Classification Algorithms for Data Mining: A Survey

Raj Kumar
Department of Computer Science and Engineering
Jind Institute of Engg. & Technolog, Jind, Haryana, India

Dr. Rajesh Verma
Department of Computer Science and Engineering
Kurukshetra Institute of Technology & Management, Kurukshetra, Haryana, India

Abstract— Classification is a model finding process that is used for portioning the data into different classes according to some constrains. In other words we can say that classification is process of generalizing the data according to different instances. Several major kinds of classification algorithms including C4.5, k-nearest neighbor classifier, Naive Bayes, SVM, Apriori, and AdaBoost. This paper provide a inclusive survey of different classification algorithms.

Keywords – Bayesian, classification, KDD, Data Mining, SVM, kNN, C4.5.

I. Introduction

Data Mining or Knowledge Discovery is needed to make sense and use of data. Knowledge Discovery in Data is the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data [1]. Data mining consists of more than collection and managing data; it also includes analysis and prediction. People are often do mistakes while analysing or, possibly, when trying to establish relationships between multiple features. This makes it difficult for them to find solutions to certain problems. Machine learning can often be successfully applied to these problems, improving the efficiency of systems and the designs of machines. There are several applications for Machine Learning (ML), the most significant of which is data mining. Numerous ML applications involve tasks that can be set up as supervised. In the present paper, we have concentrated on the techniques necessary to do this. In particular, this work is concerned with classification problems in which the output of instances admits only discrete, unordered values.

II. C4.5 Algorithm

Systems that construct classifiers are one of the commonly used tools in data mining. Such systems take as input a collection of cases, each belonging to one of a small number of classes and escribed by its values for a fixed set of attributes, and output a classifier that can accurately predict the class to which a new case belongs. Like CLS and ID3, C4.5 generates classifiers expressed as decision trees, but it can also construct classifiers in more comprehensible ruleset form.[2][3][4].

2.1 **Decision Tree:** Decision trees are trees that classify instances by sorting them based on feature values Given a set S of cases, C4.5 first grows an initial tree using the divide-and-conquer algorithm as follows:

- If all the cases in S belong to the same class or S is small, the tree is a leaf labeled with the most frequent class in S.
- Otherwise, choose a test based on a single attribute with two or more outcomes. Make this test the root of the tree with one branch for each outcome of the test, partition S into corresponding

subsets S1, S2… according to the outcome for each case, and apply the same procedure to each subset.

Decision trees are usually unvaried since they use based on a single feature at each internal node. Most decision tree algorithms cannot perform well with problems that require diagonal partitioning.
C4.5 uses two heuristic criteria to rank possible tests: information gain, which minimizes the total entropy of the subsets $\{Si\}$ and the default gain ratio that divides information gain by the information provided by the test outcomes. Attributes can be either numeric or nominal and this determines the format of the test outcomes. For a numeric attribute A they are $\{A \leq h, \ A > h\}$ where the threshold h is found by sorting S on the values of A and choosing the split between successive values that maximizes the criterion above. An attribute A with discrete values has by default one outcome for each value, but an option allows the values to be grouped into two or more subsets with one outcome for each subset. The initial tree is then pruned to avoid overfitting. The pruning algorithm is based on a pessimistic estimate of the error rate associated with a set of N cases, E of which do not belong to the most frequent class. Instead of E/N, C4.5 determines the upper limit of the binomial probability when E events have been observed in N trials, using a user-specified confidence whose default value is 0.25. Pruning is carried out from the leaves to the root. The estimated error at a leaf with N cases and E errors is N times the pessimistic error rate as above. For a subtree, C4.5 adds the estimated errors of the branches and compares this to the estimated error if the subtree is replaced by a leaf; if the latter is no higher than the former, the subtree is pruned. Similarly, C4.5 checks the estimated error if the subtree is replaced by one of its branches and when this appears beneficial the tree is modified accordingly.
The pruning process is completed in one pass through the tree C4.5's tree-construction algorithm differs in several respects from CART[5]

- Tests in CART are always binary, but C4.5 allows two or more outcomes.

