Data Mining 2 23-24

Notes

University of Pisa

M.Sc. in Data Science and Business Informatics

Contents

Chapter 1

Rule Based Models

A rule based classifier is a model that uses a **rule set** of "if-then" rules to classify instances. Each rule is expressed in the form:

$$r_i : (Cond_i) \rightarrow y_i.$$

The left side contains a conjunction of attribute test conditions, and is called **antecedent** or **precondition**, while the right side represents the predicted class, and is called the **consequent**. Each condition is defined by a set of k attribute-value pairs, such that:

$$Cond_i = (A_1 op v_1) \wedge (A_2 op v_2) \wedge \cdots \wedge (A_k op v_k) \ ,$$

where op is a comparison operator. Each attribute test is also known as a **conjunct**.

A rule r **covers** an instance x if the attributes of the instance satisfy the antecedent of the rule. Consider the following dataset:

Name	Can Fly	Gives Birth	Blood Type
Bat	Y	Y	W
Owl	Y	N	W
Crocodile	N	N	C
Platypus	N	N	W

Table 1.1: Small example dataset.

The rule $(CanFly = Y) \wedge (GivesBirth = N) \rightarrow Bird$ covers the instance "Owl".

The **coverage** of a rule is the fraction of records in the whole dataset that are covered by it. The **accuracy** (sometimes called **precision**) of a rule is the fraction of records in the dataset that satisfy the antecedent that also satisfy the consequent.

Coverage and Accuracy

$$Coverage(r) = \frac{|A|}{|D|}$$

$$Accuracy(r) = \frac{|A \cap y|}{|A|}$$

1.1 How Rule-Based Models Work

A rule based classifier classifies a test instance based on the rule triggered by the instance. Looking at the dataset pictured in Table 1.1, assume we obtained the following rule set from a training set:

$$r_1 : (CanFly = Y) \rightarrow Bird$$
$$r_2 : (GivesBirth = Y) \wedge (BloodType = W) \rightarrow Mammal$$
$$r_3 : (BloodType = C) \rightarrow Reptile$$

The instance Owl triggers the first rule, and is therefore classified as a *Bird*. The Bat triggers both the first and the second rule, which produce conflicting outcomes. None of the rules cover the example Platypus, so there's no immediate way to assign a class to this animal. The following section will explain how these issues can be solved.

1.2 Properties of a Rule Set

The rule set generated by the model can be characterized by the following two properties:

Mutually Exclusive Rule Set

The rules in a rule set R are mutually exclusive if no two rules in R are triggered by the same instance; this property guarantees that each instance is covered by at most one rule in R.

Exhaustive Rule Set

A rule set R is exhaustive if each combination of attribute values is covered by at least one rule.

Unfortunately, many rule based classifiers do not have such properties. If the rule set is not exhaustive, a default rule with an empty antecedent can be added to classify all instances that are not covered by any other rule.

If the rule set is not mutually exclusive, the rules can be organized into an **ordered rule set** (also known as **decision list**).

Ordered Rule Set

The rules in an ordered rule set R are ranked in decreasing order of priority.

The rank of the rule can be defined via either **rule-based ordering** (rules are ranked based on their quality, e.g., their accuracy) or **class-based ordering** (all rules that have the same consequent appear together). When a test instance is presented to the model, it is compared with the rules starting from the one at the top of the ranking, and the prediction will be the one appearing as the consequent of the highest ranking rule that covers the instance. If none of the rules are triggered, the default rule is reached, classifying the instance as the default class.

Another approach is to use a **voting scheme**, where, for each test instance, votes are accumulated for each class assigned to it by the rules it triggers. The prediction will correspond to the class with the highest number of votes, and votes may also be weighted depending on the rule that is producing it (for example, rules with lower accuracy will produce votes with lower weight).

The advantage of using an unordered rule set is that they're less susceptible to errors, since they are not biased by the chosen ordering. Model building is also less expensive, since the rules don't have to be sorted. On the other hand, classification can be more costly, since the same instance must be first compared to all the rules in the rule set before evaluating the votes.

