i\}_{i=1}^n], \tag{2.5}$$

and the M-step where we maximise this expression with respect to $\mathbf{\Psi}$ to get a new estimate $\mathbf{\Psi}^{(t+1)}$. In other words, we find

$$\mathbf{\Psi}^{(t+1)} = \arg\max_{\mathbf{\Psi} \in \mathbf{\Omega}} Q^{(t)}(\mathbf{\Psi}). \tag{2.6}$$

In these formulas and from here on out, the superscript (t) denotes a parameter or function estimate at step t of the EM algorithm.

It is shown by Dempster, Laird, and Rubin that the incomplete likelihood is not decreasing at each iteration of the EM algorithm, meaning that as long as this likelihood function is bounded from above, the sequence of solutions found at each iteration of the EM algorithm is guaranteed to converge to a local maximum of the likelihood function [9]. We can then define the final parameter estimate as

$$\hat{\mathbf{\Psi}} = \lim_{t \to \infty} \mathbf{\Psi}^{(t)}$$

For our example of the two normal mixtures, Equation (2.5) is given by

$$
\begin{aligned}
Q^{(t)}(\boldsymbol{\Psi}) &= \mathbb{E}_{\boldsymbol{\Psi}^{(t)}}[\ell_c(\boldsymbol{\Psi}) \,|\, \{y_i\}_{i=1}^n] \\
&= \mathbb{E}_{\boldsymbol{\Psi}^{(t)}}\left[\sum_{i=1}^{n}\left(\mathbb{1}\{Z_i=1\}\left(\log\zeta_1 - \frac{1}{2}\log(2\pi) - \log\sigma_1 - \frac{(y_i-\mu_1)^2}{2\sigma_1^2}\right)\right.\right. \\
&\quad \left.\left. + \ \mathbb{1}\{Z_i=2\}\left(\log(1-\zeta_1) - \frac{1}{2}\log(2\pi) - \log\sigma_2 - \frac{(y_i-\mu_2)^2}{2\sigma_2^2}\right)\right) \,\middle|\, \{y_i\}_{i=1}^n\right] \qquad (2.7) \\
&= \sum_{i=1}^{n}\left(\mathbb{P}_{\boldsymbol{\Psi}^{(t)}}(Z_i=1 \,|\, Y_i=y_i)\left(\log\zeta_1 - \frac{1}{2}\log(2\pi) - \log\sigma_1 - \frac{(y_i-\mu_1)^2}{2\sigma_1^2}\right)\right. \\
&\quad \left. + \ \mathbb{P}_{\boldsymbol{\Psi}^{(t)}}(Z_i=2 \,|\, Y_i=y_i)\left(\log(1-\zeta_1) - \frac{1}{2}\log(2\pi) - \log\sigma_2 - \frac{(y_i-\mu_2)^2}{2\sigma_2^2}\right)\right).
\end{aligned}
$$

We can use Bayes' theorem to compute the probabilities in the above equation:

$$
\begin{aligned}
p_{i1}^{(t)} &\equiv \mathbb{P}_{\boldsymbol{\Psi}^{(t)}}(Z_i=1 \,|\, Y_i=y_i) \\
&= \frac{\mathbb{P}_{\boldsymbol{\Psi}^{(t)}}(Z_i=1)\mathbb{P}_{\boldsymbol{\Psi}^{(t)}}(Y_i=y_i \,|\, Z_i=1)}{\mathbb{P}_{\boldsymbol{\Psi}^{(t)}}(Y_i=y_i)} \\
&= \frac{\zeta_1^{(t)} v_{\theta_1}^{(t)}(y_i)}{\zeta_1^{(t)} v_{\theta_1}^{(t)}(y_i) + (1-\zeta_1^{(t)}) v_{\theta_2}^{(t)}(y_i)},
\end{aligned}
$$

and similarly

$$
\begin{aligned}
p_{i2}^{(t)} &\equiv \mathbb{P}_{\boldsymbol{\Psi}^{(t)}}(Z_i=2 \,|\, Y_i=y_i) \\
&= \frac{(1-\zeta_1^{(t)}) v_{\theta_2}^{(t)}(y_i)}{\zeta_1^{(t)} v_{\theta_1}^{(t)}(y_i) + (1-\zeta_1^{(t)}) v_{\theta_2}^{(t)}(y_i)}.
\end{aligned}
$$

In order to maximise Equation (2.7), we simply need to compute the partial derivatives with respect to the parameter, and set them equal to zero. This way we get the solutions

$$
\zeta_1^{(t+1)} = \frac{\sum_{i=1}^n p_{i1}^{(t)}}{\sum_{i=1}^n \left(p_{i1}^{(t)} + p_{i2}^{(t)}\right)} = \frac{1}{n}\sum_{i=1}^{n} p_{i1}^{(t)},
$$

$$
\mu_j^{(t+1)} = \frac{\sum_{i=1}^n p_{ij}^{(t)} x_i}{\sum_{i=1}^n p_{ij}^{(t)}},
$$

And finally

$$
\sigma_j^{(t+1)} = \sqrt{\frac{\sum_{i=1}^n p_{ij}^{(t)} \left(x_i - \mu_j^{(t)}\right)^2}{\sum_{i=1}^n p_{ij}^{(t)}}}.
$$

2.1.2 A General Solution

In the above section we saw that the EM algorithm gives us a simple iterative procedure to find the MLE solution of the parameters in a simple mixture of normal distributions. We now want to use this method to get a general solution for a mixture distribution that can be used for predictive modelling. To this end we will again look at the mixture model given by Expression (1.2), were we will assume that the the densities of the clusters $v_{\theta_j}(y \mid x)$ are known up to a vector of parameters θ_j. This leads to the expression of the mixture model

$$w_{\mathbf{\Psi}}(y \mid x) = \sum_{j=1}^{k} h_j(x) v_{\theta_j}(y \mid x), \tag{2.8}$$

which has the likelihood function

$$\mathcal{L}(\mathbf{\Psi}) = \prod_{i=1}^{n} \sum_{j=1}^{k} h_j(x_i) v_{\theta_j}(y_i \mid x_i). \tag{2.9}$$

In the context of the EM algorithm, the training data $\{(x_i, y_i)\}_{i=1}^n$ are the incomplete data, and the cluster memberships $\{z_i\}_{i=1}^n$ are the missing data. We can now define the complete data likelihood function as

$$\mathcal{L}_c(\mathbf{\Psi}) = \prod_{i=1}^{n} \sum_{j=1}^{k} \mathbb{1}\{Z_i = j\} h_j(x_i) v_{\theta_j}(y_i \mid x_i), \tag{2.10}$$

and its respective log likelihood function as

$$\ell_c(\mathbf{\Psi}) = \log \mathcal{L}_c(\mathbf{\Psi}) = \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbb{1}\{Z_i = j\} (\log h_j(x_i) + \log v_{\theta_j}(y_i \mid x_i)). \tag{2.11}$$

To perform the E-step, we use equation 2.5:

$$\begin{aligned}
Q^{(t)}(\mathbf{\Psi}) &= \mathbb{E}_{\mathbf{\Psi}^{(t)}}[\ell_c(\mathbf{\Psi}) \mid \{(x_i, y_i)\}_{i=1}^n] \\
&= \mathbb{E}_{\mathbf{\Psi}^{(t)}} \left[\sum_{i=1}^{n} \sum_{j=1}^{k} \mathbb{1}\{Z_i = j\} (\log h_j(x_i) + \log f_{\theta_j}(y_i \mid x_i)) \,\middle|\, \{(x_i, y_i)\}_{i=1}^n \right] \\
&= \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbb{P}_{\mathbf{\Psi}^{(t)}}(Z_i = j \mid X_i = x_i, Y_i = y_i) (\log h_j(x_i) + \log v_{\theta_j}(y_i \mid x_i)).
\end{aligned}$$

We use Bayes' theorem to compute the probabilities in the above equation:

$$\begin{aligned}
p_{ij}^{(t)} &\equiv \mathbb{P}_{\mathbf{\Psi}^{(t)}}(Z_i = j \mid X_i = x_i, Y_i = y_i) \\
&= \frac{\mathbb{P}_{\mathbf{\Psi}^{(t)}}(Y_i = y_i \mid X_i = x_i, Z_i = j) \mathbb{P}_{\mathbf{\Psi}^{(t)}}(Z_i = j \mid X_i = x_i)}{\mathbb{P}_{\mathbf{\Psi}^{(t)}}(Y_i = y_i \mid X_i = x_i)} \\
&= \frac{h_j^{(t)}(x_i) v_{\theta_j}^{(t)}(y_i \mid x_i)}{\sum_{l=1}^{k} h_l^{(t)}(x_i) v_{\theta_l}^{(t)}(y_i \mid x_i)}.
\end{aligned} \tag{2.12}$$

Using this we can write $Q^{(t)}(\mathbf{\Psi})$ as

$$\begin{aligned} Q^{(t)}(\mathbf{\Psi}) &= \sum_{i=1}^{n}\sum_{j=1}^{k} p_{ij}^{(t)}(\log h_j(x_i) + \log v_{\theta_j}(y_i \mid x_i)) \\ &= \left(\sum_{i=1}^{n}\sum_{j=1}^{k} p_{ij}^{(t)} \log h_j(x_i)\right) + \sum_{j=1}^{k}\left(\sum_{i=1}^{n} p_{ij}^{(t)} \log v_{\theta_j}(y_i \mid x_i)\right). \end{aligned} \tag{2.13}$$

This means that we have split the optimisation problem in two parts that can be solved independently. It therefore becomes convenient to introduce some new notation

$$Q_c^{(t)}(h) = \sum_{i=1}^{n}\sum_{j=1}^{k} p_{ij}^{(t)} \log h_j(x_i) \tag{2.14}$$

$$Q_{mj}^{(t)}(\theta_j) = \sum_{i=1}^{n} p_{ij}^{(t)} \log v_{\theta_j}(y_i \mid x_i), \tag{2.15}$$

where the subscripts c and m stand for "cluster" and "model", respectively. In the M-step we need to solve k+1 optimisation problems to obtain the components of $\mathbf{\Psi}^{(t+1)}$.

Firstly, for each $j \in \{1, 2, \ldots, k\}$

$$\theta_j^{(t+1)} = \arg\max_{\theta_j \in \mathbf{\Theta}} Q_{mj}^{(t)}(\theta_j), \tag{2.16}$$

where $\mathbf{\Theta}$ is the parameter space of the conditional density function. The solutions to these optimisations are, of course, dependent on the family of models between the features and label we consider. Lastly, we need to find

$$h^{(t+1)} = \arg\max_{h \in \mathcal{H}} Q_c^{(t)}(h), \tag{2.17}$$

with $\mathcal{H}$ the class of cluster probability functions that we consider. The performance of our predictive model is actually very dependent on how we select this class. If we take $\mathcal{H}$ as the class of all functions that have range $[0, 1]^k$ and abide to $\sum_{j=1}^{k}$, the solution to this is simply

$$h_j(x)^{(t+1)} = p_{ij}^{(t)} \text{ for all } j \in \{1, 2, \ldots, k\} \text{ if } \exists_i : x = x_i, \tag{2.18}$$

and $h_j(x)^{(t+1)}$ arbitrary if it is not. This, of course, is not very useful for prediction purposes, as it does not tell us how to choose $h_j(x)$ outside of the training data. So clearly, in order to make sensible predictions, we need to impose some sort of conditions on h.

If we assume that h is independent of x, i.e. the class $\mathcal{H}$ contains only constant functions, the solution to Equation (2.17) is [26]

$$h_j(x)^{(t+1)} = \frac{\sum_{i=1}^{n} p_{ij}^{(t)}}{n},$$

which again is not very useful for making predictions, since it does not involve the features meaning that these are constant functions. Clearly, choosing $\mathcal{H}$ in a smart way is very important for making accurate predictions.

We see that, computationally, the only things necessary during the E-step are solving for the $p_{ij}^{(t)}$ using Equation (2.12). In this way, the two steps of the EM algorithm solve precisely the two problems that have been identified for mixture models. During the E-step, the data is clustered according to which model best describes that feature/target pair. During the M-step, for each cluster, the parameters are tuned using Equation (2.16) so that the corresponding model describes the patterns in that cluster well. During the M-step, the functions $h_j(x)$ are also constructed using Equation (2.17).

2.2 Clusterwise Linear regression

As a concrete example of how to find the solution to Equation (2.16), we will look at clusterwise linear regression, a common use of mixture models. In this method, we assume that the relation between the features $x \in \mathbb{R}^d$ and the target $y \in \mathbb{R}$ given that $Z = j$ can be modelled using a linear function

$$y = \beta_{j0} + \beta_{j1}x_1 + \cdots + \beta_{jd}x_d + \varepsilon_j,$$

where $\varepsilon_j \sim \mathcal{N}(0, \sigma_j)$. We see that the parameter vector $\theta_j = (\beta_{j0}, \beta_{j1}, \ldots, \beta_{jd}, \sigma_j)$. For our training set, define

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \text{ and } \boldsymbol{A} = \begin{bmatrix} 1 & x_{11} & \ldots & x_{1d} \\ 1 & x_{21} & \ldots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \ldots & x_{nd} \end{bmatrix}. \tag{2.19}$$

The conditional PDF of observation (X_i, Y_i) given $Z_i = j$ is

$$v_{\theta_j}(y_i \,|\, x_i) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{\left(y_i - \sum_{l=0}^{d} \beta_{jl} a_{il}\right)^2}{2\sigma_j^2} \right) \tag{2.20}$$

where a_{ij} denote elements in $\boldsymbol{A}$, where indexing of the rows begins at 1 and of the columns at 0. We use this to write Expression (2.12) as

$$\begin{aligned} p_{ij}^{(t)} &= \frac{h_j^{(t)}(x_i) v_{\theta_j}^{(t)}(y_i \,|\, x_i)}{\sum_{l=1}^{k} h_l^{(t)}(x_i) v_{\theta_l}^{(t)}(y_i \,|\, x_i)} \\ &= \frac{\frac{h_j^{(t)}(x_i)}{\sigma_j^{(t)}} \exp\left(-\frac{\left(y_i - \sum_{l=0}^{d} \beta_{jl}^{(t)} x_{il}\right)^2}{2\sigma_j^{2(t)}} \right)}{\sum_{m=1}^{k} \frac{h_m^{(t)}(x_i)}{\sigma_m^{(t)}} \exp\left(-\frac{\left(y_i - \sum_{l=0}^{d} \beta_{ml}^{(t)} x_{il}\right)^2}{2\sigma_m^{2(t)}} \right)}, \end{aligned} \tag{2.21}$$

We now write Equation (2.15) as

$$\begin{aligned} Q_{mj}^{(t)}(\theta_j) &= \sum_{i=1}^{n} p_{ij}^{(t)} \log v_{\theta_j}(y_i \,|\, x_i) \\ &= -\frac{1}{2}\log(2\pi)\sum_{i=1}^{n} p_{ij}^{(t)} - \log\sigma_j \sum_{i=1}^{n} p_{ij}^{(t)} - \frac{1}{2\sigma_j^2}\sum_{i=1}^{n}\left(p_{ij}^{(t)}\left(y_i - \sum_{l=0}^{d}\beta_{jl}a_{il}\right)^2\right). \end{aligned} \tag{2.22}$$

To find the values $\boldsymbol{\beta}_j$ that maximise this expression, we calculate its partial derivatives with respect to β_{jl} and set them equal to zero:

$$\frac{\partial Q_{mj}^{(t)}(\theta_j)}{\partial \beta_{jl}} = \sum_{i=1}^{n} p_{ij}^{(t)} a_{ir}\left(y_i - \sum_{l=0}^{d}\beta_{jl}a_{il}\right) = 0, \qquad r = (0, 1, \ldots, d)$$

or

$$\sum_{i=1}^{n}\sum_{l=0}^{d} p_{ij}^{(t)} a_{ir} a_{il}\beta_{jl} = \sum_{i=1}^{n} p_{ij}^{(t)} a_{ir} y_i.$$

If we define

$$\boldsymbol{W}_j^{(t)} = \begin{bmatrix} p_{1j}^{(t)} & 0 & \ldots & 0 \\ 0 & p_{2j}^{(t)} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & p_{nj}^{(t)} \end{bmatrix} \text{ and } \boldsymbol{\beta}_j = \begin{bmatrix} \beta_{j0} \\ \beta_{j1} \\ \vdots \\ \beta_{jd} \end{bmatrix},$$

we can write the above expression as

$$\boldsymbol{A}^T \boldsymbol{W}_j^{(t)} \boldsymbol{A}\boldsymbol{\beta}_j = \boldsymbol{A}^T \boldsymbol{W}_j^{(t)} \boldsymbol{y},$$

where the superscript T denotes the transpose of a matrix or vector. So, during the M-step we calculate a new estimate of $\boldsymbol{\beta}_j$ for each $j \in \{1, 2, \ldots, k\}$ using (as long as the matrix $\boldsymbol{A}$ has full rank, see e.g. Bingham and Fry [3, Section 3.2])

$$\boldsymbol{\beta}_j^{(t+1)} = \left(\boldsymbol{A}^T \boldsymbol{W}_j^{(t)} \boldsymbol{A}\right)^{-1} \boldsymbol{A}^T \boldsymbol{W}_j^{(t)} \boldsymbol{y}. \tag{2.23}$$

To find an estimate for σ_j, we first write Equation (2.7) using the matrix notation as

$$Q_{mj}^{(t)}(\theta_j) = -\frac{1}{2}\log(2\pi)\sum_{i=1}^{n} p_{ij}^{(t)} - \log\sigma_j \sum_{i=1}^{n} p_{ij}^{(t)} - \frac{1}{2\sigma_j^2}\left(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}_j\right)^T \boldsymbol{W}_j^{(t)}\left(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}_j\right).$$

We differentiate this expression with respect to σ_j and set it equal to zero to find

$$\frac{\partial Q_{mj}^{(t)}(\theta_j)}{\partial \sigma_j} = -\frac{\sum_{i=1}^{n} p_{ij}^{(t)}}{\sigma_j} + \frac{\left(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}_j\right)^T \boldsymbol{W}_j^{(t)}\left(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}_j\right)}{\sigma_j^3} = 0$$

which, using the new estimates of for $\boldsymbol{\beta}_j$, gives us

$$\sigma_j^{(t+1)} = \sqrt{\frac{\left(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}_j^{(t+1)}\right)^T \boldsymbol{W}_j^{(t)} \left(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}_j^{(t+1)}\right)}{\sum_{i=1}^n p_{ij}^{(t)}}}. \tag{2.24}$$

If we compare these solutions with those of regular linear regression (e.g. the book by Bingham and Fry [3, Section 3.2]), we see that they are actually very similar, with the exception that each data point gets a weight according to the inferred cluster probabilities.

We also need a way to find $h^{(t+1)}$ given $h^{(t)}$ by solving Equation (2.17), using a sensible class $\mathcal{H}$. In the next chapter which will look at some ways of doing this.

Let $\hat{v}_j(y \mid x) = v_{\hat{\theta}_j}(y_i \mid x_i)$ be our final estimates of the PDFs of Y given that $X = x$ and $Z = j$ for all $j \in \{1, 2, \ldots, k\}$. We know that $\mathbb{E}[Y \mid X = x, Z = j] = x^T \boldsymbol{\beta}_j$, so we can give the prediction rule for clusterwise linear regression using Expression (1.7) as

$$\hat{f}(x) = \sum_{j=1}^{k} \hat{h}_j(x) x^T \hat{\boldsymbol{\beta}}_j. \tag{2.25}$$

2.2.1 Discussion

Linear models make a lot of sense to be used as the basis block function that is fit to the individual clusters. They have a single solution that can be found analytically, meaning that the computational load required to fit this model to a dataset is very low, even for big datasets. A disadvantage is that it makes very strict assumptions about the relationship between target and features, namely that it is linear and that residuals are normally distributed. For many datasets, these assumptions will not hold and an ordinary least squares linear regression will often perform poorly at making predictions. Using them within a mixture model can mitigate this issue, as we can essentially split the feature space in regions in which the relationship between target and features is roughly linear, meaning that linear regression works well for modelling the individual clusters.

Of the methods discussed in Section 1.2, linear functions are also the most commonly used basis function to plot to the found clusters. The Hinging Hyperplanes, k-plane regression, Predictive and Constrained CLR, and the original Mixture of Experts methods all use them in some way.

Chapter 3

Estimating the Cluster Probability Functions

It is clear that the performance of any prediction rule that is based on mixture models depends greatly on how well we are able to estimate the cluster probability functions h_j. As is discussed in Section 2.1.2, we need to impose some conditions on the family of functions $\mathcal{H}$ that we consider. In this section, we will look at two methods of estimating these functions and see what their strengths and weaknesses are.

3.1 Logistic regression

One possible way of finding estimates of h_j is to use logistic regression, a method typically used for classification problems . In logistic regression, we estimate the posterior probabilities that sample x_i belongs to cluster j by assuming that the logit transformations of these probabilities can be expressed via linear functions [18, Section 4.4]. Because all probabilities have to sum up to one, it is conventional to express the probabilities of $k-1$ clusters relative to one reference cluster [19, Section 8.1]. If we use the last cluster as this reference, we get

$$\begin{aligned}
\log \frac{\mathbb{P}(Z=1 \,|\, X=x_i)}{\mathbb{P}(Z=k \,|\, X=x_i)} &= \boldsymbol{\gamma}_1^T x_i \\
\log \frac{\mathbb{P}(Z=2 \,|\, X=x_i)}{\mathbb{P}(Z=k \,|\, X=x_i)} &= \boldsymbol{\gamma}_2^T x_i \\
&\vdots \\
\log \frac{\mathbb{P}(Z=k-1 \,|\, X=x_i)}{\mathbb{P}(Z=k \,|\, X=x_i)} &= \boldsymbol{\gamma}_{k-1}^T x_i,
\end{aligned}$$

with $\boldsymbol{\gamma}_j = (\gamma_{j0}, \gamma_{j1}, \ldots, \gamma_{jd})^T$, with d the amount of features in the dataset. We further define $\mathbb{P}(Z=k \,|\, X=x_i) = 1 - \mathbb{P}(Z \neq k \,|\, X=x_i) = 1 - \sum_{j=1}^{k-1} \mathbb{P}(Z=j \,|\, X=x_i)$. It is easy to see that these equations guarantee that the probabilities sum up to one and remain in $[0,1]$, precisely what we want for h_j. We can find explicit expressions for the functions h_j by rewriting the above as

$$
\begin{aligned}
h_j(x_i) &= \mathbb{P}(Z = j \mid X = x_i) = \frac{e^{\boldsymbol{\gamma}_j^T x_i}}{1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}}, \quad j = 1, 2, \ldots, k-1 \\
h_k(x_i) &= \mathbb{P}(Z = k \mid X = x_i) = \frac{1}{1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}}.
\end{aligned} \tag{3.1}
$$

We see that, in order to find a solution to Equation (2.17), we need to find estimates of the parameters

$$
\boldsymbol{\gamma} = \begin{bmatrix} \gamma_{10} & \gamma_{20} & \cdots & \gamma_{k-10} \\ \gamma_{11} & \gamma_{21} & \cdots & \gamma_{k-11} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{1d} & \gamma_{jd} & \cdots & \gamma_{k-1d} \end{bmatrix}.
$$

that maximise $Q_c^{(t)}(h)$. Unfortunately, no analytical solution exists for this problem, meaning that we have to use a numerical optimisation technique to get the parameter estimates. The most commonly used technique for logistic regression is the Newton-Raphson method [18, Section 4.4.1].

