Data Mining and Data Warehousing – Classification-Lazy Learners

Motivation
Lazy Learners are the most intuitive type of learners and are used in many practical scenarios. The reason of their popularity is that they do not require extensive time for training, rather they classify on the go.

Lazy Learners ppt (For your convenience you can get them inside Learn More Quadrant)

Learning Objectives
By the end of this module, the learner will be able to:

- Define lazy learners
- Compare different types of lazy learners
- Apply lazy learners on sample data
- Analyze the accuracy of various lazy learner algorithms
- Justify the correctness of lazy learners
- Modify lazy learners for different types of data

Time Required
240 mins

Concept
Lazy Learners

Data classification process involves mainly two steps which are:
- Building a model from a given set of training data
- Applying the model to a given set of testing data

Eager Learners like Bayesian Classification, Rule-based classification, support vector machines, etc. will construct a classification model before receiving new tuples, when a set of training tuples is given. In contrary to this, an opposite strategy is to delay the process of modelling the training data until it is needed to
classify the testing data. Techniques that employ this strategy are known as Lazy Learners.

A lazy learner simply stores the training data and only when it sees a test tuple starts generalization to classify the tuple based on its similarity to the stored training tuples. Lazy learners do less work while training data is given and more work when classification of a test tuple is given.

Instance-based learning uses specific training instances to make predictions without having to maintain an abstraction derived from data. Nearest-neighbor classification also does the similar technique. Hence nearest-neighbor classifiers are termed as instance-based learners. Instance-based learning algorithms require a proximity measure to determine the similarity or distance between instances and a classification function that returns the predicted class of a test instance based on its proximity to other instances.

Lazy learners can be computationally very expensive while doing classification or predictions which do not require any model building. In contrast, eager learners spend more time computing resources for model building. Once a model is built, classifying a test tuple is extremely fast. Also, the nearest neighbor classifiers make predictions based on their local information, whereas the decision tree classifiers attempt to find a global model that fits the entire test data. Because of the local-made decisions, nearest-neighbor classifiers are quite susceptible to noise.

There are certain drawbacks of lazy learners. They are:

- There is a chance that few test records may not be classified because they do not match any training tuples. Also, there is a chance to make wrong predictions unless the approximate proximity measure and data preprocessing steps are taken.

- They are well-suited to implement on parallel hardware because they require efficient storage techniques.

- Structure of the data cannot be well-retrieved from the lazy learners.

Besides several drawbacks, lazy learners are able to model complex decision spaces which are not easily describable by other learning algorithms. Let us consider two examples of lazy learners: K-Nearest-Neighbor classifier:
Nearest Neighbors are the training instances which are relatively similar to the attributes of the test example. Nearest neighbor classifiers compare a given test tuple with training tuples that are similar to it. A nearest neighbor classifier represents each training tuple as a data point in an n-dimensional space.

When given a test tuple, a k-nearest neighbor classifier searches the pattern space for the k training tuples that are closest to the test tuple. These k training tuples are the k “nearest neighbors” of the test tuple.

Euclidean distance is used to measure the “closeness”. Euclidean distance between two points \( X = (x_1, x_2, x_3, \ldots, x_n) \) and \( Y = (y_1, y_2, y_3, \ldots, y_n) \) is

\[
D_{\text{dist}}(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}
\]

In normal words, we calculate the closeness between two points in such a way that difference between two corresponding values of the attribute in tuple \( X \) and tuple \( Y \) and the difference is squared. Total squared differences of all attributes are summed up and the value is square-rooted. Typically before calculating the closeness, the values of the attributes are normalized. In general for normalization, minimax normalization is used.

The data point is classified based on the class labels of its neighbors. In the case where the neighbors have more than one label, the data point is assigned to the majority class of its nearest neighbors. When \( k=1 \), the test data point is assigned the class of the training tuple that is closest to it.

