Cédric Beaulac

Statistical Learning Techniques : A short review

University of Toronto
Department of Statistical Science

1 Introduction

This is a personnal set of notes inspired by Hastie and *al.* [3] and Bishop [2]. Theses notes glance quickly through most of the material covered in these two books.

Even though most Machine Learning (*ML*) books tends to talk as little as possible about parameters and distributions, the basic *ML* problem is not that different from the basic statistical problem. This problem consist of recognizing patterns in some sort of data. Being popular amongst computer scientist the basic solution is to write a program that encodes a set of rules that are useful to solve the problem. Most of the time, the machine learning approach is to define some sort of error and pick weights such that it minimizes the error.

In supervised learning we typically define a set of *inputs* variables, also called *predictors*, that have some influence on a set of selected *outputs* variables or *response*. In this set up, the researcher is deciding which variables are perceived as inputs and which one are perceived as outputs. Classification and regression are typical problems that supervised learning algorithms are designed for.

Unsupervised learning is more interested in creating an internal representation of the data points. The main goal is to characterize and capture some kind of structure in the data. There's no imposed distinction between inputs or outputs in these types of problem. Clustering is a good example of a typical problem where unsupervised learning might be useful.

2 Supervised Learning

2.1 Introduction

Supervised learning is the most common approach to data analysis. In supervised learning, the researcher is interest in the relationship between a set of chosen predictors and a set of chosen response. The problem consist of learning the weights, the parameters, that defines the relationship between those variable. It is important to notice that a structure is imposed on the variables.

In this section we will see multiple approaches to supervised learning and we will use algorithms that are of statistical nature and computer science nature. We will also briefly compare these two distinct approach of data analysis through out this section.

2.2 A simple approach : The k-nearest neighbour method

Let's introduce a very simple approach that can both solve the regression and the classification problem, the k-nearest neighbour method. This quiet simple methodology consist of both prediction the numerical value or the class of Y for a particular input x by averaging over $\{y(x_i)|i \in N_k(x)\}$ where $N_k(x)$ is the set containing the k closest point x_i to x in our data set. Here is a formal definition of the estimate of $Y(x)$:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

For the regression problem, the average as computed above is the prediction and for the classification problem the predicted class is the class that is the most represented in the set $N_k(x)$. This approach is really simple, easy to program and does not requires any specific assumptions. Since it did not assume any distribution it is impossible to build a confidence interval for the estimate. It also produce a non-continuous estimates. We will discuss the possibility of smoothing this estimator in the *Kernel-based method* section.

2.3 Linear Methods

2.3.1 Introduction

Linear methods are the most popular approaches to most supervised learning problem. They are the bread and butter of applied statistician and data analyst. They are usually a good way to take a first look at the data set. They are easy to use and will solve most of your problem. Our knowledge of linear method is now deep and there exist many improvements to the basic linear methods.

2.3.2 Basic methods

In this subsection we will introduce the basic linear approach for solving both the regression and the 2-class classification problem. Let us define the model first :

$$Y = \beta_0 + \sum_{j=1}^{p} X_j \beta_j.$$

Here, we are assuming the relationship between a response variable Y and a set of predictors X as linear. We are trying to learn the set of parameter β that best explain this linear relationship. A first approach to that problem is the well known least square methods. Suppose we have the vector of inputs $X^T = (X_1, X_2, ...X_t)$ and we are trying to explain the values of Y using the following estimates :

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j X_j$$

which can be written as :

$$\hat{Y} = X^T \hat{\beta}$$

by adding $X_0 = 1$ into the vector X^T. For a particular data point (x_i, y_i) the prediction is then $\hat{y}_i = x_i^T \hat{\beta}$ and then the prediction error is $y_i - x_i^T \hat{\beta}$. Minimizing the sum of squared error : $\sum_{i=1}^{n} (y_i - x_i^T \beta)^2$ is one of the many ways to estimates the β's that best fits the data. The solution to this minimization problem is : $\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T Y$. With this solution we can solve the regression problem and the classification problem.

The mean-squared error minimizer is a good example of an efficient computer science approach to data analysis. No assumption regarding the distribution of the data is made. The researcher select a model and an error and then chose the weights that minimizes the selected error measurement. Let us see a distribution-based approach to that very classic problem.

We've already discussed one solution in order to estimates the β's which consist of minimizing the squared error, let's now define the distribution of Y by adding assumptions to the model. The first set of assumptions are that the observations y_i are uncorrelated and have constant variance σ^2 and that the x_i are fixed. Then we can also assume that the conditional expectation of Y is linear in X and that the error term is Gaussian, hence the new model :

$$Y = \mathbf{X}^T \beta + \varepsilon$$

where $\varepsilon \sim N(0, \sigma^2)$. With these assumption a wide theory of regression was born and will not be covered thorough in these notes. Notice that we could now think of building a Maximum Likelihood estimator for the β's since we have a distribution for Y. It is well known that the solution of that maximization problem is also :$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T Y$. This estimator is also the unbiased estimator with the smallest variance.

Even though this is one of the easiest approach to solve the regression problem we can already see the differences between a computer science and a statistical approach. The mean squared error regression assume no particular distribution on the data, resulting in an assumption-free analysis. The parametric approach assume the existence of a normally distributed error which makes the model a bit more complex. An honest researcher would have to prove that this assumption is reasonable. But a pro of that approach is that we now have a distribution for our parameters. We can therefore build confidence interval for our weights and test for variable significance.

Finally, quickly notice that if we suppose that we have multiples output $\mathbf{Y} = (Y_1, Y_2, ...Y_k)$, we could think of the following equivalent linear model :

$$\mathbf{Y} = \mathbf{XB} + \mathbf{E}$$

where $\mathbf{Y}$ is the $n \times k$ matrix of observations, $\mathbf{X}$ is the $n \times (p+1)$ input matrix, $\mathbf{B}$ is the $(p+1) \times k$ matrix of parameters and finally $\mathbf{E}$ is the $n \times k$ matrix of errors. Then in this scenario the squared error minimizer would once again be of the form : $\hat{\mathbf{B}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$.

2.3.3 Subset selection

Finding and selecting the optimal subset of parameters is an important part of fitting the right linear model in order to explain the output variable. Adding parameters to the model can only make the likelihood go higher but since overfitting can be a problem we must find a way to make sure that each variable we add to the model increase the likelihood significantly. This is why we must address the subset selection problem with tools that consider both the likelihood and the number of parameters.