3.1.1 The Algorithm

Using the Newton-Raphson method, we can find that getting estimates for $\boldsymbol{\gamma}$ comes down to repeatedly solving the iterative formula

$$
\boldsymbol{\gamma}^{(s+1)} = \left(\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T} \right)^{-1} \left(\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T} \boldsymbol{\gamma}^{(s)} - \frac{\partial Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma}} \right), \tag{3.2}
$$

in which the superscript (s) denotes the parameter estimates after iteration s of the logistic regression algorithm. The first order partial derivatives can be expressed using matrix notation as

$$
\frac{\partial Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}} = \boldsymbol{A}^T (\boldsymbol{P'} - \boldsymbol{H'}),
$$

where $\boldsymbol{A}$ is as in 3.1, and $\boldsymbol{P'}$ and $\boldsymbol{H'}$ are the matrices of size $n \times k-1$ with elements p_{ij} and $h_j(x_i)$ for $1 \leq j \leq k-1$, respectively. Although $\boldsymbol{\gamma}$ and $\partial Q_c^{(t)}(\boldsymbol{\gamma})/\partial \boldsymbol{\gamma}$ are technically matrices, we can express them as one-dimensional column arrays by appending each consecutive column below the first. This allows us to give the second order partial derivatives as a two-dimensional array, and we can express the Newton-Raphson method in terms of conventional matrix multiplication operations. These second order partial derivatives are given by

$$
\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T} = \begin{bmatrix} -\boldsymbol{A}^T \boldsymbol{W}_1 \boldsymbol{A} & \boldsymbol{A}^T \boldsymbol{W}_{12} \boldsymbol{A} & \ldots & \boldsymbol{A}^T \boldsymbol{W}_{1(k-1)} \boldsymbol{A} \\ \boldsymbol{A}^T \boldsymbol{W}_{21} \boldsymbol{A} & -\boldsymbol{A}^T \boldsymbol{W}_2 \boldsymbol{A} & \ldots & \boldsymbol{A}^T \boldsymbol{W}_{2(k-1)} \boldsymbol{A} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^T \boldsymbol{W}_{(k-1)1} \boldsymbol{A} & \boldsymbol{A}^T \boldsymbol{W}_{(k-1)2} \boldsymbol{A} & \ldots & -\boldsymbol{A}^T \boldsymbol{W}_{k-1} \boldsymbol{A} \end{bmatrix},
$$

in which

$$\boldsymbol{W}_j = \begin{bmatrix} h_j(x_1)(1-h_j(x_1)) & 0 & \dots & 0 \\ 0 & h_j(x_2)(1-h_j(x_2)) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_j(x_n)(1-h_j(x_n)) \end{bmatrix},$$

and

$$\boldsymbol{W}_{jq} = \begin{bmatrix} h_j(x_1)h_q(x_1) & 0 & \dots & 0 \\ 0 & h_j(x_2)h_q(x_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_j(x_n)h_q(x_n) \end{bmatrix}.$$

The full derivation of these equations is included in Appendix A.

To start the algorithm, we need some initial guess of the values of $\boldsymbol{\gamma}$. The most convenient option for this would be to use the estimate found during the previous iteration of the EM-algorithm, since the optimal parameter value is probably not going to change much between iterations. For the first iteration, we can simply initialise the method by setting $\boldsymbol{\gamma} = \mathbf{0}$, the zero vector.

There are multiple strategies we can use to stop the algorithm. First of all, we can define a stopping criterion that defines when we consider the algorithm to have converged. Since the goal of the Newton-Raphson method is to find a root of $\partial Q_c^{(t)}(\boldsymbol{\gamma})/\partial\boldsymbol{\gamma}$, it is common in literature to use the norm of the gradient vector for early stopping. In this thesis, we will stop the algorithm when

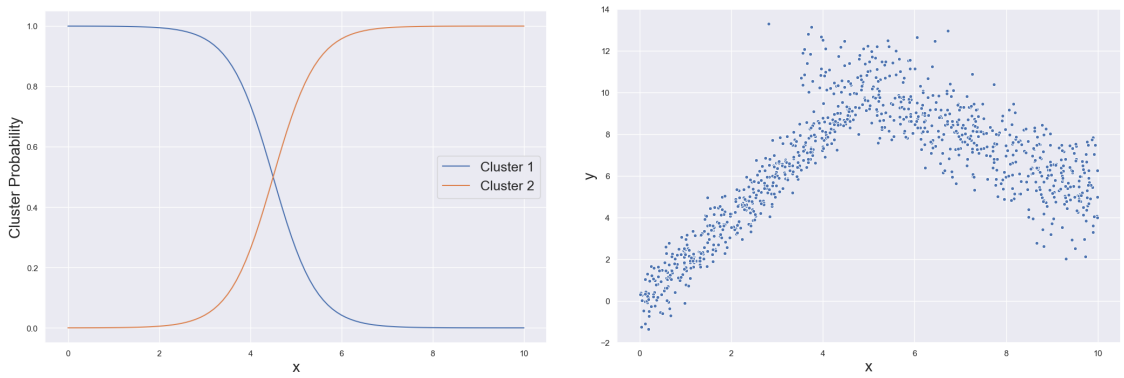
$$\left\| \frac{\partial Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}} \right\|_\infty < \tau.$$

In other words, when all components of the gradient of $Q_c^{(t)}(\boldsymbol{\gamma})$ are below a certain threshold, we consider its maximum to be found.

Unfortunately, the estimates might not actually converge. It is possible that we keep "jumping" around a local maximum, but never actually converge to it. We will therefore also set a maximum number of iterations. Furthermore, some parameter values might actually tend to infinity, which will cause integer overflow errors. A simple solution to this is to keep the parameter estimates constant whenever their size exceeds some threshold that is known to cause such errors. Of course, this precise threshold is dependent on the used data set and the used computational program.

3.1.2 Discussion

As mentioned before, logistic regression is a popular method to estimate the probabilities in a conditional categorical distribution, conditioned some features, often in the context of classification problems. It has the advantage that it provides a simple set of formulas that describe h, and there are no hyper-parameters that need to be tuned. A disadvantage of this method is that it makes pretty strict assumptions about the shapes of the cluster functions. Specifically, it assumes that all clusters are linearly separable, while in reality this will often not be the case. The typical solution that is used to solve this problem is to adjust the feature space by adding transformations of the available features. For example, when $\mathcal{X} = \mathbb{R}^2$ and the two features are denoted by x_1 and x_2, we can add x_1^2, x_2^2 and x_1x_2 to our data set, and use the logistic regression algorithm with these five features. This allows us to find clusters separated by conic sections, i.e. ellipses, hyperbolas and parabolas. A different strategy that is more congruent with the use of mixture models, is to



(a) *(b)*

Figure 3.1: *The cluster probability functions in (a) are used to sample points from two linear models, leading to the point cloud in (b).*

increase the number of clusters k. This way we can split up the non linearly separable clusters into smaller ones which can be separated by linear functions. Increasing k does however carry the risk of overfitting, as it makes it more likely that the method finds clusters containing very few data points. Another disadvantage of logistic regression is that it requires an iterative method to find solutions. Since the EM algorithm is also iterative and we need to find the logistic regression parameter estimates in each iteration of the EM algorithm, computational times can become very long for large data sets.

In the original formulation of the Mixture of Experts methodology, the functions that define the clustering in the data, called the gates, are given by the Logistic Regression Equations (3.1) [40]. The only difference is that for Mixtures of Experts, the values of $h_j(x)$ are not considered to be probabilities, and hence the constraint $\sum_{j=1}^{k} h_j(x) = 1$ is not used. This means k sets of parameters can be found. Nevertheless, predictions are still made using Equation (1.7). It is not clear whether this difference has a significant effect on the resulting predictive model. Furthermore, Gitman et. al. suggested the use of logistic regression as a classification method that can be used to cluster the data in their Predictive CLR method, alongside random forests.

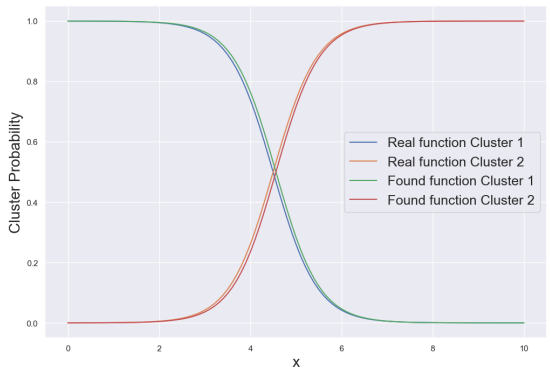
3.1.3 Example

Now that we have discussed the theoretical aspect of using logistic regression, we can take a look at how it performs on a simple example using generated data. We will be using clusterwise linear regression to model the individual clusters. This experiment, and all other experiments that will appear in this thesis, are performed using the Python programming language [38].

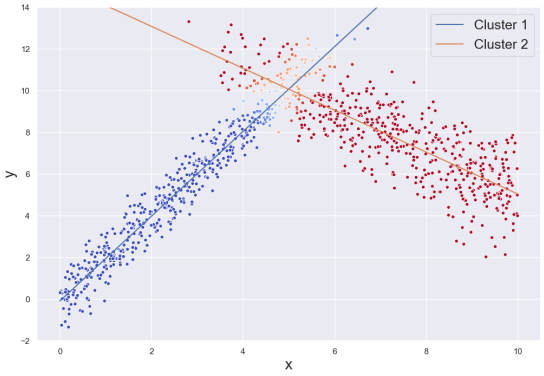
Data was generated from a mixture of two monovariate linear models that have cluster probability functions h_1 and h_2 generated by two Gaussian functions, with parameters $\mu_1 = 3$, $\sigma_1 = 1.2$, $\mu_2 = 6$ and $\sigma_2 = 1.2$, that are normalised by dividing them by the sum of both. This results in the functions visible in Figure 3.1a. The linear models used are

$$y_1 = 2x + \varepsilon_1 \qquad y_2 = 15 - x + \varepsilon_2,$$

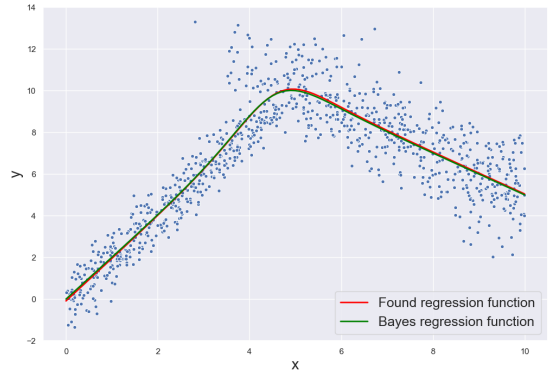
with $\varepsilon_1 \sim \mathcal{N}(0, 0.7)$ and $\varepsilon_2 \sim \mathcal{N}(0, 1.2)$. $n = 1000$ independent features are generated uniformly within $[0, 10]$. The targets are generated by sampling from y_1 with probability $h_1(x_i)$ and y_2 with probability $h_2(x_i)$ for all $i \in \{1, 2, \ldots, n\}$. This results in the point cloud visible in Figure 3.1b.



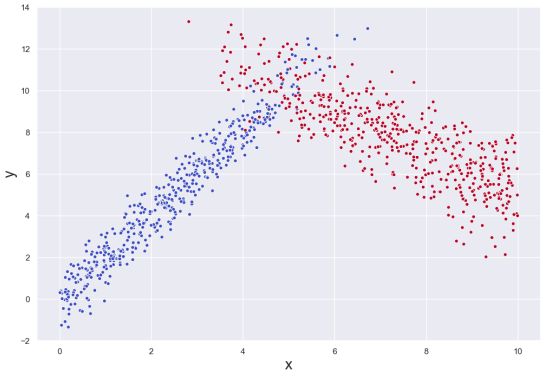
(a) The found cluster probability functions together with the real cluster probability functions.



(b) The found linear models. The colour gradient of the points indicate the inferred probability of belonging to each cluster.



(c) The found regression function together with the Bayes optimal regression model.

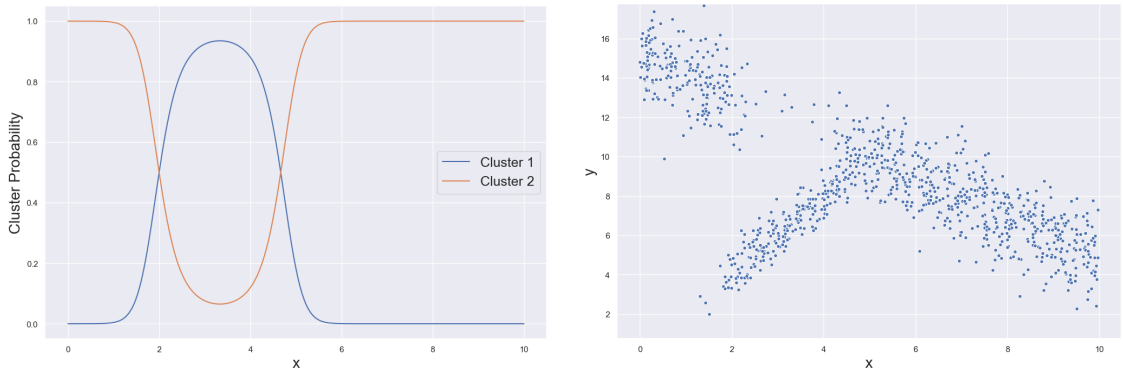


(d) The actual clustering of the data.

Figure 3.2: *The results from making a predictive model using mixture models. Logistic regression is used to estimate the cluster functions.*

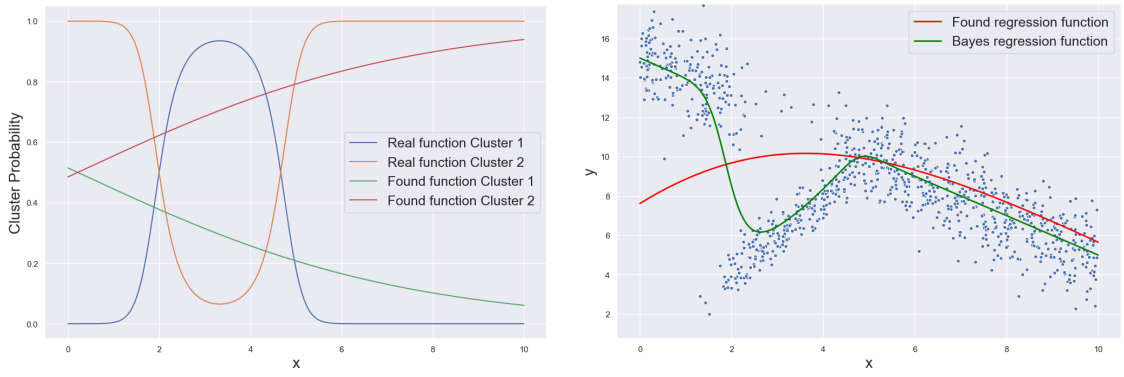
A predictive model is constructed for this data using $k = 2$, where logistic regression is used to estimate h and clusterwise linear regression to estimate g_1 and g_2. The EM algorithm is ran for 100 iterations, and each logistic regression algorithm for a maximum of 20 iteration. The results of this is visible in Figure 3.2. We see that the methods was able to find the two linear models and can approximate the true clustering functions rather well. The resulting model is presented alongside the Bayes regression function, which is the model that minimises the expected risk given in Expression (1.6). We see that the found regression model is very similar to the Bayes optimum function, which indicates that the found model will have a performance that is very close to optimal.

We will now look at an example where the two clusters cannot be separated by a linear hyperplane. We do this by adjusting the parameters of the Gaussian functions used to create the cluster probability functions to $\mu_1 = 4$, $\sigma_1 = 0.5$, $\mu_2 = 6$ and $\sigma_2 = 1$, which results in the function visible in Figure 3.3a. The corresponding set of data points is visible in Figure 3.3b. Since the two clusters are now no longer linearly separable, the logistic regression method is no longer able to find the correct clusters if we choose $k = 2$, which we can see in figure 3.4. We can, however, easily solve this problem by choosing $k = 3$, for which the results are visible in Figure 3.5. We see that, essentially, we split one of the 'true' clusters in two, resulting in three clusters that are linearly separable and can therefore be modelled using the logistic regression functions. This example shows that increasing k can indeed solve the problem where the 'true' clusters in the data are not



(a) ***(b)***

Figure 3.3: *The clustering functions in (a) are used to sample points from two linear models, leading to the point cloud in (b).*



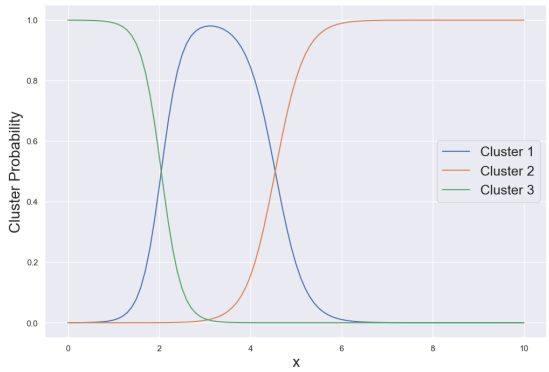
(a) *The found cluster probability functions together with the real cluster functions.*

(b) *The found regression function together with the Bayes optimal regression model.*

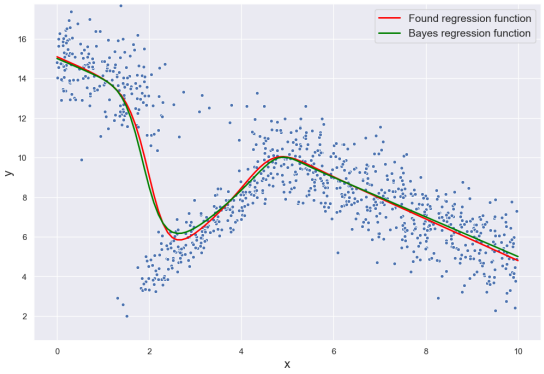
Figure 3.4: *The results from making a predictive model using mixture models. Logistic regression with $k = 2$ is used to estimate the cluster functions.*

linearly separable.

For comparison, the clusterwise regression function is again fit using $k = 2$, but for the logistic regression model, the squared values of the original features x_i^2 for all $i \in \{1, 2, \ldots, n\}$ are added as a second feature. This will allow this model to find non linear cluster boundaries. The results can be seen in Figure 3.6, where we see that this strategy is also successful in finding the correct clusters. It seems that for this simple scenario both methods of dealing with the constraint that simple logistic regression cannot deal with non-linear cluster boundaries, work well. Which of the two methods works best will depend on the given problem, and trying both, or a combination of both, is likely the best strategy. Nevertheless, from here on out we will not use the method of including transformations of the features, since the main interest in this thesis is to see how well the method of increasing k works.

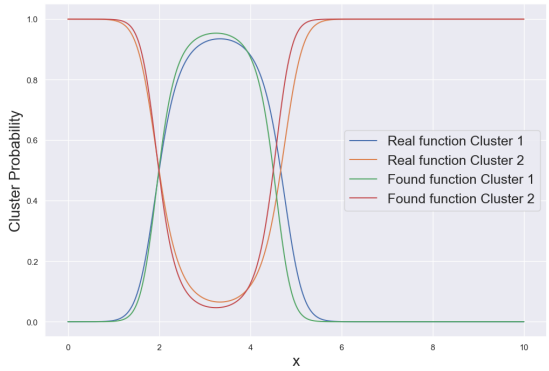


(a) The three cluster probability functions that the method finds.

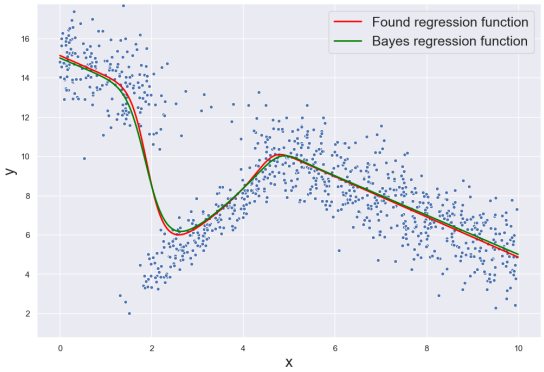


(b) The found regression function together with the Bayes optimal regression model.

Figure 3.5: *The results from making a predictive model using mixture models. Logistic regression with $k = 3$ is used to estimate the cluster functions.*



(a) The found cluster probability functions together with the real cluster functions.



(b) The found regression function together with the Bayes optimal regression model.

Figure 3.6: *The results from making a predictive model using mixture models. Logistic regression with $k = 2$ is used to estimate the cluster functions. The squared values of the original features are added as extra features in order to create non linear cluster boundaries.*

3.2 The Kernel smoothing method

Instead of solving the optimisation problem 2.17 outright, we can also try to estimating the cluster probability functions h_j non-parametrically. This will enable us to model these functions without making strong assumptions about their shape. As an example, we will consider a kernel smoothing method.

In this method, we estimate $h_j(x)$, the probability of a target point x being in cluster j, by taking the weighted average of the cluster probabilities p_{ij} for $i \in \{1, 2, \ldots, n\}$ in the training set. We want to give more weight to those training samples that are close to the target point, in other words, we want the weights to scale negatively with the distance between the points $\|x - x_i\|$, where $\|\cdot\|$ denotes the Euclidean norm. We can achieve this by using a kernel function $K_\lambda(x, x_i)$, where λ can be considered to be a smoothing parameter. A simple kernel is, for example, the boxcar kernel

$$K_\lambda(x, x_i) = \begin{cases} 1 & \text{for } \|x - x_i\| \leq \lambda \\ 0 & \text{for } \|x - x_i\| > \lambda \end{cases},$$

but since it is not continuous, the resulting estimate $\hat{h}_j(x)$ is not smooth. We therefore prefer to use kernels that are continuous. Two popular choices are the Epanechnikov kernel

$$K_\lambda(x, x_i) = \begin{cases} 1 - \left(\frac{\|x - x_i\|}{\lambda}\right)^2 & \text{for } \|x - x_i\| \leq \lambda \\ 0 & \text{for } \|x - x_i\| > \lambda \end{cases},$$

and the Gaussian kernel

$$K_\lambda(x, x_i) = e^{-\left(\frac{\|x - x_i\|}{\lambda}\right)^2}.$$

Using these kernels, we can estimate $h_j(x)$ at iteration $t+1$ of the M step of the EM algorithm by taking the kernel weighted sum of all $p_{ij}^{(t)}$ for $i \in \{1, 2, \ldots, n\}$, scaled by the sum of these weighted sums for all $j \in \{1, 2, \ldots, k\}$. In other words, we can express these estimates as

$$\begin{aligned} h_j^{(t+1)}(x) &= \frac{\sum_{i=1}^n p_{ij}^{(t)} K_\lambda(x, x_i)}{\sum_{j=1}^k \sum_{i=1}^n p_{ij}^{(t)} K_\lambda(x, x_i)} \\ &= \frac{\sum_{i=1}^n p_{ij}^{(t)} K_\lambda(x, x_i)}{\sum_{i=1}^n K_\lambda(x, x_i) \sum_{j=1}^k p_{ij}^{(t)}} \\ &= \frac{\sum_{i=1}^n p_{ij}^{(t)} K_\lambda(x, x_i)}{\sum_{i=1}^n K_\lambda(x, x_i)}. \end{aligned} \tag{3.3}$$

We see that method guarantees that $\sum_{j=1}^k h_j^{(t)} = 1$ and $h_j^{(t)} \in [0, 1]$, since all $p_{ij}^{(t)} \in [0, 1]$ as well. The last line in the expression above is actually the same as the one used in Nadaraya-Watson regression [18, Section 6.1], where the probability estimates $p_{ij}^{(t)}$ act as the targets.

