CSE 601
DATA MINING AND BIO-INFORMATICS

CLASSIFICATION ALGORITHMS

AKSHAY SHAH akshaybh (50206543)
YASH JAIN yahsnavi (50206851)
JAY SHAH jaynaren (50205647)

K - NEAREST NEIGHBOR CLASSIFICATION

This classification method considers K - Nearest Neighbor to testing record in training set, and classifies the record to most frequent class.

ALGORITHM FLOW

1. read file(s), and store it in appropriate data structure (dict) for implementation.
2. split data into n parts (value n of n-fold, in our project: 10)
3. for every fold iteration i,
   a. **merge and assign all parts (n-1 parts) except ith part as training data**
   b. **assign ith part as testing data**
   c. **perform normalization**
      i. calculate mean and standard deviation of training data.
      ii. normalize training data & testing data using above calculated mean & sd.
   d. **perform class prediction on testing data**
      i. for every record in testing data,
         1. calculate Euclidean distance to all records in training data
         2. sort training records distance in ascending order & select first k records
         3. check frequency of each class, and assign most frequent class
   e. **perform performance analysis**
      i. calculate true positives, false positives, true negatives and false negatives by comparing actual class and predicted class
      ii. calculate accuracy, precision, recall & f-measure using above values.

CHOOSING K VALUE

Since our data has only 2 classes, we are only considering odd k values, so we do get majority of unequal frequency of classes.
For **choosing value k**, we are using n-fold method to get average performance metrics and use the k which performs the best. We are using n-fold, so we can also average out of choosing less-compatible testing sets.

DISTANCE CALCULATION FOR CATEGORICAL AND CONTINUOUS DATA

For calculating **distance** for **continuous data**, we get sum of squared difference, and for **categorical data**, we add 1 for every non-matching feature, and then finally we return square root of the final sum as distance.

IMPORTANT FUNCTIONS

read_file: read file, and store it and return in appropriate data structure (dict) for implementation use

split_dict: this function splits dictionary of data and split it into n parts (used for n-fold) and return as a list of dictionaries

calculate_z_score: returns mean and standard deviation of columns of data

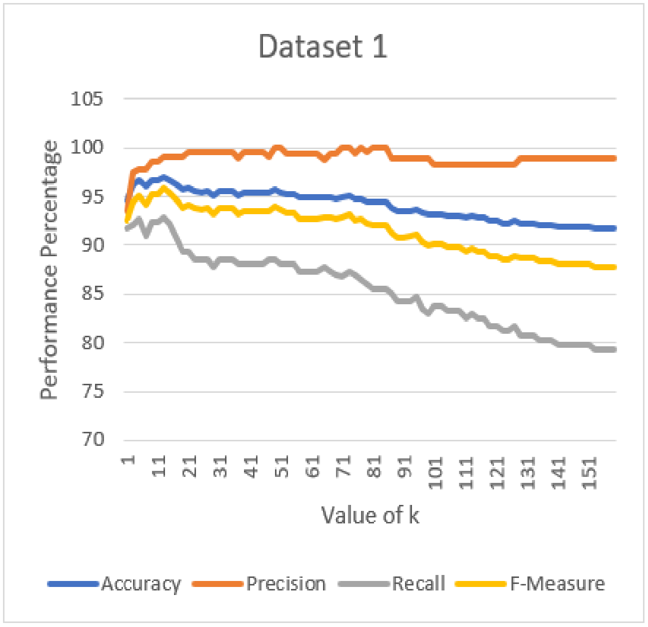
normalize_with_z_score: transforms data with given mean and standard deviation for normalization

classify_record: calls calculate_distance functions for given test record all training records, sorts all training records with distance, and return majority of classes of first k records

calculate_performance_measures: calculates true positives, false positives, true negatives and false negatives by comparing actual class and predicted class, and calculates and returns accuracy, precision, recall and f-measure

