file/after/tracklang.stypackage/after/bidipackage/after/luabidi

Classification: Basic Concepts

Classification refers to determining which class a particular piece of data belongs to. We can also predict the numerical value of a function given its inputs, using a **numerical predictor**, which is the subject of study of **Regression Analysis**.

We use a large number of example tuples, and take a look their attribute vector $\mathbf{x}$ with n dimensions. During the learning phase, we produce a set of rules for classification, by looking at a relevant feature of the dataset at node t, D_t.

When performing induction, we would test the values of the test input against the values of the nodes, and by checking the condition at each node we could eventually make a decision as to the class. DTs are popular since there is no requirement of prevoius domain knowledge.

At each node, we use a method to select the best attribute according to the data in the class (e.g. Gini coefficient, information gain). We also have to choose whether the tree is binary or has multiple children.

If all the items in D_t belong to the same class, then we make a leaf node with the class label C. Otherwise, we need to use the aforementioned node-splitting heuristic, to find the best attribute to perform a split on. Then we separate the nodes according to their values in that specific criterion.

Recursively call the procedure on each D_j according to the jth data split. The recursive procedure ends when one condition is true:

1. All the nodes in D_t belong to the same class, in which case we label the leaf node with that class value.
2. There are no remaining attributes on which to perform the split. Here the label would be the most common class in D.
3. There are no more tuples in the branch. Here the label is the majority class in D.

The computational complexity is $O(n \times |D| \times log(D))$. What if we keep adding training data D as we gain more training examples? We could use incremental versions of this algorithm to avoid re-learning the tree.

How do we chooe the best attribute to perform the split on? Since we have a map of the data attributes, we need the attribute that best splits the data. Based on the attribute measure for each attribute, we can select the best attribute to perform the split on. There are a couple of attribute meausures:

1. **Information Gain**: Here we measure the attribute which minimizes the information needed to classify the resulting tuples. It measures the least "randomness" or "impurity". The expected information needed to classify a tuple is

$$Info(D) = -\sum_{i=1}^{m} p_i log_2(p_i) \tag{1}$$

where p_i is the probability that a tuple belongs to class C_i. And p_i is really just the number of members in class C_i in the dataset D.

We would also have to measure the amount of information that would still be needed to classify members in the dataset, i.e. the amount of information we gain by splitting on attribute A. The amount we would gain is

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j) \tag{2}$$

The information gain is then described as how much information we gain by branching on A,

$$Gain(A) = Info(D) - Info_A(D) \tag{3}$$

If one attribute is continous, how do we select the best point to split? One option is to first sort the attribute values in increasing order and use the midpoint between each pair of adjacent values as the possible split point, $\frac{a_i+a_{i+1}}{2}$. Then we use the answer to calculate the information gain by splitting the datset on this value.

Gain Ratio

The information gain statistic has a bias towards attributes where more tests are performed. If there is an ID field where each attribute value is unique, then every split on this attribute will be uniques, which means that IG isn't so useful a metric anymore. The gain ratio tries to normalize the data beforehand

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times log_2(\frac{|D_j|}{|D|}) \tag{4}$$

Then the gain ratio is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)} \tag{5}$$

Gini Index

The Gini index is a measure of the impurity of an environment.

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 \tag{6}$$

Then we want to maximize the difference between the Gini impurity of the original dataset and of the proposed split.

Tree pruning

The tree pruning here refers to removing the least reliable nodes in our tree. This can make the inference process faster for a single node.

Postpruning goes back to the original tree and removes entire subtrees that are not useful. All its children are removed and the node is replace with a leaf. The **cost-complexity** analysis tries to compare the misclassification rate for the regular node and the node if we were to apply a split.

We would mvoe up from a node, and using a special set called the **pruning set**, we could measure the error rate in a node. If the regular error rate is less than the error if we applied the pruning, we go ahead an apply the pruning. However, we would need another **pruning set** which is both independent of any training and testing set we use.

Pessimistic pruning foregos the pruning set and instead tries to balance the training set. Since the training set is biased by default, we would need to counter this by some factor α to get an accurate reading.

Two issues in decision trees are **repetition** and **replication**. The former occurs when a single attribute is checked various times along a path to the leaf node. The latter occurs when entire subtrees are repeated at different points in our tree.

Memory Issues

Since some of the datsets might be too learge to fit in main memory, we can instead try a couple of alternatives. **RainForest** creates an AVC set (Attribute-Value, Classlabel) at each node Then the AVC set for each node should be able to fit in main memory?

Another algorithm called BOAT, builds several smaller trees which all fit in main memory and then use that to build the large, main tree, although it might not be quite exactly the same. This turns out to be faster than RainForest by a good bit.

Bayes Classification

This type of classifier is based on Bayes' Theorem, and we try to predict the class of an item based on its attributes, which are considered independent of each other. The form of the theorem is

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(X)} \tag{7}$$

$P(H|\mathbf{X})$ refers to the **posterior proabability** of H, since we take its probability after we make some measurement X. Then $P(X)$ is the prior probability of the event; how often it happens on its own. For a given class C_i, the posterior probability $P(\mathbf{X}|C_i$ is generally pretty expensive to compute, so we could use an approximation given all the featues are independent from each other.

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \tag{8}$$

or, the probability of a given input given that it belongs to C_i.

However, it depends on whether the value is continuous or categorical. If it is categorical, we would have to estimate the items belonging to class C_i that have the given feature. If the value is categorical, we would have to assume the values are normally distributed by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{9}$$