The question remains, which kernel function should we use? The most important difference between the Epanechnikov and Gaussian kernel is that the former has a finite support, meaning that only training points within a distance λ of the target point are used in the calculation. The

Gaussian kernel on the other had has infinite support, so every data point will be used. This means that using the Gaussian kernel can potentially lead to longer computation times compared to the Epanechnikov kernel. However, when data is sparse it might happen that there are no training data points within a distance λ away from a new data point x, meaning that $\hat{h}_j(x)$ will be undefined for this point when the Epanechnikov kernel is used. For this reason, the Gaussian kernel seems a better fit, and it is this kernel function that will be used in experiments using this method. Bear in mind that when we implement the kernel smoothing method, we need to deal with finite computer precision, meaning that even when we use the Gaussian kernel, $\hat{h}_j(x)$ can be undefined for some points. In those situations, we will set $\hat{h}_j(x) = \frac{1}{k}$.

To use these kernel functions, we have to compute the distance between points in the feature space. This means that the range of values each feature can have greatly affects the value of the kernel function. To see why this is the case, image we are building a predictive model on a data set with two features, x_1 and x_2, with he values of $x_1 \in [0, 10]$, and the value of $x_2 \in [1000, 10000]$. When we compute the distance between any two data points in this dataset, the value of x_2 is going to dominate, and the value x_1 will have very little effect! To rectify this, we can scale the values of the parameters. A simple way of doing this is to subtract the mean value of each column in the data set from the values in that column, and then divide it by the standard deviation of the values in the column. This causes the values of all features to be centred around 0, and have a standard deviation of 1. The result of this is that each feature will have the same importance when we compute the kernel function value between points.

As discussed by Larry Wasserman, the precise kernel used makes little difference in terms of estimating a function [39]. What is important is the choice of the bandwidth parameter λ. The bigger λ is, the more weight we give to training data points far away from the target point, which leads to a smoother function. This means that λ acts as a regularisation parameter that determines how smooth the estimate of h_j is. If we choose λ too small we risk overfitting the data, while choosing a value too big can lead to underfitting. At the extremes, choosing $\lambda \downarrow 0$, we essentially get back the "bad" estimate in Expression (2.18). On the other hand, if we were to take $\lambda \to \infty$, we estimate h_j as a constant function.

3.2.1 Discussion

The kernel smoothing method discussed here is nonparametric and we make very few assumptions about the shape of h_j. This has the benefit that the estimation functions can be very flexible, meaning that it has the potential to model more complicated cluster structures. It does, however, mean that we do not solve 2.17 outright, and hence are not maximising the complete data likelihood function. This means we are no longer using the proper EM algorithm, rather we are using a method that is inspired by it.

The kernel smoothing method does not "learn" a model, rather it takes some weighted average of the training data to make an estimate for a new data point. This has the potential of long computation times during evaluation if the used training data set is large. This method is also controlled by a regularisation parameter λ, which we need to optimise somehow. Methods that attempt to do this will be discussed in Section 3.2.3.

To the best of the author's knowledge, no other method that builds a predictive model by clustering the data a fitting a model to each cluster uses a non-parametric method such as the kernel smoothing method to cluster the data.

3.2.2 Example

In a similar way as we did for the logistic regression method, we will now take a look how the kernel smoothing method performs at estimating the cluster probability functions. For this purpose, we will again look at the generated data set from Figure 3.3. A regression model is build, using the Gaussian kernel and $k = 2$, for three different values for the bandwidth parameter, namely $\lambda = 0.02$ which leads to an overfitted model, $\lambda = 2$ which underfits the data, and finally $\lambda = 0.4$, which seems just about right for this data set. The result of this are visible in Figure 3.7. This example shows that choosing the right value for λ is paramount in making an accurate predictive model.

To see the what the effect is of the bandwidth of the kernel function, a test set of size 100,000 is generated using the same model as the training data. For various values of λ, the clusterwise regression model is trained on the training set, and the mean squared error (MSE) is calculated on both data sets using

$$MSE(\lambda) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(\lambda))^2,$$

where $\hat{y}_i(\lambda)$ indicates the estimate given by the clusterwise regression function with bandwidth λ. The result of this is visible in Figure 3.8, where the blue line corresponds to the scores of the training set, the orange line of the test set and the horizontal black line indicates the Bayes score, the mean squared error calculated on the test set using the Bayes regression function. We see the MSE of the training set will only decrease as the value of λ gets smaller, but the MSE of the test set has a minimum which is almost the same as Bayes score.

3.2.3 Choosing the Bandwidth Parameter

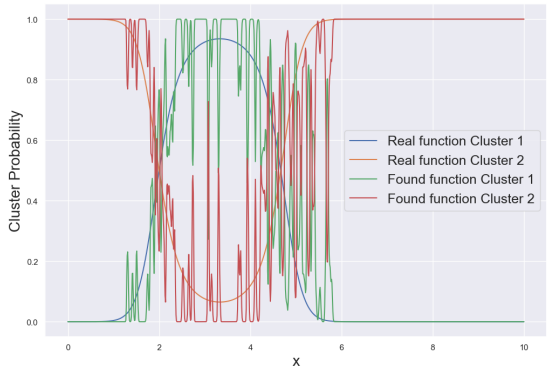
We see that the kernel smoothing method is regularised by the bandwidth parameter λ. It is clear that choosing the right value for λ is very important for making good predictions. But how do we decide what the best choice of λ is? We want to choose this parameter so that the resulting predictive model $\hat{f}_\lambda$ minimises the statistical risk $R(\hat{f}_\lambda)$, defined in Equation (1.4). Note that the distribution $\mathbb{P}_{XY}$ is generally unknown, making it impossible to calculate the true risk. It is, however, possible to estimate the risk with the use of a set of training data. let $D_n \equiv \{x_i, y_i\}_{i=1}^n$ denote the available training data. A popular method for estimating the true risk for a given training set is the use of cross-validation. In this method, the training set is randomly split in C different "folds". Then, the data in $C - 1$ of these folds is used to train the model. The mean squared error is then calculated on the data in the remaining fold, giving an estimate of the expected risk. This procedure is repeated for each fold, giving us C different estimates. We then take the average of these to get a final estimate which we call the cross-validation score CV.

If we define $c : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, C\}$, an indicator that specifies for each data point to which fold it belongs to, we can define the cross-validation score CV as

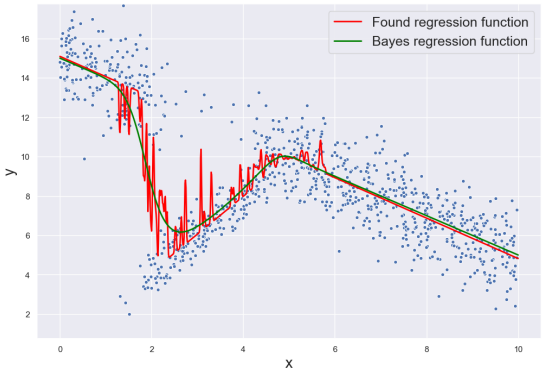
$$\hat{R}(\lambda) = CV(\hat{f}, \lambda) \equiv \frac{1}{n}\sum_{i=1}^{n} L(\hat{f}_\lambda^{-c(i)}(x_i), y_i),$$

where $\hat{f}_\lambda^{-c(i)}$ denotes the model trained using the data from all but the cth fold. We can then choose the best value of λ using

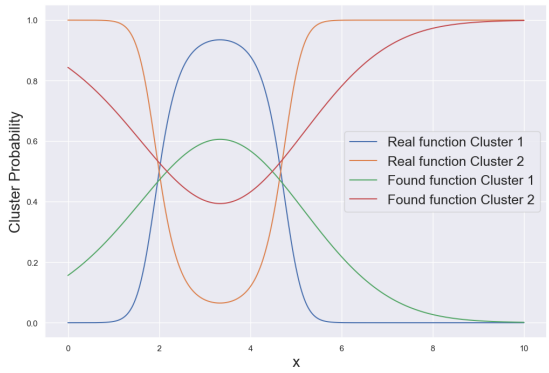
$$\lambda^* = \arg\min_\lambda \hat{R}(\lambda). \tag{3.4}$$



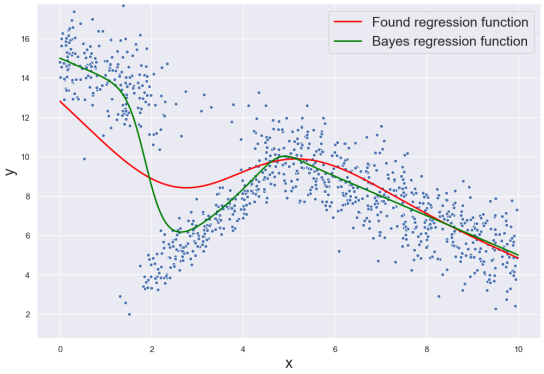
(a) The found cluster functions together with the real cluster functions, for $\lambda = 0.02$.



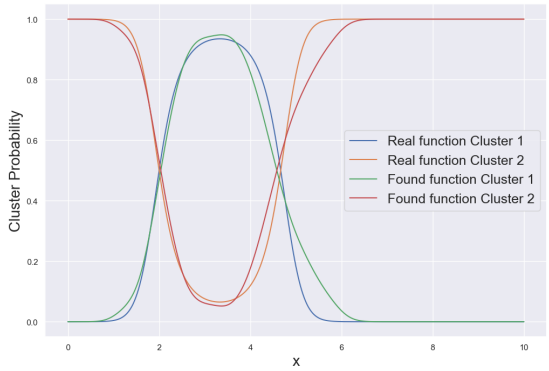
(b) The found regression function together with the Bayes optimal regression model.



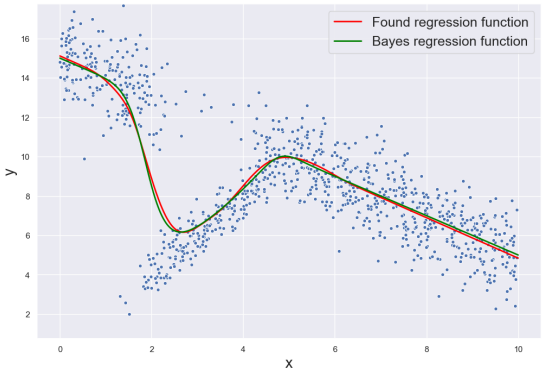
(c) The found cluster functions together with the real cluster functions, for $\lambda = 2$.



(d) The found regression function together with the Bayes optimal regression model.



(e) The found cluster functions together with the real cluster functions, for $\lambda = 0.4$.



(f) The found regression function together with the Bayes optimal regression model.

Figure 3.7: *The results from making a predictive model using mixture models. The kernel smoothing method with $k = 2$ and three different values for λ is used to estimate the cluster functions.*

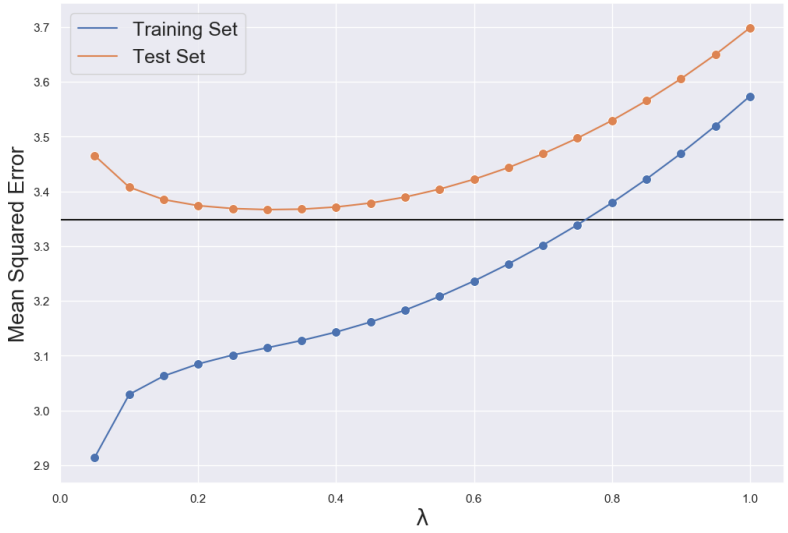


Figure 3.8: *Comparison of the mean squared error, computed on both the training and test sets, for different values of λ. The horizontal black line denotes the Bayes score.*

A thing to be aware of when using cross-validation is that it gives a biased estimate of the true risk. The performance of most predictive modelling methods is dependent on the sample size of the used training data set. The bigger the sample size, the better the method performs. During cross-validation, we essentially reduce the size of the the training data set to $\frac{C-1}{C}$ times its original size, meaning that the cross-validation score is biased upwards. Increasing C will reduce this bias, but will lead to longer computation times.

A logical limit to cross-validation is to take $C = n$, where each fold consists of just one sample. This is often called leave-one-out cross-validation (LOOCV) and it has a minimal bias. The main drawback of LOOCV is that it can be computationally intensive, because it often requires the model to be retrained n times using $n - 1$ training points. Luckily, a more computationally efficient technique exists in situations where the estimation methods for h_j and f_{θ_j} are linear fitting methods, that is methods where the training data estimates $\hat{\boldsymbol{y}}$ can be expressed as

$$\hat{\boldsymbol{y}} = \boldsymbol{L}(\boldsymbol{x})\boldsymbol{y},$$

where the matrix $\boldsymbol{L}(\boldsymbol{x})$ is called the hat matrix [15]. For most linear fitting methods, we can express the LOOCV estimate $\hat{y}_{-i}$, where the sample $\{x_i, y_i\}$ is left out of the training data, using

$$\hat{y}_{-i} = \frac{1}{1 - L_{ii}} \sum_{\substack{l=1 \\ l \neq i}}^{n} L_{il} y_l. \tag{3.5}$$

This means that it is not necessary to compute the matrix $\boldsymbol{L}_{-i}$ for each step of the LOOCV procedure. Unfortunately, even though is also a linear fitting method, the above equation does not hold for the clusterwise predictive model considered in this Section. However, we can still use it if the methods to estimate h_j and/or f_{θ_j} are linear fitting methods for which the equation holds. For kernel smoothing methods (proof is given in Appendix A) and linear regression models [36], the equation does hold. We can write the estimates of the training data using Equation (2.25) as

$$\begin{aligned}\hat{\boldsymbol{y}} &= \sum_{j=1}^{k} \hat{h}_j(\boldsymbol{x}) \odot \boldsymbol{A}\boldsymbol{\beta}_j^T \\ &= \sum_{j=1}^{k} \boldsymbol{K}\hat{\boldsymbol{p}}_j \odot \boldsymbol{A}\left(\boldsymbol{A}^T\boldsymbol{W}_j\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{W}_j\boldsymbol{y} \\ &= \sum_{j=1}^{k} \boldsymbol{K}\hat{\boldsymbol{p}}_j \odot \boldsymbol{H}_j\boldsymbol{y},\end{aligned}$$

where we use $\odot$ to denote the componentwise multiplication of two vectors of the same length. $\boldsymbol{K}$ is the scaled kernel matrix with elements

$$K_{ij} = \frac{K_\lambda(x_i, x_j)}{\sum_{l=1}^{n} K_\lambda(x_i, x_l)},$$

$\hat{\boldsymbol{p}}_j = (\hat{p}_{1j}, \hat{p}_{2j}, \ldots, \hat{p}_{nj})^T$ and $\boldsymbol{H}_j$ are the hat matrices defined as $\boldsymbol{H}_j = \boldsymbol{A}\left(\boldsymbol{A}^T\boldsymbol{W}_j\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{W}_j$.

With this notation, we can find the expressions of the LOOCV estimates as

$$\begin{aligned}\hat{y}_{-i} &= \sum_{j=1}^{k} \frac{1}{1-K_{ii}} \left(\sum_{\substack{l=1 \\ l\neq i}}^{n} K_{il}\hat{p}_{lj}\right) \frac{1}{1-(H_j)_{ii}} \left(\sum_{\substack{l=1 \\ l\neq i}}^{n} (H_j)_{il} y_l\right) \\ &= \frac{1}{1-K_{ii}} \sum_{j=1}^{k} \frac{1}{1-(H_j)_{ii}} \left(\sum_{\substack{l=1 \\ l\neq i}}^{n} K_{il}\hat{p}_{lj}\right) \left(\sum_{\substack{l=1 \\ l\neq i}}^{n} (H_j)_{il} y_l\right).\end{aligned} \tag{3.6}$$

We can choose a number of different values for λ, run the EM algorithm until the convergence criteria is met and perform LOOCV to get $\hat{R}(\lambda)$ and choose the "optimal" value of λ using 3.4. However, it might be the case that the value of λ that minimises the estimated risk is different at every iteration of the EM algorithm. It is therefore also an option to choose the best value of λ at each iteration. This way we might find a better solution than we would by keeping λ constant. Of course, this will increase the computation time needed for training the model. A compromise might be to only compute the optimal value of λ once every few iterations.

Another question is, how do we decide which values of λ to check? We can keep the values that we check the same each iteration, but perhaps it is better to automatically adapt these values. During the first few iterations of the EM algorithm, the parameter estimates will change significantly. This will make it likely that the optimal value of λ of two successive iterations can potentially also change a lot. On the other hand, for later iterations the parameter values will only change a little and we suspect that the optimal value of λ also does not change a lot each iteration.

A possible strategy is to, at iteration $t+1$, do a local search around the optimal value found at at the previous iteration $\lambda^{(t)}$. A possible set of values to be evaluated are

$$\left\{\frac{\lambda^{(t)}}{1+c^s},\ \lambda^{(t)},\ \lambda^{(t)}(1+c^s)\right\} \qquad \text{with } 0 < c < 1,$$

and with

$$s = \sum_{l=1}^{t} \left(\mathbb{1}\{\lambda^{(l)} = \lambda^{(l-1)}\} - \frac{1}{2}\mathbb{1}\{\lambda^{(l)} \neq \lambda^{(l-1)}\} \right),$$

so that we look at the previous optimal value, one smaller value and one greater value. The advantage of using this set of estimates for λ is that we can find a very precise estimate the value that minimises the LOOCV score. s gives a measure of how often we find the same "optimal" value of λ. The higher the value of s, the smaller the interval of values of λ that we evaluate. The negative term in this sum is added to insure that, in the case where the optimal value changes between successive iterations of the EM algorithm, we do not risk ending up with a "bad" estimate because we still allow the distance between checked values to increase. The reason for the extra factor $\frac{1}{2}$ is that we want s to still grow from one estimate of λ to another, that are similar distances away from the optimal value, which will cause a loop where the two situations $\lambda^{(l)} = \lambda^{(l-1)}$ and $\lambda^{(l)} \neq \lambda^{(l-1)}$ keep alternating each other.

3.2.4 Verification

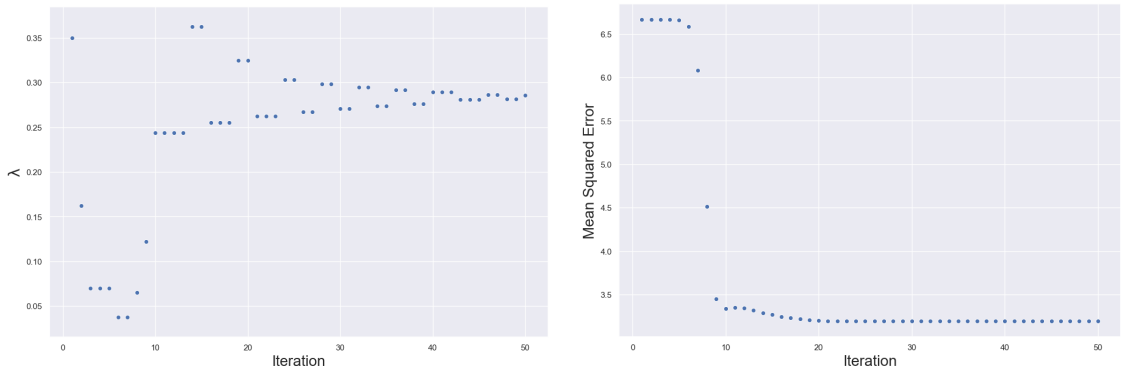
We obviously want to verify that the method discussed above indeed finds the optimal value of λ. To this end, this method is tested on the synthetic dataset from the previous example, where we used $\lambda^{(0)} = 0.7$ and $c = 0.75$. The optimal value of λ found this way is 0.286, which if we look at Figure 3.8 does seem to be around the value that minimises the MSE of the test set.

In Figure 3.9a, we see how the value of λ that the method chooses changes at each iteration. It seems that for the first few iteration, a small value of λ is chosen. This is probably because during the first few iterations, the models of the individual clusters do not fit data very well yet, so the cluster probability functions need to have large fluctuations to capture any structure in the data. After this initial period, we see the optimal value slowly converge to the real optimal value.

Figure 3.9b shows how the LOOCV score changes per iteration of the algorithm. We see that for the first few iterations this score barely changes, after which there suddenly is a huge jump. Finally, after this jump the score slowly converges. It seems that the score does not change much after about 20 iterations, even though the optimal value of λ has not been found yet. However, in Figure 3.8 the MSE function of the test set does seem almost constant for $0.2 \leq \lambda \leq 0.4$, which would explain this. We can conclude that the described method of automatically finding the optimal value of λ does work for this synthetic data set.

3.2.5 Note of Caution

There is an issue with the way we perform cross validation here that is important to note. When we are fitting a model that can make a prediction about a target value based on some features, it is very important that no information about the the target value is used as input of the model. During the M-step of iteration t of the algorithm, we fit a model that estimates the target based on the features and cluster probability estimates $p_{ij}^{(t-1)}$. Each of these estimates was computed during the previous iteration of the algorithm using the entire dataset. In other words, the are function of all data points, i.e. $p_{ij}^{(t-1)}(\{x_l, y_l\}_{l=1}^{n})$. When we perform LOOCV, we estimate y_r using $\hat{y}_{-r}$ given by Equation (3.6), which we see is a function of all $p_{ij}^{(t-1)}$. But this means that $\hat{y}_{-r}$ is also a function of the entire dataset, including y_r! This means that we build a predictive model that uses the value we wish to predict as an input. The result of this is that the LOOCV score computed this way might not have the properties typically enjoyed by the classical LOOCV score. In particular, it might not be a good estimate of the true risk.