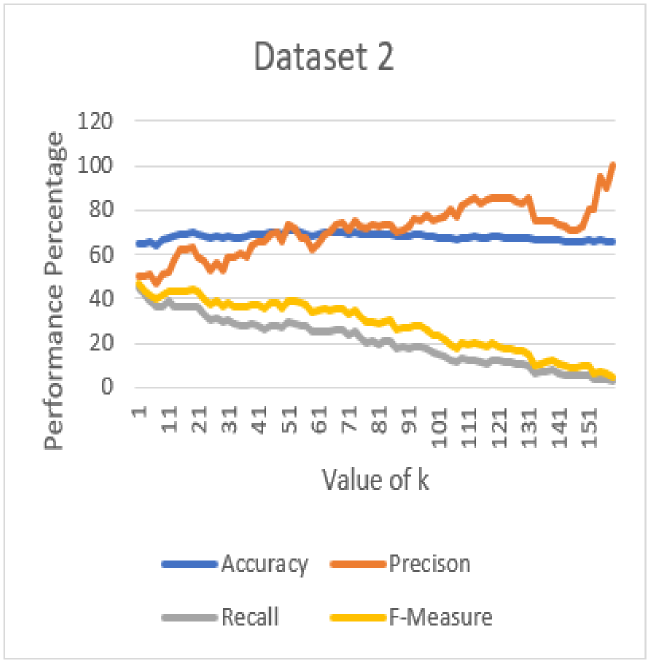
RESULTS

Dataset 1 (with 10-fold cross validation)



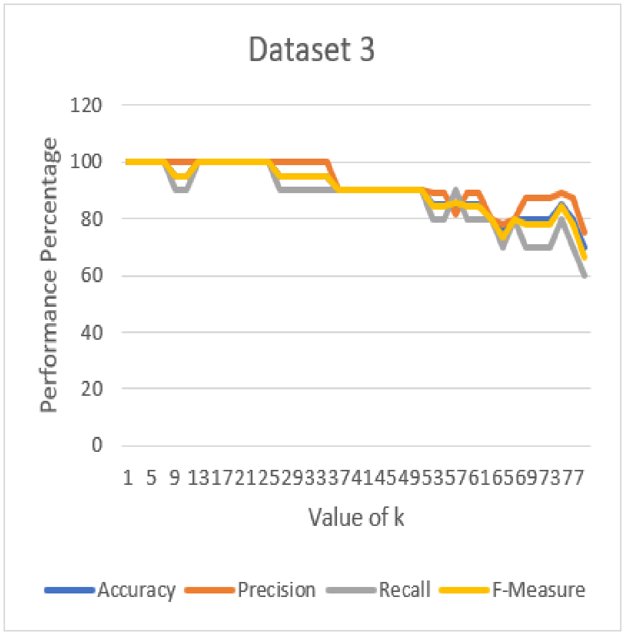
k	Accura	Precisi	Recall	F-Meas
1	94.5632	93.4549	91.7444	92.4704
3	96.1456	97.4244	92.0057	94.52
5	96.6566	97.7994	92.6907	95.044
7	95.967	97.7862	91.0182	94.1066
9	96.6566	98.6328	92.3203	95.239
11	96.6813	98.6195	92.3336	95.2543
13	97.0385	99.0741	92.8723	95.823
15	96.706	99.0598	92.1448	95.4159
17	96.1703	99.0271	90.6585	94.5899
19	95.6593	99.0118	89.3706	93.8744
21	95.8379	99.6	89.3706	94.1433
23	95.5055	99.5833	88.6289	93.7368
25	95.4808	99.6	88.5315	93.6652
27	95.5055	99.5833	88.6289	93.7368
29	95.1484	99.5833	87.6981	93.1714

