JakeCroninHW2: Decision Tree and Naive Bayesian Classifier

by Jake Cronin

Summary

A decision tree and a Naive Bayesian Classifier were implemented as separate java programs to classify data.

Given a list of classes and attributes, both algorithms learn to predict an item's class based on its attributes. Both algorithms report their accuracy based on the provided test dataset.

Decision Tree

Overview

Using the training data, the decision tree builds a data structure that resembles a dichotemous key. Essentailly, it constructs branching path with each branch representing a different value for an attribute. The branching continues until either all data in a branch is of the same class, or until there are no more attributes on which to divide the data. The predicted class for the end of a path is the majority class among the test data at the end of the branch.

Optimizing

To improve performance of the decision tree, it is important to branch on the most distinguishing attributes first. Before each split, in order to decide which attribute is the most distinguishing and should be used for the split, the gain ratio is calculated for each unused attribute. The unused attribute with the highest gain ratio is then used to split the branch.

To calculate gain ratio, the difference in entropy in the data with respect to the class before and after the split is divided by the the entropy in the data with respect to the attribute.

Performance

When training on the mushroom.train.dat dataset, my decision tree achieves 100% accuracy on the mushroom.test.dat dataset.

Naive Bayesian Classifier

Overview

The Naive Bayesian Classifier uses a bank of training data to calculate the probability of a given item being a part of a specific class based on its attributes. An item is assigned to the class that the classifier deems most probable.

Optimizing

One issue that the Naive Bayesian Classifier runs into is with zero probabilities. The probability of an item being in a class is the product of probabilities based on its attributes, so if a single attribute returns a zero probability, the resultant probabilty for the item beign in this class is zero. While this is mathematically correct, it can skew real world data.

Unless the training dataset is immensely rich, it is possible that item I of class C has an attribute value that is different from every item in class C in the training dataset. In this scenario, while item I may share many other attribues with the other training data in class C, the single outlier attribute would assign I to have a zero probability of being in class C. To handle this problem of zero probabilities, a laplacean correction can be used to assign low, non-zero, probabilities to these values on evaluation.

My Naive Bayesian Classifier has the option to evaluate data with or without a laplacian correction.

Performance

When training on the mushroom.train.dat dataset, my decision tree achieves 95.7% accuracy when using laplacian correction and 99.8% accuracy when not using laplacian correction.

This algorithm is also very fast. Training on mushroom.train.dat and evaluating the mushroom.test.dat dataset both with and without laplacian correction takes about 200ms.