CSCI 446 Artificial Intelligence Project 3 Design Report

Roy Smart Nevin Leh Brian Marsh

November 4, 2016

1 Introduction

Machine learning is a term referring to giving computers the ability to learn without explicit programming to do so. It is achieved by collecting large sets of data and analyzing them in order to find patterns that can be used in future actions. Machine learning can be implemented through a variety of algorithms. For this project, we will be focusing on four specific machine learning algorithms (MLAs): k-Nearest Neighbors, NaÃŕve Bayes, Tree Augmented NaÃŕve Bayes (TAN), and Iterative Dichotomiser 3 (ID3).

The algorithms will work to solve the problem of classification, by which new data points are classified into sets. Classification is achieved by observing the set membership of known data and making decisions based upon this knowledge. In this project we are asked to train our MLAs on five different datasets: the Wisconsin Breast Cancer Database, the Glass Identification Database, the Iris Plants Database, the Small Soybean Database, and the 1984 United States Congressional Voting Records Database. It is expected that different MLAs excel with different databases.

2 Datasets

2.1 Dataset Representation

We will define a *datum* to be a vector consisting of the class as the zeroth element and the associated attributes as the rest of the elements. Classes and attributes will be represented by an integer. Continuous data will be binned before it is inserted into each datum. The resolution of the bins will be a variable that will have to be tuned. Each dataset will be represented as a vector of datums.

2.2 Data Imputation

Imputation is the process of approximating missing values in the datasets. To our knowledge there is only one dataset that has real missing values: the Wisconsin Breast Cancer Database. The 1984 United States Congressional Voting Records Database appears to have missing values, but these can actually be interpreted as a stance on a particular issue. Since the breast cancer database has a small proportion of missing values, it is appropriate to simply eliminate datums with missing values. The authors assert that trying to train a MLA with imputed values would only create unnecessary bias in the network.

However, it is a common real-world problem to attempt to classify an unknown, incomplete datum, therefore we will perform imputation on the validation datasets. To approximate the missing values we will first try a hot-deck imputation, where missing attribute values of a given datum are constructed by selecting a random member of that datum's class and copying the value of the attribute. If the hot-deck is unsuccessful, we will attempt to fill in the missing values using a regression model developed in *Mathematica*.

2.3 Cross-validation

To partition the full datasets into test and training datasets, we will use 10-fold cross validation. This method partitions the data into ten *folds* and uses one fold for testing data and the remaining nine folds for the training dataset. This process is repeated nine more times until every fold has been used as a test dataset. We selected this method because it will allow the convergence measurement described in Section 5.3 to be applied over a larger range, which allows us to measure the rate of convergence for each MLA more accurately.

3 Machine Learning Algorithms

3.1 k-Nearest Neighbors

3.1.1 Description

k-Nearest Neighbors (k-NN) makes new classifications of data based upon the most-closely surrounding data points, with those closest given a distance metric having the most impact. The number of neighboring data points is denoted by k, the specific value of which heavily influences the effectiveness of the algorithm. Although various distance metrics are applicable to the k-Nearest Neighbors algorithm, we will be implementing the Minkowski distance metric, given by

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = \left(\sum_i |x_{j,i} - x_{q,i}|^p \right)^{1/p}$$

where we will take $p = 1$ (the Manhattan distance) for dimensionless attributes and $p = 2$ (the Euclidean Distance) for attributes with dimensions (such as length).

During the training phase, k-NN simply reads in all the datums from the training set and stores them. Then, during the testing phase k-NN can calculate the distance from the test datum to all the other datums in the training set, and selects the k nearest datums. The class prediction is then given by the majority class in the set of nearest datums.

3.1.2 Design Implications

This algorithm should be very simple to implement. Since most of the calculations for classification are done during the testing phase, the algorithm simply needs a way to store the data, calculate distances, and store these distances in a sorted list. We will use the standard library C++ sort routine to sort datums by distance.