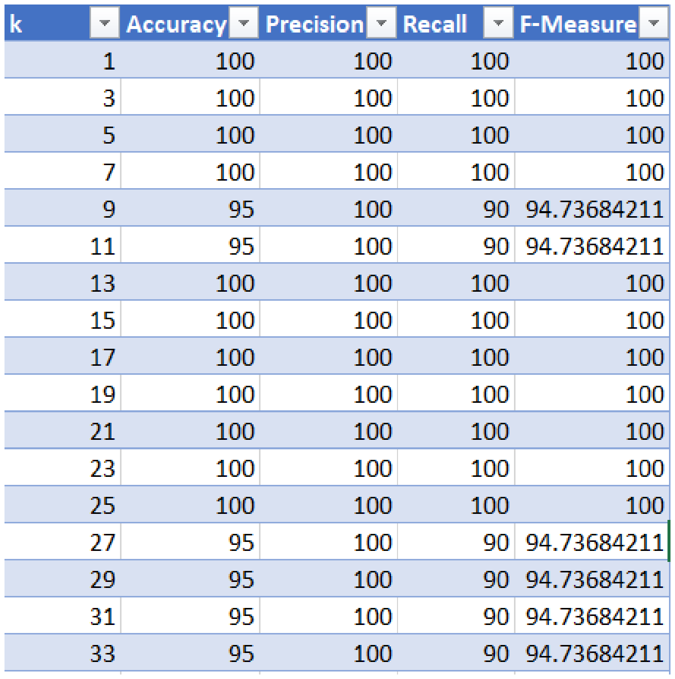
Dataset 2 (with 10-fold cross validation)



f	Accuracy	Precison	Recall	F-Measure
1	65.13587	50.562396	45.01562	46.53807838
3	64.701087	50.340498	41.69023	43.57571843
5	65.57971	51.312298	38.73757	41.55907212
7	64.275362	46.957293	36.14431	39.42686724
9	66.440217	51.269619	36.73549	41.67509579
11	67.110507	51.992077	39.35109	43.34179037
13	68.414855	57.245727	36.60101	42.96455435
15	69.492754	61.875	36.54587	43.51735677
17	69.275362	62.138889	36.55692	43.65043399
19	69.936594	63.299784	36.10705	43.68018616
21	68.849638	58.591575	35.9667	42.82717007
23	68.423913	57.217949	32.93787	40.17883118
25	67.54529	52.289377	30.68995	37.41774892
27	68.623188	56.47619	31.58989	38.9649957
29	67.536232	52.481962	29.62612	36.75387459

Dataset 3 (with testing data)





k	Accuracy	Precision	Recall	F-Measure
1	100	100	100	100
3	100	100	100	100
5	100	100	100	100
7	100	100	100	100
9	95	100	90	94.73684211
11	95	100	90	94.73684211
13	100	100	100	100
15	100	100	100	100
17	100	100	100	100
19	100	100	100	100
21	100	100	100	100
23	100	100	100	100
25	100	100	100	100
27	95	100	90	94.73684211
29	95	100	90	94.73684211
31	95	100	90	94.73684211
33	95	100	90	94.73684211

We see that for Dataset1, k=13 gives best accuracy, recall and f-measure, and good precision, so we will select k as 13 for further classification, if needed.
We also see that for Dataset2, k=19 gives best accuracy and f-measure, almost best recall, and good precision, select k as 13 for further classification, if needed.
For Dataset3, we see that k in beginning gives best performance, and it reduces gradually, as k increases towards size of training data.

With all above conclusions, we see that k value will also depend on size of training data. And for large datasets, for very small value of k, classification may get affected due to noisy data, and for larger k, we may gradually get into area of other classes.

PROS:

1. Easy to understand and implement.
2. This classifier is lazy learner, and so allows it to change during real-time use.
3. Robust to unknown probability distribution.
4. Robust to outliers.

CONS:

1. In case of local anomalies, outcome can be affected.
2. Large datasets can result in more computation time, because it is lazy learner. Most of the execution is done in testing phase.
3. More dimensions results in more training points near to testing point, which will then reduce effectiveness of k-nn logic.

NAÏVE BAYES CLASSIFICATION ALGORITHM

Naïve Bayes a classification algorithm is a supervised probabilistic learning and statistical method for classification. We create a probabilistic model and it allows and use it in determining probabilities of the outcomes. It can solve diagnostic and predictive problems. It is based on the Bayes Theorem, for with the formula

EQUATION: $\boldsymbol{P(C|X)} = \frac{\boldsymbol{P(X \mid C)\,.P(C)}}{\boldsymbol{P(X)}}$

C: Class Label	P(C): Class Prior	P(X|C): Posterior Probability.
X: Input data point	P(X): Descriptor Prior.	