The Akaike Information Criterion (AIC) is one of the popular way to help with subset selection problem. The AIC is calculated according to : $AIC = 2p - 2\ln(L)$, where p is the number of parameters and L is the likelihood. Given a set of possible models, the preferred model is the one with the **lowest** AIC. The Bayesian Information Criterion (BIC) is defined as : $BIC = p\ln(n) - 2\ln(L)$. Once again we pick the model with the lowest value, but, the AIC penalizes less strongly the number of parameters. Notice also that Yang [4] wrote a full article demonstrating that the AIC is asymptotically optimal in selecting the model with the least mean squared error while the BIC is not.

Even with these tools in hands, model selection is still a very complicated problem. The AIC and BIC can prove useful if you need to choose from a small set of models.

There also exist many techniques in order to select the best subset of size k by picking the subset that gives the smallest residuals sum of square amongst an original set of p explanatory variable. If p is small enough an exact procedure is possible, the best-subset selection procedure. If p is rather larger the Forward and Backward stepwise selection procedure could prove usefull into solving this kind of problem.

Other popular techniques are the ridge regression and the Lasso method. Both of these methods consist in minimizing the least square error, under a certain condition of the vector of parameters, more precisely :

$$\beta^{Ridge} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=i}^{p} x_{ij}\beta_j)^2$$
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 \leq t$$

which can re-written as :

$$\beta^{Ridge} = \underset{\beta}{\operatorname{argmin}} \left[\sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=i}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right]$$

Solving this result in the following estimator :

$$\hat{\beta}^{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T y$$

The Lasso uses a similar technique but instead uses the following constraint : $\sum_{j=1}^{p} |\beta_j| \leq t$. The best-subset selection completely drops all variables with coefficients smaller than Mth largest, forming a "hard-threshold". The Ridge regression does a proportional shrinkage while the Lasso is an intermediary somehow by translating each coefficient by a constant factor λ and then truncating at 0, which is called "soft-thresholding".

2.3.4 Linear basis expansion

Suppose we want to generalize the model so that it can go beyond linearity. One solution could be to replace the vector of X with a vector of transformations of X and then use linear models like we previously did. This model would be :

$$Y = \sum_{j=1}^{J} \beta_j h_j(X)$$

where $h_j(X) : \mathbb{R}^p \to \mathbb{R}$ is the jth transformation of X. What is simple about this model is that once we have set up our basis function h_j the model is linear in these new variables and estimation is performed as before. Typically the transformation function can be the product of 2 variables, then representing the interaction, any power of X or exponential and logarithmic transformation. Up to that point both the error minimizing and the likelihood maximisation approach are still viable.

An interesting way to use this linear basic expansion is via piecewise polynomial regression and splines. The idea is to use transformation functions that are indicator functions for a particular interval over X to build a piecewise model. As an example, using $h_1(X) = I(X \leq b_1)$, $h_2(X) = I(b_1 < X \leq b_2)$ and $h_3(X) = I(X > b_2)$ we will create a piecewise function constant at the mean of Y over each of the three intervals created by our basis expansion. By adding, $h_4(X) = I(X \leq b_1)X$, $h_5(X) = I(b_1 < X \leq b_2)X$ and $h_6(X) = I(X > b_2)X$ to the model we will obtain a piecewise linear fit. Finally using $h_1(X) = 1$, $h_2(X) = X$, $h_3(X) = (X - b_1)_+$ and $h_4(X) = (X - b_2)_+$ will create a continuous piecewise linear fit.

The following figure present many of these methods for piecewise regression. We start by presenting the basic least square estimator, the we use indicator function of intervals as our vector of inputs creating a piecewise constant regression, the first graphic of the second line represent piecewise linear regression for every single interval and we forced continuity on the next graphic. Finally, for the last line of figure 1 we've included x^2 as a covariates and then forced continuity of the first derivatives.

