Chapter 6
Knowledge Extraction from Support Vector Machines

Support Vector Machines have been a promising tool for data mining during these years because of its good performance. However, a main weakness of SVMs is its lack of comprehensibility: people cannot understand what the "optimal hyperplane" means and are unconfident about the prediction especially when they are not the domain experts. In this section we introduce a new method to extract knowledge with a thought inspired by the decision tree algorithm and give a formula to find the optimal attributes for rule extraction. The experimental results will show the efficiency of this method.

6.1 Introduction

Support Vector Machines, which were widely used during these years for data mining tasks, have a main weakness that the generated nonlinear models are typically regarded as incomprehensible black-box models. Lack of comprehensibility makes it difficult to apply in fields such as medical diagnosis and financial data analysis (Martens 2008).

We briefly introduce two fundamental kinds of rules. (Martens 2008): Propositional rule, which is most frequently used, is simple "If … Then … "expressions based on conventional propositional logic; The second is M-of-N rules which usually expressed as "If {at least/exactly/at most} M of the N conditions $(C_1, C_2 \ldots C_N)$ are satisfied Then Class = 1". Most of the existing algorithms extract propositional rules while only little algorithm, such as TREPAN, could extract the second rules. (Martens 2008).

There are several techniques to extract rules from SVMs so far, and one potential method of classifying these rule extraction techniques is in terms of the "translucency", which is of the view taken within the rule extraction method of the underlying classifier. Two main categories of rule extraction methods are known as decompositional and pedagogical (Diederich 2004). Decompositional approach is closely

Y. Shi et al., *Intelligent Knowledge,* SpringerBriefs in Business,
DOI 10.1007/978-3-662-46193-8_6

related to the internal workings of the SVMs and their constructed hyperplane. On the other hand, pedagogical algorithms consider the trained model as a black box and directly extract rules which relate the inputs and outputs of the SVMs.

There are some performance criteria to evaluated the extracted rules, Craven and Shavlik (Craven 1996) listed such five criteria as follows:

1) Comprehensibility: The extent to which extracted representations are humanly comprehensible.
2) Fidelity: The extent to which the extracted representations model the black box from which they were extracted.
3) Accuracy: The ability of extracted representations to make accurate predictions on previously unseen cases.
4) Scalability: The ability of the method to scale to other models with large input spaces and large number of data.
5) Generality: The extent to which the method requires special training regimes or restrictions on the model architecture.

However, the last two are hard to quantize, so we consider the first three criteria only.

First we should introduce coverage to explain accuracy and fidelity better. If the condition (that is, all the attribute tests) in a rule antecedent holds true for a given instance, we say that the rule antecedent is satisfied and the rule covers the instance. Let $n_{\mathrm{cov}ers}$ be the number of instances covered by the rule R and D be the number of instances in the data. Then we can define coverage as:

$$\mathrm{cov}erage(R) = \frac{n_{\mathrm{cov}ers}}{|D|} \tag{6.1}$$

Then we can define accuracy and fidelity easily. Let $n_{correct}$ be the number of instances correctly classified by R and $n_{coincide}$ be the number of instances which prediction by R coincides with prediction by the SVM decision function. We define them as:

$$accuracy(R) = \frac{n_{corrent}}{n_{\mathrm{cov}ers}} \tag{6.2}$$

$$fidelity(R) = \frac{n_{coincide}}{n_{\mathrm{cov}ers}} \tag{6.3}$$

There is not a definition about comprehensibility acknowledged by all. In this paper we define it as the number of attribute tests in rule antecedent in the simplest form, which means if there are two antecedents such as **If** $a_1 > \alpha$ and **If** $a_1 < \beta$ they can be simplified to the form **If** $\alpha < a_1 < \beta$.

However, the major algorithms for rule extraction from SVM have some disadvantages and limitations. There are two main decomposition methods: SVM+Prototypes and Fung. The main drawback of SVM+Prototypes is that the extracted

rules are neither exclusive nor exhaustive which results in conflicting or missing rules for the classification of new data instances. The main disadvantage of Fung is that each of the extracted rules contain all possible input variables in its conditions, making the approach undesirable for larger input spaces as it will extract complex rules lack of interpretability, which is same to SVM + Prototypes. How to solve this problem? Rules extracted from decision tree are of good comprehensibility with remarkably less antecedents as the decision tree is constructed recursively rather than construct all the branches and leaf nodes at the same time. So our basic thought is to integrate the advantage of decision tree with rule extraction methods.

