Consider a logit model as in the previous section with the threshold $l = 0.5$. In this model, for an example j, we have $y_j = 1$ if $\theta \cdot x_j \geq 0$ and $y_j = 0$ if $\theta \cdot x_j \leq 0$, and the decision boundary was determined by the hyper-plane $\theta \cdot x_j = 0$, where θ is the vector of optimal parameters. In this case, the algorithm does not allow for any margin between the default and non-default cases. The idea of SVM is to build a large margin between positive and negative responses, and for this reason, it is also called the Large Margin Classifiers.

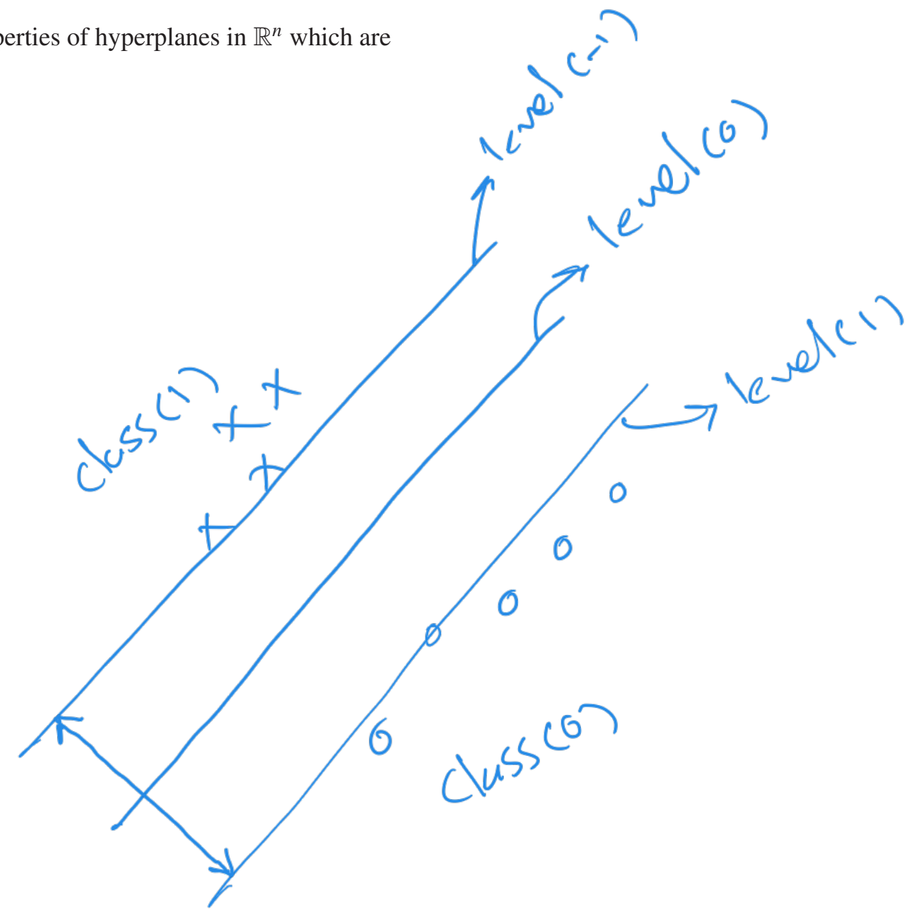
Consider the case when the two classes of default and non-default are completely separable by a hyper-plane. Heuristically, we predict $y = 1$ if $\theta \cdot x_j \geq 0$ but with a bigger margin, i.e. something as follows:

If $y_j = 1$, we would like to have $\theta \cdot x_j \geq 1$

and

If $y_j = 0$, we would like to have $\theta \cdot x_j \leq -1$.

Explaining the algorithm requires some geometrical properties of hyperplanes in $\mathbb{R}^n$ which are reviewed below:



$$\begin{aligned} \max_{\theta} \quad & \frac{2}{||\theta||} \\ \text{s.t.} \quad & \theta \cdot x_j \geq 1 \quad \text{if } y_j = 1 \\ & \theta \cdot x_j \leq -1 \quad \text{if } y_j = 0 \end{aligned}$$

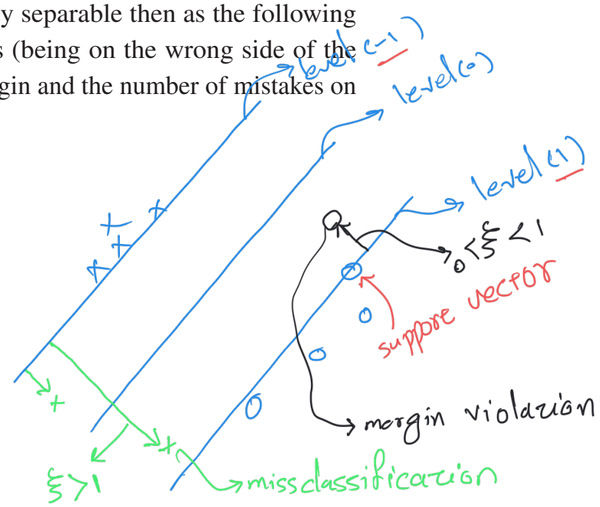
This can be equivalently written as

$$\begin{aligned} \min_{\theta} \quad & ||\theta||^2 \\ \text{s.t.} \quad & y_j \theta \cdot x_j + (y_j - 1)\theta \cdot x_j \geq 1 \end{aligned}$$

Some points are worth mentioning regarding this problem:

- This is a quadratic problem with linear constraints.
- This is a convex problem and as already discussed, it admits a unique solution.