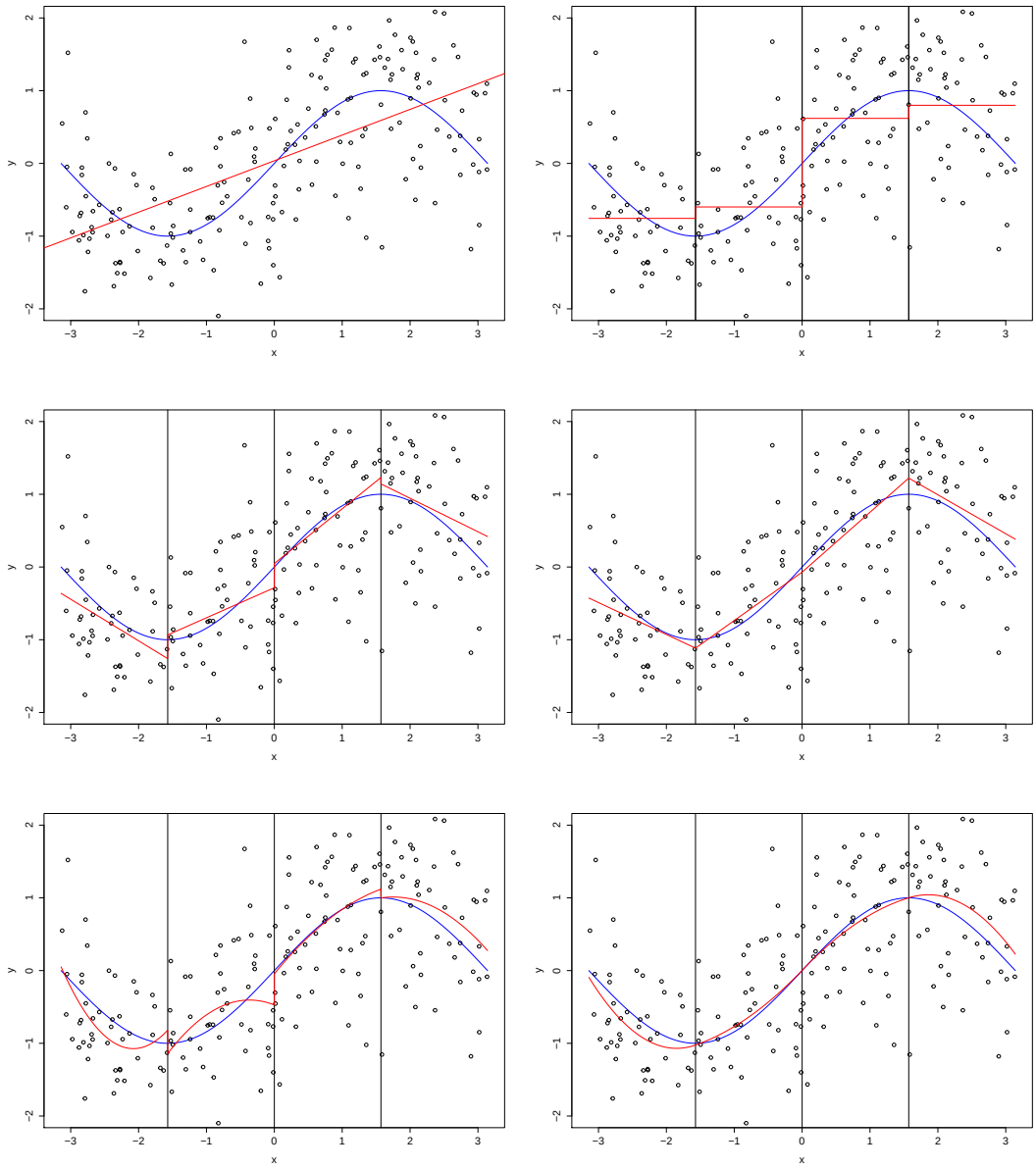

Figure 1: Graphical results of a piecewise linear regression using linear basis expansion.

An order-M spline is a piece-wise polynomial of order M regression with continuous derivatives up to order $M-2$. A cubic spline is an order-4 spline. As an example the piecewise-constant function in figure 1 is and order-1 spline.

Observe that splines can be really efficient for point estimation but as the number of parameters grows larger and changes from on interval to another, building confidence interval and interpreting the parameters become more complicated.

2.3.5 Linear methods for classification

In this short section we will quickly discuss the specifications regarding the classification using linear models. Suppose we have a discrete set of classes $\mathcal{G}$, an intuitive idea is to find a way to estimate $P(G = k|X = x)$. If there are only two classes, The logistic regression is a common solution to that problem :

$$\log \frac{P(G = 1|X = x)}{P(G = 2|X = x)} = \beta_0 + \beta^T x$$

In classification problem, it is interesting to define a decision boundary. In the logistic regression case, the decision boundary would be the hyperplane such that :

$$\begin{aligned} P(G = 1|X = x) = P(G = 2|X = x) &= \frac{1}{2} \\ \Rightarrow \frac{1}{1 + \exp(\beta_0 + \beta^T x)} &= \frac{1}{2} \\ \Rightarrow \exp(\beta_0 + \beta^T x) &= 1 \\ \Rightarrow \beta_0 + \beta^T x &= 0. \end{aligned}$$

Therefore it would be the hyperplane defined by $\{x|\beta_0 + \beta^T x = 0\}$.

Let's now focus on problems where there might be more than two different classes, let's say K classes. An intuitive approach would be to perform the linear regression over an Indicator matrix. We would build an observation matrix $\mathbf{Y}$ of size $n \times K$ with $Y_{ik} = 1$ if $G = k$ for the ith observation else 0. Once again the solution to such problem would simply be : $\hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. This method will produce estimates for $P(G = k|X = x)$. Let's call those estimates $\hat{f}_k(x)$. Even though $\sum_k \hat{f}_k(x) = 1$ some of these estimates might be negative or higher than one which can be problematic. That being said, prediction classes by picking the class with the highest estimated $f_k(x)$ mights still be correct. One way to do proper classification using linear method is by proceeding a linear discriminant analysis.

Once again, we are trying to estimate $P(G|X)$ for optimal classification. Suppose $f_k(x)$ is the density of X conditional on the $G = k$ and suppose π_k be the prior probability of class k. We know that :

$$P(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{i=1}^{K} f_i(x)\pi_i}.$$

We notice here that regarding the ability to classify having the $f_k(x)$ is sufficient. Many classification techniques are based on this result and use different models for the densities of X conditional on the class. In the section we will only discuss the linear technique which is referred as linear discriminant analysis (LDA).

Let's suppose that each class density are a multivariate Gaussian density :

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)\Sigma_k^{-1}(x-\mu_k)\right)$$

Now assume the special case when all the classes have a common covariance matrix $\Sigma_k = \Sigma \forall k$. This will lead in a log-ratio of the posterior probabilities that is linear in x :

$$\begin{aligned}\log\frac{P(G=k|X=x)}{P(G=l|X=x)} &= \log\frac{f_k(x)}{f_l(x)} + \log\frac{\pi_k}{\pi_l} \\ &= \log\frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k+\mu_l)^T\Sigma^{-1}(\mu_k-\mu_l) + x^T\Sigma^{-1}(\mu_k-\mu_l).\end{aligned}$$

This way we can build a hyperplane that act as decision boundary between any pair of classes. From this equation we build the *linear discriminant function* :

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

and define the decision rule to be : $G(x) = \text{argmax}_k\delta_k(x)$. Finally notice that we'll have to estimate these various unknown parameter.

$$\begin{aligned}\hat{\pi}_k &= N_k/N \\ \hat{\mu}_k &= \sum_{\{i|g_i=k\}} x_i/N_k \\ \hat{\Sigma} &= \sum_{k=1}^{K}\sum_{\{i|g_i=k\}}(x_i-\mu)k)(x_i-\mu_k)^T/(N-K)\end{aligned}$$

There exist multiple way to refine that technique but we'll stop here for now. Notice that we will also define many non-linear method for classification later on.

2.4 Tree-based methods

2.4.1 Introduction

Tree-based methods are a natural follow up from classical linear method as they are based on a linear basis expansion. The main concept is to split the feature space into small boxes in which the response variable is quiet similar. We must therefore be able to decide how should we form these regions in order to optimize the inference. Now assuming we've managed to partitioned the feature space into five regions R_1, R_2, R_3, R_4, R_5, this model would look like :

$$\hat{f}(X) = \sum_{m=1}^{5} c_m I\{X \in R_m\}$$

which is exactly the kind of model we've defined in section 2.3.4. If we use mean squared as our error measurement then c_m will be the mean of the response variable in that region; $c_m = \text{ave}(y_i|x_i \in R_m)$.