6.2 Decision Tree and Support Vector Machines

6.2.1 Decision Tree

Decision Tree is widely used in predictive model. A decision tree is a recursive structure that contains a combination of internal and leaf nodes. Each internal node specifies a test to be carried out on a single attribute and its branches indicate the possible outcomes of the test. So given an instance for which the associated class label is unknown, the attribute values are tested again the decision tree. A path is traced from the root to a leaf node which holds the class prediction.

A crucial step in decision tree is **splitting criterion**. The splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset. The splitting attribute is determined so that the resulting partitions at each branch are as pure as possible. According to different algorithms of splitting attribute selection people have developed lots of decision tree algorithms such as ID3, C4.5 and CART.

6.2.2 Support Vector Machines

For a classification problem in which the training set is given by

$$T = \{(x_1, y_1), \ldots, (x_l, y_l)\} \in (R^n \times \{-1, 1\})^l, \tag{6.4}$$

where $x_i = (|x_i|_1, \ldots, |x_i|_n)^T \in R^n$ and $y_i \in \{-1, 1\}, i = 1, \ldots, l$, standard C-SVM constructs a convex quadratic programming

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \xi_i, \tag{6.5}$$

$$s.t.\ \ y_i\left((w \cdot x_i) + b\right) \geq 1 - \xi_i, i = 1, \ldots, l, \tag{6.6}$$

$$\xi_i \geq 0, i = 1, \ldots, l, \tag{6.7}$$

where C is the penalty parameter to compromise this conflict of two terms in the objective function.

6.3 Knowledge Extraction from SVMs

6.3.1 Split Index

We need to sort and find the attribute of optimal performance for splitting. There are two methods for this purpose: F-value and RFE (Deng and Tian 2009).

F-value aims at displaying the difference of each attribute. For a certain attribute k, it defines:

$$[x]_k^+ = \frac{1}{l_+} \sum_{y_i=1} [x_i]_k, k = 1, \ldots, n, \tag{6.8}$$

$$[x]_k^- = \frac{1}{l_-} \sum_{y_i=-1} [x_i]_k, k = 1, \ldots, n, \tag{6.9}$$

$$[x]_k = \frac{1}{l} \sum_{i=1}^{l} [x_i]_k, k = 1, \ldots, n, \tag{6.10}$$

and then defines the F-value of attribute k as:

$$F(k) = \frac{([x]_k^+ - [x]_k)^2 + ([x]_k^- - [x]_k)^2}{\frac{1}{l_+ - 1} \sum_{y_{i=1}} ([x_i]_k - [x]_k^+)^2 + \frac{1}{l_- - 1} \sum_{y_{i=-1}} ([x_i]_k - [x]_k^-)^2} \tag{6.11}$$

The numerator reflects the extent of difference between positive and negative points on attribute k while the denominator reflects the extent of variance of positive points and negative points respectively on attribute k. So the larger $F(k)$ is, the better the attribute k could distinguish these two categories.

RFE, which is short for **recursive feature elimination**, delete the attribute with minimal absolute value component of the vector w during each iteration. On the other hand it reveals that the attribute k, which correspond to the maximal absolute value component of w : w_k, is the most important attribute as it changes slightly it could result in the maximal change in the result of decision function.