(a) *Progression of the optimal value of λ found via LOOCV at each iteration*

(b) *Progression of the LOOCV mean squared error at each iteration.*

Figure 3.9: *The results from using the automatic bandwidth selection method for the kernel smoothing method.*

The question remains if this is actually a bad thing. We know that we cannot use the LOOCV score computed this way as an indicator of how well our predictive model will perform on a new data set. This, however, is not the reason we calculate these scores. We use them to find the optimal value of λ, and it can still be used for this purpose if the biases are independent of this bandwidth parameter. It is hard to verify if this is the case, but based on the example above and other experiments that have been performed, it does seem to be the case.

Chapter 4

Algorithmic Considerations

Until now, we established a theoretical framework centred around the idea of building a predictive model under the assumption that there are multiple distinct clusters in the data. We developed a statistically motivated method of constructing a predictive rule for such data, which splits the problem in two parts. Firstly, we need to construct a function that can cluster the data based on the values of the features, and secondly we will need to make predictive model for each of the clusters. We have already seen some concrete examples of how to perform both parts, and we have seen that they work for some simple examples using synthetic data.

In order to present a robust and general algorithm that can be used to build a predictive model for a given data set, however, there are still a number of aspects that we need to consider. Specifically, we will need to decide how to initialise the method and how to choose the optimal number of parameters. But firstly, it will be argued that the method we have developed can be used in more situations than we have until now assumed.

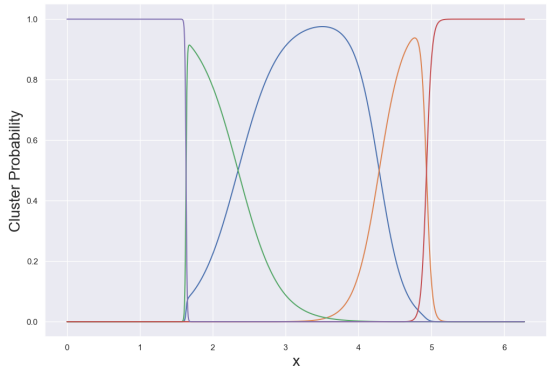
4.1 Modelling Non-Linear Functions

In the introduction of this thesis, we gave as the motivation behind developing clusterwise predictive models that in some datasets there might be some sort of heterogeneity in the data. There are a number of sub-groups in the data, and for each of these groups, a different model performs better at making predictions. Which of these groups each data point belongs to might not be given in the dataset, and therefore needs to be learned by the method. However, for a given dataset it might be hard to determine whether or not there actually are any sub-populations, and even if there are, it does not necessarily mean that a different predictive model needs to be used for each of these groups. Furthermore, it might be impossible to distinguish the different groups from one another using only the values of the features. Considering all this, it seems that the developed method has limited use. The goal of this section is to argue that this is actually not the case. To use this method, there does not necessarily need to be specific sub-groups in the data.

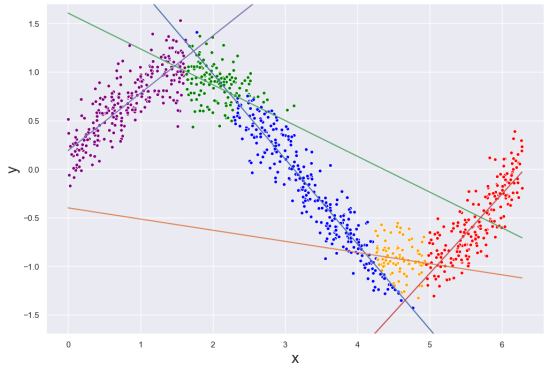
Clusterwise predictive models can also be used as a general tool to model non-linear relationships between target and features. To see an example of this, consider the following dataset. Let, for all $i \in \{1, 2, \dots, n\}$ the relationship between target y and feature x be given by

$$y_i = \sin x_i + \varepsilon_i,$$

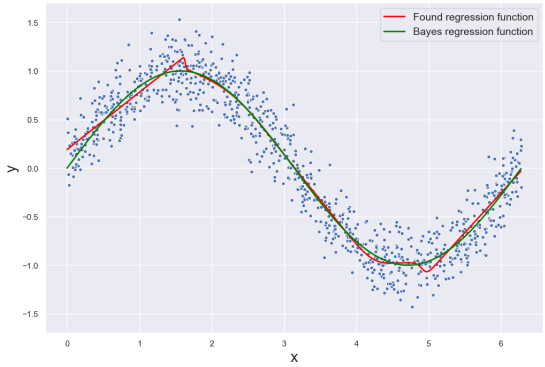
with $\varepsilon_i \sim \mathcal{N}(0, \sigma)$. Clearly, a linear model will not perform well on this dataset. If we used clusterwise linear regression, however, we can essentially break up this data in smaller subsets



(a) The found cluster probability functions.



(b) The found linear models. The colours of the data points correspond to the linear model it has the highest inferred probability of belonging to.

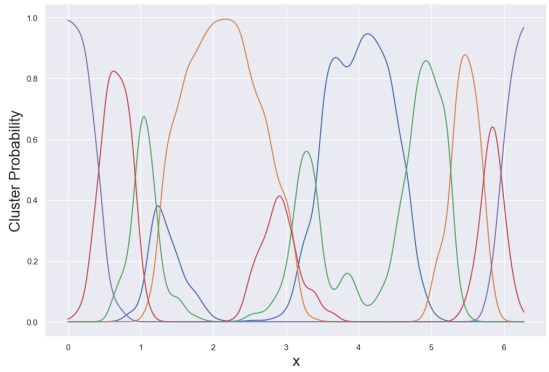


(c) The found regression function together with the Bayes optimal regression model.

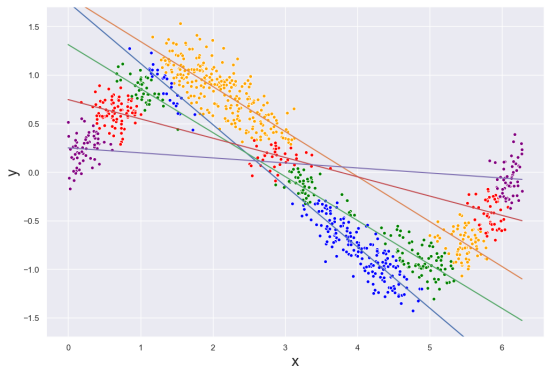
Figure 4.1: *The results from making a predictive model using clusterwise linear regression. Logistic regression is used to estimate the cluster probability functions.*

that each can be approximated by a linear model. An example of this is shown in Figure 4.1, where a clusterwise linear regression model with $k = 5$, using logistic regression to find the cluster probability functions, is fitted to data that is generated using the equation above for one period, with $\sigma = 0.2$. As we can see, the resulting regression function can approximate the Bayes regression function, which is obviously given by $f_{\text{Bayes}} = \sin x$, rather well. Of course, we could have used polynomial regression with degree 3 for this dataset as well. For datasets with more features and a more complicated non-linear relationship between target and features, however, polynomial regression might not perform well, and in those situations using a clusterwise predictive model might be better.

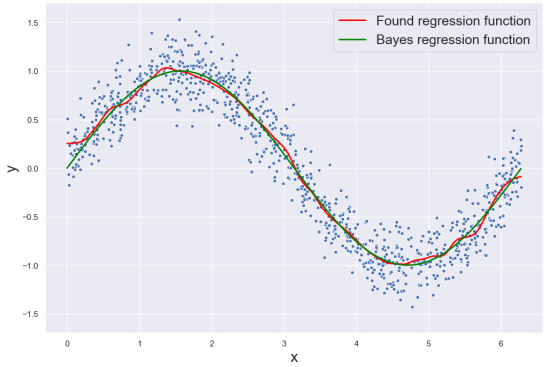
For comparison sake, let us see how the clusterwise linear regression performs on this dataset while using the kernel smoothing method to estimate the cluster probability functions. In Figure 4.2, we can see that this leads to very different linear models. The clusters in the data that this method finds can still be modelled well using linear functions, but they are no longer linearly separable. One consequence of this that we can see in these figures is that the linear models can be almost perpendicular to the gradient of the resulting regression function, which for this dataset means that the final model does not perform well near the boundaries of the range of the data, as can be seen in Figure 4.2c. This seems to indicate that models that use the kernel smoothing method to estimate h_j might perform badly at extrapolating. These experiments once again show the difference between logistic regression and the kernel smoothing method when it



(a) The found cluster probability functions.



(b) The found linear models. The colours of the data points correspond to the linear model it has the highest inferred probability of belonging to.



(c) The found regression function together with the Bayes optimal regression model.

Figure 4.2: *Each figure corresponds to the linear models that were found in the Iris dataset, for four different random initialisations.*

comes to estimating the cluster probability functions.

4.2 Initialisation

In Section 2.1 we saw that the EM algorithm requires some initial estimate of the parameter values. If we were to start with the E-step, we would need to have initial estimates of $h_j(x_i)$ for all $i \in \{1, 2, \ldots, n\}$ and all $j \in \{1, 2, \ldots, k\}$, as well as initial guesses of the model parameters. For clusterwise linear regression this would be $\boldsymbol{\beta_j}$ and σ_j for all $j \in \{1, 2, \ldots, k\}$. Alternatively, we could start the algorithm at the M-step, which means that we only need to have an initial estimates for p_{ij} for all $i \in \{1, 2, \ldots, n\}$ and all $j \in \{1, 2, \ldots, k\}$. Because of the need to have initial guesses for less values, and because the initialisation problem does not change for different methods to model the clusters, starting with the M-step seems preferable and it is this strategy that we will henceforth consider.

As is discussed in Section 2.1.1, the EM algorithm finds a local maximum of likelihood function. This function, however, will likely contain many maxima, and the one the algorithm finds will not necessarily correspond to the best predictive model. The way we initialise the methods determines which local maximum and therefore which final predictive model we find, making initialisation an important aspect of the algorithm. In fact, many initialisations will lead to a bad clustering. An

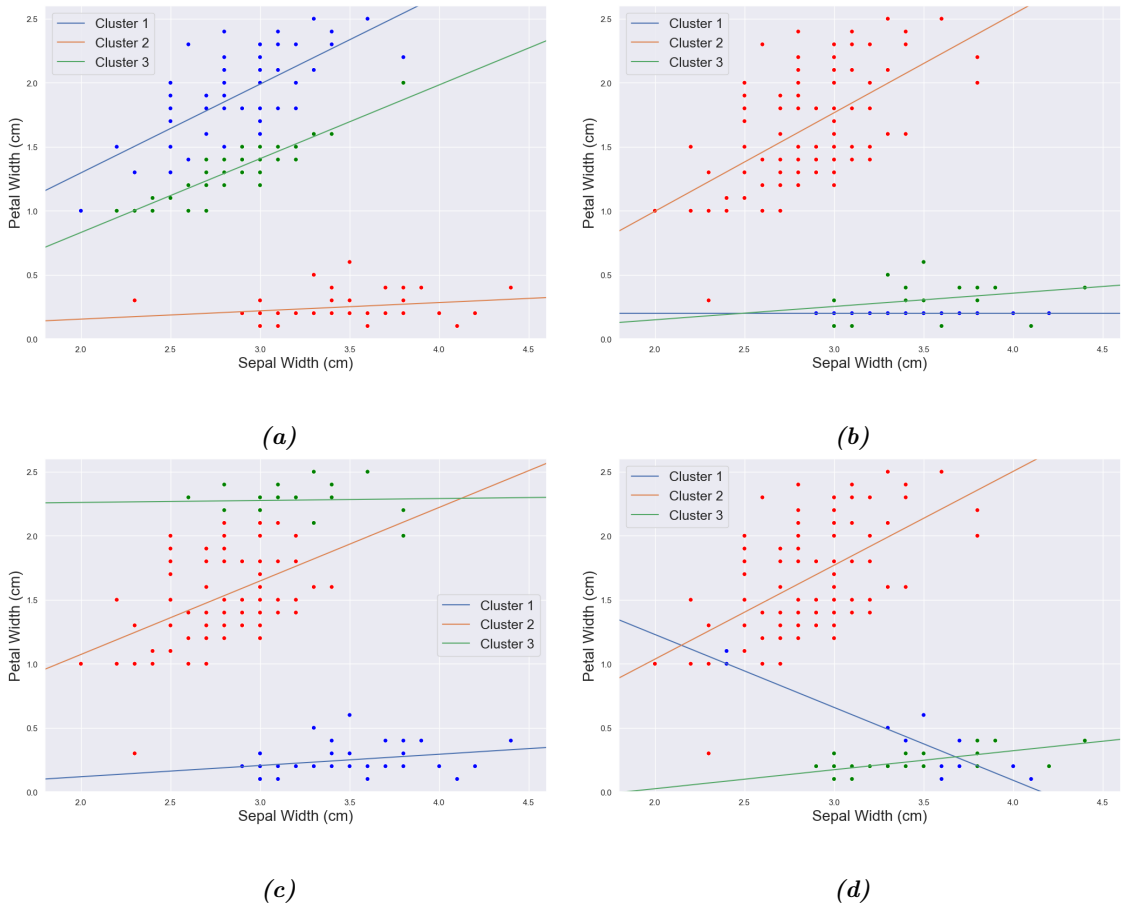


Figure 4.3: *The results from making a predictive model using clusterwise linear regression. The kernel smoothing method is used to estimate the cluster probability functions.*

example of this can be seen in Figure 4.3, where clusterwise linear regression models were fitted to the Iris dataset visible in Figure 1.1. Each plot corresponds to the linear models that were found for a different random initialisation. We see that only the models in 4.3 cluster the data in a way that is similar to the "true" clustering, which is visible in Figure 1.1b. A bad clustering might lead to a bad predictive model, which highlights the need to find an initialisation method that leads to a good final model.

4.2.1 Initial Estimates of the Cluster Probabilities

This does leave us with the question: what method should we use to initialise the EM algorithm? Since we will start at the M step, we will need initial guesses for p_{ij} for all $i \in \{1, 2, \ldots, n\}$ and all $j \in \{1, 2, \ldots, k\}$. There are, of course, many ways of getting these. One solution might be to set all these probabilities to be equal, i.e. $p_{ij}^{(0)} = \frac{1}{k}$. The problem with this is strategy is that this actually corresponds to a stationary point of the likelihood function. When all probabilities are the same, for each cluster the exact same model will be learned and hence the probabilities that each data point belongs to the clusters will remain the same. This stationary point, however, is a very unstable solution as it does not correspond to a local maximum. We can therefore easily circumvent this by adding a small deviation to each probability. We can, for example, choose $p_{ij}^{(0)} = \frac{1}{k} + \varepsilon_{ij}$ with $\varepsilon_{ij} \sim \text{Unif}(-0.01, 0.01)$ for $1 \neq j < k$, and $p_{ik}^{(0)} = 1 - \sum_{j=1}^{k-1} p_{ij}$.

Another strategy is to make the probabilities completely random. This means that we generate $q_{ij} \sim \text{Unif}(0, 1)$ and normalise them to get probabilities $p_{ij} = \frac{q_{ij}}{\sum_{j=1}^{k} q_{ij}}$. Because the initial clusters will be quite different, this method might have a decreased chance of finding multiple clusters with the same model. The large degree of randomness can mean that different initialisation lead to a very different model and hence a different performance.

It is also possible to pre-cluster the data using a conventional clustering method such as k-means [18, Section 13.2.1] or hierarchical clustering [18, Section 14.3.6]. These are hard clustering methods that will assign each data to exactly one cluster, meaning that all $p_{ij}^{(0)} \in \{0, 1\}$. If we want soft labels, or probabilities of belonging to each cluster, we can use a fuzzy clustering method, such as fuzzy c-means clustering [2].

We need to remember that the clustering in clusterwise predictive modelling is different from the way we usually define clustering. In the traditional sense, each cluster is expected to be somewhat localised and distinct from the other clusters. In the clustering we get using mixture models, this may not be the case. If we for example look at the dataset in Figure 3.3, we see that the cluster that is described by the linear model with the negative slope essentially consists of two separate areas where points are dense. Because of this, these conventional clustering methods will have a hard time of recognising it as a single cluster. It therefore remains to be seen whether the clustering found using these clustering methods will lead to a good predictive model.

4.2.2 Estimating Model Performance

In order to compare the models that result from different initialisations, we need some kind of metric of their performance. We could compare them based on their likelihood scores. This, however, is not a good strategy as a higher likelihood does not necessarily correspond to a better predictive model. As a matter of fact, the model that reaches the global maximum of the likelihood function will typically perform poorly at making predictions. To see why this is the case, imagine for example that we try to construct a clusterwise linear regression model on a dataset with a single feature and with $k = 2$. From Equation (2.22), we can see that $\lim_{\sigma_1 \downarrow 0} Q_{m1}^{(t)}(\theta_1)$ will become infinite when $y_i = \hat{\beta}_{10} + \hat{\beta}_{11} x_i \; \forall_i : p_{i1} > 0$, which will be the case when one cluster contains only two data points. Obviously, a predictive model fitted to only two data points will typically not perform well, so a prediction made for a new data point assigned to cluster 1 will often be poor. It is therefore better to compare models resulting from different initialisations using an estimate of their risk.

In Section 3.2.3, we introduce the notion of cross validation as a way of estimating the risk of a model. Unfortunately, using cross validation to compare models resulting from different initialisations is not a good idea. To see why this is the case, it is important to realise that the final model resulting from a certain initialisation is inherently connected to the used training set. If we added or removed some points from the dataset, but kept initial estimate values the same otherwise, we might get a completely different final model. Applying this concept to cross-validation, where we split our data set in C smaller sets with $\frac{C-1}{C} n$ points each, it is easy to see that the models that will be learned on each of these sets can be very different from one another, and from the model that would be found with the same initialisation values on the entire data set. This means that the CV scores will not be representative for the risk of the model found for the entire set.

A possible solution would be to split the available data in two different sets, namely a training set and a verification set. We can then train a model using the training set, and get an estimate of their risk by calculating the mean squared error on the verification set.

4.2.3 Experiments

We will now use this method of splitting the data set in a training and verification set to get an idea of how the prediction models resulting from the different initialisation methods discussed in Section 4.2.1 compare. We will once again be using the Iris data set introduced in Chapter 1 for this. The benefit of experimenting on this dataset is that we know for sure that there are distinct clusters in the data, namely the three different species of the flower. This dataset contains four measurements made on 150 different irises, namely the sepal length, sepal width, petal length and petal width, all in cm. We will be using the first three of these as the features, and the last one, the petal width, as the target we wish to predict.

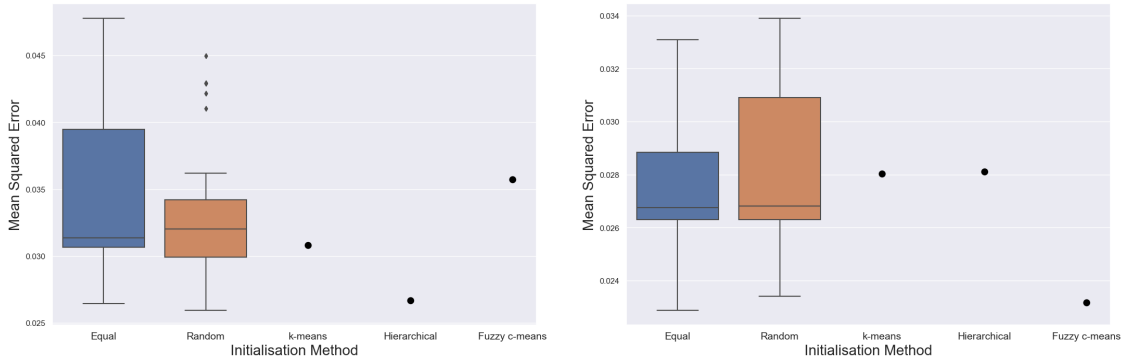
The dataset is randomly split in a training and validation set with sizes $n_t = 100$ and $n_v = 50$ respectively. For each initialisation method described above, a predictive model is constructed on the training set using clusterwise linear regression models using both cluster probability function estimation methods, with $k = 3$, the true number of clusters in the data. The mean squared error is then computed using the verification set. Since the initialisation methods using almost equal initialisation values and completely random initialisation values contain randomness, we want to see if there is much difference between different random initial values. The experiments using these methods will therefore be repeated 50 times.

In these experiments, the k-means and hierarchical clustering implementation from the scikit-learn library [32]. For the fuzzy c-means clustering, the implementation by Madson Luiz Dantas Dias [11] will be used.

The results are visible in Figure 4.4, where the found scores of the experiments regarding the methods containing randomness are summarised by boxplots, with the length of the whiskers a maximum of 1.5 times the interquartile range. There are a number of things to notice in these figures. First of all, the scores acquired from using the kernel smoothing method to estimate the cluster probability functions are on average better than the ones for which logistic regression is used. This seems to indicate that the three different clusters are not linearly separable. Another thing to notice is that the range of the found MSE scores is very big. For comparison, the empirical risk found using the same test/verification split is 0.0342 when fitting a regular linear regression regression model on the entire training set, and when we fit a linear regression model to each species of flower separately, we get an MSE of 0.0215. From this we see that the performance of the models resulting from clusterwise linear regression can range from being much worse than simple linear regression, to being almost as good as that of linear regression models when the real clustering is known. This again highlights the extreme importance of the initialisation of the method.

Another thing to notice is that initialisation via pre-clustering generally leads to good models, but in both sets of experiments, the best models where actually found using the methods with randomness in the initialisation. Lastly using almost equal initialisation values seems to, on average, lead to better models than the completely random initial values.

These experiments have only been conduction on the Iris dataset. However, for all the datasets on which clusterwise predictive modelling has been tested, the large variance in the performances of the models that are acquired with different initialisations has been observed. It seems that the more features the dataset has, the greater this variance is. In the Example in Figure 4.3, shows how different initialisations can lead to different clusterings. Nevertheless, in that example only one feature of the data is considered, and all four initialisations will lead to rather similar final model. In Chapter 6 we will look at some datasets that contain up to 13 features, and there a very large variance in performances have been observed. Some models fit to those datasets had empirical losses found on the validation set that were multiple orders of magnitude greater than that of the best model. Unfortunately, it is hard to make figures that shows two very different final models, that are acquired from two similar initialisations, as this only happens for high



(a) *Using logistic regression to estimate the cluster probability functions.*

(b) *Using the kernel smoothing method to estimate the cluster probability functions.*

Figure 4.4: *The results from trying different methods of choosing initial cluster probabilities. The experiments using near equal and completely random initial probabilities are repeated 50 times, and their results are given by boxplots.*

dimensional datasets.

4.2.4 The Initialisation Strategy

From these experiments, it seems that there is no clear best strategy in regards to choosing initial estimates for each p_{ij}. The wide range of empirical risks found underlines the importance of being able to verify the model performance.

A general strategy to find a good initialisation can be as follows. First, randomly split the available data in a training and a validation set. Secondly, train a number of models on the training set using different initialisations. Pre-clustering can be good first choices for this, followed by initialisation with a near equal probabilities. Lastly, choose the model that has the lowest mean squared error computed on the validation set. Testing a larger amount of initialisations, of course, increases to probability of finding a good model, but will increase computation time. It is therefore up to the practitioner to decide how many initialisations they want to check.