Algorithm
1. Let \( k \) be the number of nearest neighbors and \( D \) be the set of training tuples.
2. For each test example \( z = (x', y') \) do
3. Computed \( (x', x) \), the distance between \( z \) and every example, \( (x, y) \in D \).
4. Select \( D_z \subseteq D \), the set of \( k \) closest training examples to \( z \).

\[
y' = \arg \max_v \sum_{(x_i, y_i) \in D_z} I(v = y_i)
\]
5. End for

In the step-4, \( v \) is the class label, \( y_i \) is the class label for one of the nearest neighbors and \( I(.) \) is the indicator function that returns the value of 1 if its
argument is true and 0 otherwise. The algorithm is sensitive to the choice of k since every neighbor has the same impact on the classification. Hence to reduce the impact of k, we can add some weight according to the distance. Because of this, training tuples that are located far away from z have a weaker impact on the classification compared to those that are located close. Using the distance-weighted scheme, class label can be determined as:

\[ y' = \arg\max_v \sum_{(x_i, y_i) \in D_z} w_i \times I(v = y_i) \]

**Distance-Weighted Voting:**
where \( w_i = 1/d(x', x_i)^2 \)

Computing the distance between points of categorical attributes:
For given two points X and Y, a training tuple and a test tuple respectively, if there are some categorical attributes, then if both the values are same, the difference can be considered as 0. If the two are different, then the difference can be considered as 1. Many other methods are available which incorporate more sophisticated methods for various grading methods.

**Dealing with missing values:**
For given two points X and Y, a training tuple and a test tuple respectively, if in any of the points, for some attribute(s) there are missing values, and if the attribute is mapped to the range [0, 1]. For categorical attributes, we consider the difference as 1 if either one or both of the corresponding values of that attribute are missing. If the attribute is numeric, if both the points have missing values for the attribute, then the difference is taken as 1. If only one value is missing and the other is suppose v is present and normalized, then we can take the difference to be the greatest of \(|1 - v|\) and \(|v - 1|\).

**Determining the value of k:**
The value of k can be determined experimentally. Starting with k=1, error rate of the classifier is estimated for each value of k by incrementing it by 1. The k value which gives a minimum error rate can be selected as the appropriate value. In general, if the training tuples are larger in number, then the value of k should be larger. We should remember that if k is too small, then the nearest-neighbor classifier may be susceptible to over-fitting because of the noise in the training data. On the other hand, if k is too large, the nearest neighbor classifier may mis-
classify the test tuple because its list of nearest neighbor may include data points that are located far away from its neighborhood.

K-Nearest neighbor uses distance based comparisons that assigns equal weights to each attribute. So if we have noise or irrelevant attributes then it will have poor performance.

K-Nearest neighbor will be slow when classifying test tuples. If D is the training database with |D| tuples and k=1 then O(|D|) comparisons are required for classifying test tuples. If you want to decrease the number of comparisons then presort and arrange the stored tuples into search trees. In order to reduce the number of comparisons use either parallel implementation, partial distance calculations and editing the stored tuples. In partial distance we have to calculate the distance based on a subset of the n attributes. If this distance exceeds a given threshold then halt the computations of stored tuples. The editing method removes the useless training tuples. This method is also called pruning or condensing.

These classifiers produce arbitrarily shaped decision boundaries. This kind of boundaries provide a more flexible model representation compared to decision tree and rule based classifiers that are often constrained to rectilinear decision boundaries.

**Case-Based Reasoning:**

Base-based reasoning is the process of solving new problems based on the solutions of similar past problems. These classifiers use a database of problem solutions to solve new problems. Unlike nearest neighbor classifiers, which store the training tuples as points in Euclidean space, CBR stores the tuples for problem solving as complex symbolic description. Many applications of CBR can be seen in our day-to-day life. Applications are in Business to describe the product-diagnostic problems, in Medicine to identify the patient case histories to help diagnose and treat new patients, etc.

When a test tuple is given to classify it, a case-based reasoner will first check if an identical training case exists. If it is found, then the accompanying solution is returned. If no identical case is found, then the case-based reasoner will check for the training tuples which have identical components that are similar to the test tuple. If the identical training cases are considered as neighbors and these cases are represented as graphs, then subgraphs should be searched for, that are
similar to the subgraphs are involved in the test case. The case-based reasoner tries to combine the solutions of the neighboring training cases in order to propose a solution for the new case.

The case-based reasoner uses the background knowledge and problem solving strategies in order to propose a new feasible solution for the new case. But the biggest challenge is to identify a good similarity metric and suitable method for combining solutions. Cases that are redundant or have not proved useful can be discarded for improving the performance.