Classification Algorithms:

Project III

Abinash Behera – 50169804 (abinashb)
Kathleen Rychlicki - 35846580 (krychlic)
Vishwaksen Mane – 50168863 (vishwaks)

CSE 601 | December 10, 2016

Overview

The following implements three classification algorithms, Nearest Neighbor, Decision Tree and Naive Bayes in order to classify datasets. Random Forests and Boosting are also implemented based on the Decision Tree algorithm. K-fold (i.e 10-fold) Cross Validation is adopted to evaluate the performance of all the methods in terms of Accuracy, Precision, Recall and F-1 Measure.

Data

The classification algorithms are performed on two sample datasets where each line represents one data sample. The last column of each line is the class label (0 or 1) and the rest columns are feature values, each of them can be a real-value (continuous type) or a string (nominal type).

Algorithms

K - Nearest Neighbor

K-Nearest Neighbor is a non-parametric method used for classification and regression. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among 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:

1. **Divide into Train and Test Data:** Split the given dataset into train and test data.
2. **Calculate Euclidean Distance:** For each record in the test data, calculate the **Euclidean Distance** between the test data record p and each of the records present in the training data q.

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

3. **Identify K nearest neighbors:** For each record in test data, identify its nearest 'K' neighbors based on the euclidean distance (calculated in **step 2**) between the corresponding training records.
4. **Predict Labels:** By using the class labels of the nearest neighbors, determine the class label of the unknown test record by taking a majority vote scheme.
5. Repeat **steps 2, 3 and 4** until all the records in the test data have been classified as per our algorithm.

Metrics for Performance Evaluation :

		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) => (TP + TN) / (TP + TN + FP + FN)

Precision (p) = a / (a + c)

Recall (r) = a / (a + b)

F-Measure(F) = 2rp / (r + p) => 2a / (2a + b + c)

Parameter Setting - Choice of K:
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.

Handling Continuous Attributes :
We have processed the continuous data by calculating the Euclidean distance between the corresponding test record and training records and thereby its corresponding nearest neighbors are identified to classify using majority voting scheme as per KNN algorithm.

Handling Categorical Attributes :
We have adopted 0-1 loss function for handling categorical attributes. According to this scheme, if two strings are different, then the distance between them is considered to be 1. Whereas if two strings are the same, then the distance between them is considered to be 0.

Pros of KNN :

1. Can handle multiclass classification and regression
2. Simple with no training involved

Cons of KNN :

1. Scaling issues with attributes that have been scaled to prevent distance measures from being dominated by one of the attributes
2. Choosing the value of k. If k is too small, will be sensitive to noise points. If k is too large, the neighborhood may include points from other classes.
3. k-NN classifiers are lazy learners (e.g. does not build models explicitly)
4. Classifying unknown records can be relatively expensive.

K- Nearest Neighbor Results:

Performance Metrics forK-Nearest Neighbor Without Cross Validation :

	project3_dataset1.txt	project3_dataset2.txt
Accuracy	**0.9122**	**0.6086**
Precision	**0.9333**	**0.66**
Recall	**0.7777**	**0.1052**
F-Measure	**0.8484**	**0.1818**

Performance Metrics for K-Nearest Neighbor Obtained Using 10-Fold Cross Validation as follows:

	project3_dataset1.txt	project3_dataset2.txt
Accuracy	**0.9314**	**0.6452**
Precision	**0.9485**	**0.4714**
Recall	**0.8648**	**0.2126**
F-Measure	**0.9028**	**0.2822**

Naive Bayes Classification

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature

Algorithm:

1. Let X be a data sample whose class label is unknown
2. Let Hi be the hypothesis that X belongs to a particular class C_i
3. **Class Posterior Probability:** $P(H_i \mid X)$ is the posterior probability of H conditioned on X - i.e. probability that data example X belongs to class C_i given the attribute values of X.
4. **Class Prior Probability:** $P(H_i)$ is the class probability that X belongs to a particular class Ci. It is calculated as follows:
   $P(H_i) = n_i / n$ where n = total number of training data samples, n_i = number of training data samples of class C_i
5. **Descriptor Prior Probability:** P(X) is the prior probability of X - probability that observe the attribute values of X.
   Suppose X = (x1, x2,…, xd) and they are independent, then P(X) =P(x1) P(x2) …P(xd)
   $P(x_j) = nj / n$, where,

n_j = number of training examples having value x_j for attribute A_j

n = total number of training examples. P(X) is constant for all classes.

