CSC4008 Data Mining

117010279 Ziren WANG

April 2020

5 Basic Classification

5.1 Several Concept

- Supervised VS. Unsupervised learning: whether accompanied by labels or not.
- Prediction Problems: Classification VS. Numeric Prediction: predict labels or unknown values.

Classification: Two Step Process
(1) Model construction: describing a set of predetermined classes;
(2) Model usage: for classifying future or unknown objects, we estimate accuracy of the model and use models to classify new data.

5.2 Decision Tree Induction

5.2.1 Basic Algorithm for Inducing a Decision Tree

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C, then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

Figure 1: decision tree algorithm

5.2.2 Attribute Selection method

We use Entropy to measure the uncertainty associated with a random variable that taking m distinct values: $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i), where: p_i = P(Y = y_i)$. We also have conditional Entropy: $H(Y|X) = \sum_x p(x)H(Y|X = x)$. The interpretation is: higher entropy implies higher uncertainty, vice versa. Several other measures are as following:

- **Expected information (entropy) of tuple D**: $Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$
- **Information needed**: $Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$, where D is splited into v partitions by A

- **Information gained from attribute A**: $Gain(A) = Info(D) - Info_A(D)$
  Used by ID3 tree, but information gain measure is biased towards attributes with a large number of values v.
- **Gain ratio**: $splitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2(\frac{D_j}{D})$, $GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$
  Used by C4.5, the attribute with the maximum gain ratio is selected as the splitting attribute.
- **Gini Index**: $Gini_A(D) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2), where : Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$, A split D into D_1&D_2. Reduction in Impurity: $\Delta Gini(A) = Gini(D) - Gini_A(D)$.
  biased on multivalued attributes and has difficulty when of classes is large. Tends to favor tests that result in equal-sized partitions and purity in both partitions

There are two approaches to avoid overfitting: Pre-pruning & Post-pruning. And other enhancements: (1)Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals; (2)Assign the most common value of the attribute or the probability to each of the possible values to handle missing attribute values; (3)construct new attribute based on existing ones.

5.2.3 Why Decision tree induction is popular?

(1) comparable classification accuracy with other methods;
(2) convertible to simple and easy to understand classification rules;
(3) relatively faster learning speed (than other classification methods);
(4) can use SQL queries for accessing databases.

5.2.4 RainForest & BOAT

RainForest
Separates the scalability aspects from the criteria that determine the quality of the tree, then builds an AVC-list: AVC (Attribute, Value, Class label).
(1) AVC-set (of an attribute X): Projection of training dataset onto the attribute X and class label where counts of individual class label are aggregated;
(2) AVC-group (of a node n): Set of AVC-sets of all predictor attributes at the node n.

BOAT (Bootstrapped Optimistic Algorithm for Tree Construction)
(1) Use a statistical technique called bootstrapping to create several smaller samples (subsets), each fits in memory;
(2) Each subset is used to create a tree, resulting in several trees; (3) These trees are examined and used to construct a new tree T'.
Advantage of BOAT: requires only two scans of DB, an incremental alg.

5.3 Model Evaluation and Selection

5.3.1 Confusion Matrix

Actual/Predict Class	C_1	$\overline{C_1}$
C_1	True Positives (TP)	False Negatives(FN)
$\overline{C_1}$	False Positives(FP)	True Negatives(TN)

We may use a simple form:

Actual/Predict	yes	no	Total
yes	TP	FN	P
no	FP	TN	N
Total	P'	N'	P+N

Several measures are:

Measure Name	Formula
accuracy, recognition rate	$\frac{TP+TN}{P+N}$
error rate, mis-classification rate	$\frac{FP+FN}{P+N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity	$\frac{TN}{N}$
precision	$\frac{TP}{TP+FP} = \frac{TP}{P'}$
F-score, harmonic mean of precision and recall	$\frac{2 \times precision \times recall}{precision+recall}$
F_β, where β is a non-negative real number	$\frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision+recall}$

Note: $accuracy = sensitivity \frac{P}{(P+N)} + specificity \frac{N}{(P+N)}$

5.3.2 Partitioning Data Set

- **Holdout Method**
  Given data is randomly partitioned into two independent sets: (1)Training set (e.g., 2/3) for model construction; (2)Test set (e.g., 1/3) for accuracy estimation.
  We may use random sampling to partition. Repeat holdout k times, accuracy = avg. of the accuracies obtained.

- **Cross-validation** (k-fold, where k = 10 is most popular)
  Randomly partition the data into k mutually exclusive subsets, each approximately equal size. At i-th iteration, use D_i as test set and others as training set (so called Leave-one-out principle)

- **Bootstrap**
  Works well with small data sets. The Method is Sampling the given training tuples uniformly with replacement. A common one is 0.632 bootstrap (36.8% test set). Then: $Accuracy(M) = \frac{1}{k} \sum_{i=1}^{k} (0.632 \times Acc(M_i)_{test} + 0.368 \times Acc(M_i)_{train})$.

5.3.3 Estimating Confidence Intervals

Try to answer: 2 classifiers: $M_1 and M_2$, which one is better?
$H_0 : M_1$ *and* M_2 *are the same*, reject when 2 are statistically significantly different.
t test: $t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{var(M_1 - M_2)/k}$ $follows$ $t(k-1)$, where:
$var(M_1 - M_2) = \sqrt{\frac{var(M_1}{k_1} + \frac{var(M_2}{k_2}}$ and k_1 and k_2 are # of cv samples used for M_1 *and* M_2 respectively.