It is important to note that the empirical risk calculated on the validation set of our final model is a biased estimator of the true risk, since we have tested multiple models on this data set. To see why this leads to a bias, see for example the explanation by Brownlee [7]. To get an unbiased estimate of the risk, we would need a third independent data split, called the test set. This means that we would need to split our original data set in three different parts, the training set to train the models, the verification set to choose the best initialisation, and the test set to assess the performance of our final model. When we only have a small amount of data available to us, this method might not be ideal, since the three resulting datasets can be very small and might therefore not be a good representation of their population.

4.3 Finding the Optimal Number of Clusters

An important parameter of clusterwise prediction models, and of all methods that cluster the data in some way, is the number of clusters k. In the context of mixture models, it is easy to see that this parameter controls the complexity of the model. The more clusters we have, the more structure in the training set we can capture. This means that, were we to choose a high value for

k, we risk overfitting. On the other hand, choosing k too low can result in underfitting the data. It is clear that k is a regularisation parameter.

The question remains, how do we choose the best number of clusters? Since it is such an important issue, a great deal of research has already been conducted towards answering it [28]. When mixture models are used for the purpose of density estimation, the Bayesian-Schwarz Information Criterion (BIC) [35] is often used to choose the number of clusters, because under mild conditions, it does not underestimate the true number of clusters asymptotically [22]. Furthermore, Roeder and Wasserman have shown that a density estimate that uses the BIC to select the number of clusters is consistent [34].

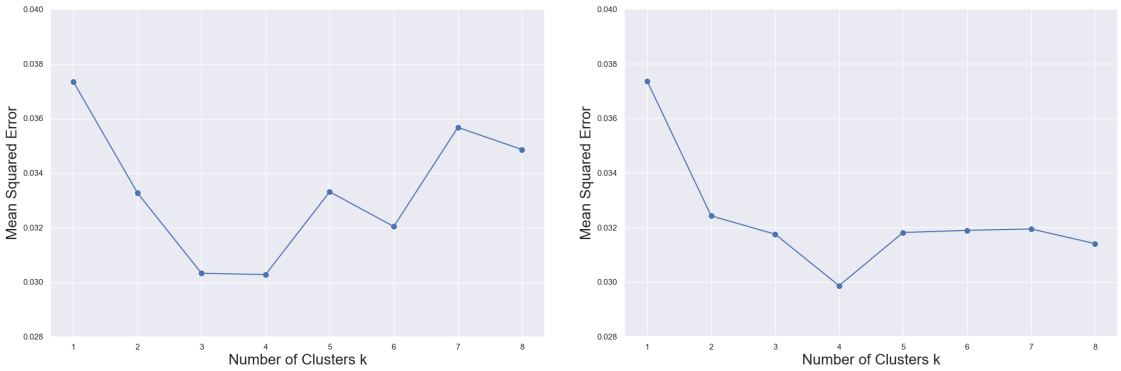
For the other popular use of mixture models, clustering the data, there does not seem to be a consensus. In cluster analysis, the way the data gets clusters is very dependent on the goal one wishes to accomplish with the clustering. Hence, it is hard to come up with one technique that will work for all goals. Some methods that are commonly used penalised likelihood methods such as the AIC and the BIC, likelihood ratio tests and the use of cross-validation. However, these methods are not ideal for the use in clusterwise predictive modelling. As is discussed in Section 4.2.2, the likelihood of the model is not representative for the performance of the model at making predictions, making any method that relies on it, such as the likelihood ratio test or information criteria such as the AIC, not ideal. In that same section it is also argued that cross-validation leads to problems, since the models found on each fold can be vastly different. For this reason, it is likely a good idea to have a separate training and validation set, and use the empirical risk to choose the optimal value of k

We could find the optimal value of k per iteration. One way of doing this would be, at iteration t with current estimate of the optimal number of clusters $k^{(t)}$, see if $k^{(t)} + 1$ or $k^{(t)} - 1$ clusters leads to a better model. This means that one needs to split one cluster into two, or merge two of them. Both approaches raise some difficulties. When splitting one cluster into two, we need to decide what part of the weight of each data point that the parent cluster has goes to each child cluster. There does not seem to be an easy way to do this, and dividing up this weight randomly between the new clusters seems unlikely to lead to a better model. On the other hand, when we merge two clusters we can simply give the new cluster the combined weight of the two original clusters for each data point. The problem, however, is that this requires us to check $\binom{k^{(t)}}{2}$ different combinations each iteration, which can greatly increase the required computation time. The choice is therefore made to run the whole EM algorithm for different values of k, and choose its optimal value based on the mean squared error computed on the validation set.

In the strategy for choosing the optimal initialisation values outlined in Section 4.2.1, we split the available data in a training, validation and test set. We can easily modify this strategy by testing models with different initialisations and different values of k. We then assess the performance of each model on the validation set and choose the model which the best performance. Finally the test set can be used to get an unbiased estimate of the statistical risk of the final model.

4.3.1 Experiments

We want to test this strategy, and to this end we will once again run a number of experiments on the Iris data set, using the same training/validation set. Clusterwise linear regression models are once again build using both methods of estimating $h(x)$. The experiments are repeated with different initial values, first using the three clustering methods described in Section 4.2.1 and after that, 50 times with the method using near equal cluster probabilities, also described in that section. These sets of experiments are performed for $2 \leq k \leq 8$. The results can be seen in Figure 4.5, which shows the best MSE found on the verification set using all resulting models, for each value of k separately. The found MSE using regular least squares linear regression is also included in these figures at $k = 1$, since this corresponds to having all data in the set be in the same cluster.



(a) *Where the cluster probability functions are estimated using logistic regression.*

(b) *Where the cluster probability functions are estimated using the kernel smoothing method.*

Figure 4.5: *The best mean squared error found on the verification set for different numbers of clusters k, for both methods of estimating the cluster probability functions. The experiments are repeated using different initialisation methods, and the best scores for each value of k are presented in these graphs. The scores found for $k = 1$ result from using ordinary least squares linear regression on the entire dataset.*

Since k is a regularisation parameter, we expect that, as k increases, the risk of the models initially decreases, and after reaching a minimum to start increasing. This pattern can indeed be seen in these figures. For the experiments where logistic regression is used, it is more clear than for the ones using the kernel smoothing method. The most likely reason for this is that the cluster probability functions that result form the kernel smoothing method can be more flexible, meaning that there can be very small clusters in the data for which the value of $h_j(x)$ is low for all $x \in \mathcal{X}$, meaning that they have only little effect on the risk of the model.

These experiments seem to confirm that the strategy to find the optimal number of clusters outlined above makes sense. A practitioner can, for example, start with $k = 2$, test multiple different initialisations and choose the model with the best empirical risk. Then, repeat this procedure for $k = 3$, $k = 4$, etc. When the empirical risk of the best model stops decreasing, or starts increasing, the optimal value of k will be known. Of course, the found value of k might correspond to a local minimum, so it remains to the practitioner's discretion to decide how many values they want to test before they can conclude when the optimal value has been found.

Chapter 5

General Algorithmic Approach

Our goal is to construct an algorithm that can build a predictive model on any given dataset. The goal of this chapter is to give the general framework of this algorithm. It will be a framework rather than a complete algorithm, because its users can decide what class of cluster probability functions $\mathcal{H}$ and predictive models $\mathcal{G}$ they want to use. The strategy derived in the previous chapter will enable us to choose the most important parameters of the method automatically, and will be part of this framework.

5.1 Using General Loss Functions

The body of the algorithmic framework will be similar to that of the EM algorithm that finds the parameters in a mixture distribution, described in Section 2.1.2. The derivation of that algorithm was statistical in nature, and relied on the class of predictive models $\mathcal{G}$ to be regression functions that are defined as the expectation of the target Y, given the features X. This means that the underlying conditional probability density functions of Y given X need to be known.

In the predictive modelling literature, however, there are a great number of methods that do not rely on some conditional probability distribution to build a predictive model, but rather do this by minimising the empirical risk function outright. If we want to use any of such methods in this algorithm, we cannot use Equations (2.12) and (2.16), meaning that we need to adjust the iterative method described in Section 2.1.2 to allow us to use a loss function to find new estimates $p_{ij}^{(t+1)}$ and $g_j^{(t+1)}$ based on the current estimates $p_{ij}^{(t)}$ and $g_j^{(t)}$. $h^{(t+1)}$ will still be found based on $h^{(t)}$ using Equation (2.17) or using the kernel smoothing method.

Let $L_{\text{model}} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be the loss function that determines the performance of the predictive model $g_j(x)$ in cluster j. We can now adjust Equations (2.16) by realising that the loss function that is implicitly being used in those equations is the negative log likelihood, in other words $L_{\text{model}}(g_j(x_i), y_i) = -\log v_\theta(y_i \,|\, x_i)$. We could easily use a different loss function within this equation, resulting in the optimisation problem

$$g_j^{(t+1)} = \arg\min_{g \in \mathcal{G}} \left(\sum_{i=1}^{n} p_{ij}^{(t)} L_{\text{model}}(g(x_i), y_i) \right),$$

which is equivalent to a weighted empirical risk minimisation. This also means that we can use a clusterwise prediction model for classification problems, we just need to use the appropriate loss function. We can use the same relation between the conditional PDF and the loss function to

change Equation (2.12) to

$$p_{ij}^{(t+1)} = \frac{h_j^{(t+1)}(x_i)e^{-L_{\text{model}}(g_j^{(t+1)}(x_i),y_i)}}{\sum_{l=1}^{k} h_l^{(t+1)}(x_i)e^{-L_{\text{model}}(g_l^{(t+1)}(x_i),y_i)}}.$$

After the algorithm has converged, we can construct a final prediction rule $\hat{f}(x)$. If the loss function used as a metric of the model performance is convex in its arguments, the prediction rule in Equation (1.7) can be used. If it is not, it might be better to make predictions based on the cluster with the highest probability, i.e. use

$$\hat{f}(x) = \hat{g_j}(x), \qquad \forall_l : h_j(x) \geq h_l(x) \tag{5.1}$$

We also saw an example of estimating the cluster probability functions that does not use Equation (2.17), namely the kernel smoothing method. It is unclear if the kernel method can be interpreted as a minimiser of empirical risk, and if so, what loss is implicitly being considered. Nonetheless, this shows that even for estimating h we are not restricted to using only the likelihood function.

We can write the total loss, which will call the clusterwise prediction loss L_{CP}, after iteration t of the algorithm as

$$L_{\text{CP}}(f^{(t)}(x_i), y_i) = \sum_{j=1}^{k} p_{ij}^{(t)} \log h_j^{(t)}(x_i) + \sum_{j=1}^{k} p_{ij}^{(t)} L_{\text{model}}(g_j^{(t)}(x_i), y_i)$$

meaning that the empirical clusterwise prediction risk at that iteration is given by

$$R_{\text{CP}}^{(t)}(f) = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{k} p_{ij}^{(t)}\mathbb{E}\left[\log h_j^{(t)}(x_i)\right] + \sum_{j=1}^{k} p_{ij}^{(t)}\mathbb{E}[L_{\text{model}}(g_j^{(t)}(x_i), y_i)]\right).$$

This is the empirical risk that is being optimised at each M-step by the algorithm. This will generally be different from the empirical risk that uses the loss function we wish to optimise $L(\hat{y}, y)$. For this reason, and using the same arguments as in Section 4.2.2, the value of this empirical risk cannot be used to assess the true performance of the predictive model.

5.2 The Algorithmic Framework

We now have all the pieces in place to summarise the entire algorithmic framework in detail.

1. Choose a class of predictive models $\mathcal{G}$ with a respective loss function L_{model}, and a class of classification models $\mathcal{H}$. Furthermore, choose a loss function that is to be used to assess the final predictive model L.
2. Split the dataset in a training set $D_\tau = \{(x_i, y_i)\}_{i=1}^{n_\tau}$, and a validation set $D_v = \{(x_i, y_i)\}_{i=1}^{n_v}$
3. Choose an initial k, such as $k = 2$.
4. Choose an initialisation. Either pre-cluster the data using an existing clustering method or choose $p_{ij}^{(0)} = \frac{1}{k} + \varepsilon_{ij}$ with $\varepsilon_{ij} \sim \text{Unif}(-0.01, 0.01)$ for $1 \neq j < k$, and $p_{ik}^{(0)} = 1 - \sum_{j=1}^{k-1} p_{ij}$.

5. Compute for all $j \in \{1, 2, \ldots, k\}$

$$g_j^{(t+1)} = \arg\min_{g \in \mathcal{G}} \left(\sum_{i=1}^{n_\tau} p_{ij}^{(t)} L_{\text{model}}(g(x_i), y_i) \right).$$

6. Compute

$$h^{(t+1)} = \arg\max_{h \in \mathcal{H}} \left(\sum_{i=1}^{n_\tau} \sum_{j=1}^{k} p_{ij}^{(t)} \log h_j(x_i) \right).$$

7. Compute for all $i \in \{1, 2, \ldots, n_\tau\}$ and all $j \in \{1, 2, \ldots, k\}$

$$p_{ij}^{(t+1)} = \frac{h_j^{(t+1)}(x_i) e^{-L_{\text{model}}(g_j^{(t+1)}(x_i), y_i)}}{\sum_{l=1}^{k} h_l^{(t+1)}(x_i) e^{-L_{\text{model}}(g_l^{(t+1)}(x_i), y_i)}}.$$

8. We can use the empirical clusterwise prediction risk on the training set to determine whether the algorithm has converged or not. To this end compute

$$R_{CP,t}^{(t+1)}(f_{kl}^{(t+1)}) = \frac{1}{n_\tau} \sum_{i=1}^{n_\tau} \left(\sum_{j=1}^{k} p_{ij}^{(t+1)} \log h_j^{(t+1)}(x_i) + \sum_{j=1}^{k} p_{ij}^{(t+1)} L_{\text{model}}(g_j^{(t+1)}(x_i), y_i) \right).$$

9. Go back to step (5), and repeat until the empirical risk on the training set no longer decreases, or until the maximum number of iterations has been reached. After this, we can construct a predictive model using Equation (1.7) or (5.1). Denote this final model by $\hat{f}_{k1}$, $\hat{f}_{k2}$, etc.

10. Go back to step (4), and repeat for the desired number of initialisations to be checked.

11. For all the constructed models, compute the empirical risk on the validation set using

$$\hat{R}_v(\hat{f}_{kl}) = \frac{1}{n_v} \sum_{i=1}^{n_v} L(\hat{f}_{kl}(x_i), y_i),$$

and select the model the model with the smallest empirical risk

$$\hat{f}_k = \arg\min_{\hat{f}_{kl}} \hat{R}_v(\hat{f}_{kl}).$$

12. Choose a different value for k, such as $k_{new} = k_{old} + 1$. Repeat the entire procedure for this new k, beginning from step (4).

13. We can stop and say we found the best model when the empirical risk stops dropping for higher values of k, i.e. when

$$\hat{R}_v(\hat{f}_{k_{old}}) > \hat{R}_v(\hat{f}_{k_{new}}).$$

We can now choose the best model using

$$\hat{f} = \arg\min_{\hat{f}_k} \hat{R}_v(\hat{f}_k).$$

This algorithm does not specify how to perform steps (5) and (6), hence it should be viewed as a framework for an algorithm rather than a complete algorithm.

Of course, there are are many possible adjustments that could be made. For example, the desired value of k might be known beforehand, meaning that we do not need to check multiple values. It might be naive to stop the algorithm when $\hat{R}_v(\hat{f}_k)$ starts increasing, as it might again start

to decrease for even bigger values of k. It is therefore likely a good a idea to check more values of this parameter to see the overall trend. In low dimensional problems, or when k is small, it can be possible that different initialisations will always lead to the same model. This was, for example, observed in the data visible in Figures 3.1 and 3.3. In those situations, it is obviously not necessary to build multiple models with different initialisations.

We can see that this model is not particularly efficient. It requires many different models to be constructed on the dataset, and most of these will be "thrown away". It is therefore advised to have the classes of models $\mathcal{G}$ and $\mathcal{H}$ to be relatively simple, and preferably have analytical solutions, as this will greatly reduce the computational load. As we saw for the kernel smoothing method, if the models of the individual clusters, or the cluster probability functions themselves, contain hyper-parameters to be tuned, cross validation can be used to adaptively find their optimal value. This cross-validation, however, can also increase computational times a lot, meaning that methods that contain such hyper parameters might not be ideal.

A huge factor in determining the computational cost necessary to perform the algorithm in its current form, is the amount of initialisations that are checked. Obviously, the more we check the greater the probability is that a good initialisation is found, but also the greater the computational times will be. This naturally leads to the question, how many should we check? There does not seem to be concise answer to this question. From experimenting with this algorithm, it seems the the variance in performance of the models acquired from different initialisations is greater when data is sparse, the feature space is high dimensional and when k is big. Perhaps an adaptive method that chooses the amount of initialisations to be checked bases on the distribution of performance of the models can be used. This, however, has not been tested in this final project.

5.2.1 Ensemble

The great amount of randomness in models produced by different initialisations make clusterwise predictive models great candidates to be used as the individual learners within a learning ensemble [33]. The principle idea behind building an ensemble is that we can combine many different predictive models to get better better estimates than is possible using any of the individual models. Perhaps the simplest form of a learning ensemble in a committee, in which we take the unweighted average of the members of the committee $(\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_m)$. In other words, the ensemble model estimate of feature x is

$$\hat{f}_{\text{ensemble}}(x) = \frac{1}{m} \sum_{i=1}^{m} \hat{f}_m(x).$$

Since in the algorithmic framework above we are already required to build multiple different models, creating an ensemble does not increase the computational load much, unless of course we want to include more models in our ensemble.

How do we decide which models to include in our committee? We know that some of the clusterwise predictive models that are learned on the training set will perform very poorly on an independent validation set. We can use as a simple heuristic to include only models that perform better at prediction that regular linear regression does, that is have a lower empirical risk calculated on the validation set than the linear regression model that is trained on the training set. The idea behind this heuristic is that we expect any "good" clusterwise predictive model to perform at least better at making predictions than regular linear regression would.

5.3 Regression Trees

Now that we have the general algorithmic framework complete, we can see how it works using a method to model the individual clusters other than linear regression. The choice is made to test regression trees. In this method, the feature space is split in M rectangular regions $R_1, R_2, \ldots, R_M$, and the prediction rule for a data point is given by

$$\hat{g}(x) = \sum_{m=1}^{M} c_m \mathbb{1}\{x \in R_m\},$$

e.g. all data points that fall in the same region get the same prediction. The value of c_m that will minimise the squared error loss is equal to the average value of the targets of the data points in the training set that fall in region m, in other words

$$c_m = \text{ave}(y_i \,|\, x_i \in R_m).$$

An example of this can be seen in Figure 5.1. For a more detailed description of this method, and a way of constructing a regression tree on a data set see, for example, *The Elements of Statistical Learning* [18, Section 9.2.2].

Regression trees are a popular method for predictive modelling. They are intuitive and easy to understand, even for non-experts. Furthermore, they can handle both numerical and categorical data, are scalable to large datasets, can handle outliers and missing data, and relevant features are automatically selected. A drawback, however, is that their performance is often not great. This is in part due to the non-smooth nature of the regression functions they produce, which can be seen in Figure 5.1. When regression trees are used in a clusterwise predictive model, smooth transitions between different trees can be made, which should make the resulting regression functions smoother and, hopefully, achieve a better performance than is possible with a single tree.

5.3.1 Regularisation

Regression trees are regularised by their size, i.e. by the amount of regions M the feature space is divided in. Obviously, choosing a good value for this parameter is very important for making predictions. Just like the bandwidth parameter λ in the kernel smoothing method, we can choose the optimal value of M adaptively, at each iteration of the algorithm. This will again be done by comparing the cross-validation score as a metric for the model performance for different values of M, and selecting the value that minimises this score.

Since the kernel smoothing method is a linear fitting method, a computationally efficient method for doing leave-one-out cross-validation exists. A regression tree is not a linear fitting method, and hence the method described in Section 3.2.3, unfortunately. For this reason it is probably better to use C-fold cross-validation on bigger datasets. On iteration $t+1$ of the algorithm, we can do a local search around the optimal value found at the previous iteration. A possible set of values to check is thus

$$\{M^{(t)} - 1, M^{(t)}, M^{(t)} + 1\}.$$

When regression trees are used in a clusterwise predictive model in combination with the kernel smoothing method to estimate the cluster probability functions, the optimal value of both λ and M will need to be found. This means that, at each iteration, 9 combinations of parameter values will need to be checked, which will have quite an effect on the computation cost.

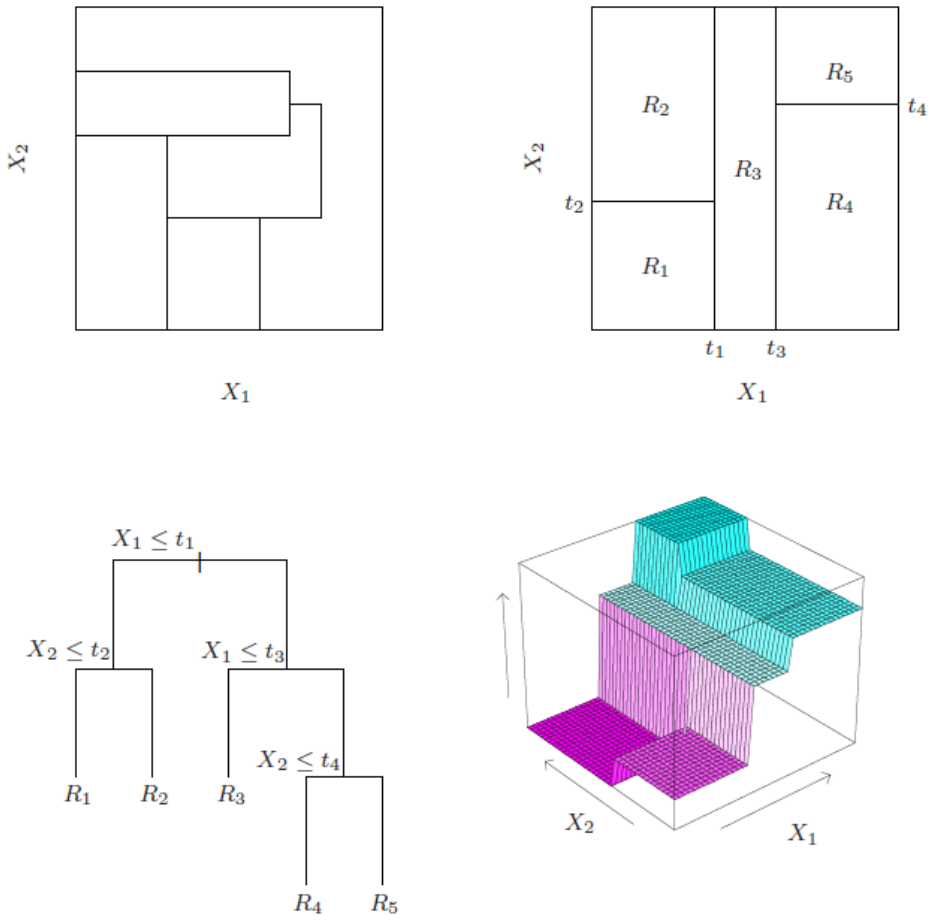


Figure 5.1: *An example of a regression tree. The upper left image shows a partitioning of the feature space that is not possible to get with a regression trees, since the regions are not all rectangular in shape. The upper right image shows a partitioning that is possible to get. The lower left image shows how the partitioning of the feature space can be represented by a tree structure. The lower right image shows the final prediction model for this regression tree. Reprinted from* The Elements of Statistical Learning *by Hastie et. al. [18].*

5.3.2 Example

We will now test the clusterwise predictive model that uses regression trees to model the individual clusters on the synthetic dataset visible in Figure 3.3. Both logistic regression and the kernel smoothing method will be used to estimate the cluster probability functions, using $k = 3$. The regression tree implementation from the scikit-learn library [32] will be used, which uses the CART algorithm [6] to construct the trees. This implementation allows the user to give sample weights to data points, meaning that weighted empirical risk minimisation is possible. For convenience sake, the size of the tree will be determined by a single parameter, namely the amount of regions M the feature space is split in, controlled by the max_leaf_nodes parameter in the scikit-learn implementation. The adaptive method to find its optimal value described above will be used.