A natural way to form these regions would be to perform successive binary partition in the space of explanatory variables. Because of this, Tree-based methods are sometimes named Recursive Partitioning Analysis (RPA). By proceeding that way, we will be able to represent each regions as terminal nodes in a tree which will create an easy to understand decision methods in order to predict the value of a new data point. This popular method for tree-based regression and classification is called CART (Classification And Regression Tree). This is of course the reason why we classify these methods under the category of *tree-based methods* even though we will later learn about methods that do partitioned the features spaces without creating an associated decision tree.

The following figure represent a partition of the feature space and an associated decision tree :

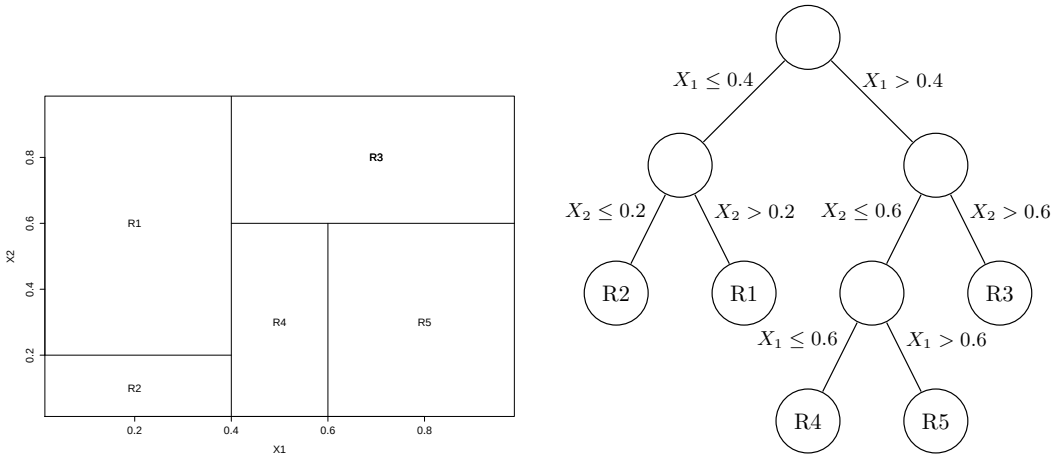

Figure 2: Decision tree

2.4.2 Fitting a decision tree

Let's now discuss how shall we build a decision tree in order to solve a regression problem. Suppose our data consist of N observations consisting of one response and p inputs. Now suppose that we partition the feature space into M region then the linear model we propose is :

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

Once again note that the average over the region m is the choice of $\hat{c}_m$ that will minimize the error if we use the sum of squares as our error measurement. Now the difficult part is to find the best binary partition. By best partition we mean the partition that minimize our error measurement across the data set. Starting with all the data we must choose the

variable j on which we will perform a split and a split point s. This process will result in creating two region $R_1(j,s) = \{X|X_j \leq s\}$ and $R_2(j,s) = \{X|X_j > s\}$. The goal would be to find j and s that solve :

$$\min_{j,s} \left[\sum_{x_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{c}_2)^2 \right]$$

where $\hat{c}_1 = \text{ave}(y_i|x_i \in R_1(j,s))$ and $\hat{c}_2 = \text{ave}(y_i|x_i \in R_2(j,s))$. To solve this problem we will basically try every possible solution since there is a finite of them. For a fixed variable j, since there is N observations there is $N-1$ possible splits and we can scan through all of the splits to find the optimal one. Then we can scan through all the p inputs in order to find the minimizer of the above equation which will result in the best pairs (j,s) according to our data set. We then repeat the process on each of the resulting region separately.

The next problem we have to take care of is how large we should grow our tree. Of course if we build regions that only contains one observations each our error would be precisely 0 but that would grotesquely overfit the data. The generic strategy consist of growing a large tree T_0 where we stop the splitting process as soon as we reach some minimum node size. Then we *prune* the large tree. This process is done by adding a cost for complexity and building a new minimization criterion. Hatie & al. [3] use the following tree pruning model. Let $|T|$ be the number of terminal nodes and let's define

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

Using these we can now define a more refined criterion that penalize huge trees :

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

In this equation, α serves as a tuning parameter where large values of α creates smaller trees T_α. To create these trees we will successively collapse the internal node that produces the smallest increase in $\sum_{m=1}^{|T|} N_m Q_m$ and continue until we produce a single-node tree. We'll then have a sequence of subtree that contains T_α for any α.

For a K-class classification problem the technique is similar but we have to define a new error measurement. Let

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

be the observed proportion of class k in the region m. We can now define multiple error measurements, among other the Gini index and the deviance :

$$\text{Gini index} : \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$$

$$\text{Deviance} : -\sum_{k=1}^{K} p_{mk} \log(p_{mk})$$

The fitting methods are otherwise the same.

2.4.3 Related models

In this section we will quickly introduce other technique which resolves around partitioning the feature space into regions where the responses variable behave similarly. Even though these are part of the *Tree-based methods* section, most of those methods do not uses binary split of inputs and are therefore impossible to represent as a tree.

We will begin by introducing the patient rule induction method (PRIM) which also consist of finding boxes in the feature space, but is looking for boxes in which the response average if high. Since this technique is somehow looking for maxima it is also called *bump hunting*. We start with one big box containing all the data. The box is then compressed along one face by a small amount and we choose this face according to the one creating the compressed box with the highest mean. The process is then repeat until the compressed box contains some minimum number of data points. The PRIM reverses the process and expand along any edge is such an expansion increases the box mean. In most cases we will use cross-validation in order to choose the optimal box size. We can then eliminates all the data points from that box and repeat the process in order to build a sequence of box with high average response values. This technique only works for regression problems or 2-class classification.

Let's now discuss the hierarchical mixtures of experts (HME) procedure. In this model tree splits are not hard decision but rather soft probabilistic ones. In that precise model, the terminal nodes are called experts and do not represent a prediction but rather a model for the output variable. For a regression problem the Gaussian linear regression model is used and for classification the linear logistic regression model is used. The set of parameters is different from one terminal node to another. The non-terminal nodes are called gating network and have outputs of the form :

$$g_j(x, \gamma_j) = \frac{e^{\gamma_j^T x}}{\sum_{k=1}^{K} e^{\gamma_k^T x}}$$

and probabilistically splits data points among the possible models. The resulting model is a mixture model with the mixture probabilities determined by the gating network models. The parameter estimation is done using the likelihood, commonly an EM algorithm will be used.