5.3.4 ROC Curves (Receiver Operating Characteristics Curves)

- **Base**: the trade-off between true positive rate and false positive rate.

- **Plot**: Rank the test tuples in decreasing order. Then the one that is most likely to belong to the positive class appears at the top of the list. X-axis: $FPR = \frac{FP}{N} = 1 - Specificity$; Y-axis = $\frac{TP}{P} = Sensitivity$.

- **Criteria**: The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model. A model with perfect accuracy will have an area of 1. Thus we simply compare their AUC (Area Under Curve) and draw the conclusion.

5.3.5 Issues Affecting Model Selection

- Accuracy: classifier accuracy: predicting class label;

- Speed: time to construct models (training time) and time to use models (classification/prediction time);

- Robustness: handling noise and missing values;

- Scalability: efficiency in disk-resident databases;

- Interpretability: understanding and insight provided by the model.

5.4 Bayes Classification Methods

5.4.1 Principle

Bayes Theorem: $P(C|x) = \frac{P(C)P(x|c)}{P(x)}$, where: $P(C|x)$ is posterior; $P(C)$ is prior; $P(x|C)$ is likelihood and $p(x)$ is evidence.

Prediction: Given training data $\boldsymbol{X}$, posterior probability of a hypothesis H, $P(H|X)$, follows the Bayes' theorem: $P(H|\mathbf{X}) = \frac{P(\mathbf{X})P(H)}{P(\mathbf{X})} = P(\mathbf{X}) \times \frac{P(H)}{P(\mathbf{X})}$. Predicts $\boldsymbol{X}$ belongs to C_i if and only if the probability $P(C_i|\boldsymbol{X})$ is the highest among all the $P(C_k|\boldsymbol{X})$ for all the k classes.

Simplified Sssumption: attributes are conditionally independent (i.e., no dependence relation between attributes): $P(\boldsymbol{X}|C_i) = \prod_{k=1}^{n} p(x_k|C_i) = p(x_1|C_i) \times p(x_2|C_i) \times ... \times p(x_n|C_i)$, which reduces the computation cost significantly.

Potential Problem: Zero-probability. Notice Naive Bayesian predication requires each conditional probability be Non-zero, otherwise the predicted probability, $P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$ will be zero. We may use **Laplacian correction** to handle, that is adding 1 sample to each case.

5.4.2 Comments

- Advantages: (1) Easy to implement; (2) Good results obtained in most of the cases.
- Disadvantages: Having a strong assumption: class conditional independence. therefore loss of accuracy in reality because dependencies exist among variables.

5.5 Rule-Based Classification

Despite of usual classification algorithm we used, one can also use "IF-THEN" rule to predict which class dose a sample belong to. There are several assessment criteria for the rule:
(1) n_{covers} = # of tuples covered by rule R;
(2) $n_{correct}$ = # of tuples correctly classified by rule R;
Thus we have: $\mathbf{coverage(R)} = \frac{n_{covers}}{|D|}$; $\mathbf{accuracy(R)} = \frac{n_{correct}}{n_{covers}}$, where D is training data set.
Usually, we use sequential covering method to extract rules from training data directly, such as FOIL, AQ, CN2, RIPPER. A general procedure is as following: Generally speaking, rule-based classification is less accurate than

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set of class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

```
(1)  Rule_set = {}; // initial set of rules learned is empty
(2)  for each class c do
(3)      repeat
(4)          Rule = Learn_One_Rule(D, Att_vals, c);
(5)          remove tuples covered by Rule from D;
(6)          Rule_set = Rule_set + Rule; // add new rule to rule set
(7)      until terminating condition;
(8)  endfor
(9)  return Rule_Set;
```

Figure 2: sequential covering algorithm

other normal classification algorithms, including Naive Bayes.

5.6 Improving Techniques for Classification Algorithms

5.6.1 Bagging

- Training: Give a data set D of d tuples, at each iteration i, sampling D_i, which contains d tuples, from D with replacement. For each D_i, construct a classifier M_i on D_i;

- Prediction: When an unknown sample X input, for each classifier M_i, return its own class predication. The bagged classifier M^* counts the votes of $M_i, i = 1, 2, ...m$. In the end, M^* assigns the most voted class label to X (A regression model will take the average of predictions from each M_i).
- Accuracy: often significantly better than a single classifier derived from D, also quite robust for noise data.

5.6.2 Boosting

Boosting is generalized from bagging, which not only takes simple average of M_i, but also with weights. This is a iterative process. An example of AdaBoost is as following:

Algorithm: AdaBoost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

(1) initialize the weight of each tuple in D to $1/d$;
(2) **for** $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Eq. 8.34)
(6) **if** $error(M_i) > 0.5$ **then**
(7) go back to step 3 and try again;
(8) **endif**
(9) **for** each tuple in D_i that was correctly classified **do**
(10) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(11) normalize the weight of each tuple;
(12) **endfor**

To use the ensemble to classify tuple, X:

(1) initialize weight of each class to 0;
(2) **for** $i = 1$ to k **do** // for each classifier:
(3) $w_i = log \frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4) $c = M_i(X)$; // get class prediction for X from M_i
(5) add w_i to weight for class c
(6) **endfor**
(7) return the class with the largest weight;

Figure 3: AdaBoost

5.6.3 Handling Imbalanced Data Sets

In reality, we may deal with training data sets which contains numerous negative tuples, while positive one is little, or vice versa. Several methods are:

- Oversampling: re-sampling of data from positive class;
- Undersampling: randomly eliminate tuples from negative class;
- Threshold-Moving: moves the decision threshold t, so that the rare class tuples have more probability to be classified.