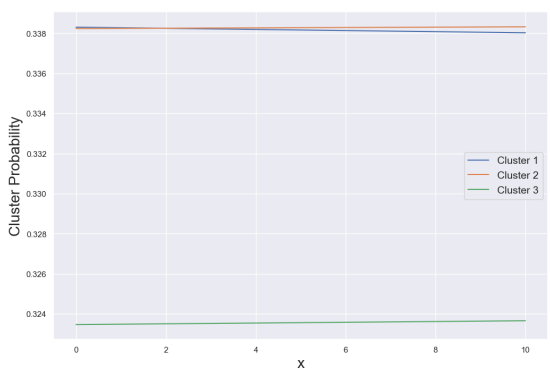
The results from using logistic regression are visible in Figure 5.2. We see in Figure 5.2b that the three regression trees seem to divide themselves over the point cloud, but each tree does cover it for the entire domain. The result of this is that no distinctive clusters can be found that can be modelled using logistic regression functions. We see in Figure 5.2a that the found cluster probability functions are almost horizontal lines, with almost the same value for all three functions. This means that the final predictive model, visible in Figure 5.2c is essentially the unweighted average of the three trees, which results in a regression function very similar to a regression tree with $M = 10$, instead of the smooth functions we hoped for. We have to conclude that building a clusterwise predictive model using regression trees in combination with logistic regression does not produce something that is very distinct from what can be achieved by building a single tree to the data.

So what about using this method in combination with the kernel smoothing method to estimate the cluster probability functions? The result of that experiment can be seen in Figure 5.3. As we can see, the individual regression trees have a very similar structure as before. The found probability cluster functions, however, are now completely different. These functions are all over the place, and we can no longer distinguish distinct clusters in the data. Nevertheless, these functions do cause smoother transitions between the different regions in the feature space, meaning that the final regression function is smoother than it would be when a single regression tree is used. As can be seen in Figure 5.3c, the final model is relatively smooth, and approximates the Bayes optimum function quite closely.

5.3.3 Discussion

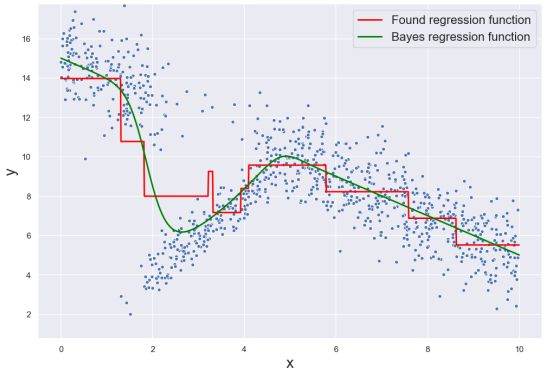
We have seen that when a clusterwise predictive model is build using linear functions as the building block model, each individual function typically models part of the data well, and the rest not. This means that the feature space gets divided in distinctive regions, each in which a different function is used to model the data. The cluster probability functions determines the location of the regions, and also ensures smooth transitions between them. When regression trees are used as the basis function, however, the interpretation of the final model becomes very different. The k trees seem to divide themselves over the range of the target value for the entire domain of the features. This means that we can no longer talk about distinctive clusters that are found in the data. In this scenario, the cluster probability functions determine, for each $x \in \mathcal{X}$, in what ratio the average of the trees should be taken as to minimise the chosen loss function. This requires very flexible functions, meaning that logistic regression cannot be used. The kernel smoothing method, on the other hand, seems appropriate for this task.

The result of this is that clusterwise predictive models that using regression trees are not as interpretable as ones that use linear functions, or even as single regression trees. These models should therefore be considered to be black-box s. Nevertheless, their performance at predicting is what we care about most, and how well they do at that will be looked into in the next chapter.



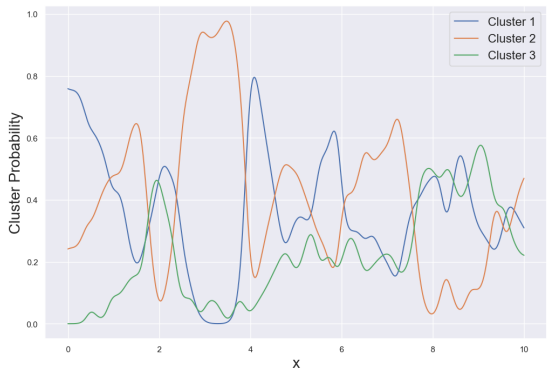
(a) The found cluster probability functions.

(b) The found regression trees.



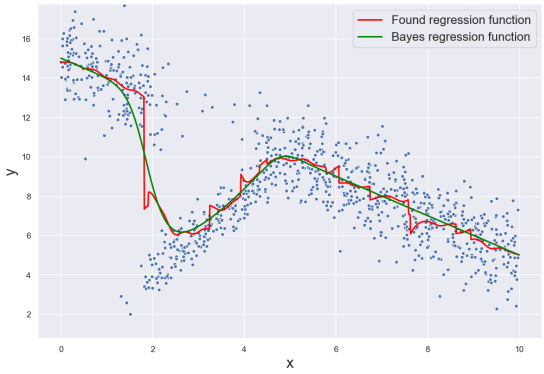
(c) The resulting regression function together with the Bayes optimal regression model.

Figure 5.2: *The results from making a clusterwise predictive model using regression trees to model the individual clusters. Logistic regression is used to estimate the cluster probability functions.*



(a) The found cluster probability functions.

(b) The found regression trees.



(c) The resulting regression function together with the Bayes optimal regression model.

Figure 5.3: *The results from making a clusterwise predictive model using regression trees to model the individual clusters. The kernel smoothing method is used to estimate the cluster probability functions.*

It seems that in the literature of methods that build a predictive model by clustering the data and than model each clusters, the use of regression trees has not before been suggested to fit to the individual clusters. However, a number of popular predictive modelling methods exist that combine multiple decision trees into a single model. For example, the random forest, which is the average of a large number of trees that are each fit to a bootstrap sample of the training set, often using a subset of the available features [5].

Chapter 6

Assessment of Performance

We have developed a general algorithmic framework that enables us to build a clusterwise predictive model for a given dataset. Throughout this thesis, we have seen a number of experiments performed on simple datasets, were this general method is shown to work well. Of course, we want to know how well this method works for more complicated, real world datasets. In particular, we want to see how its performance compares to that of other methods that in some way cluster the data and build models for each cluster, like the ones discussed in Section 1.2, as well as some other popular predictive models. The decision was made to test this method on the three open datasets that are also used in the paper by Gitman et. al., who tested various versions of the Predictive CLR and Constrained CLR methods they developed, as well as the k-planes regression method [24]. Those three datasets are the Boston Housing [17], the Abalone [30] and the Auto-mpg [37] datasets.

6.1 Datasets

The three used datasets are all publicly available through, e.g., the UCI Machine Learning repository [13]. The Boston Housing set contains data on 506 neighbourhoods in various areas around the city of Boston. It contains 13 features, and the target value is the median house price of the neighbourhood. The Abalone dataset contains data from a study conducted towards abalones, a type of mollusc with a peculiar ear-shaped shell lined of mother of pearl. It contains 4177 data points with 6 numerical features and one categorical one, namely the sex which can be "male", "female" or "infant". This last feature will be encoded with two dummy variables, one for "male" and one for "female". The target value is the amount of rings in the shell, which is an indicator of the age of the abalone. The last data is the Auto-mpg dataset which contains data on 398 types of cars. The target value is the fuel consumption of the car, measured in miles per gallon. There are 8 features, one which is the name of the car which we ignore, since it is unique for each instance. One of the remaining 7 features is the origin of the data. Apparently, data comes from three different sources. Again, two dummy variable will be used to encode this categorical feature. This dataset contains 6 missing values in the "horsepower" column. Their values are imputed with the median value of this column, which is 93.5. This is perhaps a naive way of dealing with this missingness, but since there is so little missing data, this simple solution is likely adequate.

As is discussed in Section 3.2, we will scale all features by subtracting the mean and dividing by the standard deviation. Each dataset will be randomly split in a training set, validation set and test set with sizes equal to 50%, 25% and 25% of the data respectively.

6.2 Experiments

We will now test on these datasets the performance of the three clusterwise predictive modelling methods that have been discussed throughout this thesis, namely the ones that use linear regression to model the individual clusters, both using logistic regression and the kernel smoothing method to estimate the cluster probability functions, and the version using regression trees to model the individual clusters and the kernel smoothing method for the cluster probability functions. Furthermore, we will test ensemble versions of all of these methods.

We will compare the performance of the clusterwise predictive models to that of a number of popular regression methods. Firstly, we want to test how their performance compares to their building block methods, that is ordinary least squares linear regression and regression trees. Furthermore, we will test random forests, k-nearest neighbours [18, Section 2.3.2] and support vector regression (SVR) [12]. For all of these methods, except for linear regression, the implementation from the scikit-learn library will be used. The optimal hyper-parameters of these methods will be found using a grid search, where for each parameter combination in the grid the model will be build on the training set and their performance is assessed on the validation set. Finally, for the model with the best performance, the empirical risk is calculated on the test set.

For the regression tree and random forest method, the parameters that will be optimised are the maximum number of regions to divide the feature space in, the minimum number of samples that each region needs to contain and, for the random forest method, the number of features considered when constructing the individual trees. Furthermore, in the random forest method the number of trees is set to 100. In the k-nearest neighbours method, the parameters used in the grid search are the amount of neighbours k, and p, the coefficient in the Minkowski distance used to find the k nearest neighbours. Furthermore, data points are weighted by the inverse of their distance. The parameters that are found for the support vector regression method are ε, the threshold parameter, where the loss between the true target y and prediction $\hat{y}$ scales linearly if $|\hat{y} - y| > \varepsilon$, and is 0 otherwise, and C, the regularisation parameter that gives a squared L2-norm penalty to $|\hat{y} - y| - \varepsilon$, if this difference is greater than 0. All other parameters in these methods are kept at their defaults. The optimal parameter values that were found this way are given in Appendix B.

An important part of the algorithmic framework is to test multiple different initialisations. Each non-ensemble clusterwise predictive modelling method will be initialised using the three pre-clustering methods discussed in Section 4.2.1, alongside a number of random initialisations. Because the computational times needed to construct the models is much greater when the kernel smoothing method and/or regression trees are used compared to when logistic regression or regression trees are used, the decision is made to test different amounts of random initialisations for the methods. For the methods using linear models and logistic regression to estimate the cluster probability methods, 50 random initialisations are checked. For the other two methods, 10 are checked except for the tree based method tested on the Abalone data set, for which only 5 random initialisations are checked. These experiments are repeated for different values of k, until the value of this parameter that yields the best performance on the validation set is found. For the ensemble based methods, a larger number of models are fitted using different initialisations for the found best value of k. Again, the three pre-clustering methods and a number of random initialisations are used. For the clusterwise linear regression methods, 200 random initialisations are used, except for the version using the kernel smoothing method, where only 100 are made for the Abalone dataset. For the tree based method, 100 random initialisations are used for the Boston housing and Auto-mpg sets, and 50 for the Abalone dataset.

Gitman et. al. [16] used these same datasets to test the performance of their Predictive CLR method, both using logistic regression and random forest to cluster data, and their constrained CLR method as well as the k-plane regression method. They also tested ensemble versions of all these methods. Furthermore, they also tried random forests, support vector regression and ordinary least squares linear regression. They used a grid search to find optimal parameter values,

Table 6.1: *The results of the experiments conducted to assess the performance of the different clusterwise predictive models. For three different datasets the found mean squared error computed on the test set for the best model found using each method are presented. The "method" column denotes which version of the clusterwise predictive modelling methodology is used, with "Linear" and "Tree" denoting whether linear regression of regression trees are used to model the individual clusters, and "Logistic" and "Kernel"denote whether the cluster probability functions are estimated using logistic regression or the kernel smoothing method. Furthermore, the results acquired form using a number of popular regression models on these datasets is also included. The best results for each dataset, both from clusterwise predictive models and from the other methods, have been highlighted.*

Method	Boston housing	Abalone	Auto-mpg
Linear/Logistic	17.79	4.448	9.547
Linear/Logistic Ensemble	**13.07**	**4.320**	9.171
Linear/Kernel	17.29	5.006	9.073
Linear/Kernel Ensemble	15.85	4.963	**8.966**
Tree/Kernel	15.12	7.917	14.42
Tree/Kernel Ensemble	14.14	8.597	12.71
Linear regression	22.52	5.044	11.07
Regression Tree	18.72	6.057	12.32
Random Forest	**11.00**	4.947	11.56
k-nearest neighbours	22.86	5.790	11.73
SVR	13.27	**4.607**	**10.96**

Table 6.2: *The optimal number of clusters k found on all datasets, for all the non-ensemble clusterwise predictive models.*

Method	Boston housing	Abalone	Auto-mpg
Linear/Logistic	5	10	3
Linear/Kernel	3	4	6
Tree/Kernel	4	4	6

using 5 repetitions of 10-fold cross validation. Since they did not test the final models on an independent test set, however, their reported values have a bias. They also do not mention how they initialise their method, other than saying that it is "random". Nevertheless, we can use the values they report as an indicator of how our methods compare to theirs.

The found mean squared errors of the best model found for each method, computed on the test set, for all three datasets, are presented in Table 6.1. The optimal amount of clusters that were found for each method, for each dataset, are given in Table 6.2. Figures showing the lowest mean squared error found on the validation sets for different value of k have been included in Appendix B. This appendix also contains the amount of models that are included in each of the tested ensembles.

6.3 Discussion of Performance

There are a number of things that we can learn from these results. First of all, building an ensemble of different clusterwise predictive seems to increase the performance of the method most of the time. This difference does seem to vary between datasets. For the Boston housing set, the difference is the most substantial. The effect seems to be least significant on the Abalone dataset, for which the performance of the tree based method actually decreased when using an ensemble.

Another thing to notice is that there is no clear winner between the three different clusterwise predictive modelling methods we tested. In fact, on each dataset a different method led to the best performance. This seems to indicate that the dataset is quite important in regard to how well each method performs. We can also see that the performance of the basic predictive model can be an indicator of the performance of the mixture model that uses it. On the Boston housing set, where a single regression tree led to a lower mean squared error than linear regression, the tree-based clusterwise predictive models performed better than the linear model-based methods. These last models, however, performed better on the other two data sets, on which linear regression led to a better performance compared to a single regression tree.

We can also see from these results that the clusterwise predictive models have a good performance overall, when we compare it to the popular regression techniques. On the abalone and Auto-mpg sets we can see that some versions of clusterwise predictive models achieved a better performance than all of these conventional methods, and on the Boston housing set only Random Forest and SVR performed better. This seems to indicate that clusterwise predictive models are quite powerful and useful.

If we compare the results from the experiments conducted for this thesis with the ones from the paper by Gitman et. al., we see that the results are quite comparable. This is not very surprising since our methodologies share a lot of similarities. Only on the Boston housing do the experiments conducted by Gitman et. al. result in significantly better performance. We do have to keep in mind though that our methods of testing the algorithm differ somewhat, meaning that the results cannot be compared one-to-one. Overall, we cannot conclude that either of our methodologies of constructing a predictive model is significantly better than that of the other, but the algorithmic framework we developed is more flexible.

6.4 Interpreting the Final Models.

A big advantage of clusterwise predictive models that use linear regression as basis function to fit to the clusters is that they are interpretable. The data points get clustered in groups for which the same predictive model fits the data well. We expect that the members of each cluster are in some way similar to one another. In the introduction of this thesis way gave as motivation behind researching these methods that there might be some sub-populations in the data. This means that, when we fit a clusterwise predictive model, looking at the clusters that are found and their corresponding linear model, we might be able to recognise these sub-populations and perhaps gain some insight about how they differ from one another, and what the important features of each of these are. Of course, as is argued in Section 4.1, the different clusters that are found do not necessarily all correspond to a sub-population.

In this section we will take a closer look at the final models acquired using clusterwise linear regression to see if we can gain any of such insights for the datasets that we have studied. The found parameter values as well as the average feature and target values and the size of each cluster are given in tables in Appendix B. Here, data points are assigned to a cluster when they have the greater inferred probability of belonging to that cluster than to the other clusters. As we saw in Section 5.3, the interpretation of the method that uses regression trees to model the individual

clusters is quite different, and it seems that this method is not very interpretable. The models found using this method will therefore not be looked at here. Note that, since the main interest in this final project is making predictions, these analyses of the acquired models are not very extensive.

6.4.1 Boston housing

For the Boston housing dataset, using logistic regression to estimate the logistic regression functions, five clusters were found. Cluster 1 contains, on average, the neighbourhoods with the highest median value of homes. It contains neighbourhoods with low crime rate, little industry, and houses with large amounts of rooms. We can see that the value of the "chas" parameter, which corresponds to a dummy variable indicating whether or not the neighbourhood is next to the Charles River or not, is very high for this cluster. However, non of the neighbourhoods that have the greatest inferred probability of belonging to this cluster are located next to this river, meaning that this parameter is not very significant. We can also see that the parameters concerning crime rate and average number of rooms per dwelling have a larger magnitude for this cluster than for the others. Clusters 2, 3 and 4 correspond, on average, to neighbourhoods that contain lower valued houses. We can see that the found absolute parameter values are generally lower than for Clusters 1 and 5, which contain more expensive neighbourhoods. This seems to indicate that there is a greater variance in median value of a house between richer neighbourhoods than between poorer ones. The price of neighbourhoods in Cluster 5 seems to be influenced the most by the accessibility to radial highways, the proportion of residential land zoned for lots over 25,000 sq.ft. and the full-value property-tax rate, whereas for the other clusters these features are not as impactful. It seems that this cluster contains the oldest neighbourhoods of the area. Furthermore, the parameter value of the feature that gives the proportion of black people per town is much smaller for Cluster 1 and 5, compared to the the poorer clusters. Coincidentally, the members of Clusters 1 and 5 also have the average greatest proportion of black people of all the clusters

When the kernel smoothing method is used to estimate the cluster probability functions, we get quite different results. Firstly, only three clusters are found. Based on the feature values though, all three clusters seem somewhat similar. The average house price is the greatest for the neighbourhoods in Cluster 3, and its corresponding model also has the greatest parameter values, which seems to affirm the notion that the median house value between richer neighbourhoods varies more than between poorer neighbourhoods. In that Cluster, the most the features that have the greatest parameter values are the full-value property-tax rate, the accessibility to radial highways and the distances to five Boston employment centres. For Cluster 2, this is the average number of rooms per dwelling, which has a very low value for the other two clusters. Finally, for Cluster 1, which contains on average the poorest neighbourhoods, the most impactful feature seems to also be the average number of rooms per dwelling.

6.4.2 Abalone

Ten clusters were found in the Abalone dataset when logistic regression was used to estimate the cluster probability functions. It seems that four main groups exist among the clusters. Clusters 1, 6 and 10 contain, on average, the oldest abalones. It seems that the biggest molluscs are not necessarily the oldest. In fact, the heaviest and largest abalones are found more commonly in Clusters 2, 4, and 7, which have a lower average amount of rings than those in the first group. Clusters 5, 8, and 9 are the three clusters with the lowest average age. As expected, these clusters contain mainly infant molluscs, which are also generally smaller and lighter. Cluster 3 seems to be a separate group, containing abalones that are on the smaller and lighter side, but also relatively old.

The first group, containing older molluscs of moderate size and weight, seems to have have the biggest parameter values. For these, the weight related features seem to have the greatest importance. For the members of Clusters 2, 4, 7, which are generally bigger abalones of moderate age, the size of the shell seems to be as impactful as the weight of the abalone in determining the age. For the group containing the youngest molluscs, again the weight related parameter value are the largest. Finally, for Cluster 3, the feature with the largest parameter value is the length of the shell.

When the kernel smoothing method was used to estimate the cluster probability functions, only four clusters were found. Based on the average amount of rings of the members in each clusters, there is some difference between the clusters, with Cluster 1 containing the oldest abalones, and Clusters 3 and 4 the youngest. Based on their feature values though, all four clusters seem similar. Looking at the found coefficient values of the found linear models, we again see that they are the highest for the cluster that contains on average the oldest molluscs. For Clusters 1, 2 and 3, the parameters of the weight related features have the largest magnitudes, and for Cluster 4 the features with the largest parameter values are the size related ones.

6.4.3 Auto-mpg

Finally, let us take a look at the clusters that were found in the auto-mpg dataset. When logistic regression is used to estimate the cluster probability function, three clusters are found. Cluster 1 has the highest average mileage per gallon. It contains cars that have a low amounts of cylinders, a small engine and low weight. For the cars this cluster the amount of horsepower and the year it is build seem to have a larger impact on their fuel consumption than for the other clusters. Exactly half of the cars in this cluster come from Source 3, and only a few from Source 1. The cars in Cluster 2 have the worst fuel consumption on average, not surprising since it contains cars with big engines with many cylinders, and high weight and amount of horsepower. Almost all cars in this cluster come from Source 1. Relative to the other clusters, the amount of cylinders and the origin of the data seem to have a large impact on the mileage per gallon of the cars in this cluster. Contrary to the other two clusters, the weight of the car does not seem to have a large impact on fuel consumption for members of this cluster. Finally, the cars in cluster 3 seems to be somewhere in between those of Clusters 1 and 3 in terms of fuel consumption as well as for most feature values. The data from more than half of the cars in this cluster come from Source 1, and similar amounts from the other two origins. The feature with the largest parameter value for this cluster seems to be the weight of the car, same as it is for Cluster 1.

When the cluster probability functions are estimated using the kernel smoothing method, six clusters are found. Based on the average feature values, Clusters 1 and 3 seem to be similar, containing heavy cars with big engines that perform badly in terms of fuel consumption. Clusters 4 and 6 contain the cars with a high mileage per gallon, and both consist of light, less powerful cars. Lastly, Clusters 2 and 5 seem to contain cars that are roughly in the middle of this spectrum. Based on this, it seems that the same three main sub-populations found in the last experiment are also found here, except they are all split in two clusters. Looking at the found parameter values of the linear models, it seems that the two clusters that belong to the same sub-populations can have quite different models. It seems that logistic regression was not able to properly split these sub-populations in two, while the kernel smoothing method was. This might be the reason why this last method performed better.