But two figures as follows show a dilemma that we may not get a desired result while taking each one separately into consideration. Figure 6.1 shows that the attribute x_1 has a maximal w_1 as the gradient of the decision line, but $F(1)$ is too low,

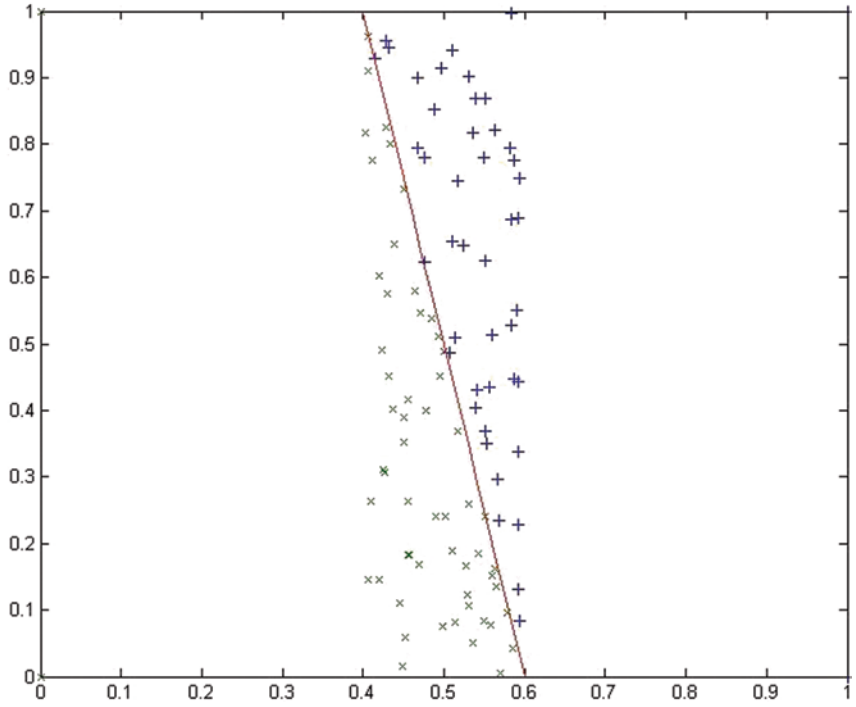

Fig. 6.1 Example of attribute with large w_1 but low $F(1)$

so the attribute x_1 is not a good attribute for splitting. Figure 6.2 shows that x_1 has a large $F(1)$ but a small w_1 and similarly we won't select x_1 as the splitting attribute.

So we could say both F-value and RFE are not always effective and stable and so they are not an excellent criterion to evaluate the splitting capacity.

Here we introduce a new criterion called **Split Index** to balance the effect of these two factors. The Split Index of attribute k could be computed as the formula:

$$SI(k) = F(k) * |w_k| \tag{6.12}$$

It is easy to compute and obviously we should normalize the training data to make sure that all the attributes are under the same condition. We assume that the training data we mentioned later has been normalized.

In order to test the rationality of (6.12) we use it in the two data showed in the figures above. The attribute x_2 has maximal SI value rather than x_1 which has large component w_1 on the first data. When applying to the second data the attribute x_2 has maximal SI value rather than x_1 which has larger $F(1)$. The results are better using SI value for splitting after computation.