- CART uses the Gini diversity index to rank tests, whereas C4.5 uses information-based criteria.
- CART prunes trees using a cost-complexity model whose parameters are estimated by cross-validation; C4.5 uses a single-pass algorithm derived from binomial confidence limits.
- This brief discussion has not mentioned what happens when some of a case's values are unknown. CART looks for surrogate tests that approximate the outcomes when the tested attribute has an unknown value, but C4.5 apportions the case probabilistically among the outcomes.

Limitations of C4.5 Algorithm: Some limitations of C4.5 are discussed below.
(a) Empty branches: Constructing tree with meaningful value is one of the crucial steps for rule generation by C4.5 algorithm. In our experiment, we have found many nodes with zero values or close to zero values. These values neither contribute to generate rules nor help to construct any class for classification task. Rather it makes the tree bigger and more complex.
(b) Insignificant branches: Numbers of selected discrete attributes create equal number of potential branches to build a decision tree. But all of them are not significant for classification task. These insignificant branches not only reduce the usability of decision
(c) Over fitting: Over fitting happens when algorithm model picks up data with uncommon characteristics. This cause many fragmentations is the process distribution. Statistically insignificant nodes with very few samples are known as fragmentations [6]. Generally C4.5 algorithm constructs trees and grows it branches 'just deep enough to perfectly classify the training examples'. This strategy performs well with noise free data. But most of the time this approach over fits the training examples with noisy data. Currently there are two approaches are widely using to bypass this over-fitting in decision tree learning [7].Those are:

- If tree grows very large, stop it before it reaches maximal point of perfect classification of the training data
- Allow the tree to over-fit the training data then post-prune tree.

III. THE K-NEAREST NEIGHBOR ALGORITHM

Suppose that an object is sampled with a set of different attributes, but the group to which the object belongs is unknown. Assuming its group can be determined from its attributes; different algorithms can be used to automate the classification process. A nearest neighbor classifier is a technique for classifying elements based on the

classification of the elements in the training set that are most similar to the test example. With the k-nearest neighbor technique, this is done by evaluating the k number of closest neighbors [8]
In pseudocode, k-nearest neighbor classification algorithm can be expressed fairly compactly [8]:

```
k ← number of nearest neighbors
    for each object X in the test set do
        calculate the distance D(X,Y) between X and every object Y  in the training set
        neighborhood ← the k neighbors in the training set closest to X
        X.class ←SelectClass(neighborhood)
    end for
```

3.1 Distance Metric:
All of the training samples are stored in an n-dimensional pattern space. When given an unknown sample, a k-nearest neighbor classifier searches the pattern space for the k training samples that are closest to the unknown sample. "Closeness" is
defined in terms of Euclidean distance, where the Euclidean distance, where the Euclidean distance between two points, X=(x1,x2,……,x_n) and Y=(y1,y2,….,y_n) is [9] :

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

X and Y are the two compared objects and n is their number of attributes. However, for distance calculations involving symbolic (nominal) data, other methods are necessary; for symbolic data a similarity metric is used to measure distance, the most basic of which is the overlap metric. The overlap metric simply tests for equality between two values, so that different values get distance 1 whereas equal values get distance 0[9]:

$d_{overlap}\ (x , y) = 0$ when $x=y$

And

$d_{overlap}\ (x , y) =1$, when $x \neq y$

The unknown sample is assigned the most common class among its k nearest neighbors. When k=1, the unknown sample is assigned the class of the training sample that is closest to it in pattern space. Nearest neighbor classifiers are instance-based or lazy learners in that they store all of the training samples and do not build a classifier until a new(unlabeled) sample needs to be classified. This contrasts with eager learning methods, such a decision tree induction and backpropagation, which construct a generalization model before receiving new samples to classify. Lazy learners can incur expensive computational costs when the number of potential neighbors (i.e., stored training samples)with which to compare a given unlabeled smaple is great. Therefore, they require efficient indexing techniques. An expected lazy learning methods are faster ata trainging than eager methods, but slower at classification since all computation is delayed to that time. Unlike decision tree induction and backpropagation, nearest neighbor classifiers assign equal weight to each attribute. This may cause confusion when there are many irrelevant attributes in the data. Nearest neighbor classifiers can also be used for prediction, that is, to return a real-valued prediction for a given unknown sample. In this case, the classifier retruns the average value of the real-valued associated with the k nearest neighbors of the unknown sample.