6. **Descriptor Posterior Probability:** $P(X|H_i)$ is posterior probability of X given H_i - probability that observe X in class C_i.

   Assume X=(x1, x2,…, xd) and they are independent, then

   $P(X|H_i) = P(x1|H_i)\ P(x2|H_i) \ldots P(xd|H_i)$

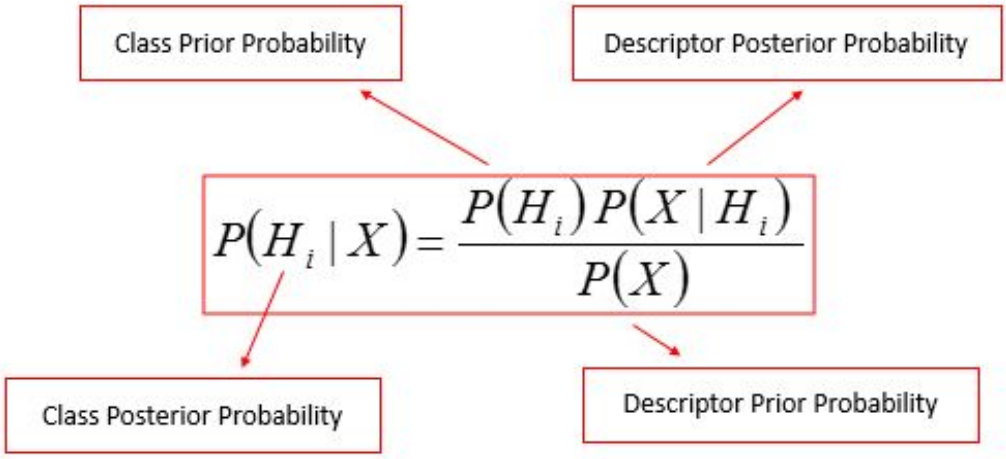
   $P(x_j|H_i) = n_{i,j} / n_i$, where

   $n_{i,j}$ = number of training examples in class C_i having value x_j for attribute A_j

   n_i = number of training examples in C_i
7. Thereby Bayes theorem provides a way of calculating posterior probability $P(H_i | X)$ from P(H),P(X) & P(X| H)
8. **Classification:** To classify means to find the highest $P(H_i | X)$ among all classes C1,…Cm

   Example: If P(H1|X)>P(H0|X), then X buys computer

   If P(H0|X)>P(H1|X), then X does not buy computer



Parameter Setting - Avoiding Zero Probability Problem in Bayes Classification:

Descriptor posterior probability goes to 0 if any of probability is 0. Thus Laplacian correction (or Laplacian estimator) is used to avoid zero probability by adding 1 to each case. This is based on the assumption that since our training dataset is large and thus adding 1 to each case to make a negligible difference to our final estimated probabilities and thereby help us to avoid the zero probability problem.

Handling Continuous Attributes:

For handling continuous attributes we estimate $P(x_i|C)$ through a Gaussian density function with mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

Handling Categorical Attributes:
For handling categorical attributes we have estimated $P(x_i|C)$ as the relative freq of samples having value xi as i-th attribute in class C. If Ak is categorical, then $P(xk|C_i)$ is the number of tuples of class C_i in D having the value xk for Ak , divided by $|C_i,D|$, the number of tuples of class C_i in D.

Pros of Naive Bayes Classifier:

1. Exhibits high accuracy and speed when applied to large databases
2. It is a simple classifier which assumes attribute independence.
3. Comparable in performance to decision trees.

Cons of Naive Bayes Classifier:

1. It does not work well with correlated data since it is based on the idea of attribute independence.

Naive Bayes Classification Results:

Performance Metrics for Naive Bayes Without Cross Validation :

	project3_dataset1.txt	**project3_dataset2.txt**
Accuracy	**0.9122**	**0.6341**
Precision	**0.9047**	**0.7155**
Recall	**0.8636**	**0.9821**
F-Measure	**0.8837**	**0.7102**

Performance Metrics for Naive Bayes Obtained Using 10-Fold Cross Validation as follows:

	project3_dataset1.txt	**project3_dataset2.txt**
Accuracy	**0.9464**	**0.6523**
Precision	**0.9130**	**0.6821**
Recall	**0.9545**	**0.9562**
F-Measure	**0.9333**	**0.6921**

Decision Tree

Algorithm:

We have used the **C4.5 algorithm** to build our decision tree. Out of all the available algorithms, C4.5 is the most versatile as it can handle both contiguous and discrete data very well. Also, it uses entropy and information gain to calculate the next best attribute for splitting. C4.5 is an extension of the ID3 algorithm as it employs similar splitting techniques. However, it does not have the shortcomings of the original ID3 algorithm.