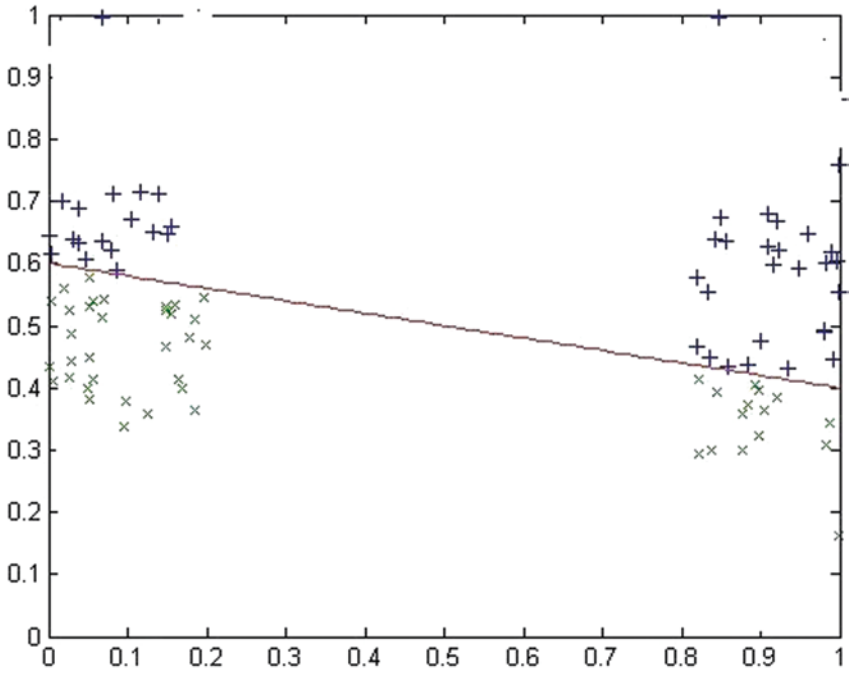

Fig. 6.2 Example of attribute with large $F(1)$ but low w_1

6.3.2 Splitting and Rule Induction

We choose the attribute k_i with maximal SI value as the splitting attribute during the ith iteration. In order to get rules with good comprehensibility we want to get subsets of k_i as pure as possible, which means we want to extract rules like **if** $a_{k_1} \leq \alpha$ **then label** -1 and **if** $a_{k_1} \geq \beta$ **then label 1** with a perfect accuracy. α and β are named split points, which should make sure that the instances are covered as much as possible with coincide label. In addition a constraint inequality must be satisfied: $\alpha \leq \beta$.

If $\alpha = \beta$ the algorithm ends with two rules mentioned above because all the attributes are covered. While $\alpha < \beta$ the rules cannot give the label of instances with $\alpha < a_{k_1} < \beta$, and a_{k_1} is of no use to these instances. We define the rest instances which satisfy $\alpha < a_{k_1} < \beta$ as the training data for the second iteration with a_{k_1} deleted and select a new attribute a_{k_2} with maximal SI value. The procedure could hold on until some stopping criteria are matched.

The method to compute α and β is crucial because the split points are closely related to the quality and performance of the extracted rules. The first method is to compute the cross point that the optimal decision hyperplane learned by SVM intersect the normalized border as showed in Fig. 6.3. The advantage is stability as they

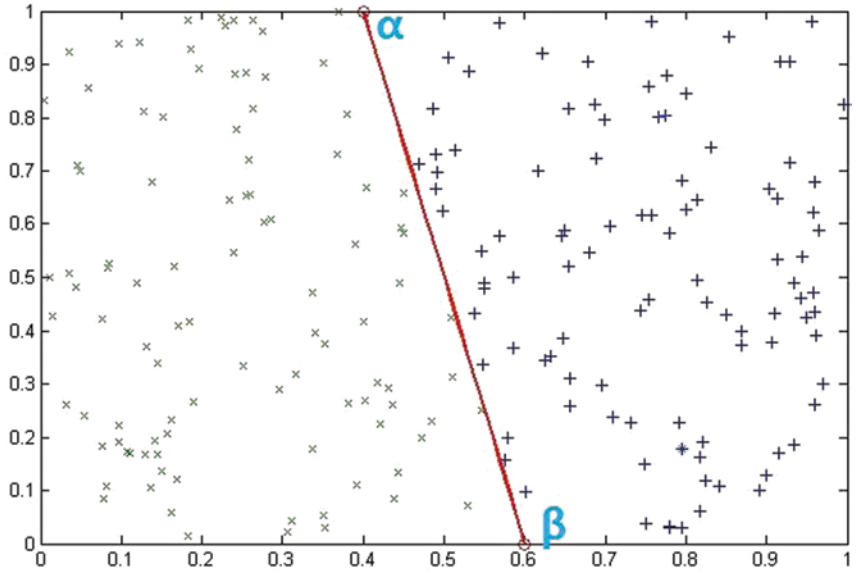

Fig. 6.3 and β are cross points the decision hyperplane intersect the border

are inducted directly from the SVM decision hyperplane. But the intuitive solutions may be hard to compute especially dealing with high dimensional data.

The main idea is to construct a statistic which is a good estimation of the split points and easy to compute. First we assume that the negative and positive points satisfied $0 \le a_{k_1} \le \alpha_i$ and $\beta_i \le a_{k_1} \le 1$ respectively on attribute k_i during ith iteration. So if $\alpha_i \le \beta_i$ we can induct these two rules:

$$if \quad a_{k_i} \le a_i, \quad then -1 \tag{6.13}$$

$$if \quad a_{k_i} \ge a_i, \quad then\ 1 \tag{6.14}$$

The accuracy of these two rules is 100 % on the training data.

