CSE 601- Data Mining and Bioinformatics, Fall 2013

Project 3

Classification Algorithms

(Naïve Bayes, k-Nearest Neighbor, Decision Tree & Random Forest)

Submitted by:
Akshay Viswanathan
Aman Sharma
Vinay Amrit

Introduction

In this project, we are required to implement three classification algorithms which are **Nearest Neighbor, Decision Tree,** and **Naïve Bayes**. We are also required to implement **Random Forests** based on implementation of Decision Tree. A classification technique is a systematic approach to building classification models from an input dataset. Each technique employs a learning model that best fits the relationship between the attribute set and the class label of the input data. Therefore the key objective of the project is to create a good learning model which is generalized, and so predict the class labels of previously unknown records.

Algorithms

1. *Naïve Bayes*:

Algorithm Basics:

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. It assumes that the presence or absence of a particular feature is unrelated to the presence or absence of any other feature, given the class variable. Parameter estimation for naive Bayes models uses the method of maximum likelihood. Naive Bayes classifiers can handle an arbitrary number of independent variables whether continuous or categorical.

Given a set of variables, X = {x1,x2,x...,xd}, we want to construct the posterior probability for the event Cj among a set of possible outcomes C = {c1,c2,c...,cd}. In a more familiar language, X is the predictors and C is the set of categorical levels present in the dependent variable. Using Bayes' rule:

$$p(C_j \mid x_1, x_2, \ldots, x_d) \propto p(x_1, x_2, \ldots, x_d \mid C_j) p(C_j)$$

where p(Cj | x1,x2,x...,xd) is the posterior probability of class membership, i.e., the probability that X belongs to Cj. Since Naive Bayes assumes that the conditional probabilities of the independent variables are statistically independent we can decompose the likelihood to a product of terms:

$$p(X \mid C_j) \propto \prod_{k=1}^{d} p(x_k \mid C_j)$$

and rewrite the posterior as:

$$p(C_j \mid X) \propto p(C_j) \prod_{k=1}^{d} p(x_k \mid C_j)$$

Using Bayes' rule above, we label a new case X with a class level Cj that achieves the highest posterior probability.

Flow of Code:

readFiles()- Reads the txt files and puts the data into the Vector<vector>. And put the data from files into dataset1 and dataset2. For string value "PRESENT" it adds 1, and for "ABSENT" it adds 0 of type double.

main() – Calls the process(), which is the main function to start evaluating the algorithm. It is called twice for each dataset.

We need to find the posterior distribution for the prediction of the sample belonging to Label 0 or 1 given the label. We will find the parameters mean and variance using the training data.

We proceed by first using Cross Validation to partition the dataset into the K values as per the input and tag the dataset as Testing and Training data-set.
samplingClassification() and dimensionClassification()- We feed the training data-set into the algorithm and evaluate the parameters mean and covariance for each dimension in each dataset and save it in a data structure.

calcPosterior()- We use these covariance and mean values to compute the posterior probabilities using the Normal distribution.

accuracy()- we then compare the posterior probabilities of the 2 labels and infer that the label that has the higher probability is the one the sample belongs to. At the end of the algorithm we compute the accuracy, precision, recall and f-measure for all the predicted values with respect to the given values and average the values to provide the result.

Parameters:

Zero probability:

The Laplacian correction (or Laplace estimator) is a way of dealing with zero probability values. problem.We assume that our training set is so large that adding one to each count that we need would only make a negligible difference in the estimated probabilities, yet would avoid the case of zero probability values.

Pros:

1. Bayesian classifier is a statistical classifier, and predicts class membership probabilities.
2. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.
3. Comparable in performance to Decision Trees.

Cons

1. *Doesn't work with co-related data. Most real-life data are correlated so it doesn't serve the best purpose.*

2. *Nearest Neighbor*:

Algorithm Basics:

The ***k*-nearest neighbor's algorithm** (*k*-NN) is a non-parametric method for classification. That predicts objects' "values" or class memberships based on the *k* closest training examples in the feature space. *K-NN* is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. The *k*-nearest neighbor algorithm is amongst the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

Algorithm: The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, *k* is a user-defined constant, and an unlabeled vector is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point. A commonly used distance metric for continuous variables is Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

To classify an unknown record:

1. Compute distance to other training records
2. Identify *k* nearest neighbors
3. Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)
4. Since *KNN* predictions are based on the intuitive assumption that objects close in distance are potentially similar, it makes good sense to discriminate between the *K* nearest neighbors when making predictions, i.e., let the closest points among the *K* nearest neighbors have more say in affecting the outcome of the query point. This can be achieved by introducing a set of weights *W*, one for each nearest neighbor, defined by the relative closeness of each neighbor with respect to the query point. Thus determine the class labels according to
   Weight factor, w = 1/d^2

Flow of Code:

readFiles()- Reads the txt files and puts the data into the Vector<vector>. And put the data from files into dataset1 and dataset2. For string value “PRESENT” it adds 1, and for “ABSENT” it adds 0 of type double.

main() – Calls the process(), which is the main function to start evaluating the algorithm. It is called twice for each dataset.