If the two sets of default and default-free cases are not fully separable then as the following figure illustrates, the margin violation and miss-classifications (being on the wrong side of the margin) can occur. Note that there is a trade off between the margin and the number of mistakes on the training data. How can this be addressed?



The idea to solve this problem is to make the margin more flexible but make margin violation and miss-classification expensive. The margin can be more flexible by introducing slack variables and a regularization parameter as follows:

$$\begin{aligned} \min_{\theta,\xi_j\geq 0} \quad & ||\theta||^2 + C\sum_{j=1}^{m}\xi_j \\ \text{s.t.} \quad & y_j\theta\cdot x_j + (y_j-1)\theta\cdot x_j \geq 1-\xi_j, \text{ for all } j \end{aligned} \tag{5.5}$$

- This problem is still a convex quadratic optimization that admits a unique minimum.
- Parameter C can be thought of as a regularization parameter. It can be treated as a hyper-parameter and be tuned in cross validation.
- Every constraint can be satisfied if ξ_j are sufficiently large.
- Small C allows some of the constraints to be ignored which leads to possibly Large margins.
- Large C makes it hard to ignore most constraints which leads to narrow margins.

Loss Function

Consider the unique optimal point of (5.5). Since at the optimal point (which is guaranteed to exist due to the convexity of the problem), we have $\xi_j \geq 0$, then the constraint can be actually written as

$$\xi_j = \max(1-y_j\theta\cdot x_j-(y_j-1)\theta\cdot x_j, 0).$$

This last equation is equal to $\xi_j \geq 1 - y_j \; \theta \cdot x_j - (y_j - 1)\,\theta \cdot x_j, \; \xi_j \geq 0$

$$\xi_j = (1-y_j)\max(0, 1+\theta\cdot x_j) + y_j \max(0, 1-\theta\cdot x_j)$$

$$\min_{\theta, \xi_j} \; \|\theta\|^2 + C\sum_{j=1}^{m} \xi_j \quad \text{s.t. } \xi_j \geq \max(0, \alpha)$$

$$\Longrightarrow \min_{\theta} \; \|\theta\|^2 + C\sum_{j=1}^{m} (1-y_j)\max(0, 1+\theta\cdot x_j) + y_j \max(0, 1-\theta\cdot x_j)$$

By substituting this into Equation (5.5), we obtain the following unconstrained problem

$$\min_{\theta} \quad \frac{\|\theta\|^2}{2} + C\sum_{j=1}^{m} [(1-y_j)\max(0, 1+\theta\cdot x_j) + y_j \max(0, 1-\theta\cdot x_j)]. \qquad C>0$$

By dividing the objective function by 1/2 and rescaling C (i.e. changing C to $2C$), from the last equation, we obtain:

$$\Longrightarrow \min_{\theta} \left[\frac{\|\theta\|^2}{2} + C\sum_{j=1}^{m} [(1-y_j)\max(0, 1+\theta\cdot x_j) + y_j \max(0, 1-\theta\cdot x_j)] \right] \tag{5.6}$$

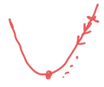
Therefore, from Equation (5.6), the loss function (or the cost function) is equal to

$$L(\theta) = \frac{\|\theta\|^2}{2} + C\sum_{j=1}^{m} \left[(1-y_j)\max(0, 1+\theta\cdot x_j) + y_j \max(0, 1-\theta\cdot x_j)\right].$$