ALGORITHM FLOW

1. **Generation of test data and training data**
   We used k-fold Cross Validation method to generate training and test set.
2. **Training the probabilistic model using the training data**
   We parsed the training data split it in the number of parts as many as the class labels. For each split, we compute posteriors probabilities
   a. **Computing the class prior**
      We compute the priors for each class label.
   b. **Computing the posterior values of each attribute**
      For continuous valued attributes we compute mean and variance.
      For discrete we find the conditional probabilities for that class label.
   c. We store the posterior values in a map.
3. **Test the model trained with the test data**
   a. **For each data calculate the posterior probability for class label.**
      We multiple the posterior of each attribute in the data point and the class prior to compute the final probability of that data point belonging to that class label.
   b. **Compare the probabilities and assign the label with the higher probability**
4. Compare the results with the true values original dataset and evaluate algorithm using accuracy, precision, recall, f-measure.
5. **Repeat the steps for the k-iteration of k-fold cross validation.**
6. Compute the average accuracy, precision, recall, f-measure for the dataset.

USE OF K-FOLD CROSS VALIDATION (TEST AND TRAINING DATA)

In order to evaluate the correctness of the probabilistic model created for Naïve Bayes we use K-FOLD cross validation technique.
We have used k=10. Hence the dataset is divided in 10 parts. In every iteration of cross validation, we choose a different fold as a test data, and the rest of the data is used as training.
We have chosen to map the data points to their line number and we pass the line numbers around the function so that we save stack space

HANDLING OF CONTINUOUS VALUED ATTRIBUTES

Naïve Bayes cannot be directly applied to data having continuous valued attributes. We have assumed the continuous values of the feature are distributed according to Gaussian Distribution.
Hence for each feature, instead of pre-computing the conditional probabilities we just compute the mean and the variance required for the distribution.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

HANDLING CASES OF ZERO PROBABILITY

Zero probability happens when occurrences of a class label and a certain attribute value together then the frequency-based probability estimate will be zero.
With the posterior 0.0 our whole probability becomes 0.
To counter this problem, we use **Laplacian Smoothing/Correction.**
In this method we add 1 to every value on the numerator.
E.g.: P(X = x1|C)=5/10, P(X =x2|C) = 5/10, **P(X = x3 | C) = 0/10**
We can correct this as : P(X = x1|C)=6/13, P(X = x2|C) = 6/13, **P(X = x3|C) = 1/13**

As the dataset given show no sign of Zero probability, we did not implement the Laplacian correction in the project.

IMPORTANT METHODS AND VARIABLES

1. **resultMap:** To store the results for all the iterations.
2. **truthMap:** To store the truth values of the data points.
3. **Boolean isDiscrete []** It's a map to store if the feature is discrete or continuous.
4. **posteriorProbabilities:** Map storing the Map of posterior values for each class label.
5. **descriptorPorbabilites:** Map to store the descriptor probabilities for each feature.
6. **classPrior:** Map to store the class prior probabilities for each feature value.

7. **trainDescriptorPrior:** This method precomputes the descriptor prior probability of each value of each feature and stores the result in the **descriptorPrior Map.**

8. **generateDiscretePosteriors:** This method handles the categorical data and precomputes the conditional probabilities of each value of each feature and stores the result in the **Map which is again stored in the posteriorProbabilities Map for all labels.**

9. **computePosterior:**
   This method handles the continuous data and precomputes the mean and variance for the continuous valued attribute stores the result in the **posteriorMap** and returns the map to the calling function.

10. **computePDF:** This function is used to compute the probability using **Gaussian Probability distribution** accepting the mean, current value and the variance of the feature and returns final posterior probability.

11. **trainData:** This function splits the training data into N parts. Where N is the number of class labels. We call the computePosterior for each split and store the results in the **posteriorMap.**

RESULTS

Following are the evaluation results Naïve Bayes classification on the two datasets provided.