2.4.4 Random Forest

Bagging or bootstrap aggregating is a technique that consist of constructing many bootstrap sample at first, then build trees out of these sample and finally average the prediction for a regression case or build a committee of trees that caste vote for a classification problem. By proceeding that way we will greatly reduce the variance of the estimate. Even though this procedure is efficient enough on its own the idea behind Random Forest is the build a large collection of *de-correlated* trees in order to reduce even more the variance of the resulting estimates.

The average of b i.i.d. random variable with each having the variance σ^2 will be σ^2/b. That being said, if the variables are not independent but they rather have pairwise correlation ρ, the variance of the average is $\rho\sigma^2 + (1-\rho)\sigma^2/b$. As b grows larger the second term will converge to zero but the first term will remain. The concept of random forest is to improve the variance reduction of bagging by reducing the correlation between trees, hence reducing the first term. This is achieved in the tree-growing process by randomly selecting the inputs variables in each tree.

We won't detail the process more than this, but random forest are becoming more and more popular everyday and it is due to the impressive efficiency of that technique.

2.5 Neural Network

2.5.1 Introduction

The Neural Network is a very popular tool right now since it is very efficient at prediction for both the classification and regression problem. Even though it is impossible to interpret the model, its predictive power is so impressive that the model is widely used in many machine learning problem.

A Neural Network can be perceived as a generalization of the basic regression model where we now allow for non-linear combinations of the various inputs. More precisely, in the 1-layer Neural Network we allow for one more layer of non-linear transformation of the linear combination of the inputs. In a multiple layer Neural Network we produce many very complex non-linear functions in order to approximate the output which results in a precise prediction but rather non-understandable relationship between the explanatory variable and the response. Let's explain with more detail what exactly is a Neural Network.

2.5.2 Simple Model

For simplicity we will only look at the K-class classification problem, but know that we could use a similar model for a regression model. We have a size p vector X of inputs and a size K vector Y of outputs where $y_j = 1$ if and only if this observation is part of the jth class and 0 otherwise. Let us begin with the 1-layer Neural Network. We'll define elements of the layer as the following : $Z_m = \sigma(\alpha_m^T X)$ creating a size M layer of derived features Z_m where α_m is a size $p+1$ vector of weights (parameters). As usual, let's define $f_k(X)$ as the estimate for y_k and we'll build these estimate out of the derived features space the following way : $f_k(X) = g_k(\beta_k Z)$ where β_k is a size $M+1$ vector of weights.

σ is named the activation function and notice that if it is the identity function, the model collapse into a basic linear regression where g is the link function. By introducing this non-linear transformation σ, a much more complex relationship between our variables is allowed. Among many possible activation function the most commonly used are the Sigmoid ; $\sigma(w) = \frac{1}{1+\exp(-w)}$,the Tanh ; $\tanh(w) = \frac{\exp(w)-\exp(-w)}{\exp(w)+\exp(-w)}$ and the Rectified Linear Unit ; $\text{ReLU}(w) = \max(0, w)$. Ideally, we would want a differentiable activation function.

2.5.3 Fitting a Neural Network

Now let's discuss here how we are going to do inferences on the multiples unknown parameters we have. First of all, we will denote the complete set of parameters by θ, which consist of $\{\alpha_m; m = 1, 2, ...M\}$ and $\{\beta_k; k = 1, 2, ..., K\}$. Therefore there is $M(p + 1) + K(M + 1)$ parameters to be estimated. Being more naturally approached from a computer science perspective there is no distribution here therefore inference is not build upon maximising some kind of likelihood. For the basic problem we need first to define a measure of error and we will fix our parameter by minimizing the error.

As usual any error function could do, for regression we commonly use the sum-of-squared errors :

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2,$$

and for classification cross-entropy is commonly used :

$$R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log(f_k(x_i)).$$

To begin, notice that these is clearly no hope of finding an analytical solution to the equation $\nabla R(\theta) = 0$ therefore we will have to resort to iterative numerical procedures. θ will then be estimated by minimizing the error term with a version of the gradient descent algorithm that has been conceived especially for Neural Networks that consist of two parts, a forward pass where we perform inference and a backward pass where we perform learning. The technique is called back-propagation.

The general idea consists of computing first $f_k(x), \forall k$ from which we can compute the error $R(\theta)$, this is the forward pass. Then, from these errors we can compute $\frac{\partial R}{\partial \beta}$ and $\frac{\partial R}{\partial \alpha}$ to optimize via gradient descent the weights, this is the backward pass. Let's explain both part with a bit more details.

We start by initializing the weights somehow. With the weights being fixed, the forward pass consist of starting from the vector X and going forward through the network until reaching $f_k(X)$. More precisely, during the forward part of the algorithm you compute $f_k(X), \forall k$ by using the fact that $f_k(X) = g_k(\beta_k^T Z)$ where $Z_m = \sigma(\alpha_m^T X)$. Now that we've computed $f_k(X), \forall k$ we can proceed at calculating the error term : $R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2$, for instance.

Now that we've move forward through the network in order to compute the error term, we're going to move backward through the network in order to compute derivatives, using the chain rule, that are needed to estimate the weights that minimizes the error term. We perceive this as going backward because we uses $f_k(X)$ to compute first the error term, then we'll move backward through the first layer in order to compute $\frac{\partial R_i}{\partial \beta_{km}}$ and then proceed to estimate β_{km} by doing one iteration of a gradient descent algorithm. Once $\{\beta_k; k = 1, 2, ..., K\}$ have been estimated that way, we'll keep moving backward in the network in order to compute $\frac{\partial R_i}{\partial \alpha_{ml}}$ using the chain rule and the proceed to estimate α_{ml} by doing one iteration of a gradient descent algorithm.

With these new weights we could start over from the forward pass until a desired level of convergence. This two-pass procedure is what is known as back-propagation, it was also named the delta rule.