$\min L(\theta)$

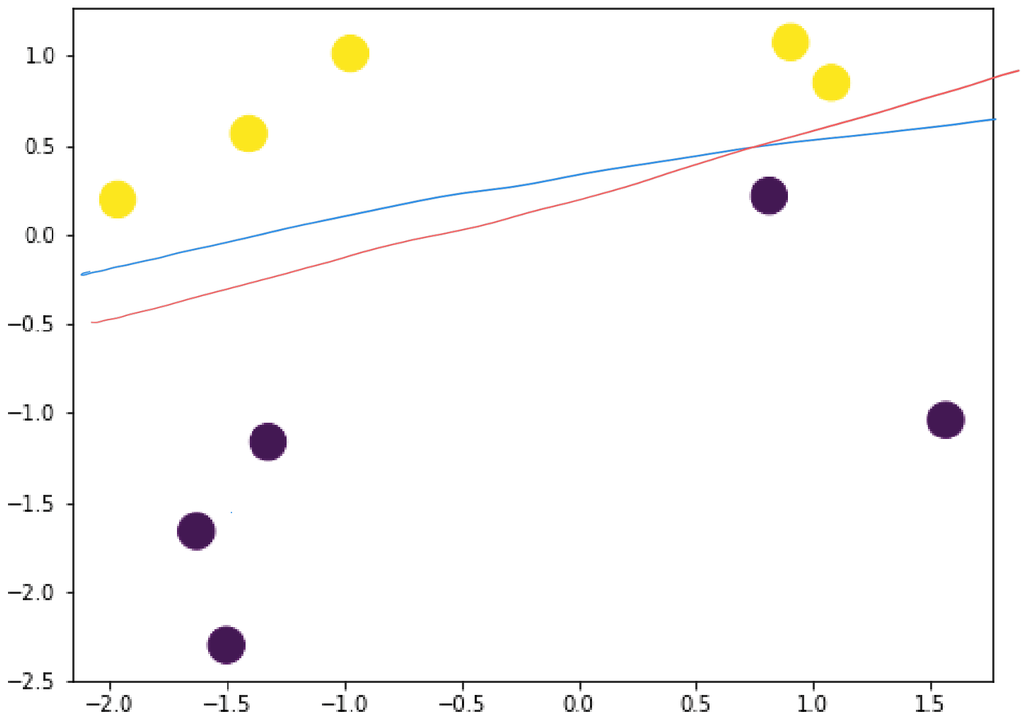
θ

R The cost function of SVM learning is convex and hence the optimization problem (5.6) admits a unique solution. Therefore, by using gradient descent and a starting point, we can be sure that the algorithm is convergent to the global optimal point.

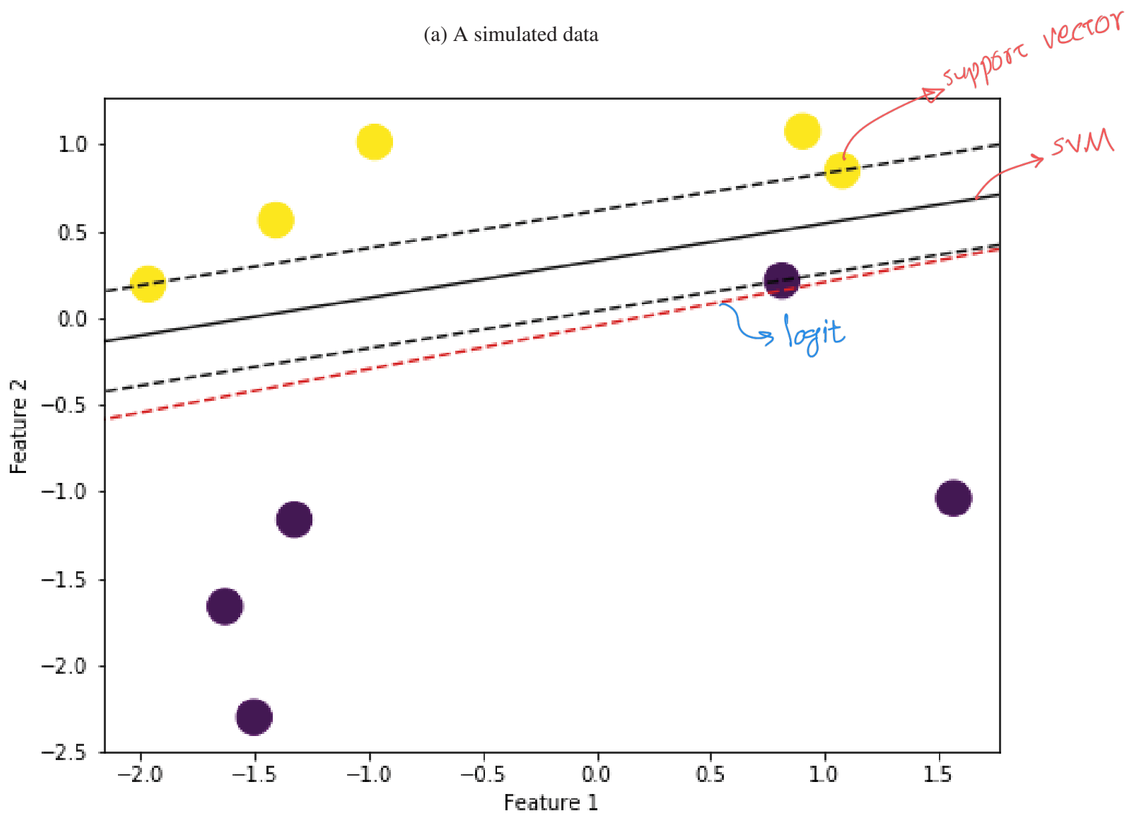


R Compare this loss function with the one from the logistic regression, what do you observe?