Chapter 7

Conclusions

We started out this project with the goal of constructing a mathematical model that can make predictions on a statistical population which contains some sort of heterogeneity. That is, multiple sub-populations exist within the overall population, and for each of these groups a different predictive model performs well. The goal of the method we developed was to cluster the data, and for each cluster fit a predictive model. These separate models can then be combined into what is known as a mixture model.

In finding a method of estimating the parameters in such a mixture model, we sought inspiration in the statistical literature, where mixture model have long been an important tool for density estimation and model based clustering. We managed to write this task as a likelihood maximisation problem, which we solved using the EM algorithm. This gave us a general method that allows us to construct a predictive model given what we called the cluster probability functions, and a basis predictive model that can perform predictions on each separate cluster. A natural choice for such a model was linear regression, which led to the notion of clusterwise linear regression.

As concrete examples of methods that could be used to construct the cluster probability functions, we looked at logistic regression and the kernel smoothing method. This first method is parametric and tries to separate clusters using hyperplanes, which limits the shapes of clusters it can found. The kernel smoothing method is non-parametric and can find clusters of much more flexible shapes. We saw that this last method is regularised by the bandwidth parameter λ, and we developed a methodology for finding the optimal value of this parameter using leave-one-out cross-validation at each iteration of the EM algorithm.

The argument was made that the developed methodology can be used as a general method of constructing a non-linear model. This means that there does not necessarily need to be heterogeneity in the data to use the method. In order to generalise the idea of clusterwise predictive into an algorithmic approach, there are a number of questions that need to be answered. We looked at how to initialise the EM algorithm, and saw that different initialisations could lead to vastly different predictive models, with a huge range of performances. We concluded from this that it would be necessary to try multiple different initial values and verify the performance of the resulting model on an independent validation set, enabling us to choose the best model. In terms of choosing the optimal number of clusters k in the dataset, the best strategy seems to be to test multiple different values of this parameter and to plot the validation scores of the best model for each value. When a clear minimum is found in this plot, we can assume that it corresponds to the optimal number of clusters.

The strategy of fitting a clusterwise predictive model relied on likelihood maximisation, limiting the type of models that can be fit to the individual clusters. Fortunately, we could easily change this procedure to allow the use of general loss functions, opening the door to the use of any predictive

modelling technique in the method. As an example of this, we looked at a mixture model combining multiple regression trees using the kernel smoothing method to estimate the cluster probability functions, and saw that this results in regression functions that are a lot smoother than those acquired from a single tree, which could lead to significantly improved performance. Furthermore, the idea was coined to build an ensemble of multiple clusterwise predictive models to further increase their predictive ability.

Finally, we looked at how the performance of the developed methods compares to that of a number of popular regression technique on some open datasets. The results of these experiments were promising, and the developed method of clusterwise predictive modelling seems like a very powerful technique. In these experiments, we also saw that building an ensemble of multiple methods does indeed lead to a better performance than is possible to get using only one model.

7.1 Strengths and Weaknesses

The greatest strength of the developed algorithmic framework is that it very versatile. Practitioners can decide for themselves which techniques to use to model the individual clusters, and to construct the cluster probability function. This means that the method can easily be customised to fit many different situations, problems and goals. This also means that the user has great control over the complexity of the model. When simpler building block techniques, such as linear regression and logistic regression, are used, the resulting model becomes very interpretable while still being very flexible. When more complex modelling techniques, such as the kernel smoothing method and regression trees, are used, the method becomes less interpretable, but will be able to model more complex patterns in the data.

Looking at the potential weaknesses of clusterwise predictive models, the greatest one seems to be that there is no good way of initialising the method that guarantees that the resulting model performs well at making predictions. The solutions that we came up with, to test many different initialisations, works but it does require constructing many different models, which has a significant effect on the computational load. Furthermore, since we always need to keep part of our dataset separate to validate the models, we have less data available to train the model. We also cannot retrain the model on the entire dataset when a good initialisation has been found, because we don't know which initial values to use for the new points and even if we did know, we have no guarantee that this will lead to the same final model.

Another big weakness of this method currently is the required computational load. Even without considering that the algorithmic frameworks requires training many different models on the same dataset, its iterative nature means that it will take much more time to fit the predictive model compared to many other techniques. When the used basis functions also have hyper parameters that need to be tuned, the computational load increases even further, since this means that cross-validation has to be applied many times. Even when computationally efficient methods exist for this, such as for clusterwise linear regression using the kernel smoothing method to estimate the cluster probability functions, this effect can be quite significant. For reference, the experiments in Chapter 6 conduction on the Abalone data that use the kernel smoothing method all took over 4 hours to complete on an off-the-shelf laptop. One should keep in mind that that the used implementation was not very optimised.

7.2 Suggestions for Future Research

Since the developed method is more of a algorithmic framework rather than a strict algorithm, there are many possible variations of it possible. In this thesis we have only looked at regression

problems. The method could also easily be used for classification problems, it only requires using the appropriate class of models $\mathcal{G}$ to fit to the individual clusters. For example, we could use a method that attempts to solve a binary classification problem using a separating hyperplane, such as logistic regression and support vector machines. In this context, the cluster probability functions will split the feature space into regions in which the two classes can be separated by such hyperplanes. This makes clusterwise predictive modelling an alternative method of estimating non-linear decision boundaries with these methods, other than, for example, using higher order terms for logistic regression or the kernel trick for support vector machines.

In a similar way that we generalised the fitting of the predictive models using loss functions in Section 5.1, we can generalise step 6 in the algorithmic framework in Section 5.2 by recognising that the loss function that is minimised there is the cross-entropy loss. We could exchange this expression to the minimisation of a general loss function, meaning that any classification method can be used to estimate the cluster probability function. For example, support vector machines could be used by using the hinge loss.

The expression in step 7 of the algorithmic framework is an estimate of the true probability that sample i belongs to cluster j after iteration $t+1$. We could, however, generalise this estimate by including a learning rate η. This leads to

$$p_{ij}^{(t+1)} = \frac{h_j^{(t+1)}(x_i) e^{-\eta L_{\text{model}}(g_j^{(t+1)}(x_i), y_i)}}{\sum_{l=1}^{k} h_l^{(t+1)}(x_i) e^{-\eta L_{\text{model}}(g_l^{(t+1)}(x_i), y_i)}}.$$

this allows us to regularise the final predictive model more, but it does mean that there is another parameter that needs to be tuned, which leads to a greater computational load.

Feature selection is typically an important aspect in constructing a good predictive model, but we have not discussed this aspect in this thesis. It is easy to image that the optimal set of features is different in each cluster. Deriving a strategy to find this optimal set in a clusterwise fashion might therefore greatly increase the performance of clusterwise predictive modelling.

The developed methodology shares a lot of similarities with the Mixture of Experts method. The clusterwise predictive model that uses logistic regression to estimate the cluster probability functions and linear regression to model the clusters is very similar to the original form of the Mixture of Experts method. However, a lot of modifications of this method have been studied, such as incorporating multiple levels of cluster functions, the use of different basis functions, multiple ways to find the optimal model structure and methods other than the EM algorithm to construct the final model [40]. Nevertheless, some ideas that have been discussed in this thesis have not been attempted for the Mixture of Experts method, such as the use of a non-parametric method to cluster the data, the adaptive method of finding the optimal parameter values in the used models and the use of regression trees to model the clusters. It would be interesting to see how these ideas work in combination with these modifications of the Mixture of Experts methodology.

Bibliography

[1] Richard Anstee. The Newton–Raphson Method. `https://www.math.ubc.ca/~anstee/math104/newtonmethod.pdf`. 63

[2] James C. Bezdek, Robert Ehrlich, and William Full. FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2):191–203, 1984. 35

[3] Nicholas H. Bingham and John M. Fry. *Regression Linear Models in Statistics*. Springer-Verlag London, 2010. 14, 15

[4] Leo Breiman. Hinging hyperplanes for regression, classification, and function approximation. *IEEE Transactions on Information Theory*, 39(3):999–1013, 1993. 5

[5] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001. 49

[6] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984. 46

[7] Jason Brownlee. What is the Difference Between Test and Validation Datasets? `https://machinelearningmastery.com/difference-test-validation-datasets/`, 2017. 37

[8] Scott A. Czepiel. Maximum Likelihood Estimation of Logistic Regression Models: Theory and Implementation. `https://czep.net/stat/mlelr.pdf`, 2012. 65

[9] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:1–38, 1977. 7, 9

[10] Wayne S. DeSarbo and William L. Cron. A maximum likelihood methodology for clusterwise linear regression. *Journal of Classification*, 5(2):249–282, sep 1988. 5

[11] Madson Luiz Dantas Dias. fuzzy-c-means: An implementation of Fuzzy C-means clustering algorithm. `https://github.com/omadson/fuzzy-c-means`, 2019. 36

[12] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support Vector Regression Machines. In M C Mozer, M I Jordan, and T Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997. 51

[13] Dheeru Dua and Casey Graff. UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`, 2017. 50

[14] Ronald A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2):179–188, sep 1936. 1

[15] Charles J. Geyer. Statistics 5601 Notes : Smoothing. `http://www.stat.umn.edu/geyer/5601/notes/smoo.pdf`, 2003. 27

[16] Igor Gitman, Jieshi Chen, Eric Lei, and Artur Dubrawski. Novel Prediction Techniques Based on Clusterwise Linear Regression. *arXiv e-prints*, page arXiv:1804.10742, April 2018. 6, 51

[17] David Harrison and Daniel L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, 1978. 50

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 edition, 2009. 5, 16, 17, 23, 35, 44, 45, 51

[19] David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression.*, volume 47. John Wiley & Sons, Inc., Hoboken, New Jersey, 3rd editio edition, 1991. 16

[20] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 1991. 6

[21] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 2, pages 1339–1344 vol.2, 1993. 6

[22] Brian G. Leroux. Consistent Estimation of a Mixing Distribution. *Ann. Statist.*, 20(3):1350–1360, 1992. 38

[23] Jonathan Q. Li and Andrew R. Barron. Mixture Density Estimation. In S A Solla, T K Leen, and K Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 279–285. MIT Press, 2000. 5

[24] Naresh Manwani and Pidaparthy S. Sastry. K-plane regression. *Information Sciences*, 292:39–56, 2015. 5, 50

[25] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions: Second Edition.* Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA, feb 2008. 8

[26] Geoffrey J. McLachlan, Sharon X. Lee, and Suren I. Rathnayake. Finite Mixture Models. *Annual Review of Statistics and Its Application*, 6(1):355–378, mar 2019. 5, 12

[27] Geoffrey J. McLachlan and David Peel. *Finite Mixture Models.* Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA, sep 2000. 5, 7

[28] Geoffrey J. McLachlan and Suren Rathnayake. On the number of components in a Gaussian mixture model. *WIREs Data Mining and Knowledge Discovery*, 4(5):341–355, 2014. 38

[29] Paul D. McNicholas. *Mixture model-based classification.* Taylor & Francis, 2016. 5

[30] Warwick J. Nash and Tasmania Marine Research Laboratories. The Population biology of abalone (Haliotis species) in Tasmania. 1, Blacklip abalone (H. rubra) from the north coast and the islands of Bass Strait. 1994. 50

[31] Karl Pearson. Contributions to the Mathematical Theory of Evolution. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 185(0):71–110, jan 1894. 7

[32] F Pedregosa, G Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 36, 46

[33] Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006. 43

[34] Kathryn Roeder and Larry Wasserman. Practical Bayesian Density Estimation Using Mixtures of Normals. *Journal of the American Statistical Association*, 92(439):894–902, 1997. 38

[35] Gideon Schwarz. Estimating the Dimension of a Model. *Ann. Statist.*, 6(2):461–464, 1978. 38

[36] George A. F. Seber and Alan J. Lee. *Linear regression analysis*. Wiley-Interscience, 2003. 27

[37] Carnegie Mellon University. StatLib Datasets Archive. `http://lib.stat.cmu.edu/datasets/`. 50

[38] Guido van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 1995. 19

[39] Larry Wasserman. *All of Nonparametric Statistics*. Springer Science+Business Media, Inc., New York, 2006. 24

[40] Seniha E. Yuksel, Joseph N. Wilson, and Paul D. Gader. Twenty Years of Mixture of Experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1177–1193, aug 2012. 6, 19, 58

Appendix A

Various Proofs

A.1 Proof of Equation (1.5)

Proof. Let $f : \mathcal{X} \to \mathcal{Y}$ be any prediction rule (without loss of generality assume it is not random). We have

$$
\begin{aligned}
R(f) &= \mathbb{E}\left[(f(X) - Y)^2\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[(f(X) - Y)^2 \,\middle|\, X\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[(f(X) - \mathbb{E}\left[Y \,|\, X\right] + \mathbb{E}\left[Y \,|\, X\right] - Y)^2 \,\middle|\, X\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[(f(X) - \mathbb{E}\left[Y \,|\, X\right])^2 \,\middle|\, X\right]\right. \\
&\quad \left. + 2\mathbb{E}\left[(f(X) - \mathbb{E}\left[Y \,|\, X\right])(\mathbb{E}\left[Y \,|\, X\right] - Y) \,|\, X\right] + \mathbb{E}\left[(\mathbb{E}\left[Y \,|\, X\right] - Y)^2 \,\middle|\, X\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[(f(X) - \mathbb{E}\left[Y \,|\, X\right])^2 \,\middle|\, X\right]\right. \\
&\quad \left. + 2(f(X) - \mathbb{E}\left[Y \,|\, X\right]) \times 0 + \mathbb{E}\left[(\mathbb{E}\left[Y \,|\, X\right] - Y)^2 \,\middle|\, X\right]\right] \\
&= \underbrace{\mathbb{E}\left[(f(X) - \mathbb{E}\left[Y \,|\, X\right])^2 \,\middle|\, X\right]}_{>0} + R(f^*).
\end{aligned}
$$

Thus $R(f) \geq R(f^*)$ for any prediction rule f. □

A.2 Derivation of Equation (3.2)

Proof. First, rewrite the log likelihood function of Expression 2.14 as

$$
\begin{aligned}
Q_c^{(t)}(\boldsymbol{\gamma}) &= \sum_{i=1}^{n}\sum_{j=1}^{k} p_{ij}^{(t)} \log h_j(x_i) \\
&= \sum_{i=1}^{n}\left(\sum_{j=1}^{k-1} p_{ij}^{(t)} \log \frac{e^{\boldsymbol{\gamma}_j^T x_i}}{1+\sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}} + p_{ik}^{(t)} \log \frac{1}{1+\sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}}\right) \\
&= \sum_{i=1}^{n}\left(\sum_{j=1}^{k-1} p_{ij}^{(t)} \left(\boldsymbol{\gamma}_j^T x_i - \log\left(1+\sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}\right)\right) - p_{ik}^{(t)} \log\left(1+\sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}\right)\right) \qquad \text{(A.1)} \\
&= \sum_{i=1}^{n}\left(\sum_{j=1}^{k-1} p_{ij}^{(t)} \boldsymbol{\gamma}_j^T x_i - \sum_{l=1}^{k} p_{il}^{(t)} \log\left(1+\sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}\right)\right) \\
&= \sum_{i=1}^{n}\left(\sum_{j=1}^{k-1} p_{ij}^{(t)} \boldsymbol{\gamma}_j^T x_i - \log\left(1+\sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}\right)\right),
\end{aligned}
$$

where we also changed the argument of $Q_c^{(t)}$ to $\boldsymbol{\gamma}$ to emphasise that those are the parameters we wish to estimate. To maximise this expression, we differentiate with respect to $\boldsymbol{\gamma}_r$ for all $f \in \{1, 2, \ldots, k\}$ and set it equal to zero

$$
\frac{\partial Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}_j} = \sum_{i=1}^{n}\left(p_{ij}^{(t)} x_i - \frac{x_i e^{\boldsymbol{\gamma}_j^T x_i}}{1+\sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l^T x_i}}\right) = \sum_{i=1}^{n} x_i (p_{ij}^{(t)} - h_j(x_i)) = 0, \quad \text{for } j = 1, 2, \ldots, k-1. \qquad \text{(A.2)}
$$

Since we cannot solve this problem analytically, we need to use a numerical optimisation technique. A powerful iterative method that can be used to find a root of a function is the Newton-Raphson method. In this method, the root of a function $f(x)$ is approximated by starting with some initial guess $x^{(0)}$ and estimating the function with a linear function at this point using its first order Taylor series, and finding the point where this linear function is zero, i.e.

$$
f(x^{(0)}) + f'(x^{(0)})(x - x^{(0)}) = 0.
$$

If we solve this for x we find

$$
x = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})}.
$$

We can then take $x^{(1)} = x$ as the new estimate of the root of $f(x)$, and repeat this procedure, resulting in the iterative formula.

$$
x^{(s+1)} = x^{(s)} - \frac{f(x^{(s)})}{f'(x^{(s)})},
$$

where the superscript (s) denotes the estimate after iteration s. When a number of assumptions are met, this algorithm will converge to a root of the function $f(x)$ [1].

In the case of finding solutions to Equation (A.2), the Newton-Raphson algorithm is

$$\boldsymbol{\gamma}^{(s+1)} = \boldsymbol{\gamma}^{(s)} - \left(\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T} \right)^{-1} \frac{\partial Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma}}. \tag{A.3}$$

The second derivative of $Q_c^{(t)}(\boldsymbol{\gamma})$ with respect to $\boldsymbol{\gamma}_j$ and $\boldsymbol{\gamma}_q^T$ is dependent on whether $j = q$ or not. When $j = q$, it is given by

$$\begin{aligned}
\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}_j \partial \boldsymbol{\gamma}_j^T} &= -\sum_{i=1}^{n} x_i \frac{\partial}{\partial \boldsymbol{\gamma}_j^T} \left(\frac{e^{\boldsymbol{\gamma}_j x_i}}{1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l x_i}} \right) \\
&= -\sum_{i=1}^{n} x_i \frac{\left(1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l x_i}\right) x_i^T e^{\boldsymbol{\gamma}_j x_i} - e^{\boldsymbol{\gamma}_j x_i} x_i^T e^{\boldsymbol{\gamma}_j x_i}}{\left(1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l x_i}\right)^2} \\
&= -\sum_{i=1}^{n} x_i \frac{x_i^T e^{\boldsymbol{\gamma}_j x_i} \left(\left(1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l x_i}\right) - e^{\boldsymbol{\gamma}_j x_i} \right)}{\left(1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l x_i}\right)^2} \\
&= -\sum_{i=1}^{n} x_i x_i^T h_j(x_i)(1 - h_j(x_i)),
\end{aligned}$$

and when $j \neq q$ by

$$\begin{aligned}
\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}_j \partial \boldsymbol{\gamma}_q^T} &= -\sum_{i=1}^{n} x_i \frac{\partial}{\partial \boldsymbol{\gamma}_q^T} \left(\frac{e^{\boldsymbol{\gamma}_j x_i}}{1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l x_i}} \right) \\
&= -\sum_{i=1}^{n} x_i \frac{-e^{\boldsymbol{\gamma}_j x_i} x_i^T e^{\boldsymbol{\gamma}_q x_i}}{\left(1 + \sum_{l=1}^{k-1} e^{\boldsymbol{\gamma}_l x_i}\right)^2} \\
&= \sum_{i=1}^{n} x_i x_i^T h_j(x_i) h_q(x_i).
\end{aligned}$$

We can now construct a simple iterative algorithm that finds a local maximum of Expression (A.1). For convenience, it will be expressed using matrix multiplication. We can write Expression for (A.2) this way as

$$\frac{\partial Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}} = \boldsymbol{A}^T (\boldsymbol{P}' - \boldsymbol{H}'),$$

where $\boldsymbol{A}$ is as in 3.1, and $\boldsymbol{P}'$ and $\boldsymbol{H}'$ are the matrices of size $n \times k - 1$ with elements p_{ij} and $h_j(x_i)$, respectively. Although $\boldsymbol{\gamma}$ and $\partial Q_c^{(t)}(\boldsymbol{\gamma})/\partial \boldsymbol{\gamma}$ are technically matrices, we can express them as one-dimensional column arrays by appending each consecutive column below the first. This allows us to express the second derivatives as a two-dimensional array, and we can express the Newton-Raphson method in terms of conventional matrix multiplication operations.