3.2 Naive Bayes

3.2.1 Description

Naive Bayes uses the principle of Bayesian learning to solve the classification problem. This algorithm is said to be naive because it considers the attributes to be conditionally independent when performing classifications. Under this scheme, the probability distribution of each class C is given the set of attributes $x_1, ..., x_n$ is written by [Russel and Norvig, 2010] as

$$\mathbf{P}(C|x_1, ..., x_n) = \alpha \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C) \tag{1}$$

where, using the maximum likelihood hypothesis, $\mathbf{P}(C)$ prior probability of the class C in the training dataset, $\mathbf{P}(x_i|C)$ is the likelihood of attribute x_i given C, and α is a normalization constant. To find $\mathbf{P}(C)$ we simply

say it is equal to the proportion of C observed in the training dataset. Likewise we can calculate the likelihood using

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

where the probabilities are calculated using the number of occurrences in the training dataset. So, using equation 1 we can calculate the probability of a particular set of attributes being classified as class C. Then, we simply select the class with the largest probability for our prediction. In this way, naive Bayes can make predictions using only the probability ratios constructed from the training dataset. The probability distribution in the naive Bayes algorithm can be viewed as a Bayesian network, shown in Figure 1 where each attribute is only connected to the class, and there are no relationships between attributes.

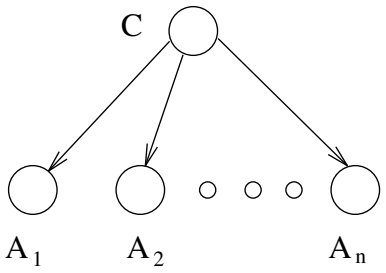

Figure 1: *An example of a Bayesian network for naive Bayes. The root C is the class, the leaves $A_1, ..., A_n$ are the attributes [Friedman et al., 1997]*

3.2.2 Design Implications

This algorithm should be very straightforward to implement. During the training phase, we simply loop through the dataset and tally how many times each class takes on every value of each attribute. During the testing phase, the algorithm uses the equations described in Section 3.2.1 to select the most probable class.

3.3 TAN

3.3.1 Description

Tree-Augmented Naive Bayes (TAN) is an extension to the naive Bayes algorithm described in Section 3.2. First described by [Friedman et al., 1997], TAN relaxes the assumption that the attributes are conditionally independent and allows each attribute to depend on only one other attribute. It accomplishes this by constructing a fully-connected, undirected graph out of the attributes in the Bayesian network for the naive Bayes algorithm (Figure 1). TAN then assigns a weight to each connection in the graph using the *mutual information function*, given by

$$I_P(\mathbf{X}; \mathbf{Y}) = \sum_{\mathbf{x},\mathbf{y}} P(\mathbf{x}, \mathbf{y}) \log \left(\frac{P(\mathbf{x}, \mathbf{y})}{P(\mathbf{x})P(\mathbf{y})} \right).$$

Applying the this weight to the edges allows us to construct a minimum spanning tree of the graph. We can then make the spanning tree directed by choosing a root node and setting the direction of all edges to be outward from it. The result is exemplified in Figure 2. Using this representation, we can calculate the analog of equation 1, the probability distribution using the expression given by [Zheng and Webb, 2010]

$$\mathbf{P}(C|x_1, ..., x_n) = \alpha \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C, \pi(x_i)) \tag{2}$$

where $\pi(x_i)$ is the parent of the attribute x_i in the directed minimum spanning tree. Then, using a similar process to that described in Section 3.2 we can classify a set of attributes by selecting the class with the highest probability calculated from equation 2.