K	Data Set 1	Data Set 2
1	Accuracy: 94.64% Precision: 91.30% Recall: 95.45% F-Measure: 93.33%	Accuracy: 67.39% Precision: 62.50% Recall: 71.43% F-Measure: 66.67%
2	Accuracy: 92.86% Precision: 88.89% Recall: 88.89% F-Measure: 88.89%	Accuracy: 69.57% Precision: 45.45% Recall: 83.33% F-Measure: 58.82%
3	Accuracy: 96.43% Precision: 85.71% Recall: 100.00% F-Measure: 92.31%	Accuracy: 78.26% Precision: 76.47% Recall: 68.42% F-Measure: 72.22%
4	Accuracy: 91.07% Precision: 90.00% Recall: 85.71% F-Measure: 87.80%	Accuracy: 69.57% Precision: 52.63% Recall: 66.67% F-Measure: 58.82%
5	Accuracy: 91.07% Precision: 94.74% Recall: 81.82% F-Measure: 87.80%	Accuracy: 63.04% Precision: 64.29% Recall: 42.86% F-Measure: 51.43%
6	Accuracy: 96.43% Precision: 100.00% Recall: 90.48% F-Measure: 95.00%	Accuracy: 60.87% Precision: 50.00% Recall: 55.56% F-Measure: 52.63%
7	Accuracy: 96.43% Precision: 96.15% Recall: 96.15% F-Measure: 96.15%	Accuracy: 84.78% Precision: 75.00% Recall: 69.23% F-Measure: 72.00%
8	Accuracy: 96.43% Precision: 93.33% Recall: 93.33% F-Measure: 93.33%	Accuracy: 78.26% Precision: 50.00% Recall: 60.00% F-Measure: 54.55%
9	Accuracy: 92.86% Precision: 96.00% Recall: 88.89% F-Measure: 92.31%	Accuracy: 60.87% Precision: 44.44% Recall: 50.00% F-Measure: 47.06%
10	Accuracy: 87.50% Precision: 88.00% Recall: 84.62% F-Measure: 86.27%	Accuracy: 69.57% Precision: 50.00% Recall: 50.00% F-Measure: 50.00%
Average	**Average Accuracy: 93.57%** **Average Precision: 92.41%** **Average Recall: 90.53%** **Average F-Measure: 91.32%**	**Average Accuracy: 70.22%** **Average Precision: 57.08%** **Average Recall: 61.75%** **Average F-Measure: 58.42%**

RESULT ANALYSIS:

1. **Using K-Fold Cross Validation increases the average Accuracy/Precision:** The training data is not always well distributed. Hence taking multiple combination of the training dataset, our average accuracy tends to increase. Thus K-Fold cross validation will make sure that we work with different training set combination at every iteration. Thus an instance of a bad training set is averaged out, and we get an optimum average accuracy.

2. **Dimensionality**
   The time taken to run the Naïve Bayes Classifier was:
   Data Set 1: 0.3 s and Data Set 2: 0.2 s
   We can say that if the dimensions of the dataset increase (by number of rows and number of features/columns) training time will increase linearly.

3. **Number of Training samples taken**
   Our accuracy is also affected by the number of training samples we take. If the number of training examples are too low then our probabilistic model will not be able to classify accurately.

4. **Degree of dependency between attributes:** Given the high accuracy in data set in we can assume say that the degree of co-relation between attributes in dataset 1 might be low. But with dataset 2 the accuracy is lower hence we can say that the degree of correlation between attributes might be high. But this is not enough to come to any solid conclusion.

PROS:

1. **Computationally Fast and Scalable.**
   Given the time results; Naïve Bayes is faster to train than other models.

2. **Highly Scalable.**
   Naïve Bayes, is highly scalable and works well with high dimension without loss in accuracy and training time.

3. **Can be parallelized.**
   The build process can be parallelized. We can distribute the attributes among different processors and compute the conditional probability.
4. **Simple Implementation.**

5. **Can be used for binary as well as multiclass classification.**

CONS:

1. **Reliance on Independence of Attributes.**

   The Naïve Bayes works with the assumption that the attributes are not correlated. Hence Naïve Bayes won’t work well with data having relationships among them, E.g. Recommendation systems.

2. **Need of large number of samples.**

   Accuracy is also affected by the number of training samples as well as the quality. If the number of training examples are too low or the data isn’t well distributed, it will not cover the whole domain of values; thus, the precision and accuracy of the classification will be low.