The *k-nearest neighbors' algorithm* is amongest the simplest of all machine learning algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors. *k* is a positive integer, typically small. If *k* = 1, then the object is simply assigned to the class of its nearest neighbor. In binary (two class) classification problems, it is helpful to choose *k* to be an odd number as this avoids tied votes.

The same method can be used for regression, by simply assigning the property value for the object to be the average of the values of its *k* nearest neighbors. It can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. The neighbors are taken from a set of objects for which the correct classification (or, in the case of regression, the value of the property) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. In order to identify neighbors, the objects are represented by position vectors in a multidimensional feature space. It is usual to use the Euclidian distance, though other distance measures, such as the Manhanttan distance could in principle be used instead. The *k*-nearest neighbor algorithm is sensitive to the local structure of the data.

IV. Naive Bayes

Given a set of objects, each of which belongs to a known class, and each of which has a known vector of variables, our aim is to construct a rule which will allow us to assign future objects to a class, given only the vectors of variables describing the future objects. Problems of this kind, called problems of supervised classification, are ubiquitous, and many methods for constructing such rules have been developed. One very important one is the naive Bayes method—also called idiot's Bayes, simple Bayes, and independence Bayes. This method is important for several reasons. It is very easy to construct, not needing any complicated iterative parameter estimation schemes. This means it may be readily applied to huge data sets. It is easy to interpret, so users unskilled in classifier technology can understand why it is making the classification it makes. And finally, it often does surprisingly well: it may not Probabilistic approaches to classification typically involve modeling the conditional probability distribution $P(C|D)$, where C ranges over classes and D over descriptions, in some language, of objects to be classified. Given a description d of a particular object, we assign the class $argmax_c P(C = c|D = d)$. A Bayesian approach splits this posterior distribution into a prior distribution $P(C)$ and a likelihood $P(D|C)$:

$$argmax_c\, P(C = c|D = d) = argmax_c \frac{P(D = d|C = c)P(C = c)}{P(D = d)} \qquad (1)$$

The denominator $P(D = d)$ is a normalising factor that can be ignored when determining the maximum *a posteriori* class, as it does not depend on the class. The key term in Equation (1) is $P(D = d|C = c)$, the likelihood of the given description given the class (often abbreviated to $P(d|c)$). A Bayesian classifier estimates these likelihoods from training data, but this typically requires some additional simplifying assumptions. For instance, in an attribute-value representation (also called *propositional* or single-table representation), the individual is described by a vector of values $a_1, \ldots, a_n$ for a fixed set of attributes $A_1, \ldots, A_n$. Determining $P(D = d|C = c)$ here requires an estimate of the joint probability $P(A_1 = a_1, \ldots, A_n = a_n|C = c)$, abbreviated to $P(a_1, \ldots, a_n|c)$. This joint probability distribution is problematic for two reasons:
(1) its size is exponential in the number of attributes n, and (2) it requires a complete training set, with several examples for each possible description. These problems vanish if we can assume that all attributes are independent given the class:

$$P(A_1 = a_1, \ldots, A_n = a_n|C = c) = \prod_{i=1}^{n} P(A_i = a_i|C = c) \qquad (2)$$

This assumption is usually called the *naive Bayes assumption*, and a Bayesian classifier using this assumption is called the *naive Bayesian classifier*, often abbreviated to 'naive Bayes'. Effectively, it means that we are ignoring interactions between attributes within individuals of the same class.