***Reference**: http://idb.csie.ncku.edu.tw/tsengsm/course/dm/Paper/ec45.pdf*

Implementation: Object oriented programming greatly helps in building, maintaining and handling the complex logic of trees. We have defined a custom DTC4Point5Node class, which is a node in the decision tree. This custom class has attributes which can be used to store values like threshold, considered attribute, boolean for leaf or node and the class value if this is a leaf. Additionally, we use the DecisionTreeC4Point5 class to maintain the decision tree itself. This abstraction helps in handling the complex C4.5 algorithm pretty well.

1. We have used the top-down approach for building the decision tree.
2. Base Case 1: If the input dataset belongs to the same class, then we create a leaf and attach the corresponding class to the leaf.
3. Base Case 2: If the attribute values for all the input test cases are the same, then we create a leaf and find the most frequent class.
4. For every attribute, we find the information gain and select the attribute which provides the highest information gain.
5. Then, we attach the selected attribute with the highest information gain to the current node in the tree and recurse on the subsets obtained by splitting using this selected attribute.

Best Feature: We select the best feature based on the information gain values of all the features. We calculate the information gain for all the attributes which have been unused upto this level and then choose the attribute with the highest gain for splitting. This greedy strategy helps in building a near optimal tree which performs well in most cases. However, the greedy strategy does not ensure that the obtained tree will be the smallest possible tree.

Attributes with Discrete Data: Handling discrete data points is pretty straight forward. We create as many children for the current node as the number of discrete data points. For e.g. if the current attribute in consideration has discrete values and each value gets repeated multiple times then we group the data points based on these discrete values and calculate the information gain. Then, we recurse on this grouping based on discrete values for growing the tree.

Attributes with contiguous data: We follow the C4.5 algorithm for handling contiguous attributes ie.e data points with real numbers. As per the algorithm, we calculate the information gain for this attribute as follows:

a) For each data value find the information gain by splitting the data points into two subsets. The first subset consists of all the data points with corresponding values for this attribute less than or equal to the selected data value. The second subset consists of the rest of the data points.

b) We select the threshold as the data point with value such that the obtained subsets provide the highest information gain.

Post Processing for Decision Trees:

1. **Post-Pruning - Avoids Overfitting.**
   a) Grow the decision tree to its entirety
   b) Trim the nodes of the decision tree in a bottom up fashion.
   c) If generalization error improves after trimming, replace sub-tree by a leaf node.
   d) Class label of leaf node is determined from majority class of instances in the sub-tree.

Pros of Decision Tree :

1. Greedy/Eager Learners
2. Inexpensive to construct
3. Fast at classifying unknown records
4. Easy to interpret for small-sized trees

Cons of Decision Tree:

1. Building the optimal smallest decision tree for a given set of data is an NP-complete problem. So, we use the greedy technique to build our tree.
2. Determining the best split of the records can be difficult and also need to determine when to stop splitting.
3. Complex trees can be hard to interpret
4. Duplication in the same sub-tree can be possible

Performance Metrics for Decision Trees Without Cross Validation :

	project3_dataset1.txt	**project3_dataset2.txt**
Accuracy	**0.9220**	**0.6327**
Precision	**0.92**	**0.421**
Recall	**0.87**	**0.38**
F-Measure	**0.83**	**0.41**

Performance Metrics for Decision Trees Obtained Using 10-Fold Cross Validation as follows:

	project3_dataset1.txt	**project3_dataset2.txt**
Accuracy	**0.9315**	**0.6434**
Precision	**0.9105**	**0.327**
Recall	**0.8825**	**0.297**
F-Measure	**0.8692**	**0.305**

Random Forests

We extend our decision tree implementation to build the random forests. Random forests builds on the ensemble learning method. Here, we construct T decision trees based on N sized samples chosen with replacement from the original data set where N is the size of the original training dataset. While constructing the decision tree, we select m random features from the original set of features at each node. This randomization helps in avoiding highly correlated features which can influence the final output. We use the results obtained from all these decision trees to find the final class. For each test case, the final class is the mode/ highest vote of all the classes obtained from the T decision trees.