DECISION TREE, RANDOM FOREST & BOOSTING (ADABOOST)

Methods common to each of the above algorithm

fileRead: It reads the files, and stores the important information like data and true label in arraylist and hashmap.

partition: Based on the column and value, it splits the input data into left and right.
For discrete data type, it splits matching rows in left and rest on right.
For continous data, it splits rows with values less than or equal to, to the left, and greater than, to right.

calcGain: It calculates the information using the gini for parent, left and right node.

calcGini: It calculates the gini index for the input rows by counting the number of 1s and 0s.

getLabel: For a given test row and tree, recursively traverse tree and follow left/right part based on the condition and return the label

getAccuracy: It calculates the accuracy using the formula as no of correctly classfied data/ total no of data.

getPrecision: It calculates the precision using the formula as no of true positive data/ sum of true positive and false positive.

getRecall: It calculates the recall using the formula as no of true positive data/ sum of true positive and false negative.

getFMeasure: It calculates the Fmeasure using the formula as twice no of true positive data/ sum of twice of true positive, false negative, false postive.

Decision tree

Algorithm Flow

1. For every fold,
    a. Split the data into test data and training part.
    b. Build tree
        i. find the best split
        ii. calculate information gain
        iii. if gain == 0
            1. mark as leaf, store label and return.
        iv. partition the data into left and right
        v. calculate information gain
        vi. recursively build tree for left and right data
    c. Test testing data
    d. calculate accuracy, precision, recall and f-measure.
2. //end of fold loop.
3. Calculate average accuracy, average precision, average recall, average f-measure.

Important Methods

CART: For each fold, it splits the data into test and train. Builds tree using train data, and predicts data for the test rows. And calculates the accuracy, precision, recall and f-measure for current fold. Finally, after all folds are completed, it calculates average of each of the above measures.

buildTree: It finds the best split for given set of rows. if the gain is 0, it marks as Leaf node and sets the label as true label of input rows. Else it splits the data into left and right for the question obtained above. Recursively it builds left tree and right tree.

findBestSplit: For each attribute, each value, it splits the data and calculates the information gain (using Gini). It returns the attribute and value for which we get the best gain.

testData: For each test row, it traverses the built tree, based on condition move to left or right and returns the label when it reaches the leaf node.

RESULTS

For dataset1 (with 10-fold cross validation)
Accuracy = 93.49736379613357
Precision = 92.27053140096618
Recall = 90.09433962264151
f-measure = 91.16945107398568

For dataset2 (with 10-fold cross validation)
Accuracy = 62.17316017316017
Precision = 45.258426966292134
Recall = 48.125
f-measure = 45.562130177514796

ANALYSIS :
From the results obtained, it looks like attributes in dataset 1 are more related to each other resulting in high accuracy. Whereas attributes in dataset2 doesn't looked co-related, resulting in not so high accuracy.

PROS :

1. The main advantage is interpretability. Decision trees are "white boxes" in the sense that the acquired knowledge can be expressed in a readable form,
2. while KNN, SVM are generally black boxes, i.e. you cannot read the acquired knowledge in a comprehensible way
3. For not very complex data sets, decision tree has accuracy comparable to other classification algorithm.
4. Decision tree to easy and Inexpensive to construct
5. It is extremely fast at classifying unknown records as with every decision we select only one path.
6. Easy to interpret for small-sized trees.
7. It can deal with noisy or incomplete data.

CONS:

1. For complex data, tree might get very large resulting in exponential calculation growth and complexity.
2. High variance and unstable: Since we use greedy strategy, decision tree's variance in finding the right starting point of the tree can greatly impact the final result. i.e minor changes early on can have big impacts later.
3. Decision Tree's do not work best if you have a lot of un-correlated variables.
4. Decision Trees do not work well if you have smooth boundaries (it becomes very unstable).