If $\alpha_i > \beta_i$ the accuracy of these two rules descends and we should find a better estimation. Set $S_{i_{pos}}$ to be the set that contains the value of attribute k_i on positive instances and $a_{i_{pos}}$ that satisfies:

$$a_{i_{pos}} \ge \alpha_i \tag{6.15}$$

$$\forall a_s \in S_{i_{pos}} \quad we\ have\ a_{i_{pos}} \le a_s \tag{6.16}$$

Then we can yield this rule based on the fact that $a_{i_{pos}}$ is no less than α_i and β_i:

$$if\ a_{k_i} \geq a_{i_{pos}},\ then\ 1 \tag{6.17}$$

According to (6.13) we yield another rule:

$$if\ a_{k_i} > \alpha_i,\ then\ 1 \tag{6.18}$$

Its accuracy is also 100% and we consider (6.17) and (6.18) at the same time. We set the statistics $\overline{\beta}_i$ to be the median of $a_{i_{pos}}$ and α_i:

$$\overline{\beta}_i = (a_{i_{pos}} + \alpha_i)/2 \tag{6.19}$$

Now (6.17) and (6.18) could be replaced by:

$$if\ a_{k_i} \geq \overline{\beta}_i,\ then\ 1 \tag{6.20}$$

Similarly we have corresponding rule on the negative instances:

$$if\ a_{k_i} \leq \overline{\alpha}_i,\ then -1 \tag{6.21}$$

While:

$$\overline{\alpha}_i = (a_{i_{neg}} + \beta_i)/2 \tag{6.22}$$

$\overline{\alpha}_i$ and $\overline{\beta}_i$ are convergent with little error compared to α_i, β_i, $a_{i_{pos}}$ and $a_{i_{neg}}$. We can get formula as follows:

$$\overline{\alpha}_i = \begin{cases} \alpha_i & if\ \alpha_i \leq \beta_i; \\ (a_{i_{neg}} + \beta_i)/2 else. \end{cases}$$

$$\overline{\beta}_i = \begin{cases} \beta_i & if\ \alpha_i \leq \beta_i; \\ (a_{i_{pos}} + \alpha_i)/2 else. \end{cases}$$

And two yielded rules could have unique form:

$$if\ a_i \leq \overline{\alpha}_i,\ then -1 \tag{6.23}$$

$$if\ a_i \geq \overline{\beta}_i,\ then\ 1 \tag{6.24}$$

But one problem should be taken into consideration: the estimated statistics $\bar{\alpha}$ and $\bar{\beta}$ are strongly relied on α_i and β_i because they can also change the value of $a_{i_{pos}}$ and $a_{i_{neg}}$ according to (6.15). If there is an outlier the statistics biases too much. We mark α_{abnor} for this "abnormal" training data and α_{nor} while deleting the outlier. $|\alpha_{abnor} - \alpha_{nor}|$ may be great as the outlier plays an important role.

To eliminate the influence of outliers we need to make the data set linear separable in order that the label is coincided with what the SVM predict. According to the thought of pedagogical rule extraction algorithm known as **learn what SVM has learned** we could make the training set linear separable through 3 steps: (1) perform linear SVM on the normalized training data and get the decision function; (2) change the label into what the decision function predicts; (3) do the **second learning** on the linear separable data and get new decision function. After these steps we erase the outliers and $\bar{\alpha}$ and $\bar{\beta}$ are good approximation of split points.

In order to stop the iteration we construct two stopping criterion: (1) no attribute left; (2) $\alpha_i = \beta_i$ such that all the instances are covered by the rules extracted. Nevertheless, sometimes these criteria are too idealized. We should do some changes to make the criteria practical. If we take **comprehensibility** into consideration we should limit the number of antecedent because rules with too many antecedents are hard to comprehend and interpret especially when the training data is of high dimension. So the first criterion could be changed as follows: (1) The number of antecedents reaches the maximal threshold (5 usually).

