CSE 601: Data Mining and Bioinformatics

Project 3: Classification Algorithms

Submitted by:

Rahul Mallu (rmallu@buffalo.edu) - 50245594

Sai Yaswanth Vitta (saiyaswa@buffalo.edu) – 50246600

Santhosh Phani Vishnu Aita (saita@buffalo.edu) – 50246573

Introduction:

Classification is a supervised learning algorithm where the training data is accompanied by the labels indicating the class of the observation and the goal is to assign the previously unseen records to a class as accurately as possible. The main process of the classification is to build a classifier and using the same for classification.

- To implement three classification algorithms namely Nearest Neighbor, Decision Tree, and Naïve Bayes.
- To implement Random Forest and boosting on the implementation of Decision tress
- To perform a 10-fold Cross Validation to evaluate all the performance metrics i.e. Accuracy, Precision, Recall, F1-score of all the methods.

K Nearest Neighbors:

K-Nearest Neighbors is an essential classification algorithm in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. KNN has been used in statistical estimation and pattern recognition already from earlier years as a non-parametric technique.

Algorithm:

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor. It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables, the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

Implementation:

- We normalize our data in an array using z-score normalization. The formula for this is as shown below:

$$Normalized(ei\ ) = \frac{ei - E}{std(E)}$$

- For easier estimation, we normalize the data to make scores on the variables comparable.

- We perform 10-fold cross validation on our data set. One round of cross validation involves partitioning the data into subsets, where analysis is performed on one set of subsets (training set) and validate analysis on the rest (testing set). We perform this for multiple rounds with different partitions to reduce variability.
- For each record in the testing set we find k records or data points in the training set that are the closest to it. These points can be considered the record's neighbors.
- Prediction is done by assigning the majority class label these neighbors corresponding to the test record.
- Euclidean distance is used to measure the distance and identify the neighbors.
- The value of k can be varied till we get maximum accuracy and once the predicted labels are found we compare it with the true labels and calculate various measures.

Pros and Cons of K Nearest Neighbors:

Pros:

- K-NN algorithm is very simple to understand and equally easy to implement.
- It does not explicitly build any model, it simply tags the new data entry-based learning from historical data. New data entry would be tagged with majority class in the nearest neighbor.
- K-NN can be used both for classification and regression problems.
- K-NN is a non-parametric algorithm which means there are assumptions to be met to implement K-NN.

Cons:

- k-NN classifiers are lazy learners i.e. It does not build models explicitly.
- Classifying unknown records are relatively expensive
- It has Scaling issues - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes

Choice of Parameters:

- Choosing the optimal value of k is best done by inspecting data at first.Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. The optimal K for most datasets has been between 3-10. After running the code several times for different values of k, k=5 was observed to be an optimal value.

Results:

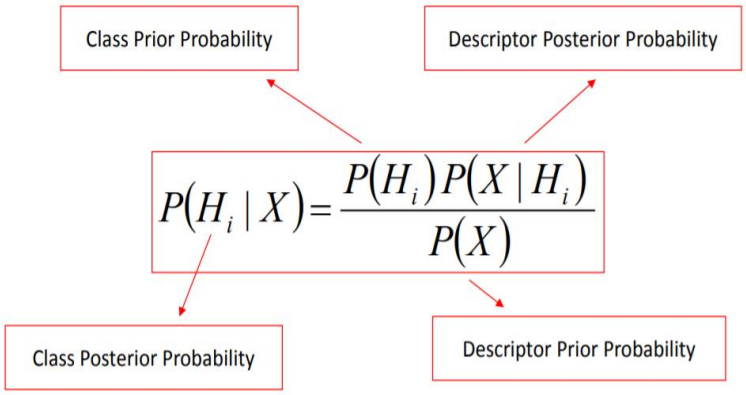
KNN Classification				
Dataset	Accuracy	Precision	Recall	F1 - measure
Project3_dataset1.txt	94.721177944	95.438992738	89.702247765	92.412714461
Project3_dataset2.txt	68.626271970	57.924908424	40.878908372	47.493548226

Naïve Bayes:

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Even if these features depend on each other or upon the existence of the other features, all these properties independently contribute to the probability.