Process(Vector<Vector> dataset)- Divides the data into Training and Test part. We are taking 75% of data as test data and 25% as training data at first. It then process the test and train data and displays the result.
Process(test,train)- It normalizes the data by calling MinMaxNormalize() method on both datasets for training and testing data. It also calculates the distance between the rows.

Classify()- classifies the class label according to count value and also by weight. And then assigns the class label for test data.

Distance()- Calculates the eculidean distance.

MinMaxNormalize()- Normalizes the data in each column according to minimum and the maximum value in each column.

CrossValidate()- It performs the 10 fold cross validation on the data, by setting one part as test data , and rest as train data at each time. The data is divided in equal parts, by adding the leftover data buffer into each partition.

Parameters:

The best choice of *k* depends upon the data; generally, larger values of *k* reduce the effect of noise on the classification, but make boundaries between classes less distinct.

Estimation through Cross validation: The general idea is to divide the data sample into a number of *v* folds randomly. For a fixed value of *k*, we apply the *KNN* model to make predictions on the *v*th segment (i.e., use the *v-1* segments as the examples) and evaluate the error. The most common choice for this error for classification it is most conveniently defined as the accuracy (the percentage of correctly classified cases). This process is then successively applied to all possible choices of *v*. At the end of the v folds (cycles), the computed errors are averaged to yield a measure of the stability of the model .The above steps are then repeated for various *k* and the value achieving the highest classification accuracy is then selected as the optimal value for *k* which in our case comes out to be k=5.

Pros:

1. Simplest of all to evaluate.
2. Do not need training of data.

Cons:

1. The accuracy of the *k*-NN algorithm can be severely degraded by the presence of noisy or irrelevant features.
2. It does not build models explicitly.
3. Different from eager learners such as decision tree induction.
4. Classifying unknown records are relatively expensive.

3. *Decision Trees*:

Algorithm Basics:

Decision tree learning is a method commonly used in data mining. The goal is to create a model that predicts the value of a target variable based on several input variables. Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

The basic algorithm for constructing decision tree is as follows.

```
TreeGrowth ($E$, $F$)
if stopping_cond($E$,$F$) = $true$ then
    $leaf$ = createNode().
    $leaf.label$ = Classify($E$).
    return $leaf$.
else
    $root$ = createNode().
    $root.test\_cond$ = find_best_split($E$, $F$).
    let $V = \{v | v$ is a possible outcome of $root.test\_cond\}$.
    for each $v \in V$ do
        $E_v = \{e \mid root.test\_cond(e) = v$ and $e \in E\}$.
        $child$ = TreeGrowth($E_v$, $F$).
        add $child$ as descendent of $root$ and label the edge ($root \rightarrow child$) as $v$.
    end for
end if
return $root$.
```

Flow of Code:

readFiles()- Reads the txt files and puts the data into the Vector<vector>. And put the data from files into dataset1 and dataset2. For string value “PRESENT” it adds 1, and for “ABSENT” it adds 0 of type double.

main() – Calls the process(), which is the main function to start evaluating the algorithm. It is called twice for each dataset.

Process(Vector<Vector> dataset)- Divides the data into Training and Test part. We are taking 50% of data as test data and 50% as training data at first. It then process the test and train data and displays the result.
Process(test,train,Boolean, int)- It converts both datasets for training and testing data into matrix. It then calls createTree() to build the tree, and calls VaidateRow() on test data to classify the test data set class labels.

ValidateRow()- Classifies the test data set rows according to the decision tree and assigns the class labels.

CreateNode()- Creates the nodes according to the checking(split) condition based on the parent node. It first creates the root node of the tree, by evaluating the Sort () & InternalGini() methods , calculating the best splitting point in the dataset. Its been recursively called to build the decision tree in createTree().

InternalGini()- It calculates the gigni value for each column and then find the minimum value for each column and stores it into the ArrayList.

Gini()- It calculates the gini and the gini split for the column(attribute).

CrossValidate()- It performs the 10 fold cross validation on the data, by setting one part as test data , and rest as train data at each time. The data is divided in equal parts, by adding the leftover data buffer into each partition.It then performs cross validation on decision tree and prints the average Accuracy, Precision, Recall, F-Measure.

BinaryTree.java- Stores the structure of the decision tree, and creates node through addNode() method by comparing the node key(split) value.

Parameters:

Splitting Data based on Gini impurity:
Used by the CART (classification and regression tree) algorithm, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. Gini impurity can be computed by summing the probability of each item being chosen times the probability of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

We used Gini as the measure of splitting the continuous data given in the dataset. The split position is determined on each node as, by sorting the data based on, each, individual column to determine its min Gini split value by finding all gini value between each change in dataset value. And then compare all the values to find the column which has the minimum gini split value, this column is then used as the splitting attribute.

$$\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2,$$ where P(i|t) is relative frequency of i at node t.

Pros:

1. Simple to understand and interpret
2. Requires little data preparation.
3. Able to handle both numerical and categorical data.
4. Possible to validate a model using statistical tests
5. Robust and Performs well with large datasets.