On the other hand some rules may be redundant because their coverage is too low. We can prune them to keep the rules in rule set efficient and easy to understand. We can also integrate a rule into a "father" rule which developed during the last iteration with one antecedent less. This process could repeat, but it may reduce the accuracy of the "pruned" rules. Now the stopping criteria could be changed into:

1) The number of antecedents reaches the maximal threshold (5 usually) or no attribute left.
2) $\alpha_i = \beta_i$ Such that all the instances are covered by the rules extracted.
3) Too little instances remain in the training data.

For these rules are on the normalized data we should convert them into rules on original training data. The final step is to refer to the meaning of each attribute and change the norm such as "attribute k" into its real meaning.

Now we can summarize the algorithm as follows:

Algorithm 6.1 (Rule extraction from SVM using Split Index)

1) Divide the data into two parts: training data and test data;
2) Normalize the training set and do linear SVM on it, change the label into what the SVM predict;
3) Do linear SVM and get the decision function;
4) Compute Split Index value and choose the attribute a_{k_i} with maximal value as splitting attribute;
5) Compute α_i and β_i, then extract two rules respectively;

6) Delete the points covered by these two rules and make the instances rest to consist of the new training data with a_{k_i} deleted;
7) Repeat step 3–6 with $i \leftarrow i+1$ until any of the stopping criterion is matched;
8) Get the rule set and prune redundant rules;
9) Yield corresponding rules on original training data;
10) Do tests on test data and evaluate the criterion of the rules in rule set;

6.4 Numerical Experiments

We choose the wine data as our experimental data from UCI repository.

The data is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines, and we select two types among the three for our two-class classification task and the first 130 instances are reserved. For the comprehensibility of our results we need to illustrate the meaning of each attribute for detail:

1) Alcohol;
2) Malic acid;
3) Ash;
4) Alcalinity of ash;
5) Magnesium;
6) Total phenols;
7) Flavanoids;
8) Nonflavanoid phenols;
9) Proanthocyanins;
10) Color intensity;
11) Hue;
12) OD280/OD315 of diluted wines;
13) Proline

We randomly select 65 instances as training data while the rest consist of the test data. During the first iteration the a_{13} has the maximal SI value and we have $\alpha_1 = 0.5043,\ \beta_1 = 0.326,\ a_{1_{pos}} = 0.5257, a_{1_{neg}} = 0.3138, \overline{\alpha}_1 = 0.32, \overline{\beta}_1 = 0.515$ after computation. According to (6.23) and (6.24) we yield two rules:

$$if\ a_{13} \le 0.32, \quad then\ 1 \tag{6.25}$$

$$if\ a_{13} \ge 0.515, \quad then\text{-}1 \tag{6.26}$$

On the second iteration the splitting attribute is a_2, and we have $\alpha_2 = 0.2617$, $\beta_2 = 0.2483$,

$a_{2_{pos}} = 0.3087, a_{2_{neg}} = 0.2416, \bar{a}_2 = 0.245, \overline{\beta_2} = 0.2852$, so we get two rules:

$$if\ a_2 \le 0.245, \quad then\ 1 \tag{6.27}$$

$$if\ a_2 \ge 0.2852, \quad then\text{-}1 \tag{6.28}$$

On the second iteration the splitting attribute is a_3 and there are only two instances in the training data, so we end the algorithm according to the stopping criterion (3). Then we yield rule set on the original training data:

$$R1: if\ \ \mathrm{Pr}oline \le 726.64, \quad then\ 1 \tag{6.29}$$

$$R2: if\ \ \mathrm{Pr}oline \ge 1000, \quad then -1 \tag{6.30}$$

$$R3: if\ \ 726 < \mathrm{Pr}oline < 1000\ and\ Malic\ Acid \le 1.62, \quad then\ 1 \tag{6.31}$$

$$R4: if\ \ 726 < \mathrm{Pr}oline < 1000\ and\ Malic\ Acid \ge 1.74, \quad then -1 \tag{6.32}$$

The following table shows the performance of rules on the test data (Table 6.1):

Table 6.1 EXPERIMENTAL RESULTS ON WINE TEST DATA

Rule	Fidelity	Accuracy	Coverage	Number of antecendent
R1	0.94	0.97	36/65	1
R2	0.95	1	20/65	1
R3	1	1	1/65	2
R4	1	1	4/65	2