V. Support vector machines (SVM)

SVM was first introduced by Vapnik [10] and has been very effective method for regression, classification and general pattern recognition [11]. It is considered a good classifier because of its high generalization performance without the need to add a priori knowledge, even when the dimension of the input space is very high. It is considered a good classifier because of its high generalization performance without the need to add a priori knowledge, even when the dimension of the input space is very high. The aim of SVM is to find the best classification function to distinguish between members of the two classes in the training data. The metric for the concept of the "best" classification function can be realized geometrically. For a linearly separable dataset, a linear classification function corresponds to a separating hyperplane *f(x)* that passes through the middle of the two classes, separating the two. Once this function is determined, new data instance x_n can be classified by simply testing the sign of the function $f(x_n)$; x_n belongs to the positive class if $f(x_n) > 0$.

Because there are many such linear hyperplanes, SVM guarantee that the best such function is found by maximizing the margin between the two classes. Intuitively, the margin is defined as the amount of space, or separation between the two classes as defined by the hyperplane. Geometrically, the margin corresponds to the shortest distance between the closest data points to a point on the hyperplane. Having this geometric definition allows us to explore how to maximize the margin, so that even though there are an infinite number of hyperplanes, only a few qualify as the solution to SVM. To ensure that the maximum margin hyperplanes are actually found, an SVM classifier attempts to maximize the following function with respect to $\vec{\mathbf{w}}$ and b:

$$L_P = \frac{1}{2} \| \vec{\mathbf{w}} \| - \sum_{i=1}^{t} \alpha_i y_i (\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_\mathbf{i} + b) + \sum_{i=1}^{t} \alpha_i \qquad (2)$$

where t is the number of training examples, and α_i , $i = 1, \ldots, t$, are non-negative numbers such that the derivatives of L_P with respect to α_i are zero. α_i are the Lagrange multipliers and L_P is called the Lagrangian. In this equation, the vectors $\vec{\mathbf{w}}$ and constant b define the hyperplane. A learning machine, such as the SVM, can be modeled as a function class based on some parameters α. Different function classes can have different capacity in learning, which is represented by a parameter h known as the VC dimension [12]. The VC dimension measures the maximum number of training examples where the function class can still be used to learn erfectly, by obtaining zero error rates on the training data, for any assignment of class labels on these points. It can be proven that the actual error on the future data is bounded by a sum of two terms. The first term is the training error, and the second term if proportional to the square root of the VC dimension h. Thus, if we can minimize h, we can minimize the future error, as long as we also minimize the training error, SVM can be easily extended to perform numerical calculations.
Here we discuss two such extensions. To extend SVM to perform regression analysis, where the goal is to produce a linear function that can approximate that target function. Careful consideration goes into the choice of the error models; in support vector regression, or SVR, the error is defined to be zero when the differences between actual and predicted values are within a epsilon amount. Otherwise, the *epsilon insensitive* error will grow linearly. The support vectors can then be learned through the minimization of the Lagrangian. An advantage of support vector regression is reported to be its insensitivity to outliers. One of the initial drawbacks of SVMis its computational inefficiency. However, this problem is being solved with great success. One approach is to break a large optimization problem into a series of smaller problems, where each problem only involves a couple of carefully chosen variables so that the optimization can be done efficiently. The process iterates until all the decomposed optimization problems are solved successfully. A more recent approach is to consider the problem of learning an SVM as that of finding an approximate minimum enclosing ball of a set of instances.

VI. The Apriori algorithm

One of the most popular data mining approaches is to find frequent itemsets from a transaction dataset and derive association rules. Finding frequent itemsets is not trivial because of its combinatorial explosion. Once frequent itemsets are obtained, it is straightforward to generate association rules with confidence larger than or equal to a user specified minimum confidence.
Apriori is a seminal algorithm for finding frequent itemsets using candidate generation [13]. It is characterized as a level-wise complete search algorithm using anti-monotonicity of itemsets, "if an itemset is not frequent, any of its superset is never frequent". By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. Let the set of frequent itemsets of size k be F_k and their candidates be C_k . Apriori first scans the database and searches for frequent itemsets of size 1 by accumulating the count for each item and collecting those that satisfy the minimum support requirement. It then iterates on the following three steps and extracts all the frequent itemsets.