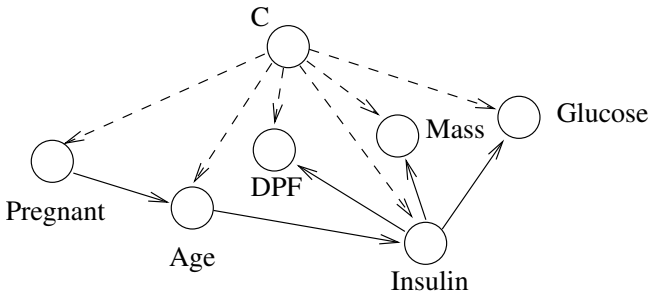

Figure 2: *An example of the directed minimum spanning tree produced by TAN. Dashed lines represent the original naive Bayes representation and the solid lines represent the spanning tree [Friedman et al., 1997].*

3.3.2 Design Implications

To develop the TAN algorithm, we will need to implement a tree structure to represent the naive Bayesian network and the associated minimum spanning tree. To find the minimum spanning tree we will need an implementation of Prim's algorithm. If appropriate we will use the implementation of Prim's algorithm provided in the Boost C++ libraries, otherwise we will develop our own implementation of Prim's algorithm. Once the minimum spanning tree is constructed we can appropriate the probability calculations developed for naive Bayes to calculate the value of equation 2 for classification.

3.4 ID3

3.4.1 Description

Iterative Dichotomiser 3 (ID3) is an algorithm that creates a decision tree and uses it to make inductive inferences about a set of data. The essence of this approach is to create a decision tree that correctly classifies all the individuals in the training data. Since the goal of this solution is to generalize classification, it makes sense to use the smallest tree that can still correctly classify all of the training data. The naive way of finding the shortest tree would be to generate all possible trees and select the shortest one according to[Quinlan, 1986]. The problem that one encounters is the very large, but finite, number of decision trees that are capable of correctly classifying all data. The computation required to do this is unfeasible large so a different approach must be used. ID3 provides a good, but not optimal, solution to this problem.

ID3 is an iterative process that generates a short tree by splitting the tree one attribute at a time. The first attribute to be used is the one with the most information gain. Information gain can described as how much entropy the system lost by splitting on an attribute. The equation for entropy is as follows:

$$H(S) = \sum_{i=1}^{n} \left(-p_i \log_2 p_i\right).$$

where p_i is the proportion of the number of datums in class i to the total number of datums in the set S. To use this equation the entropy of the whole system is determined first. Then the sum of the entropy for each new branch is calculated and subtracted from the total entropy. We try to branch on each attribute that has not been used yet. Since the goal is to maximize this difference so the tree will be small and more general, we choose the attribute that resulted in the greatest reduction of entropy. The reduction of entropy is defined as information gain.

The rest of the algorithm is just iterating through as the tree is made and redo the calculation on an attribute that has not been split yet. The tree is finished when all leaf nodes are a single class or when attributes to split on have run out. In this case each leaf node that does not have a class is assigned the most common class in that leaf.

One bias that ID3 introduces is the fact that attributes that have a large amount of possibilities will naturally have a higher gain and, therefore, will be chosen first. To combat this we will use gain ratio instead.

Gain ratio takes into account the number of possibilities for each algorithm.

Another problem is that the tree still might not be general enough and could be trimmed down further. To help with this we will implement reduced error pruning. The general idea is to fully create a tree and then prune branches while checking if the missing branch still results in the same classification for the pruning set. The pruning set is a subset of the of the training set that is not used in the normal training process.

3.4.2 Design Implications

This algorithm will affect the structure of our code greatly. A tree structure will have to be created that can contain subsets of the training set and also leaf values for when a classification is found. In addition the training set is going to have to be divided to get a validation set to perform reduced error pruning. The main body of the learning part of the algorithm will be in a while loop with the conditions above. The search part of the algorithm will just consist of transversing the tree according to the attributes belonging to the query.

4 Software Architecture

To reliably compare the four machine learning algorithms on five different datasets, we will use inheritance and C++ virtual classes that allows us to perform algorithm- and dataset-independent analyses on the MLAs. The inheritance structure is described in Figure 3. Each of the datasets will be encapsulated in an associated