Implementation:

- Split the data into two halves based on class labels
- Calculate the mean and standard deviation for each numerical attribute of both the splits of training data.
- For each test sample we calculate the 'Class Posterior Probability'.



$P(H_i)$ = (no of samples of class H_i)/ (Total length of training data) = Class Prior Probability
P(X|Hi) (for numerical attribute) = Gaussian_Pdf(X,mean,std)
P(X|Hi) (for categorical attribute) = $n_{i,j}$ /ni
Where $n_{i,j}$ is number of training examples in class C_i having value x_j for attribute A_j
P(X) = n_j /n_i
Where n_j is number of training examples having value x_j for attribute A_j and n is the total number of training examples

- Based on which Class Posterior Probability is greater we label the test sample accordingly. i.e. if P(0|X) > P(1|X) we label the test sample as 0.

Pros and Cons of Naïve Bayes:

Pros:

- It is simple to implement, computationally fast and works well with high dimensions
- Its easily trained, even with a small data set and insensitive to irrelevant features
- It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- Relies on independence assumption and will perform badly if this assumption is not met
- Naive Bayes considers that the features are independent of each other. However, in real world, features depend on each other.

Results:

Naïve Bayes				
Dataset	Accuracy	Precision	Recall	F1 - measure
Project3_dataset1.txt	0.934899749	0.917870936	0.904442635	0.910031838
Project3_dataset2.txt	0.707724329	0.575085073	0.620701549	0.591568145

Decision Tree:

Decision Tree is a supervised classification algorithm where a decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. It is on of the predicative modelling approaches used in data mining and machine learning.

Algorithm:

Firstly, a decision tree is built using the training data. And then the decision tree is used for prediction. To build a decision tree, we calculate the gini and information gain of each attribute and split using the attribute which has the maximum information gain. Then this process is recursively done till we get terminal nodes. To predict the class of a test data, we traverse through the decision tree going on the right if the test data has a value greater than the splitting value for that attribute otherwise going to the left if it has a lesser value. We continue this process until we get to the terminal node. The label of terminal node is the class label for test data.

Implementation:

- We build the tree in top down manner for splitting.
- For the current training data set, implement a binary split of the data as follows:
    - For each attribute, iterate through each value of the attribute and choose the value which gives the lowest value of the cost function.
    - The cost function used in Gini Index is as follows:

$$\left(1 - sum\left(p\left(\frac{c}{t}\right) * p\left(\frac{c}{n}\right)\right)\right) * (splitSize/total\_numberOfRecords)$$

    Where $p\left(\frac{c}{n}\right)$ is the relative frequency of the records belonging to class c for all the records at node n.
- At each binary split, create an object of class Node which contains the index of the split attribute and value of the attribute at which split is done.
- If the split attribute is numerical:
    - All the training records with the split attribute value less than or equal to the split value is assigned to the left child node and the rest of records to the right child node of the current node.
- If the split attribute is categorical:
    - All the training records with the split attribute value equal to the split value are assigned to the left child node and the rest of records to the right child node of the current node

- Build the rest of the tree recursively by iterating the above steps for the left and right child.
- Stop the tree from growing when the number of records in either of left or right child becomes zero. Convert both the children into a terminal node and assign a final class label to both according to the most common class label out of all the records at that node.
- Recursively navigate through each node of the tree for classifying each test data, going to the left.

Pros and Cons of Decision Trees:

Pros:

- Ability of selecting the most discriminatory features.
- Comprehensibility so that can be used in Rule Generation problem
- Data classification without much calculations
- Dealing with noisy or incomplete data
- Handling both continuous and discrete data (you must choose proper algorithm)

Cons:

- They are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.
- They are often relatively inaccurate.
- Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.
- The high classification error rate while training set is small in comparison with the number of classes

Results:

For number of trees = 5

Decision Tree				
Dataset	Accuracy	Precision	Recall	F1 - measure
Project3_dataset1.txt	0.9308	0.8926	0.9364	0.9127
Project3_dataset2.txt	0.5971	0.4218	0.4645	0.4343

Random Forest:

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks.