1. Generate C_{k+1}, candidates of frequent itemsets of size k +1, from the frequent itemsets of size k.
2. Scan the database and calculate the support of each candidate of frequent itemsets.
3. Add those itemsets that satisfies the minimum support requirement to $F_{k+1.}$

Apriori algorithm is given in the below figuer1:

```
$F_1$=(Frequent itemsets of cardinality 1);
for($k = 1; F_k \neq \phi; k++$) do begin
    $C_{k+1}$ = apriori-gen($F_k$); //New candidates
    for all transactions $t \in$ Database do begin
        $C_t'$ = subset($C_{k+1}, t$); //Candidates contained in $t$
        for all candidate $c \in C_t'$ do
            $c.count++$;
        end
        $F_{k+1} = \{C \in C_{k+1} \quad | c.count \geq$ minimum support $\}$
    end
end
Answer $\cup_k F_k$;
```

Fig.1: Apriori Algorithm

In the above Fig. 1. Function apriori-gen in line 3 generates C_{k+1} from F_k in the following two step process:

1. Join step: Generate R_{K+1}, the initial candidates of frequent itemsets of size k + 1 by taking the union of the two frequent itemsets of size k, P_k and Q_k that have the first k−1 elements in common.

$R_{k+1}=P_k \cup Q_k= \{item_1, item_2,......,item_k, item_{k'}\}$

$P_k = \{item_1, item_2,......,item_k, item_k\}$

$Q_k = \{item_1, item_2,......, item_{k'}\}$

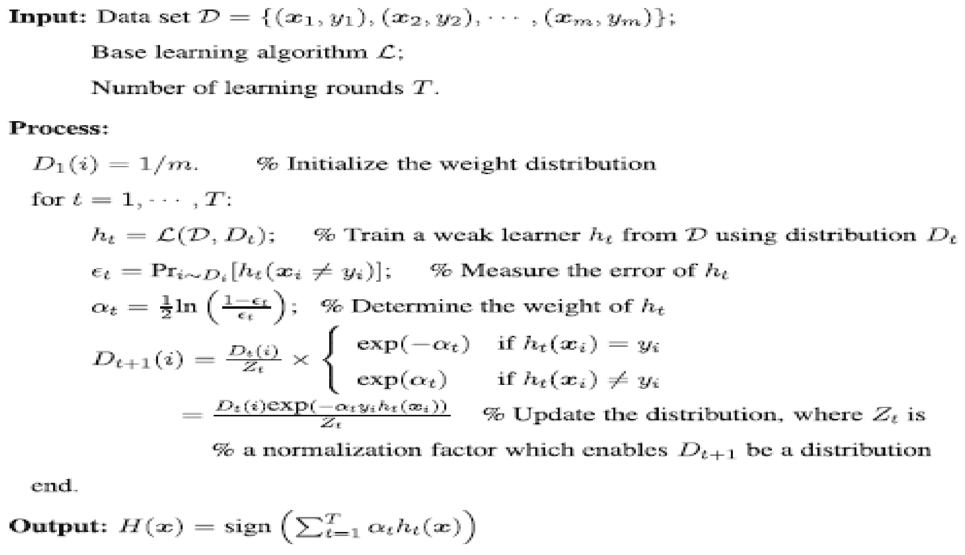
Where $item_1<item_2<.........<item_k<item_{k'}$

2. Prune step: Check if all the itemsets of size k in R_{k+1} are frequent and generate C_{k+1} by removing those that do not pass this requirement from R_{k+1}. This is because any subset of size k of C_{k+1} that is not frequent cannot be a subset of a frequent itemset of size k + 1.

Function subset in line 5 finds all the candidates of the frequent itemsets included in transaction t. Apriori, then, calculates frequency only for those candidates generated this way by scanning the database. It is evident that Apriori scans the database at most k_{max+1} times when the maximum size
of frequent itemsets is set at k_{max}.