1.3 Building a Rule Set

Rule extraction methods can be either:

- **Direct**, if the rules are extracted from the data itself;
- **Indirect**, if the rules are extracted from some other model (e.g., Decision Trees).

1.3.1 Direct Methods for Rule Extraction

To illustrate how direct methods work, we'll consider a widely-used algorithm called **RIPPER** (Repeated Incremental Pruning to Produce Error Reduction). This algorithm scales almost linearly with the number of training examples, and is particularly suited for datasets with imbalanced class distributions. It also works well with noisy data, since it uses a validation set to prevent overfitting.

RIPPER uses the **sequential covering** algorithm to extract rules from data. This algorithm uses a greedy strategy to build rules, one class at a time. For binary problems, the majority class is chosen as the default, and the algorithm learns the rules to detect only the minority class. For multiclass problems, the classes are first ordered by prevalence in the dataset; then, starting from the least prevalent class y_1, all elements belonging to it are labeled as positive, while all the rest, belonging to $y_2, y_3, \ldots y_c$, are labeled as negative. The sequential covering algorithm learns a set rules that discriminates between these positive and negative classes. Next, all instances in y_2 (the second least prevalent class) are labeled as positive, while all instances belonging to $y_3, y_4, \ldots y_c$ are labeled as negative, and a new rule set is constructed. This process is repeated until only one class remains, y_c, which is designated as the default one.

Algorithm 1 Sequential covering algorithm.

1: E = TR instances, A = set of attribute-value pairs
2: $Y_\sigma = \{y_1, y_2, \ldots, y_k\}$
3: $R = \{\}$
4: **for** each $y \in Y_\sigma - \{y_k\}$ **do**
5: **while** stopping cond is False **do**
6: $r \leftarrow$ Learn-One-Rule(E, A, y)
7: Remove TR instances from E that are covered by r.
8: $R \leftarrow R \vee r$
9: **end while**
10: **end for**
11: Insert default rule: $R \leftarrow R \vee (\{\} \rightarrow y_k)$

The algorithm always starts with an empty decision list, R, and extracts rules for each class following the ordering specified by their prevalence. The Learn-One-Rule function iteratively extracts all rules for the current class, and all training instances

covered by each rule found is removed from E. The rule is then added to the bottom to the rule list, and the loop repeats until the specified stopping criterion is met.

Rule Evaluation

In the Learn-One-Rule function, the algorithm must search for an optimal rule by growing one in a greedy fashion. It starts with a rule with an empty antecedent, $r : \{\} \rightarrow +$. Then, new conjuncts are gradually added to the antecedent in order to improve the rule's accuracy.

RIPPER uses the **FOIL's First Order Inductive Learner) information gain** as the measure to choose which conjunctive to add to the rule's antecedent.

FOIL's Information Gain

Given p_0 and n_0 the number of positive and negative examples covered by the original rule, and p_1 and n_1 the number of positive and negative examples covered by the new rule, the FOIL's information gain is defined as:

$$FOIL's\ inf.gain = p_1 \times \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

RIPPER starts with a rule $r : A \rightarrow +$. It then adds one conjunct, B, generating the rule $r : A \wedge B \rightarrow +$, and the information gain is calculated for this addition. This step is repeated for different conjuncts, and the rule with the highest information gain is chosen to replace the original rule. The function stops once there's no additions that improve the information gain, so the rule covers only positive instances. Additionally, all instances covered by the rule are removed from the training set.

RIPPER also performs pruning of the rules to improve the generalization error based on their performance on a validation set. After generating a rule, the following metric is computed:

$$v = \frac{(p - n)}{(p + n)} \ ,$$

where p/n are the number of positive/negative validation instances covered by that rule. If this measure improves after removing a conjunct, the latter is permanently pruned, and the measure is again evaluated for the next conjunct. The check follows the reverse order to the one established by the insertion of conjuncts during generation. Note that a pruned rule may cover both positive and negative examples of the training set; this means that the rule is less adapted to the training data, but performs better on unseen examples.