■ **Example 5.8** Figure 5.2 provides a comparison between the decision boundaries of the Logit and SVM models. ■



(a) A simulated data



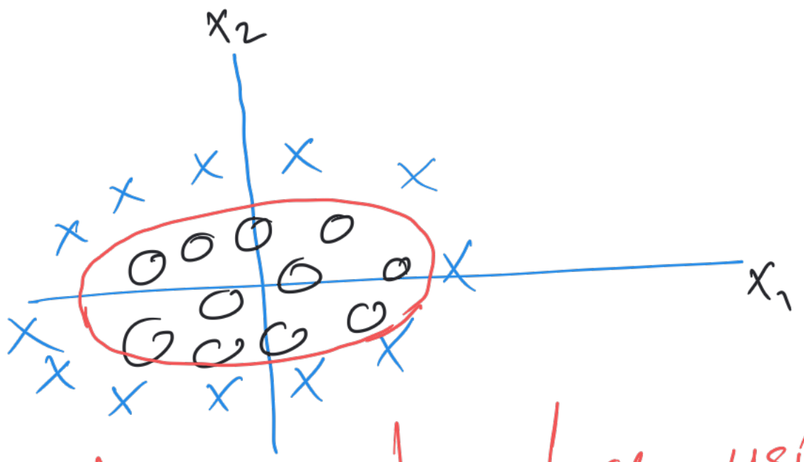
(b) Logistic vs SVM decision boundaries

Figure 5.2: A comparison of decision boundaries of SVM and Logit models

W6L2

5.2.3 Kernel Methods and Support Vector Machine Learning

The decision boundaries obtained in the previous section are linear. However, some data would not obey a linear decision boundary as it is shown by the following graph:



Non-linear decision boundary using logistic regression

$h_\theta(x) = \text{sigmoid}(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

predict $y=1$ if $h_\theta(x) \geq 0.5 \iff \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 \geq 0$

" $y=0$ if $h_\theta(x) < 0.5 \iff \underbrace{\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2}_{} < 0$

after estimating θ_i, $i = 0, 1, 2, 3, 4$, it leads to a non-linear decision boundary

Kernels are functions that can help us producing ~~non-linear~~ decision boundaries. In general, there are four types of Kernels:

- Linear: This is basically the kernel that was used in the previous section leading to a linear decision boundary.
- Polynomial. If instead of $\theta \cdot x_j$, we use $(\theta \cdot x_j + r)^d$, where r is a constant and $d \geq 2$ is a positive integer. Then the decision boundary will not be linear. This defines the polynomial kernel.
- Gaussian or Exponential Kernel. Since this is an important kernel, we provide the kernel implementation in detail for this case. We recall the training dataset $\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}$.

Definition 5.2.1 For any two points x and $\tilde{x}$ in the feature space, the similarity of x and $\tilde{x}$ is defined by:

$$\text{smlt}(x,\tilde{x}) = e^{-\lambda(||x-\tilde{x}||^2)}$$

where $\lambda > 0$ is a constant. This similarty function is also called the Gaussian kernel as it is based on a Gaussian density.

Note the following two properties:
if x is very close to $\tilde{x}$ then $\text{smlt}(x,\tilde{x}) \approx$

and if x is very far from $\tilde{x}$ then $\text{smlt}(x,\tilde{x}) \approx$

The Gaussian kernel algorithm is as follows:

- For each training example x_j, we choose a landmark shown by l_j. The landmark l_j is chosen in such a way that $\text{smlt}(l_j,x_j)$ is close to one. For instance, l_j can be chosen exactly as x_j. In what follows, this is how the landmark is chosen for us.
- For a given example (x_k,y_k), by using the land marks $(l_1,l_2,...,l_m)$, we obtain a new set of features by using the corresponding similarity functions. More precisely, the feature $x_k = (x_{k0},x_{k1},...,x_{kn})$ now transforms to

$$f^{(k)} = (f_0^{(k)},f_1^{(k)},....,f_m^{(k)}),$$

where $f_i^{(k)} = \text{smlt}(x_k,l_i)$, $i = 0,1,...,m$.

- To obtain the optimal parameter Θ corresponding to the new features, we use the same loss function as SVM:

$$\min_{\Theta} \quad \frac{||\Theta||^2}{2} + C\sum_{j=1}^{m}[(1-y_j)\max(0,1+\Theta\cdot f^{(j)}) + y_j \max(0,1-\Theta\cdot f^{(j)})].$$