2.5.4 Deep Neural Network

Now that we've introduced the idea of a simple 1-layer neural Network it is important to quickly discuss the idea of a deeper neural net. We could generalize the concept we've introduced previously by simply adding more hidden layers. The definition and computation of this layer would be done exactly like the first layer was created. Inference would also work as previously define; the back-propagation algorithm would work similarly with the forward step having to pass through one more layer before obtaining $f_k(X)$ and the the backward step would need to evaluate an entire new set of derivative for the new layer.

Deep neural networks are very popular in image recognition and have proven to be quiet efficient at it winning most of the competitions in that field in the last few years. That being said it is still not clear how deep and how wide neural Networks should be. Ba [1] discuss the idea that very wide neural network could be as efficient as very deep network. We also know that overfitting could be a problem in Neural Network.

In conclusion, it seems that multiple layer Neural Network are built and work similarly as 1-layer neural network but we also know that for all of these network the question of the optimal number of layers and the size of each of these layers is left unanswered.

2.6 Kernel based methods

3 Unsupervised Learning

3.1 Introduction

Remember that in Unsupervised Learning there are no such things as predictors and response variable. We will not impose or force a certain structure on the data set, but we are still trying to observe and recognize patterns.

We can use such methods in order to reduce the dimensionality of a data set before applying more conventional supervised learning methods. We could also be interested in finding clusters of data, which is in a way trying to split the data into some sort of natural groups. Another important objective is to be able to model data density. Finally Unsupervised learning can also provides us with tools in order to find hidden causes.

3.2 Clustering

3.2.1 Introduction

Cluster analysis is based of the idea of forming groups of observations that are more similar to each other than to those in other groups. Mathematically speaking, suppose we have p variables and n observations and we are trying to form natural groups. We will use all of our p variables and try to identify data points that are significantly close to one another in the feature space and call these points a group.

In cluster analysis we are interested in the number of groups that belong in our data set, identifying the group that each data point belong to and predicting the group of a potential observation.

3.2.2 K-means algorithm

Let's begin by one of the main usage of Unsupervised learning, clustering. Here, we are going to consider the problem of identifying groups, or clusters, of data points in a multidimensional space. On intuitive approach that leads to an efficient algorithm is the K-means algorithm. Suppose we have a set of n d-dimensional quantitative variables $\mathbf{x}_i$, $i \in \{1, ..., n\}$ and suppose that they seem to naturally form K different clusters. This can be somehow perceived as a classification problem, we are trying to form the right groups, or precisely identify the right cluster for every single points. The idea behind that technique is to find a set of K middle points representing what is the center of mass of each of these clusters. We would ideally place that center of mass at a point that minimize some kind of sum of distance between each point belonging to a cluster and the center of that cluster.

Remember that we are actually trying to form groups among our variables, so let's define $r_{ik} \in \{0, 1\}$ such that $r_{ik} = 1$ if and only if $\mathbf{x}_i$ belong to the cluster k. Therefore, we want to both optimizing the position of this center of mass, m_k and the assignment r_{ik} of every data point such that we do have the center of mass and we do identify well what are the elements composing these natural groups. In other words with are trying to find :

$$\min_{\{m\},\{r\}} \sum_{i=1}^{N} \sum_{k=1}^{K} r_{ik} d(m_k, \mathbf{x}_i),$$

where d is any distance. From now on we will use the following : $||m_k - \mathbf{x}_i||^2$. Since it is impossible to find a global minimum, we will use an iterative algorithm that minimize both parts separately. Notice that it is easy to do the assignments when we know all the center points m_k by simply assigning $\mathbf{x}_i$ to the cluster that has the closest center point. It is also easy to compute the cluster center point that minimize the above equation when the assignments r_{ik} are fixed by simply taking the means of the $\mathbf{x}_i$ that belongs to that precise cluster since the mean minimizes the quadratic distance we are using.

We can therefore define an algorithm in order to solve this clustering problem :

1) Randomly initialize the k means m_k.

2) Compute the assignments that minimizes the errors :

$$r_{ik} = \begin{cases} 1 & \text{if } k = \underset{j}{\operatorname{argmin}} ||\mathbf{x}_i - m_j||^2 \\ 0 & \text{otherwise.} \end{cases}$$

3) Using those assignments, compute the new means :

$$m_k = \frac{\sum_i r_{ik} \mathbf{x}_i}{\sum_i r_{ik}}$$

4) Return to step 2) and repeat until a desired level of convergence is attained.

The figure 3 demonstrate an example of how the k-means algorithm works on a sample of 2 dimensional observations divided in 3 clusters. The first plot shows the data set. In the second image we initialize the 3 random means. The third graphic shows the original assignments and then the forth plot shows the new means based on this first assignments. The fifth graphic represents the final position of the means after 100 iterations while the sixth graph represents the final assignment.