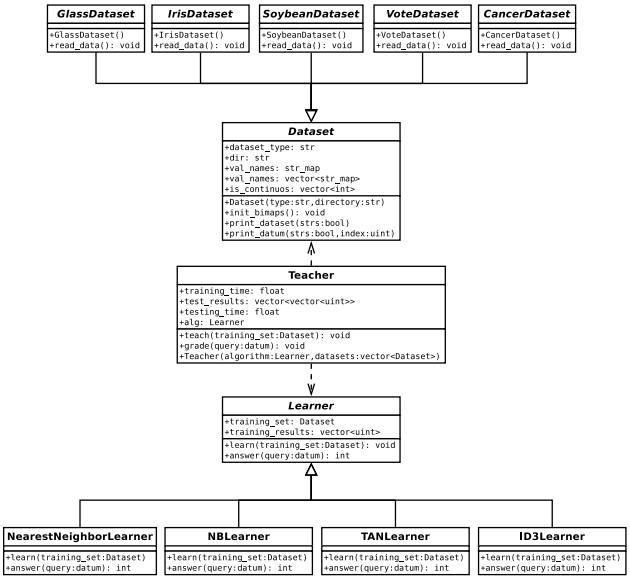

Figure 3: *A high-level description of the program architecture in UML.*

class, e.g. the Glass Identification Database will be stored in a class denoted `GlassDataset`. This class will inherit from the virtual class `Dataset`, which defines a dataset as a vector of integers, and also a set of associated functions for selected subsets of the dataset, printing the dataset, etc.

We will also implement a virtual class called `Learner`, which operates on a single training dataset. `Learner` defines the two functions `learn()` and `answer()` which will be used to perform training and testing respectively. Therefore, each of the four MLAs will inherit from learner and implement the two functions `learn()` and `answer()` based on the algorithm descriptions in Section 3.

To relate MLAs to datasets, we will define a `Teacher` class that is responsible for training and validating the MLAs, and measuring their performance. This class will also be responsible for performing the convergence measurement described in Section 5.3.

Not depicted in Figure 3 is a class called `Validation`. This class will be responsible for the 10-fold cross validation described in Section 2.3. `Validation` will partition the datasets into training and testing datasets and provide them to the `Teacher` to train and validate the MLAs.

5 Experiment Design

5.1 Algorithm Accuracy

5.1.1 Precision

Precision is the proportion of results that are relevant according to [Russel and Norvig, 2010]. This is better described as the fraction of the number of correct classifications for a class and the total amount of datums that were classified as that class. This process will have to be done one class at a time and then averaged. This is a good metric to look at because it tells you the proportion of correct classifications to all classifications. To do this we will have to save the true positive, false positive, true negative, and false negative counts for each class as validation occurs.

5.1.2 Recall

Recall is the proportion of relevant datums that that are correctly classified. This is better described as the fraction of the number of correct classifications for a class and the total number of datums that should have been classified as that class. This metric seems similar to precision but is useful because it tells you the proportion of correct classifications to the actual number of datums in that class. In essence it tells you how many classifications you missed rather than how many were correct. Again the process will have to be done one class at a time and then averaged.

5.2 Algorithm Time-Complexity

For algorithm time complexity we will use a simple timer to get an estimation of the run time. Since a timer sometimes is not necessarily a good metric for time-complexity we will also attempt to determine the time complexity by inspection of each algorithm.

5.3 Algorithm Convergence

Algorithm Convergence is a metric that shows how well the algorithm performs with different amounts of training data. The idea is to incrementally add data to the training set and see how well validation performs at each increment. This is an interesting metric to have because it gives us an indication as to how much training data is needed for each algorithm. This will add a step to our validation process since we will have to validate every time we increment the amount of training data.

6 Summary

Our goals with this project is to correctly implement the above algorithms and train each algorithm with the data sets provided. We will then see how well the algorithm performs in terms of time complexity, convergence, precision, and recall. We hope to determine what datasets each algorithm excels on and come to a reasonable conclusion as to why this is the case.

References

[Friedman et al., 1997] Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29(2):131–163.

[Quinlan, 1986] Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Russel and Norvig, 2010] Russel, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson Education, Upper Saddle River, New Jersey 07458, 3rd edition.

[Zheng and Webb, 2010] Zheng, F. and Webb, G. I. (2010). *Tree Augmented Naive Bayes*, pages 990–991. Springer US, Boston, MA.