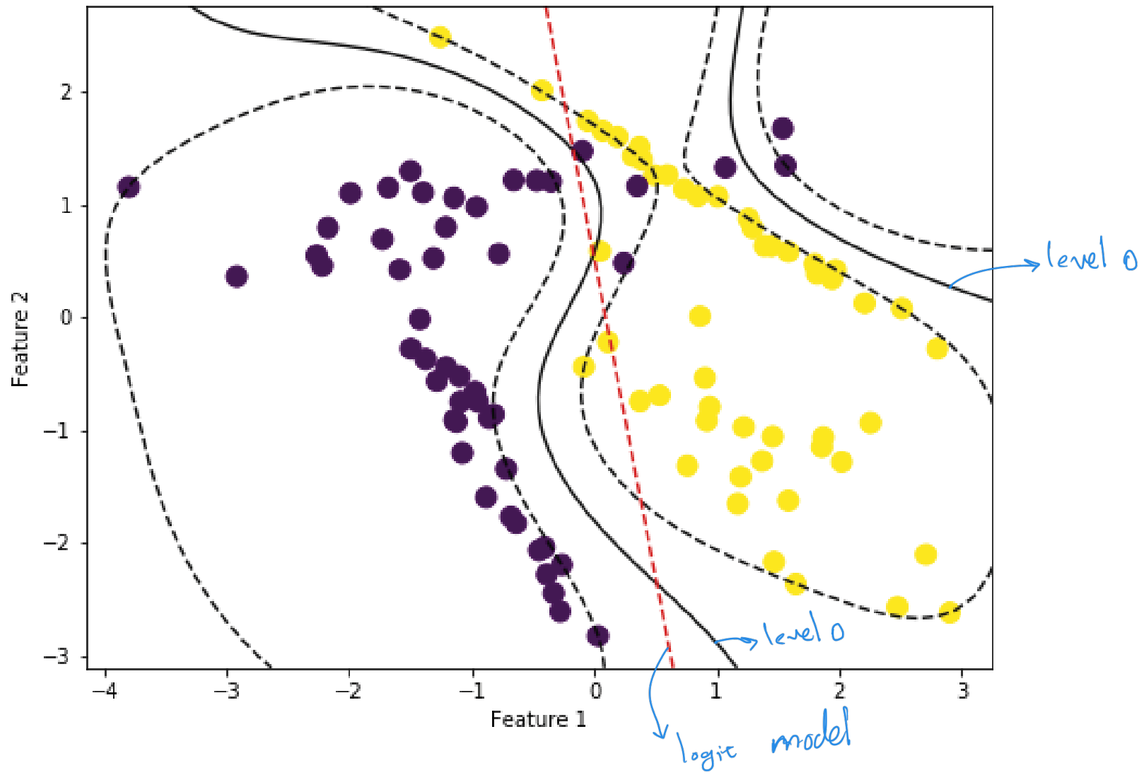


Figure 5.3: Counter plots of a comparison of decision boundaries of SVM with a Gaussian kernel and Logit models

  - Finally, after the training and obtaining the optimal parameter Θ, for an unseen feature x, first compute new features f, and then make the prediction $y = 1$, if $\Theta \cdot f \geq 0$, and prediction $y = 0$ otherwise.

- Similarly, other types of kernels with similar algorithm can be constructed such as Polynomial, Chi-Square kernel, Sigmoid, etc.

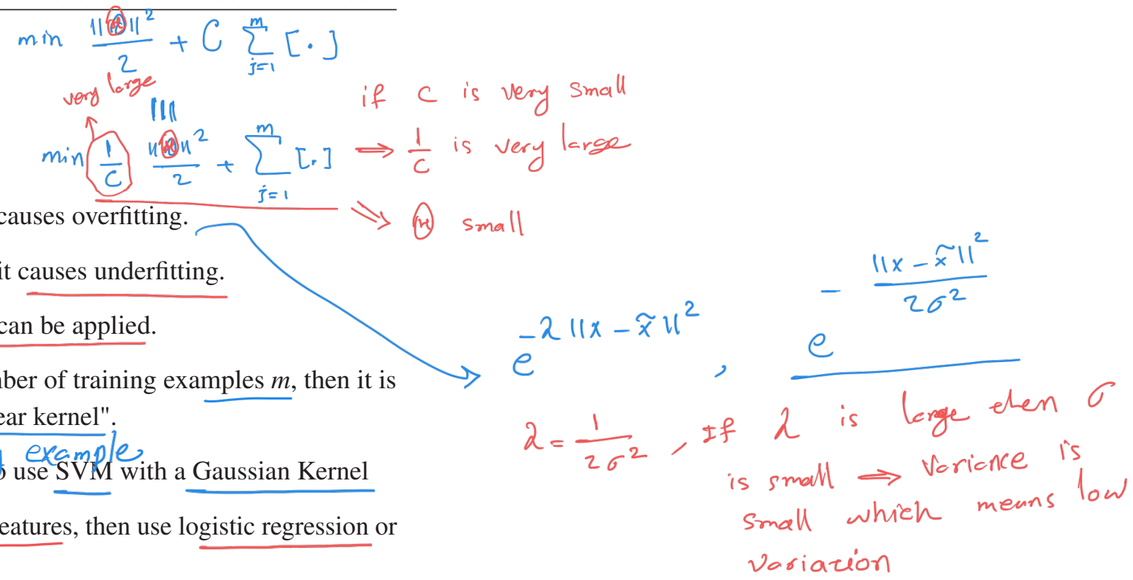
Figure 5.3 compares the decision boundaries of SVM with a Gaussian kernel with that of Logit model when there are only two features.

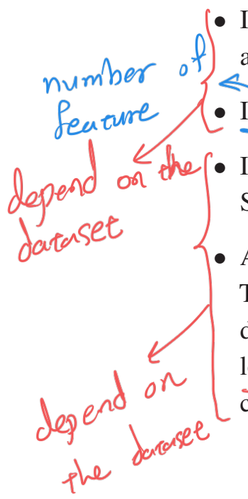
R Note that unlike the Logit model, the prediction function (or hypothesis) of SVM cannot be interpreted as probability. It is purely a deterministic classification function with binary outcomes.

Advise in Applying SVM and Kernels:

- Although in theory one can use any function as a kernel, the choice is actually limited as the convergence of the optimization problem of SVM packages might not be guaranteed.