Cons:

1. The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts.
2. Mechanisms such as pruning are necessary to avoid overfitting.

4.Random Forests:

Algorithm Basics:

The random forest starts with a "decision tree" which, in ensemble terms, corresponds to our weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets.

The random forest takes this notion to the next level by combining trees with the notion of an ensemble. Thus, in ensemble terms, the trees are weak learners and the random forest is a strong learner. Here is how such a system is trained; for some number of trees *T*:

1. Sample *N* cases at random with replacement to create a subset of the data (see top layer of figure above). The subset should be about 66% of the total set.
2. At each node:
    1. For some number *m* (see below), *m* predictor variables are selected at random from all the predictor variables.
    2. The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.
    3. At the next node, choose other m variables at random from all predictor variables and do the same.

Depending upon the value of *m*, there are three slightly different systems:

- Random splitter selection: *m* =1
- Breiman's bagger: *m* = total number of predictor variables
- Random forest: *m* << number of predictor variables. Brieman suggests three possible values for m: ½√*m*, √*m*, and 2√*m*

Running Random Forest. When a new input is entered into the system, it is run down all of the trees. The result may either be an average or weighted average of all of the terminal nodes that are reached, or, in the case of categorical variables, a voting majority.

Flow of Algorithm:

readFiles()- Reads the txt files and puts the data into the Vector<vector>. And put the data from files into dataset1 and dataset2. For string value "PRESENT" it adds 1, and for "ABSENT" it adds 0 of type double.

main() – Calls the process(), which is the main function to start evaluating the algorithm. It is called twice for each dataset.

Process(Vector<Vector> dataset)- Divides the data into Training and Test part. We are taking 25% of data as test data and 75% as training data at first. It then process the test and train data and displays the result. For each Tree in the random forest it divides the data with random and send it to each tree, and also selects the number of random attributes for each node for each tree.
It then calls all the methods used to create the decision tree, with a Boolean label passed as "True" for Random Forest.

ValidateRow()- Classifies the test data set rows according to the decision tree and assigns the class labels according to majority voting amongst all trees.

Parameters:

For each tree

- We chose training set by choosing *N* times (*N* is the number of training examples) with replacement from the training set , where N is number of trees in our case we used N=5.
- For each node, randomly choose *m* features and calculate the best split we chose m=10

Pros:

1. It is unexcelled in accuracy among current algorithms.
2. It runs efficiently on large data bases.
3. It can handle thousands of input variables without variable deletion.
4. It gives estimates of what variables are important in the classification.
5. It converts a group of "weak learners" together to form a "strong learner.
6. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
7. It has methods for balancing error in class population unbalanced data sets.
8. Generated forests can be saved for future use on other data.

Results:

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

Accuracy= (a+d)/(a+b+c+d)

Recall= a/(a+b)

Precision = (a)/(a+c)

F-Measure= (2a)/(2a+b+c)

Naïve Bayes

For 10 fold Cross validation

	Dataset 1	***Dataset 2***
Accuracy	0.9443	0.6623
Precision	0.9364	0.6640
Recall	0.9739	0.9768
F-Measure	0.9403	0.6631

K –Nearest Neighbor -By Count

	Dataset 1	***Dataset 2***
Accuracy	0.961	0.671
Precision	0.977	0.625
Recall	0.905	0.341
F-Measure	0.94	0.441

For 10 fold Cross validation

	Dataset 1	***Dataset 2***
Accuracy	0.9701	0.6589
Precision	0.9814	0.50939
Recall	0.9391	0.4067
F-Measure	0.95879	0.4391

K –Nearest Neighbor -By Weight

	Dataset 1	***Dataset 2***
Accuracy	0.961	0.678
Precision	0.977	0.538
Recall	0.905	0.489
F-Measure	0.94	0.503

For 10 fold Cross validation

	Dataset 1	***Dataset 2***
Accuracy	0.9701	0.671
Precision	0.9814	0.526
Recall	0.9391	0.487
F-Measure	0.95879	0.494

Decision Tree

	Dataset 1	***Dataset 2***
Accuracy	0.915	0.636
Precision	0.874	0.75
Recall	0.876	0.038
F-Measure	0.875	0.0738

For 10 fold Cross validation

	Dataset 1	***Dataset 2***
Accuracy	0.939	0.636
Precision	0.916	0.459
Recall	0.929	0.439
F-Measure	0.924	0.438

Random Forests

	Dataset 1	***Dataset 2***
Accuracy	0.951	0.652
Precision	0.977	0.568
Recall	0.875	0.467
F-Measure	0.923	0.512

References:

1. http://cis.poly.edu/~mleung/FRE7851/f07/naiveBayesianClassifier.pdf
2. http://en.wikipedia.org/wiki/Naive_Bayes_classifier
3. http://www.statsoft.com/Textbook/
4. Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, Elsevier 2006, ISBN 1558609016. This part of the lecture notes is derived from chapter 6.4 of this book.
5. http://citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics/
6. http://www.statsoft.com/textbook/k-nearest-neighbors.