Performance Metrics for Random Forests Without Cross Validation :

	project3_dataset1.txt	**project3_dataset2.txt**
Accuracy	**0.866**	**0.6435**
Precision	**0.902**	**0.5121**
Recall	**0.722**	**0.6550**
F-Measure	**0.795**	**0.4921**

Performance Metrics for Random Forests Obtained Using 10-Fold Cross Validation as follows:

	project3_dataset1.txt	**project3_dataset2.txt**
Accuracy	**0.9435**	**0.6538**
Precision	**0.9141**	**0.5369**
Recall	**0.8536**	**0.6701**
F-Measure	**0.9023**	**0.5801**

Pros of Random Forests :

1. Improves accuracy because more diversity can be incorporating, thus reducing variances
2. Improves efficiency because searching among subsets of features is much faster than searching the complete set.

Cons of Random Forests :

1. Random forests is usually slower in runtime.

Boosting

Boosting for decision tree builds upon the ensemble method in order to avoid misclassification of hard to classify data points.
First, we assign uniform weights to all the data points. We then use multiple rounds of weak classifiers to build C4.5 decision trees for each round using a bootstrap sample. Then, these weak classifiers are used to classify the original dataset and based on the error calculated using the following formula, we update the weights of each data point. The weights of the misclassified data points are increased and the weights of the correctly classified data points are decreased. This completes the boosting process. After the boosting has been done, we use this ensemble of classifiers to classify each validation data point. First we assign 0 weights to all the classes. The weights of these classes are updated based on the accuracy of each classifier for this corresponding validation data point. The final predicted class is the class with highest weighted average.

ADA Boost Algorithm:

The Adaptive Boosting algorithm is the most famous boosting algorithm currently in practice. We have referred the ADA boosting algorithm mentioned in the "Data Mining: Concepts and Techniques" book for our implementation. The algorithm is as follows:

Input:
D, a set of d class-labeled training tuples;
k, the number of rounds (one classifier is generated per round); a classification learning scheme.

Output:
A composite model.

Method:
(1) initialize the weight of each tuple in D to 1/d;
(2) for i = 1 to k do // for each round:
(3) sample D with replacement according to the tuple weights to obtain Di ;
(4) use training set Di to derive a model, Mi ;
(5) compute error(Mi), the error rate of Mi (Eq. 8.34)
(6) if error(Mi) > 0.5 then
(7) go back to step 3 and try again;
(8) endif
(9) for each tuple in Di that was correctly classified do
(10) multiply the weight of the tuple by error(Mi)/(1 − error(Mi)); // update weights
(11) normalize the weight of each tuple;
(12) endfor
To use the ensemble to classify tuple, X:
(1) initialize weight of each class to 0;
(2) for i = 1 to k do // for each classifier:
(3) wi = log 1−error(Mi) error(Mi) ; // weight of the classifier's vote
(4) c = Mi(X); // get class prediction for X from Mi
(5) add wi to weight for class c
(6) endfor
(7) return the class with the largest weight;

Performance Metrics for Boosting Without Cross Validation :

	project3_dataset1.txt	project3_dataset2.txt
Accuracy	**0.9136**	**0.6283**
Precision	**0.714**	**0.46**
Recall	**0.76**	**0.37**
F-Measure	**0.73**	**0.408**

Performance Metrics for Boosting Obtained Using 10-Fold Cross Validation as follows:

	project3_dataset1.txt	project3_dataset2.txt
Accuracy	**0.9525**	**0.7548**
Precision	**0.72**	**0.68**
Recall	**0.847**	**0.71**
F-Measure	**0.68**	**0.73**

Pros of Boosting :

1. Allows the learner to fit better and thereby improves accuracy.
2. Efficiently handles high dimensional spaces and large number of records.

Cons of Boosting :

1. Harder to tune than other models and you can easily overfit.
2. Lack of interpretability compared to linear classifiers
3. Cannot efficiently handle noisy datasets

K-Fold Cross Validation

We have successfully implemented K-fold cross validation across all the classification algorithms mentioned above.

K-fold cross validation is one way to improve over the holdout method. The data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set. Then the average error across all k trials is computed.