In order to write the expression for $\partial^2 Q_c^{(t)}(\boldsymbol{\gamma})/\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T$ in matrix form, we first define

$$\boldsymbol{W}_j = \begin{bmatrix} h_j(x_1)(1-h_j(x_1)) & 0 & \dots & 0 \\ 0 & h_j(x_2)(1-h_j(x_2)) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_j(x_n)(1-h_j(x_n)) \end{bmatrix},$$

and

$$\boldsymbol{W}_{jq} = \begin{bmatrix} h_j(x_1)h_q(x_1) & 0 & \dots & 0 \\ 0 & h_j(x_2)h_q(x_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_j(x_n)h_q(x_n) \end{bmatrix}.$$

Now, we can write

$$\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}_j \partial \boldsymbol{\gamma}_j^T} = -\boldsymbol{A}^T \boldsymbol{W}_j \boldsymbol{A},$$

and

$$\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}_j \partial \boldsymbol{\gamma}_q^T} = \boldsymbol{A}^T \boldsymbol{W}_{jq} \boldsymbol{A},$$

which gives us the expression

$$\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T} = \begin{bmatrix} -\boldsymbol{A}^T \boldsymbol{W}_1 \boldsymbol{A} & \boldsymbol{A}^T \boldsymbol{W}_{12} \boldsymbol{A} & \dots & \boldsymbol{A}^T \boldsymbol{W}_{1(k-1)} \boldsymbol{A} \\ \boldsymbol{A}^T \boldsymbol{W}_{21} \boldsymbol{A} & -\boldsymbol{A}^T \boldsymbol{W}_2 \boldsymbol{A} & \dots & \boldsymbol{A}^T \boldsymbol{W}_{2(k-1)} \boldsymbol{A} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^T \boldsymbol{W}_{(k-1)1} \boldsymbol{A} & \boldsymbol{A}^T \boldsymbol{W}_{(k-1)2} \boldsymbol{A} & \dots & -\boldsymbol{A}^T \boldsymbol{W}_{k-1} \boldsymbol{A} \end{bmatrix}.$$

The term $\left(\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T}\right)^{-1} \frac{\partial Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma}}$ in Expression (A.3) can, in some cases, be very small which leads to numerical inaccuracy due to round-off errors [8]. We will therefore use the slightly different form

$$\begin{aligned} \boldsymbol{\gamma}^{(s+1)} &= \boldsymbol{\gamma}^{(s)} - \left(\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T}\right)^{-1} \frac{\partial Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma}} \\ &= \left(\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T}\right)^{-1} \left(\frac{\partial^2 Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}^T} \boldsymbol{\gamma}^{(s)} - \frac{\partial Q_c^{(t)}(\boldsymbol{\gamma}^{(s)})}{\partial \boldsymbol{\gamma}}\right), \end{aligned}$$

which gives us the iterative formula we sought. □

A.3 Proof of Equation (3.5) for the Kernel Smoothing Method

Proof. We can give the general form of the prediction rule given by Nadaraya-Watson regression analogously with Equation (3.3), writing $\hat{y}(x) = h_j^{(t+1)}(x)$ and $y_i = p_{ij}^{(t)}$, as

$$\hat{y}(x) = \sum_{i=1}^{n} \frac{K_\lambda(x_i, x)}{\sum_{r=1}^{n} K_\lambda(x_r, x)} y_i.$$

So that we can express the LOOCV estimate of data point (x_i, y_i) as

$$\begin{aligned}
\hat{y}_{-i} &= \sum_{\substack{l=1 \\ l \neq i}}^{n} \frac{K_\lambda(x_l, x_i)}{\sum\limits_{\substack{r=1 \\ r \neq i}}^{n} K_\lambda(x_r, x_i)} y_l \\
&= \sum_{\substack{l=1 \\ l \neq i}}^{n} \frac{K_\lambda(x_l, x_i)}{\sum_{r=1}^{n} K_\lambda(x_r, x_i) - K_\lambda(x_i, x_i)} y_l \\
&= \sum_{\substack{l=1 \\ l \neq i}}^{n} \frac{\frac{K_\lambda(x_l, x_i)}{\sum_{r=1}^{n} K_\lambda(x_r, x_i)}}{1 - \frac{K_\lambda(x_i, x_i)}{\sum_{r=1}^{n} K_\lambda(x_r, x_i)}} y_l \\
&= \sum_{\substack{l=1 \\ l \neq i}}^{n} \frac{L_{il}}{1 - L_{ii}} y_l \\
&= \frac{1}{1 - L_{ii}} \sum_{\substack{l=1 \\ l \neq i}}^{n} L_{il} y_l.
\end{aligned}$$

□

Appendix B

Experimental Setup

B.1 Optimal Hyperparameter Values

As is discussed in Section 6.2, the optimal values of the hyperparameters of the non-clusterwise predictive modelling methods are found via a grid search. The values that were found this way are:

Boston housing

- Regression tree: $max\_leaf\_nodes = 26$, $min\_samples\_leaf = 8$
- Random forest: $max\_leaf\_nodes = 49$, $min\_samples\_leaf = 2$, $max\_features = \sqrt{n\_features}$
- k-nearest neighbours: $p = 7$, $n\_neighbors = 2$
- Support vector regression: $\varepsilon = 0.4$ and $C = 140$

Abalone

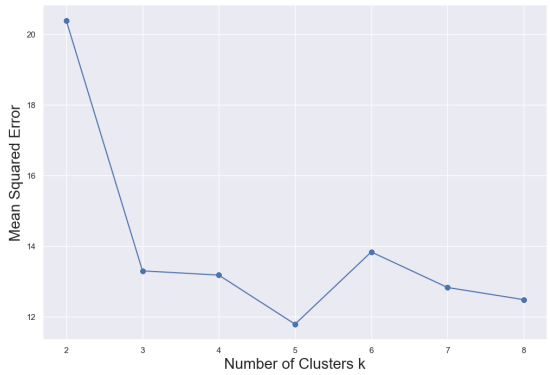
- Regression tree: $max\_leaf\_nodes = 50$, $min\_samples\_leaf = 16$
- Random forest: $max\_leaf\_nodes = 109$, $min\_samples\_leaf = 4$, $max\_features = n\_features$
- k-nearest neighbours: $p = 2$, $n\_neighbors = 11$
- Support vector regression: $\varepsilon = 1.2$ and $C = 100$

Auto-mpg

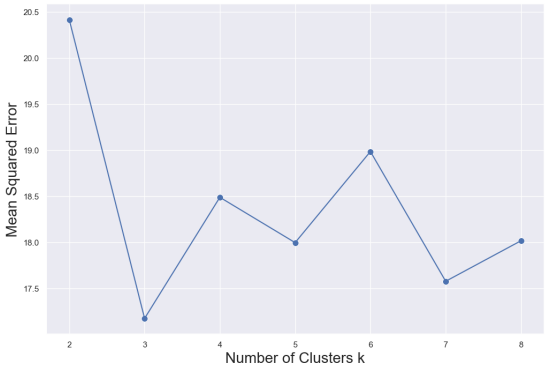
- Regression tree: $max\_leaf\_nodes = 63$, $min\_samples\_leaf = 1$
- Random forest: $max\_leaf\_nodes = 75$, $min\_samples\_leaf = 1$, $max\_features = n\_features$
- k-nearest neighbours: $p = 1$, $n\_neighbors = 5$
- Support vector regression: $\varepsilon = 0.7$ and $C = 100$

B.2 Optimal Number of Clusters

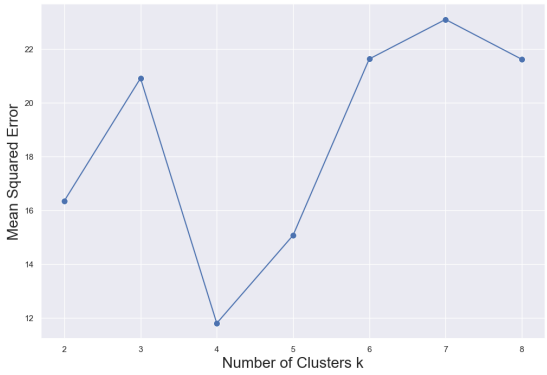
The figures below show the influence of the amount of clusters k on the validation scores.



(a) *For the method that uses linear regression to model the individual clusters, and where the cluster probability functions are estimated using logistic regression.*

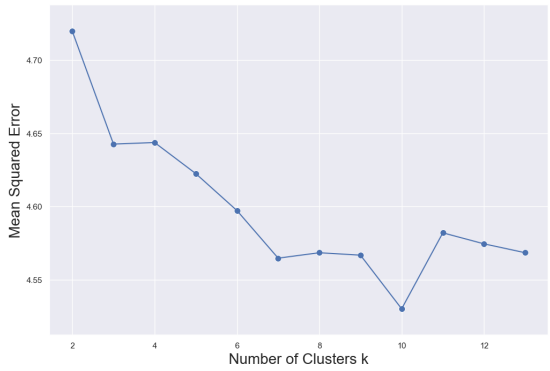


(b) *For the method that uses linear regression to model the individual clusters, and where the cluster probability functions are estimated using the kernel smoothing method.*

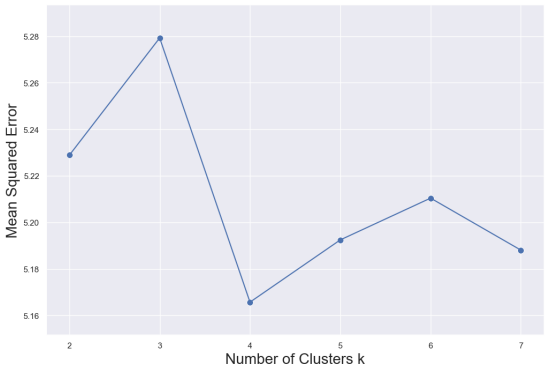


(c) *For the method that uses a regression tree to model the individual clusters, and where the cluster probability functions are estimated using the kernel smoothing method.*

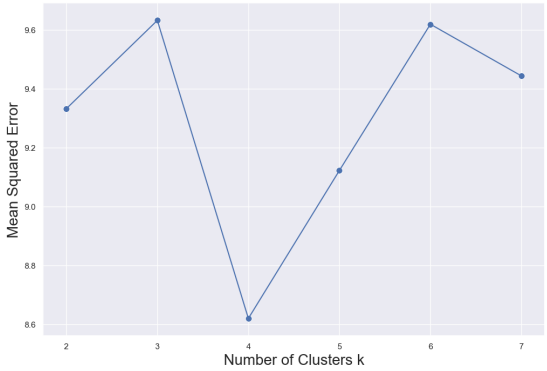
Figure B.1: *The best mean squared error found on the verification set for different numbers of clusters k, for the Boston housing dataset*



(a) *For the method that uses linear regression to model the individual clusters, and where the cluster probability functions are estimated using logistic regression.*

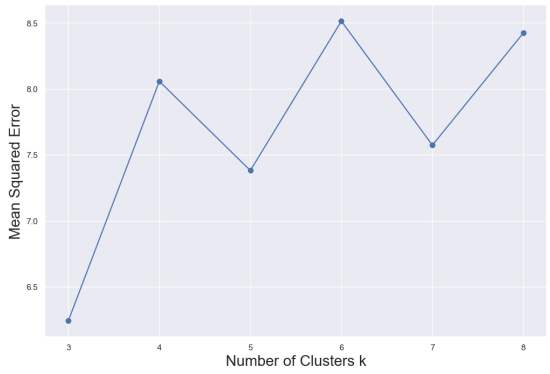


(b) *For the method that uses linear regression to model the individual clusters, and where the cluster probability functions are estimated using the kernel smoothing method.*

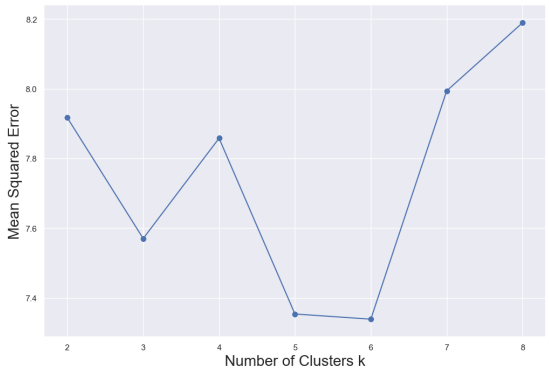


(c) *For the method that uses a regression tree to model the individual clusters, and where the cluster probability functions are estimated using the kernel smoothing method.*

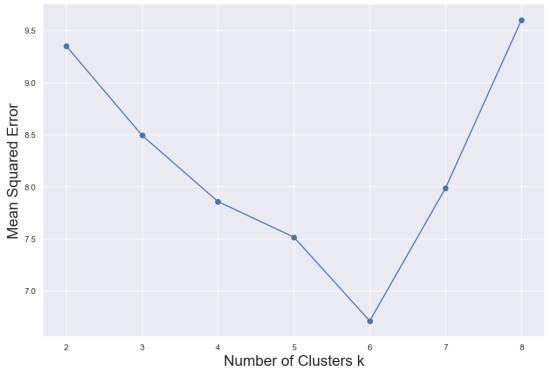
Figure B.2: *The best mean squared error found on the verification set for different numbers of clusters k, for the Abalone dataset*



(a) For the method that uses linear regression to model the individual clusters, and where the cluster probability functions are estimated using logistic regression. $k = 2$ did not lead to stable results and is therefore excluded.



(b) For the method that uses linear regression to model the individual clusters, and where the cluster probability functions are estimated using the kernel smoothing method.



(c) For the method that uses a regression tree to model the individual clusters, and where the cluster probability functions are estimated using the kernel smoothing method.

Figure B.3: *The best mean squared error found on the verification set for different numbers of clusters k, for the auto-mpg dataset*

B.3 Amount of Models in the Ensembles

The amount of models that were below the performance threshold and are included in each of the ensemble methods are given in the table below. Note that on the Abalone dataset, no model using the kernel smoothing method to estimate the cluster probability methods performed better than simple linear regression, so the threshold was increased to 5.3 for the method using linear regression as basis model, and 10 for the regression tree based method.

Method	Boston housing	Abalone	Auto-mpg
Linear/Logistic Ensemble	86	189	123
Linear/Kernel Ensemble	102	28	198
Tree/Kernel Ensemble	27	20	75

B.4 Found Clusterwise Regression Models

In the following tables, the found parameter values of the linear regression models found by the best clusterwise predictive models acquired from the experiments in Chapter. Furthermore, the amount of data points that have the greatest inferred probability of belonging to each cluster, as well as the mean feature and target values of those points, are included as well. The average feature values are of the unscaled features. To get a description of each feature, see the respective data repository.

Table B.1: *The coefficient values of each linear model found for the Boston Housing dataset, where logistic regression is used to estimate the cluster probability function.*

Coefficient	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Intercept	71.6239	22.0313	20.2417	20.1406	-9.96114
Crim	-58.4987	-1.29293	-4.16685	-1.01464	-6.48431
Zn	0.26357	-1.73316	0.550313	1.00148	-26.9686
Indus	0.411438	0.0527926	-0.167711	1.63955	3.9374
Chas	259.564	0.176366	0.0372985	0.745033	-0.136107
Nox	0.0895774	-4.43947	0.586153	0.0780161	-7.55418
Rm	7.8666	0.106476	2.82875	3.63894	2.091
Age	-3.39959	-3.96013	-2.19881	-0.251191	12.7562
Dis	-1.28602	-3.57487	-1.89754	-0.495841	-7.39604
Rad	1.28919	3.70496	2.83711	-0.0554287	28.8313
Tax	-3.91165	-2.89525	-1.72754	-0.851645	-16.8762
Ptratio	-2.24231	-2.04228	-0.532055	-1.23905	-11.6104
Black	-9.70285	-0.770217	1.38795	1.23886	-9.25765
Lstat	0.605685	-2.03449	-0.682873	-4.2529	-13.214

Table B.2: *The average feature and target values and size of each cluster found in the Boston Housing dataset, where logistic regression is used to estimate the cluster probability function.*

Feature	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Crim	0.188824	10.0998	1.05606	4.40987	4.49214
Zn	30.9167	1.53191	7.84021	19.8778	0.909091
Indus	3.78738	15.4915	10.2886	11.7544	15.5055
Chas	0	0.0638298	0.0309278	0.177778	0.136364
Nox	0.482452	0.608298	0.533793	0.555222	0.644773
Rm	7.20879	5.79413	6.0789	6.41691	6.462
Age	51.2024	92.0043	61.3196	63.1422	94.95
Dis	4.31915	2.81842	4.04121	4.55974	2.07031
Rad	4.69048	16.5532	5.73196	9.91111	17.0455
Tax	275.69	533.66	342.897	431.356	534.818
Ptratio	16.6167	19.383	18.8041	18.6667	18.6045
Black	389.289	356.795	364.404	356.596	384.503
Lstat	6.44643	19.6528	11.8978	11.8393	10.6332
Medv	35.7	16.0	20.3	21.7	28.9
Size	42	47	97	45	22

Table B.3: *The coefficient values of each linear model found for the Boston Housing dataset, where the kernel smoothing method is used to estimate the cluster probability function.*

Coefficient	Cluster 1	Cluster 2	Cluster 3
Intercept	21.8295	22.2426	22.8704
Crim	-1.09308	0.419531	-3.86636
Zn	1.75469	-0.218291	2.46913
Indus	0.355984	-1.16377	7.19375
Chas	0.13212	0.258022	1.62876
Nox	-1.36986	0.123011	-5.25493
Rm	3.79415	6.83767	0.608284
Age	-0.308676	-3.1987	-1.29835
Dis	-1.65769	-2.5144	-8.71839
Rad	0.036829	2.00163	10.8277
Rax	-0.674673	-1.45349	-12.974
Ptratio	-1.97645	-1.97607	-2.86368
Black	1.05016	1.21903	-0.10625
Lstat	-1.73482	1.82999	-8.68144

Table B.4: *The average feature and target values and size of each cluster found in the Boston Housing dataset, where the kernel smoothing method is used to estimate the cluster probability function.*

Feature	Cluster 1	Cluster 2	Cluster 3
Crim	4.10771	2.21486	4.11047
Zn	12.0931	12.488	11.4044
Indus	11.212	9.82024	11.7137
Chas	0.0490196	0.060241	0.102941
Nox	0.548451	0.53589	0.579118
Rm	6.21013	6.30333	6.45679
Age	62.6078	69.5241	76.4191
Dis	4.22739	3.66025	3.25893
Rad	7.91176	7.59036	13.4559
Tax	388.01	361.181	463.882
Ptratio	18.9716	17.9554	18.4824
Black	360.56	375.709	367.82
Lstat	11.9094	12.9998	12.0806
Medv	21.1	23.3	25.6
size	102	83	68

Table B.5: *The coefficient values of each linear model found for the Abalone dataset, where logistic regression is used to estimate the cluster probability function.*

Coefficient	1	2	3	4	5	6	7	8	9	10
Intercept	10.3	8.1	10.4	8.4	9.2	10.4	8.6	9.1	8.7	13.0
Length	1.1	-1.3	4.3	0.8	1.4	-0.9	-1.4	2.1	0.2	1.0
Diameter	0.5	1.3	-0.9	2.8	0.5	1.8	1.1	-1.1	1.2	1.0
Height	1.5	1.0	-1.0	1.0	-0.5	-0.1	1.0	1.0	0.4	-0.1
Whole weight	5.9	2.8	0.0	-4.5	7.4	9.2	1.0	3.4	4.4	11.5
Shucked weight	-4.9	-2.3	-2.1	1.2	-4.2	-5.7	-0.3	-1.6	-3.8	-7.0
Viscera weight	-2.8	-0.2	0.3	1.1	-2.3	-1.8	0.8	-1.8	-0.2	-4.3
Sex: female	-0.1	0.8	1.1	0.5	-0.3	0.3	-0.4	-0.1	-0.5	0.8
Sex: male	-0.1	0.8	0.4	-0.2	0.6	-0.4	-0.7	-0.2	-0.1	0.5

Table B.6: *The average feature and target values and size of each cluster found in the Abalone dataset, where logistic regression is used to estimate the cluster probability function.*

Feature	1	2	3	4	5	6	7	8	9	10
Length	0.57	0.62	0.43	0.59	0.51	0.47	0.58	0.41	0.40	0.54
Diameter	0.44	0.49	0.35	0.46	0.39	0.37	0.46	0.31	0.30	0.43
Height	0.16	0.17	0.12	0.15	0.13	0.11	0.16	0.11	0.11	0.16
Whole weight	0.98	1.22	0.53	1.01	0.68	0.63	0.96	0.57	0.58	0.89
Shucked weight	0.38	0.51	0.20	0.46	0.31	0.26	0.49	0.27	0.26	0.35
Viscera weight	0.21	0.27	0.11	0.23	0.15	0.13	0.21	0.12	0.11	0.20
Sex: female	0.30	0.57	0.29	0.06	0.24	0.47	0.33	0.15	0.29	0.28
Sex: male	0.62	0.42	0.64	0.70	0.08	0.18	0.44	0.23	0.17	0.44
Rings	12.5	9.9	9.8	9.8	8.2	10.1	8.9	7.6	6.5	14.9
Size	134	313	159	223	398	139	149	232	87	254

Table B.7: *The coefficient values of each linear model found for the Abalone dataset, where the kernel smoothing method is used to estimate the cluster probability function.*

Coefficient	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Intercept	13.1022	9.50239	8.71896	9.03005
Length	-0.653901	-0.276903	1.88981	-1.07668
Diameter	2.33484	3.167	-0.94483	0.923798
Height	0.471804	-1.24141	0.00133938	3.11591
Whole weight	11.3495	10.2332	4.28548	-0.710304
Shucked weight	-9.08068	-8.87736	-2.32932	-0.447487
Viscera weight	-2.37521	-1.99962	-1.41399	0.279656
Sex: female	0.319759	0.355656	0.312927	-0.219074
Sex: male	0.621903	0.712232	0.288879	-0.212015

Table B.8: *The average feature and target values and size of each cluster found in the Abalone dataset, where the kernel smoothing method is used to estimate the cluster probability function.*

Feature	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Length	0.520929	0.529136	0.531515	0.512344
Diameter	0.407493	0.412761	0.413723	0.39808
Height	0.144336	0.139795	0.142654	0.134181
Whole weight	0.831637	0.831351	0.869685	0.788263
Shucked weight	0.350372	0.345975	0.382538	0.351557
Viscera weight	0.180158	0.179751	0.190577	0.172245
Sex: female	0.353982	0.327273	0.272308	0.267071
Sex: male	0.351032	0.440909	0.367692	0.341426
Rings	13.6	10.4	8.7	8.7
Size	339	440	650	659

Table B.9: *The coefficient values of each linear model found for the Auto-mpg dataset, where logistic regression is used to estimate the cluster probability function.*

Coefficient	Cluster 1	Cluster 2	Cluster 3
Intercept	20.9419	23.6317	21.3287
Cylinders	2.7243	-3.17144	1.00917
Displacement	0.797433	-0.877339	-0.837559
Horsepower	-5.43802	-0.723479	-0.729198
Weight	-6.19564	-0.371986	-7.88939
Acceleration	0.703285	-1.62928	-0.455186
Model year	4.28373	1.54112	1.66155
Origin: 1	-1.31474	-4.4255	0.932244
Origin: 2	0.658091	-2.16214	-0.79533

Table B.10: *The average feature and target values and size of each cluster found in the Auto-mpg dataset, where logistic regression is used to estimate the cluster probability function.*

Feature	Cluster 1	Cluster 2	Cluster 3
Cylinders	4.125	6.32609	5.35593
Displacement	109.458	258.25	168.458
Horsepower	77.9688	122.766	97.3475
Weight	2415.56	3474.86	2668.66
Acceleration	17.3521	14.8891	14.8864
Model year	78.4583	74.7717	75.3729
Origin: 1	0.1875	0.967391	0.542373
Origin: 2	0.3125	0.0108696	0.237288
Mpg	30.7	18.6	23.9
Size	48	92	59

Table B.11: *The coefficient values of each linear model found for the Auto-mpg dataset, where the kernel smoothing method is used to estimate the cluster probability function.*

Coefficient	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Intercept	22.9017	21.9799	22.444	22.0717	23.1206	22.7784
Cylinders	-3.79569	-0.865199	-2.19833	1.83784	0.598303	6.26181
Displacement	-2.53687	1.05886	1.40586	4.36493	-6.64082	-7.38378
Horsepower	1.92351	1.32777	-1.52078	-7.00099	-2.36871	-4.44452
Weight	-0.69752	-7.49506	-2.11299	-6.43442	1.98459	-6.35145
Acceleration	-0.71111	0.969715	-1.06883	-0.578526	-1.74963	-0.447791
Model year	1.10389	0.932988	1.35204	3.31279	1.9389	2.88896
Origin: 1	-3.1253	1.38217	-2.01811	-1.37886	-2.0575	-3.8499
Origin: 2	-2.54946	-0.698587	-0.458213	0.961343	-0.746989	-0.371621

Table B.12: *The average feature and target values and size of each cluster found in the Auto-mpg dataset, where the kernel smoothing method is used to estimate the cluster probability function.*

Feature	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
cylinders	6.16949	5.54839	6.41667	4.75	5.3913	4.38235
Displacement	234.441	191.065	262.833	137.786	198	131.676
Horsepower	111.559	103.516	138.771	89.5357	103.478	81.5294
Weight	3214.78	2965.03	3504.12	2580.32	3025.7	2516.41
Acceleration	15.1508	15.3484	13.7417	16.0786	16.0478	16.5353
Model year	75.0169	74.1935	73.75	77.9643	76.5652	78
Origin: 1	0.79661	0.548387	0.708333	0.214286	0.826087	0.705882
Origin: 2	0.0508475	0.258065	0.125	0.25	0.130435	0.176471
Mpg	20.0	21.0	19.0	27.6	22.2	30.0
Size	59	31	24	28	23	34