VII. AdaBoost

The AdaBoost algorithm [14] proposed by Yoav Freund and Robert Schapire is one of the most important ensemble methods, since it has solid theoretical foundation, very accurate prediction, great simplicity (Schapire said it needs only “just 10 lines of code”), and wide and successful applications.
Let X denotes the instance space and Y the set of class labels. Assume Y = {−1, +1}. Given a weak or base learning algorithm and a training set $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$ where $\mathbf{x}_i \in X$ and yi $\in Y$ (i = 1, . . . ,m), the AdaBoost algorithm works as follows[15]:



```
Input: Data set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$;
       Base learning algorithm $\mathcal{L}$;
       Number of learning rounds $T$.
Process:
  $D_1(i) = 1/m$.    % Initialize the weight distribution
  for $t = 1, \cdots, T$:
      $h_t = \mathcal{L}(\mathcal{D}, D_t)$;    % Train a weak learner $h_t$ from $\mathcal{D}$ using distribution $D_t$
      $\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i \neq y_i)]$;    % Measure the error of $h_t$
      $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$;    % Determine the weight of $h_t$
      $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(x_i) \neq y_i \end{cases}$
                 $= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$    % Update the distribution, where $Z_t$ is
                         % a normalization factor which enables $D_{t+1}$ be a distribution
  end.
Output: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
```

Fig.2: AdaBoost Algorithm

First, it assigns equal weights to all the training examples ($\mathbf{x}_i$, y_i) (i $\in$ {1,2………..,m}). Let D_t Denote the distribution of the weights at the *t*-th learning round. From the training set and D_t the algorithm generates a weak or base learner h_t : X → Y by calling the base learning algorithm. Then, it uses the training examples to test $h_{t,}$ and the weights of the incorrectly classified examples will be increased. Thus, an updated weight distribution D_{t+1} is obtained. From the training set and D_{t+1} AdaBoost generates another weak learner by calling the base learning algorithm again. Such a process is repeated for T rounds, and the final model is derived byweighted majority voting of the T weak learners,where theweights of the learners are determined during the training process. In practice, the base learning algorithm may be a learning algorithm which can use weighted training examples directly; otherwise the weights can be exploited by sampling the training examples according to the weight distribution D_t. The pseudo-code of AdaBoost is shown in the below Fig. 2

VIII.CONCLUSION

These classification algorithms can be implemented on different types of data sets like data of patients, financial data according to performances. On the basis of the performance of these algorithms, these algorithms can also be used to detect the natural disasters like cloud bursting, earth quake, etc.

IX. REFERENCE

[1] Fayyad, Piatetsky-Shapiro, Smyth, and Uthurusamy: “Advances in Knowledge Discovery and Data Mining”, AAAI/MIT Press 1996
[2] Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann Publishers, San Mateo
[3] Hunt EB, Marin J, Stone PJ (1966) Experiments in induction. Academic Press, New York
[4] Quinlan JR (1979) Discovering rules by induction from large collections of examples. In: Michie D (ed)
[5] Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees,Wadsworth
[6] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publish, 2001

[7] A. B. M. S. Ali and K. A. Smith , On learning algorithm for classification, *Applied Soft Computing* Dec 2004. pp. 119-138.
[8] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. Introduction to Data
Mining, Addison Wesley, 2006
[9] D. Randall Wilson and Tony R. Martinez. Functions. Improved Heterogeneous Distance In Journal of Artificial Intelligence Research (January 1997), pp. 1-34.
[10] Vapnik VN. Statistical learning theory, New York: Wiley;1998.
[11] ristianini N, Shawe-Taylor J. An introduction to support vector machines and other kernel-based learning methods. Cambridge, UK: Cambridge University Press; 2000
[12] Vapnik V (1995) The nature of statistical learning theory. Springer, New York
[13] Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceeding of the 20th VLDB conference, pp 487–499
[14] Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139
[15] Xindong Wu et.al, “Top 10 Algorithms of Data Mining”, Springer-Verlag London, 2007.