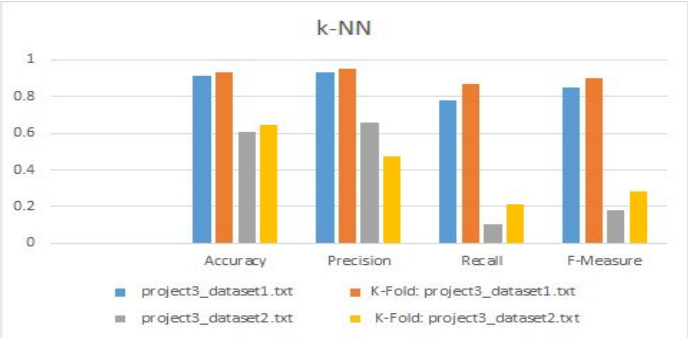
Pros of K-Fold Cross Validation:

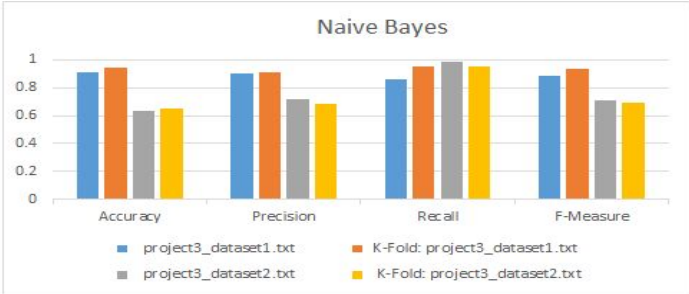
1. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times.
2. The variance of the resulting estimate is reduced as k is increased.

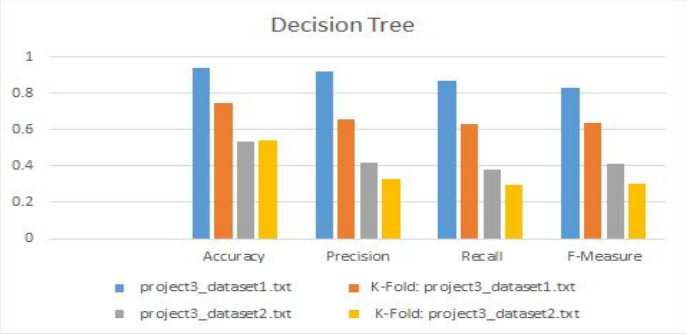
Cons of K-Fold Cross Validation:

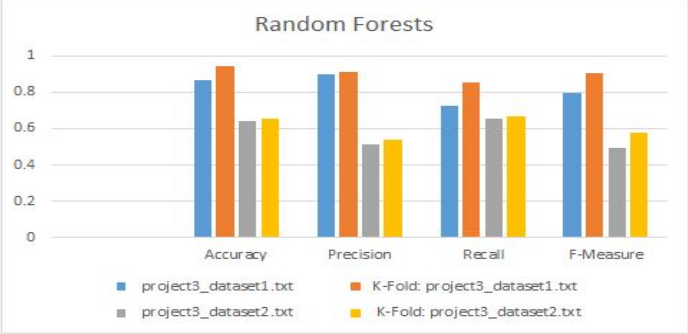
1. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation.

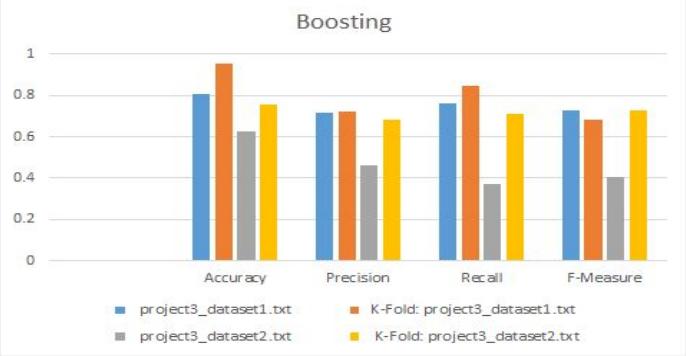
Performance Comparison between Different Classification Algorithms :











Conclusion:

We have successfully implemented K-nearest neighbor, Naive Bayes, Decision Trees, Random forest and ADA Boosting algorithms on the given datasets. Moreover we also implemented K-Fold Cross Validation on the above mentioned techniques and reported the corresponding performance statistics such as accuracy, precision, recall and f-measure.