RANDOM FOREST

ALGORITHM FLOW

1. For every fold,
    a. Split the data into test data and training part.
    b. For every iteration
        i. Sample training data using bagging with replacement
        ii. Build tree
            1. At each node, select random 20% attributes.
            2. find the best split
            3. calculate information gain
            4. if gain == 0
                a. mark as leaf, store label and return.
            5. partition the data into left and right
            6. calculate information gain
            7. recursively build tree for left and right data
        iii. Test testing data
    c. //end of iteration loop
    d. Calculate final prediction label based on voting method.
    e. calculate accuracy, precision, recall and f-measure.
2. //end of fold loop.
3. Calculate average accuracy, average precision, average recall, average f-measure.

IMPORTANT METHODS:

RandomForest: For each fold, it splits the data into test and train. For each iteration, it samples data using bagging with replacement. Builds tree using train data, and predicts data for the test rows. After all iterations, it calculates final prediction label using voting for each test row.
And calculates the accuracy, precision, recall and f-measure for current fold. Finally, after all folds are completed, it calculates average of each of the above measures.

buildTree: pick random 20% attributes at each node. It finds the best split for given set of rows. if the gain is 0, it marks as Leaf node and sets the label as true label of input rows. Else it splits the data into left and right for the question obtained above. Recursively it builds left tree and right tree.

findBestSplit: For each attribute, each value, it splits the data and calculates the information gain (using Gini). It returns the attribute and value for which we get the best gain.

testData: For each test row, it traverses the built tree, based on condition move to left or right and returns the label when it reaches the leaf node.

finalTestData: For each training row, it counts no of 1s and 0s as predicted label, and selects one with max vote as final label.

Results

For dataset1 (with 10-fold cross validation, no of trees = 5 and no of attr = 20%))
Accuracy = 95.78207381370827
Precision = 94.76190476190476
Recall = 93.86792452830188
f-measure = 94.31279620853081

For dataset1 (with 10-fold cross validation, no of trees = 7 and no of attr = 20%))
Accuracy = 94.76923076923077
Precision = 92.28571428571429
Recall = 94.28571428571429
f-measure = 94.28571428571429

For dataset2 (with 10-fold cross validation, no of trees = 5 and no of attr = 20%)
Accuracy = 64.71861471861472
Precision = 49.00662251655629
Recall = 46.25
f-measure = 47.58842443729903

For dataset2 (with 10-fold cross validation, no of trees = 7 and no of attr = 20%))
Average accuracy = 68.65036231884059
Average precision = 55.283572335816906
Average recall = 52.718262001156745
Average f-measure = 52.44530061522774

PROS:

1. Better accuracy than decision tree as it adds more diversity and reduces variances.
2. fast and better efficiency as we can consider subsets of attributes at each node.
3. runs efficiently on large database and we take only subset of data at each node.
4. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

CONS:

1. Not easy to visualize as compared to decision tree.
2. If the data contain groups of correlated features of similar relevance for the output, then smaller groups are favored over larger groups.
3. For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels.

BOOSTING

ALGORITHM FLOW

1. For every fold,
    a. Split the data into test data and training part.
    b. Calculate initial weight for entire training data.
    c. For every iteration
        i. Sample training data using bagging with replacement based on weight.
        ii. Build tree
            1. find the best split
            2. calculate information gain
            3. if gain == 0
                a. mark as leaf, store label and return.
            4. partition the data into left and right
            5. recursively build tree for left and right data
            6. Test training data.
            7. calculate Error as ratio of sum of weight of misclassified data to total weight
            8. calculate alpha and update weight
            9. Test testing data for constructed tree
    d. //end of iteration loop
    e. Calculate final prediction label based on weighted average of alpha obtained in each iteration.
    f. calculate accuracy, precision, recall and f-measure.
2. //end of fold loop.
3. Calculate average accuracy, average precision, average recall, average f-measure.

IMPORTANT METHODS

Boosting:

1. For each fold, it splits the data into test and train. For each iteration, it samples data using bagging with replacement based on weight. Builds tree using sampled train data.
2. Calculates error and alpha based on number of misclassified data.
3. Updates weights of misclassified data. Predicts label for the test rows.
4. After all iterations, it calculates final prediction label using weighted average of alpha for each test row & calculates the accuracy, precision, recall and f-measure for current fold.
5. Finally, after all folds are completed, it calculates average of each of the measures.

buildTree:

It finds the best split for given set of rows. if the gain is 0, it marks as Leaf node and sets the label as true label of input rows. Else it splits the data into left and right for the question obtained above. Recursively it builds left tree and right tree.

testData:
For each test row, it traverses the built tree, based on condition move to left or right and returns the label when it reaches the leaf node.

finalTestData:
For each training row, it calculates sums of alpha which outputs predicted label as 0 and 1.