The generation of rules is interrupted once a stopping condition is verified, such that the complexity of the model is high enough to generalize well, but not so high that it overfits the training data. Some common stopping conditions are evaluated based on the **Minimum Description Length** (**MDL**). The MDL measures the cost of a model as:

$$Cost(M, D) = Cost(D|M) + \alpha \times Cost(M) \ ,$$

where M and D are the model and the data, respectively, and α is a tuning hyperparameter (usually set to 0.5). The first term of the addition encodes the misclassification error, while the second term uses node encoding (number of children) plus encoding of the splitting condition. The cost is evaluated in terms of how many bits are needed to encode the rule set: if the addition of a rule would increase the length of the set by at least d bits, then RIPPER stops adding rules (by default, d is 64 bits). This is a form of Pessimistic Error Estimate, since it evaluates the generalization error of the model as:

$$R(T) = R_{emp} + \Omega \times \frac{k}{l} \ ,$$

where $R_{emp}(T)$ is the training error, Ω is a trade-off hyperparameter that represents the cost of adding a new rule, k is the size of the rule set, and l is the number of training instances.

RIPPER also performs additional optimization steps to determine whether the rules in the set can be replaced by better alternatives. For each rule r, two new rules are considered as replacement:

- A replacement rule r^*: a new rule is grown from scratch;
- A revised rule r': conjuncts are added to the rule r to extend it.

The rule set for r is compared with the rule sets for r^* and r', choosing the rule that minimizes the MDL.

1.3.2 Indirect Methods for Rule Extraction

Indirect methods generate a rule set by using the output of some other model, typically an unpruned decision tree. In a decision tree, each path connecting the root to a leaf can be expressed as a classification rule, where each attribute test condition encountered on the path is a different conjunct of the antecedent, and the (majority) class in the leaf node is the consequent. This section will focus on the approach followed by the algorithm C4.5rules.

A rule is generated from each path in the tree. For each rule $r : A \rightarrow y$ in the rule set, alternative rules $r' : A' \leftarrow y$ are considered, where A' is obtained by removing

one of the conjuncts in A. The simplified rule with the lowest pessimistic error rate is retained as a replacement if the error rate is also lower than that of the original rule. Eventual duplicates of the new rule are eliminated from the rule set.

After generating the rule set, C4.5rules uses a class-based ordering to rearrange the rules, so that all rules predicting the same class appear close together in the same subset. The description length of each subset is calculated, and the classes are arranged in increasing order of their total description length. This way, the subset with the lowest description length is given priority over the others, since it is assumed to contain the best set of rules.

1.4 Characteristics of Rule Based Models

Rule based classifiers are very similar to decision trees, and have about the same expressiveness. Both models construct rectilinear decision boundaries in the input space, and assign a class to each partition. Rule based classifiers, however, can allow multiple rules to be triggered for the same instance, while in decision trees, each instance can only follow one specific path. Because of this, rule based models can approximate more complex functions.

Like decision trees, they can handle different types of attributes, both continuous and categorical, and can work for both binary and multiclass classification tasks. Additionally, rule based classifiers often produce models that are easier to interpret but have comparable performance to decision trees.

They can also handle redundant attributes, since if two or more highly correlated, only one of them is chosen to be added as a conjunct. Since irrelevant attributes will show poor information gain, rule based models will tend to avoid choosing them as conjuncts. Still, as seen for decision trees, if the problem is sufficiently complex, sometimes irrelevant attributes may be chosen over other more relevant ones that show poor information gain individually, but would be useful when interacting with others.

They cannot handle missing values in the test set, as the positioning of the rules in a rule set follows a specific ordering strategy, so if a test instance is covered by multiple rules they may produce conflicting outputs.

Since RIPPER uses a class-based ordering strategy, emphasizing classes with fewer instances, these models are very well suited for imbalanced class distributions.

Bibliography

[1] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson, 2018.