Algorithm:

- Choose *T* number of trees to grow. Choose *m<M* (M is the number of total features)
- number of features used to calculate the best split at each node (typically 20%)

For each tree-

- Choose a training set by choosing *N* times (*N* is the number of training examples) with replacement from the training set
- For each node, randomly choose *m* features and calculate the best split
- Use majority voting among all the trees

Implementation:

- Predefine the initial parameter T as the number of trees to grow.
- Predefine the initial parameter m, as the number of attributes chosen randomly to calculate the best split attribute for each node. We have chosen m as the square root of the number of attributes in the data.
- Let N be the size of original training data. For each tree, perform the following:
    - Generate a training set of size N by randomly choosing training records from the original training data with replacement.
    - For the creation of each node of the decision tree, select m attributes randomly and calculate the best split attribute from those m attributes.
- To classify each test data, make a prediction on each of the tree formed and assign the test data to the class label which is returned majority of the times among all the trees.

Pros and Cons of Random Forest:

Pros

- It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
- It runs efficiently on large databases and has reduced variance (relative to regular trees)
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets

Cons:

- Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.
- Unlike decision trees, the classifications made by random forests are difficult for humans to interpret.
- Not as easy to visually interpret

Results:

For number of trees = 5

Random Forest				
Dataset	Accuracy	Precision	Recall	F1 - measure
Project3_dataset1.txt	0.9514	0.9611	0.9110	0.9350
Project3_dataset2.txt	0.6471	0.4976	0.4143	0.4294

Boosting:

The term 'Boosting' refers to a family of algorithms which converts weak learner to strong learners. Boosting is an ensemble method for improving the model predictions of any given learning algorithm. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor.

Algorithm:

- Initially, set uniform weights on all the records

At each round,

- Create a bootstrap sample based on the weights
- Train a classifier on the sample and apply it on the original training set
- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

- For classifier i, its error is:

$$\varepsilon_i = \frac{\sum_{j=1}^{N} w_j \delta(C_i(x_j) \neq y_j)}{\sum_{j=1}^{N} w_j}$$

where w_j = weight of datapoint

- The classifier's importance is represented as:

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

where ε_i = The sum of all the misclassified weights / Total sum of the weighted matrix

- The weight of each record is updated as:

$$w_j^{(i+1)} = \frac{w_j^{(i)} \exp\left(-\alpha_i y_j C_i(x_j)\right)}{Z^{(i)}}$$

i.e. if the predicted label and original label is the same then yiCi(xj) is updated as 1 else for misclassified label its updated as -1

- If the error rate is higher than 50%, start over

Final prediction is weighted average of all the classifiers with weight representing the training accuracy

Implementation:

- Assign an initial weight of (1/number of records) to all the training records.
- Repeat the below until certain number of trees (5) are grown:
- Create a bootstrap sample randomly from the original training data based on the current weights of the training records.
- Train a Decision Tree model on the sample and apply it on the original training data.
- Calculate the error for the model as follows:

$$error = \frac{sum\ of\ weights\ of\ misclassified\ records}{\text{sum of weights of all records}}$$

- If the error is greater 0.5, start again from creating bootstrap
- Calculate the model's importance, alpha as follows:

$$alpha = 0.5 * \ln(\frac{1 - error}{\text{error}})$$

- The weight of each record i, is updated as follows:

$$wi = wi * \exp(-a * yi * C(xi))$$

Where a is the alpha for the current model,
y_i is the class label predicted by the model,
$C(x_i)$ is the true label for that record

- Normalize the weights so that they lie in range [0,1].
- To classify each test data, make a prediction on each of the tree formed and the final prediction is based on the weighted average of all the classifiers with weight representing the alpha values of each classifier.

Pros and Cons of Boosting:

Pros:

- Fast, simple, versatile and easy to program
- No prior knowledge needed about weak learner
- Provably effective given Weak Learning Assumption

Cons:

- Weak classifiers too complex leads to overfitting.
- Weak classifiers too weak can lead to low margins and can also lead to overfitting
- From empirical evidence, AdaBoost is particularly vulnerable to uniform noise

Results:

For number of trees = 5

Boosting				
Dataset	Accuracy	Precision	Recall	F1 - measure
Project3_dataset1.txt	0.9494	0.9369	0.9208	0.9277
Project3_dataset2.txt	0.6451	0.4856	0.4546	0.4542