- Perform feature scaling before using Gaussian kernel.
- If C is very small, the model becomes underfitted.
- If C is very large, the model becomes overfitted.
- If λ is large, the new features' variation is low, and it causes overfitting.
- If λ is small, the new features' variation is hight, and it causes underfitting.
- For multi-class classifications, "one vs all" algorithm can be applied.
- If the number of features n is large relative to the number of training examples m, then it is advised to use logistic regression, or SVM with a "linear kernel".
- If n is small and m is intermediate, then it is advised to use SVM with a Gaussian Kernel
- If n is small and m is large, manually create/add more features, then use logistic regression or SVM without a linear kernel.
- A neural network (which we have not yet studied) with the right architecture could (POTENTIALLY) provide better fitting and prediction than any of these models, but even if it does, neural network still does not have two advantages of SVM, one the convexity of the loss function in linear kernels guaranteeing a unique optimal point, and second, the speed of convergence.





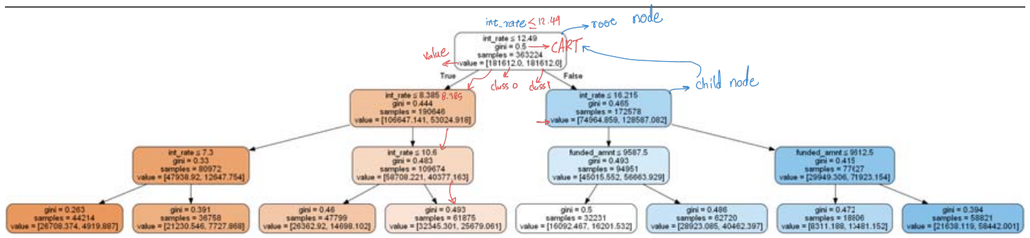


Figure 5.4: An example of a tree model constructed using the P2P dataset

W6L3

5.3 Decision Tree Models

In SVM models and GLM models (such as logistic regression), we could successfully capture some non-linearity in the model and express a dependent random variable based on some features. However, these models cannot tell us much about the interaction between the features or even the features themselves. For instance, let say we want to know which features in a credit card dataset has played a major role in the failure of the card holders. This is something that we can achieve using tree-based models.

Tree based models partition a feature space into certain sub-spaces based on some criteria. They start with a root node and through a (binary) splitting create child nodes. The child nodes (also called internal or split nodes) are then splitted accordingly; this process continues up until the point where no more splitting is feasible based on the pre-specified criteria. These final nodes are called leaf or terminal nodes. Tree models can be used both for regression and classification. Figure 5.4 shows an example of a tree model using P2P platform data with only continuous features.

R So we can either split the data in a binary manner or use a multi-splitting approach. So which method should we choose? The multi-splitting strategy could fragment the data too quickly, causing the loss of information for the child nodes. In addition, the multi-splitting can be still achieved using a binary method; therefore, we choose a binary method in splitting the tree nodes at each stage.

In order to understand tree models conceptually, we consider the following simple example in Algo trading. First note the abbreviations:

Below 50 day Moving Average: BMA

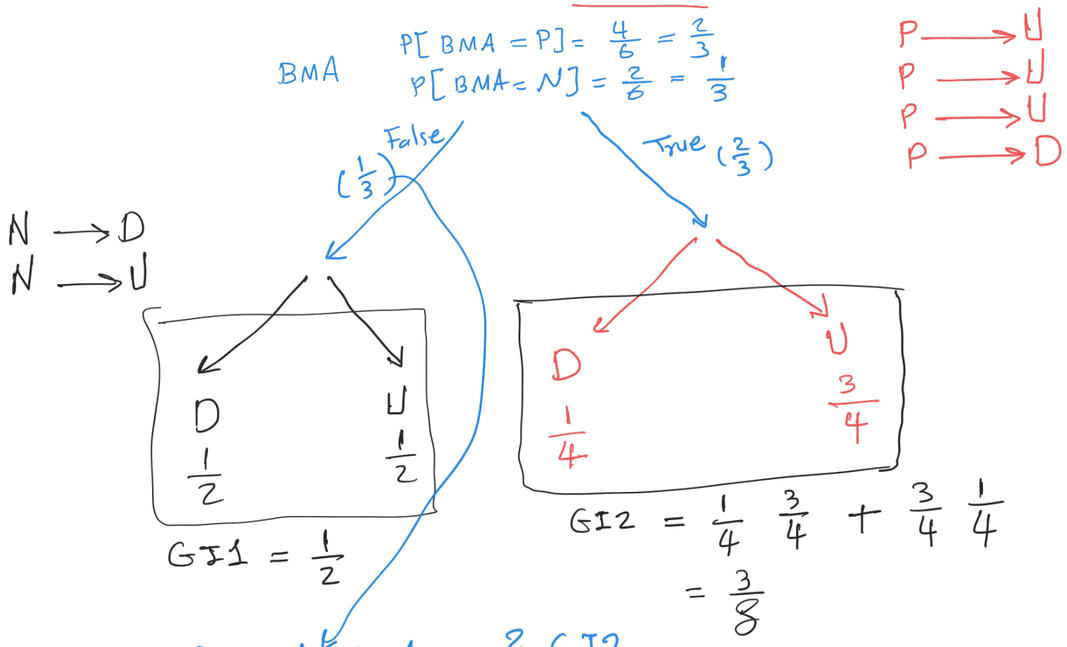
Today's open higher than yesterday's close: OC

Today's trading volume: TV

	BMA	OC	TV	Return
stock #1	Positive (P)	Yes (Y)	H	U
stock #2	P	Y	L	U
3	P	NO (N)	H	U
4	P	Y	H	D
5	Negative (N)	N	H	D
6	N	Y	L	U

features → response
U: up
D: down
dataset

Which root node should we choose? In order to achieve this, we need an optimality criteria and we choose GINI index for this. Gini index is a measure of how often a randomly chosen element would be incorrectly miss-classified based on the distributions of labels in the set. So it is a missclassification measure and it is also called Gini impurity.