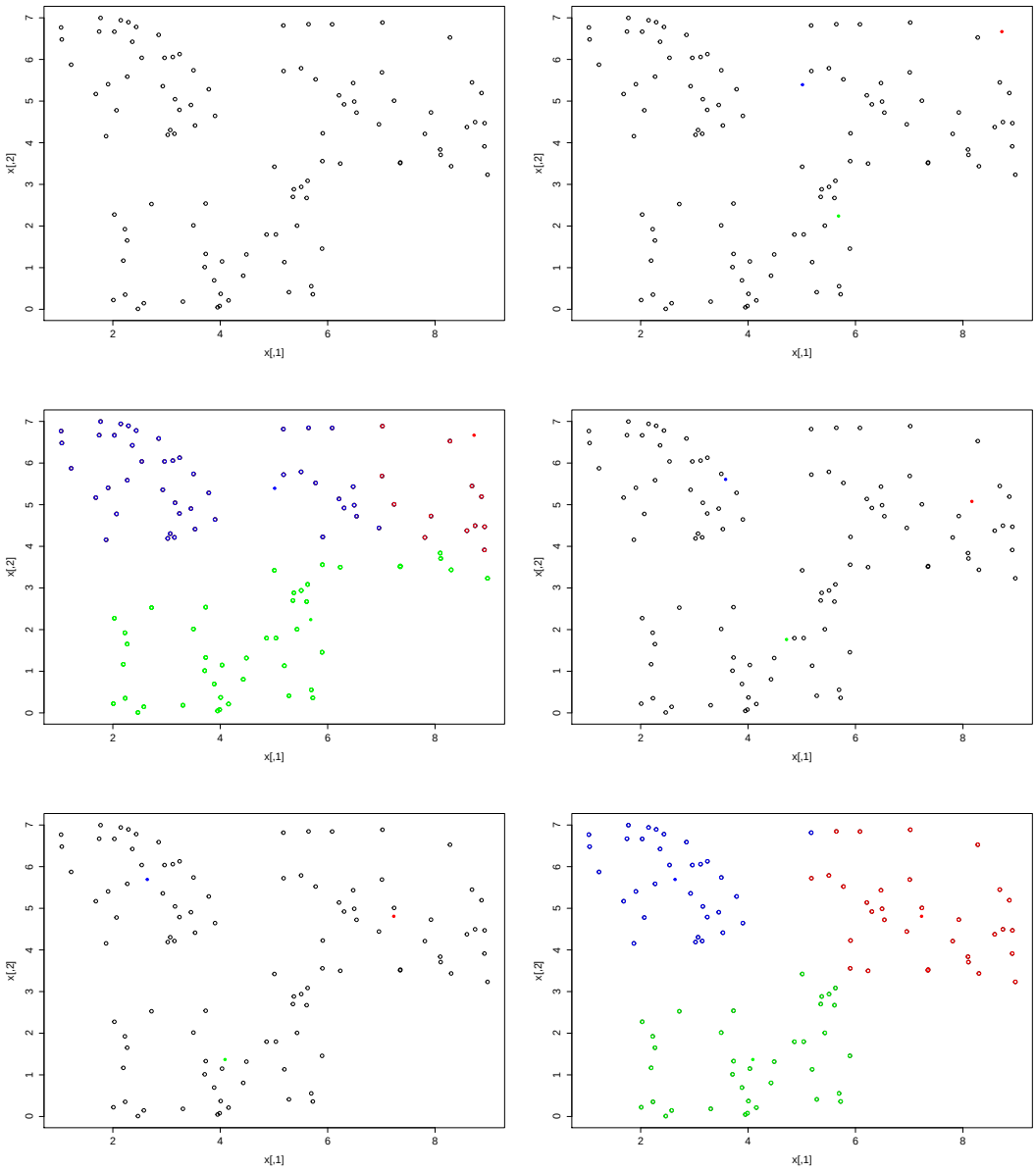

Figure 3: K-means algorithm

3.2.3 Gaussian Mixture Model

A more statistical approach would be to try to solve this clustering problem using a more parametric approach. We will use Gaussian Mixture Model to explain the various clusters. By adding a density assumption we will be able to define likelihood enabling us to solve the problem totally differently and to use tools like AIC and BIC to try to find the optimal number of clusters for example.

Let us now define more clearly the problem we are trying to solve. Once again we are trying to build clusters or natural groups of data. The difference here is that we will assume

that every cluster represents observations from a particular distribution and that the entire data set consist of a mixture model :

$$p(x) = \sum_{k=1}^{K} \pi_k f_k(x)$$

with π_k the mixing coefficients, where :

$$\sum_{k=1}^{K} \pi_k = 1 \text{ and } \pi_k \geq 0 \forall k$$

The problem consist now of estimating the various parameters of the respective densities $f_k(x)$. Even though the following procedure could be done with any density, we will assume for now that we are actually looking at a Gaussian Mixture Model (GMM) i.e. $f_k(x) = N(\mu_k, \Sigma_k)$.

As we are now dealing with densities, maximizing the likelihood seems like an appropriate solution in order ton find the optimal parameters :

$$\begin{aligned}
\log P(X|\pi, \mu, \Sigma) &= \log \prod_{i=1}^{N} P(x_i|\pi, \mu, \Sigma) \\
&= \sum_{i=1}^{N} \log P(x_i|\pi, \mu, \Sigma) \\
&= \sum_{i=1}^{N} \log \sum_{k=1}^{K} \pi_k N(x_i|\mu_k, \Sigma_k).
\end{aligned}$$

Notice that it is impossible to maximize analytically this value with respect to $\{\pi_k, \mu_k, \Sigma_k\}$. Like for the k-means algorithm we will proceed via a step-wise maximisation process. The trick here is to introduce a latent variable z_i which represents the assignments of x_i, in other words, z_i represent which Gaussian generated the observation x_i. Let us define z such that $p(z = k) = \pi_k$, then :

$$\begin{aligned}
p(x) &= \sum_{k=1}^{K} p(x, z = k) \\
&= \sum_{k=1}^{K} p(z = k) p(x|z = k) \\
&= \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)
\end{aligned}$$

The commonly use solution to solve this maximization problem is the Expectation-Maximization (EM) algorithm. The Expectation step consist of computing the posterior probability that each Normal distribution generates each data point, from which we'll build an approximation for z, and therefore we'll assign probabilistically each data points to one of the Gaussian. During the Maximisation step you assume that the data is generated the way computed during the previous step then estimates the various parameters of the Normal distributions using maximum Likelihood estimators.

The algorithm looks like the following :

1) Initialize the means μ_k, the covariances Σ_k and the mixing coefficients π_k.

2) Evaluate the responsibilities :

$$\gamma_{nk} = p(z_n = k|x_z) = \frac{\pi_k N(x_n|\mu_k, \Sigma_k}{\sum_{i=1}^{K} \pi_i N(x_n|\mu_i, \Sigma_i}$$

3) Estimate the parameters using the responsibilies :

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} x_n$$
$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k)(x_n - \mu_k)^T$$
$$\pi_k = \frac{N_k}{N}$$

where $N_k = \sum_{n=1}^{N} \gamma_{nk}$.

4) Repeat from step 2) until a desire level of convergence.

Picture 4 shows how the EM algorithm behaves. We've only drawn the means through out the iterations in the following plots. The first one represents the data set while the second images shows where we initialized our means. The 3 following pictures represents where is located the means for the 5th, 10th and 50th iterations. Finally the very last plot shows how the mean evolved during the 50 iteration of the algorithm before stabilizing near the true mean of the Gaussian distributions that generated the data.

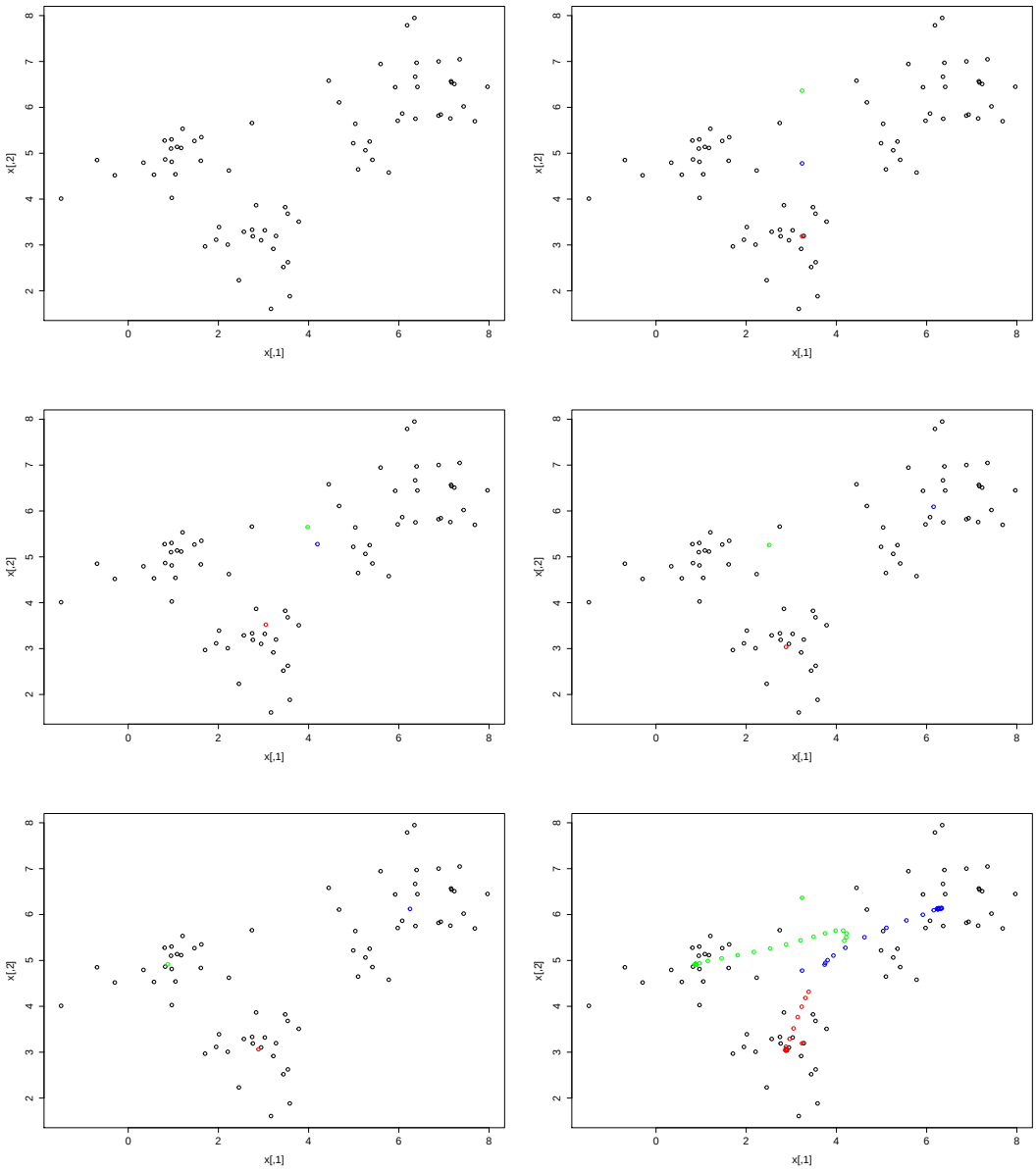

Figure 4: EM algorithm

The cluster analysis is another great problem where the differences between a computer science approach and a statistical approach are easy to observe. The k-means algorithm did not require us to assume anything about how the data is distributed, it defined an error measurement and minimized it. The result are quiet precise and the model seems powerful for a model that is totally assumption free. That being said, the classifications was rather strict and it let little place to doubt and nuance. Without any distribution assumption there is no such thing such as variance and points that were really close to getting assigned to another group are unknown to the researcher.

Using Gaussian mixture models forces us to assume a particular distribution for the data

but the classification is now explained by probability and gives us much more flexibility. We can use probability as a measure of our belief that a point comes from a particular cluster. Also, by using a parametric model, we could now use tools like that AIC or the BIC to select the best number of clusters that is necessary to explain the data set behaviour.

3.3 Principal Components Analysis

3.3.1 Introduction

When we've introduced the Unsupervised learning section we've said that dimensionality reduction is one of the main uses of the unsupervised approached. Among the many dimensionality-reduction methods, the Principal Components Analysis (PCA) is the most popular instance. Since high-dimensional data can be hard to handle for some classifier, it is useful to apply a dimensionality-reduction method, such as PCA, on some data set for visualization or preprocessing purpose.

In order to reduce the dimension of the data set, PCA will project the data to a much lower dimensional space. Even though we are trying to greatly reduce the dimension of our data set we still need to detect differences among our data points, this is why PCA tries to subsequently project the data on the direction in space with the highest variance.

3.3.2 Performing a principal components analysis

Suppose we have N observations $\{x_n\}_{n=1}^N$ of D dimensions, i.e. $x_i \in \mathbb{R}^D$. The purpose of PCA is to reduction the dimensionality of the data set by projecting to a much lower dimensional space $M << D$:

$$x \approx Uz + a$$

where U is a $D \times M$ matrix and z_i a M-dimensional variable.

Since we are trying to project onto a space that catches the difference among the data point the idea we will need the covariance matrix C of the data set. Remember that :

$$C = \frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})(x_n - \bar{X})^T$$

and that it is possible to build a spectral decomposition of a covariance matrix :

$$C = U \Sigma U^T.$$

where U are the eigenvector and Σ is a matrix with eigenvalues on the diagonal which represents the variance of the data in the direction of the associated eigenvector. By selecting the M biggest eigenvalues and associated eigenvector we will select the subset of orientations of size M that best represents the total variability in the data set. We will then project x onto this subspace :

$$z = U_{1:M}^T x.$$

z is now our new data set that lives on a lower dimension space.

References

[1] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2654–2662. Curran Associates, Inc., 2014.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2 edition, 2009.

[4] Y. Yang. Can the strengths of aic and bic be shared? *Biometrika*, 92(4):937–950, 2005.