RESULTS:

For dataset1 (with 10-fold cross validation, no of trees = 3)
Average accuracy = 93.03021978021977
Average precision = 88.43463898857146
Average recall = 93.27753727753729
Average f-measure = 90.67831779276665

For dataset1 (with 10-fold cross validation, no of trees = 5)
Accuracy = 95.07908611599296
Precision = 94.23076923076923
Recall = 92.45283018867924
f-measure = 93.33333333333333

For dataset2 (with 10-fold cross validation, no of trees = 3)
Average accuracy = 65.18115942028986
Average precision = 49.636961078137546
Average recall = 44.90640864982969
Average f-measure = 46.0491196682143

For dataset2 (with 10-fold cross validation, no of trees = 5)
Accuracy = 65.85281385281385
Precision = 48.682119205298015
Recall = 45.0
f-measure = 46.30225080385852

Ideally boosting should have given high accuracy compared to decision tree, but since data size is less, and not all attributes seems related, accuracy increases only by certain percentage for both the dataset.

PROS:

1. No need to apply feature scaling for the algorithm to do well. And the features can be a mix of binary, categorical and continuous types.
2. We get better accuracy with only modest memory and runtime requirements to perform prediction, once the model has been trained.
3. Covers training error, so normally it leads to better accuracy.
4. No prior knowledge needed about weak learner
5. No parameters to tune (except T)

CONS

1. Harder to tune than other models, because you have so many hyperparameters and you can easily overfit.
2. Not easy to visualize as compared to decision tree.
3. Not very speedy to train or score.
4. AdaBoost is particularly vulnerable to uniform noise.

DECISION TREE, RANDOM FOREST, BOOSTING (ADABOOST)

CHOICE DESCRIPTION

CONTINUOUS FEATURE:
For a given value of continuous feature, we divide the test row into left if its value for corresponding feature is less than or equal to given value else we keep it in right.

CATEGORICAL FEATURE:
For a given value of categorical feature, we divide the test row into left if its value for corresponding feature matches to the given value else we keep it in right.

BEST FEATURE:
We have used information gain to decide while splitting the data. Higher the value of information gain, better the value.
We finally pick the value with highest gain to split the data.

POST PROCESSING:

After creating the decision tree, we reverse tree in bottom-up manner and calculated the error by replacing each subtree with a leaf and label as maximum of class count in that subtree.
If the error did not increase even after this trimming, we replace the subtree with the leaf.
This way, with no decrease in accuracy, we get a smaller decision tree, and hence the testing process speeds up.

CROSS VALIDATION

The K-Fold cross validation technique will make sure that we work with different combination of training dataset. It is possible that for some iterations the accuracy results were lower.
This indicates the data taken to train does is not well distributed. Hence taking multiple combination of the training dataset, our average accuracy tends to increase.

Implementation:
We have performed 10-fold cross validation. where we divide data into 10 parts and in each iteration, different part is treated as test and others part combined are taken as training data.
Ideally all folds should be of same size except for last one where we put all remaining data. We calculate accuracy, precision, recall and f-measure for each fold. Finally, we calculate average of each measure across all folds.

K-NN VS NAÏVE BAYES VS DECISION TREE COMPARITIVE STUDY

	K-NN	Naïve Bayes	Decision Tree
Difficulty to Understand	Easy	Easy	Moderate
Difficulty to Implement	Easy	Easy	Moderate
Difficulty to Visualize model	Moderate	Difficult, its probabilistic model	Easy to visualize tree model
Time for training	Very Less	Moderate	Takes reasonable amount of time
Time for testing	Depends on amount of training data (K-NN is lazy learner)	Very Less	Very Less
Parallelization possible?	Yes	Yes	Yes