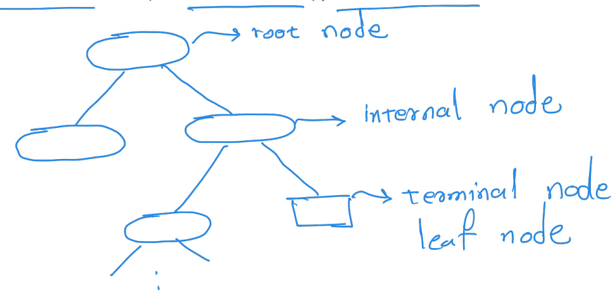


Next, in a similar way, we calculate WGI(OC) and WGI(TV), and among BMA, OC, TV we choose the root node to be the one with the lowest WGI.

W6L4

There are several algorithm and criteria to obtain the optimal tree. In what follows, we use classification and regression trees (CART) method. There are other methodologies such as ID3 and its later versions, C4.5 and C5.0 which we will not discuss.

Terminologies: root node, terminal nodes (or leave nodes), internal node.



Recall our dataset $\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}$ where m is the number of training examples and for $j = 1,2,...,m$, $x_j = (x_{j0},x_{j1},x_{j2},...,x_{jn})$ is the set of features of entity j and y_j is the corresponding response of example j. In the previous section, we have considered $y_j \in \{0,1\}$. However, decision trees can be applied both in regression and multi-class classification. So the response variable can be either continuous or categorical. There are two important questions that any decision tree algorithm needs to answer:

- Which feature to choose at each node?
- For a chosen feature, how the splitting point is determined?

init_rate <= ?

In what follows, we provide a more rigorous insight about this.

Regression Trees

Suppose that there are M regions R_j, $j = 1,2,...,M$, and we model the response as a constant c_j in region R_j, $j = 1,2,...,M$. Then basically, the response is modeled as:

$$\text{response} = y(x) = \sum_{j=1}^{M} c_j 1_{\{x\in R_j\}} = \begin{cases} c_1 & \text{if } x\in R_1 \\ c_2 & \text{if } x\in R_2 \\ \vdots & \\ c_M & \text{if } x\in R_M \end{cases}$$

Given example

If the regions and c_k are known then the above formula could be used to make a prediction for an unseen data x. Suppose that we fix the number of the regions M a priori, i.e. M is a known constant. If $M = 1$, which means that there is only one region R_1, then we can find the best estimator in the region by solving the quadratic problem $\min_c \sum_{i\in R_1}(y_i - c)^2$. The solution of this problem is of course for c to be the average over the region.

However, the whole idea of a tree model is to have multiple regions. One way of characterizing optimal regions for $j = 1,2,...,M$ is as follows:

$$\min_{R_j} \sum_{j=1}^{M} \sum_{i\in R_j} (y_i - \hat{y}_{R_j})^2$$

M is assumed to be known here

where $\hat{y}_{R_j}$ is the estimator for region j. Since a region could have any geometrical shape, obtaining an optimal region without further assumptions becomes infeasible. An easier problem would be to

assume that the regions are high dimensional rectangles or boxes and to minimize $\sum_{j=1}^{M}\sum_{i\in R_j}(y_i-\hat{y}_{R_j})^2$. However, even with this extra assumption and without any further restrictions, this problem is computationally expensive or even infeasible. Hence, we apply the following methodology.

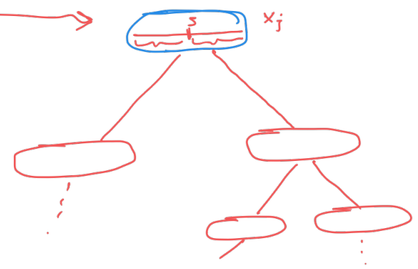
Suppose that we are at a given node, and we want to determine the next split, then we seek the splitting feature j and split value s that solve the following problem:

$$\min_{j,s}\left(\min_{c_1}\sum_{i:\,x_i\in R_1^{j,s}}(y_i-c_1)^2+\min_{c_2}\sum_{i:\,x_i\in R_2^{j,s}}(y_i-c_2)^2\right),\quad R_1^{j,s}=\{x:x_j\le s\}\text{ and }R_2^{j,s}=\{x:x_j>s\}$$

Since for the inner optimization problems, the region is fixed, this problem is equivalent to the following one:

$$\min_{j,s}\left(\sum_{i:\,x_i\in R_1^{j,s}}(y_i-\bar{c}_{R_1^{j,s}})^2+\sum_{i:\,x_i\in R_2^{j,s}}(y_i-\bar{c}_{R_2^{j,s}})^2\right)\tag{5.7}$$

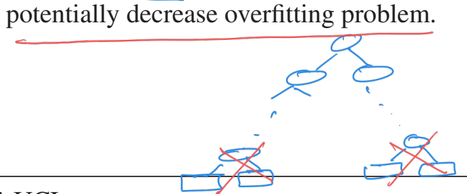
Let us know informally summarize, the above discussions. We start from the root node (the top node) and split the node by the previous optimization problem, hence creating two child nodes. Then we repeat this splitting process for the child nodes. This is a top-down method as we start from the top node, and it is also called recursive binary splitting. The method is also referred as "greedy" because at each node, the tree is developed to other nodes based on the best split at that node which might overlook some good splits in future possible splits that have been ignored, i.e. one current split might not be very optimal, but it could potentially lead to next good nodes. This argument is visualized below:



This tells us about the feature and splitting point but how large should a tree be? The splitting of a new node is continued until a stopping criterion is triggered, for example, we might stop developing the tree if the number of observation in each region has fallen below a specific level.

How to make a prediction? For an unseen data, we use the tree model and find out the region for which this data belongs to; we then use the average of the training observation at that region as a prediction.

Pruning: If no stopping rule is applied, a tree could grow very large which results in a good fit to the training dataset at the cost of a large error on the test dataset, i.e. tree models are prone to overfitting if they are left free to grow. A stopping rule (such as a minimum number of observation in each node) will cease a tree from growing, however, it will not fix the overfitting problem of the tree completely. A smaller tree would have a lower variance at the cost of a higher bias. Also a smaller tree will be also easier to interpret. The question is of course how to obtain a sub-tree from a larger one which could potentially decrease overfitting problem.



Definition 5.3.1 The process of obtaining a sub-tree from a larger one to decrease the variance is called pruning.

Approach 1. One option is to have a prespecified target c_0 and at each split calculate the residual sum of squares (RSS) and stop growing the tree if $RSS > c_0$. The problem with this option is that it might overlook some good splits that could happen at future nodes.

Approach 2. A second approach would be to grow a large tree and then prune it back to an optimal sub-tree. But how can this be achieved? The idea is similar to Lasso regression (l^1-penalty), and we penalize the growth of a tree by using a sort of a regularization parameter. This is done through a **cost complexity pruning** explained using the following procedure. Let $|T|$ denote the number of terminal nodes in the tree. Furthermore, for $k = 1, 2, ..., |T|$, we set:

$$N_k = card(\{x_i \in R_k\})$$

N_k is the total number of elements in $\{x_i \in R_k\}$

where R_k is the region (more precisely rectangle) corresponding to the kth terminal node,

$$\hat{c}_k = \frac{1}{N_k} \sum_{x_i \in R_k} y_i$$

$$Q_k(T) = \frac{1}{N_k} \sum_{x_i \in R_k} (y_i - \hat{c}_k)^2$$

$$C_\alpha(T) = \sum_{k=1}^{|T|} N_k Q_k(T) + \alpha |T|$$

(α: hyperparameter)

$$= \sum_{k=1}^{|T|} \left(\sum_{x_i \in R_k} (y_i - \hat{c}_k)^2 \right) + \alpha |T|$$

$\min_T C_\alpha(T)$ $\quad \alpha \geq 0$

For a fixed α, we can find (via a procedure that we do not explain here) a sub-tree T_α that minimizes $C_\alpha(T)$, i.e. $T_\alpha = \text{argmin}_T C_\alpha(T)$. We can start from $\alpha = 0$ which provides the largest tree, T_0. Then as we increase α, the growth of the tree is penalized, and we obtain trees (sub-trees of T_0) with lesser nodes which are nested sequentially; this process is called cost-complexity pruning.

$T_0 \supset T_{\alpha_1} \supset T_{\alpha_2} \supset \cdots \supset T_{\alpha_k}$

Finally, we need to choose the best α. A cross validation argument can be then used to obtain the optimal α. In tree models, normally K-fold cross-validation is used to choose the best α. The K-fold procedure and a summary of the tree algorithm are explained below, see Gareth et al. (2013).

For a fixed training dataset, the tree algorithm together with the cost complexity pruning follow from the following two steps:

(1) Grow a large tree using the training dataset and a stopping rule. A common stopping rule is to stop only when each terminal nodes has fewer than some pre-specified number of observations.

(2) Use the cost-complexity pruning to obtain a sequence of nested sub-trees from the large tree: $T_0 \supset T_{\alpha_1} \supset T_{\alpha_2} \supset \ldots$.

Choosing α:

- Apply the K-fold cross validation to choose the best α. In order to do so, first divide the training dataset into K folds using random sampling. Then for each $k = 1, 2, ..., K$ do as follows:
  - Repeat Steps 1and 2 on all but the kth fold of the training dataset.
  - Since the kth portion is left out, it can be used to evaluate the mean squared prediction error which is a function of α.

  For each value of α, average the results and pick α that minimizes the average error.
- Return the sub-tree in Step 2 that corresponds to the best chosen α.

In what follows, we visualize the whole procedure using three-fold cross validation: