CSE 601 – Data Mining and Bioinformatics (Fall 2017)

Project 3:
Classification Algorithms

Team Members:

1. **Siddharth Pateriya** spateriy@buffalo.edu **Person #: 50206348**
2. **Hrishikesh Saraf** hsaraf@buffalo.edu **Person #: 50205927**
3. **Ajay Gandhi** agandhi3@buffalo.edu **Person #: 50207403**

Introduction:

What has been expected:

- To implement all three classification algorithms by yourself: Nearest Neighbor, Decision Tree, and Naïve Bayes.
- To implement Random Forests based on your own implementation of Decision Tree.
- To implement Boosting based on your own implementation of Decision Tree.
- To adopt 10-fold Cross Validation to evaluate the performance of all methods on the provided two datasets in terms of Accuracy, Precision, Recall, and F-1 measure.

What has been done:

- Implemented all three classification algorithms by ourself:
    a. Nearest Neighbor – Takes value of 'k' and name of input dataset at runtime and performs classification using k-NN and provides performance results in the end.
    b. Decision Tree – Takes input dataset, recursively splits data into two parts based on an attribute selection measure until a leaf node is created. Predicts labels by traversing the created decision tree.
    c. Naïve Bayes – Takes input dataset, classifies it using Naive Bayes classification, and performs 10-fold cross validation to compute average performance metrics.
- Implemented Random Forests and Boosting based on our own implementation of Decision Tree.
- Adopted 10-fold Cross Validation in order to provide fair evaluation of our models and evaluated the performance of all methods on the provided two datasets in terms of Accuracy, Precision, Recall, and F-1 measure and shown these results in the end of each algorithm.

Classification

- Classification is the process of creating models for different applications which can be used to predict outcome based on certain input data.
- Classification models are built (or trained) on some training data, and these models help to make future predictions on certain test data.
- So to summarize, the goal of classification is to accurately predict the target class for each case in the data. For example, a classification model could be used to identify loan applicants as low, medium, or high credit risks.

- A few example of Classification Algorithms (or models) are:
  1. K-Nearest Neighbors
  2. Naïve Bayes
  3. Decision Trees
  4. Support Vector Machine
  5. Logistic Regression

Part 1: k-Nearest Neighbors

- The K-Nearest Neighbors algorithm (or k-NN) is a classification or regression algorithm in which an object is assigned a class or a label based on the values of its ‘k’ neighbors.

- The k-NN algorithm is one of the simplest of all machine learning algorithms: in which an object is classified depending on its neighbors; the object being assigned to the class most common amongst its k nearest neighbors.

- k-NN is also called as lazy learning model (instance-based learning), because all the computation is deferred until classification.

A) Steps for k-NN:

- The main data is read and the total data is divided into two subsets, training and testing data set, we can also have two separate input data one for training and one for testing.

- In order to make sure that the model is built properly and the results obtained are fair, we can use cross-fold validation in which every time a different part of the original data set is used as testing data, and the remaining part is used as training data.

- The results obtained in the end are averaged in order to get correct performance measures for the given data set.

- The training and testing data set example contains multi-dimensional vectors each with a class label.

- We have to choose the value of ‘k’, the number of nearest neighbors of the unclassified label to compare and assign.

- Once the training data set and the testing data set is created and the value of ‘k’ is determined, we need to perform following steps:

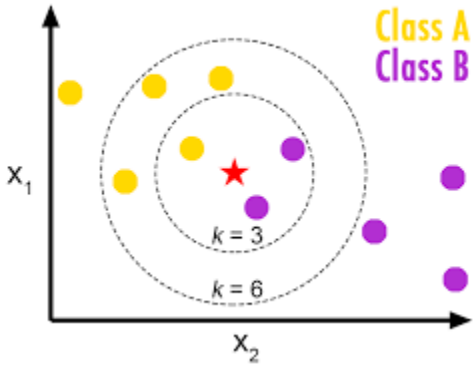
  a. Pick an unclassified data value from the testing data set.
  b. Calculate its distance from each data point in the training data set (distance metric used can be Euclidean Distance).
  c. Pick top ‘k’ closest neighbors and check the labels.
  d. Assign the label which occurs in majority to our testing data value.

e. Repeat the steps over all the testing data set values.

- Calculate performance measures such as accuracy, precision, recall and f-measure to check how well the model performs on the given data set.

B) How to choose 'k':

- It is very important to choose a good value for 'k'.



- 'k' **too low** : Choosing a very less value of k is not a good choice because, data is classified without having required information. This may lead to misclassification.

- 'k' **too high** : Choosing a high value of k is not good because classification of data maybe affected by outliers. This leads to wrong classification.

Figure 1

- Choosing an appropriate 'k' : There is no fix way to do that, value of 'k' depends on the input data set and changes accordingly.

- As we can see from ***Figure 1*** :
  - ✓ k = 3 – Classifies the star (test data) as purple circle (Class B).
  - ✓ K = 6 – Classifies the star (test data) as yellow circle (Class A).

- Thus from our example we can see changes in choosing 'k' can lead to misclassification easily.

C) How to Calculate Euclidean Distance:

- To select 'k' nearest neighbors we have to calculate distance between two data points based on their multi-dimensional vectors (we choose Euclidean Distance Metric).

- To find distance we can use Euclidean Distance, as follows: Here, p,q are two data points from the data set.

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

- There are two types of attribute values we can encounter viz. Continous and Categorical. In terms of continuous data we can simply calculate the distance like we normally do, however if the attribute data value is categorical we compare the two data points and add 0 or 1 depending on whether the values match or not.

- For eg: data_value_1 = ‘present’, data_value_2 = ‘absent’ and data_value_3 = ‘present’
  In this case: distance (data_value_1,data_value_2) = 1 (as they are unequal)
  distance (data_value_1,data_value_3) = 0 (as they are equal)

D) Results:

1. dataset1.txt

k	Accuracy	Precision	Recall	F-measure
3	96.14%	97.88%	91.56%	0.944
5	96.66%	97.8%	92.76%	0.95
10	97.00%	99.03%	92.83%	0.957
30	95.26%	99.5%	88.12%	0.934
50	95.26%	99.5%	88.12%	0.934
100	93.33%	98.9%	83.85%	0.906

2. dataset2.txt

k	Accuracy	Precision	Recall	F-measure
3	64.72%	50.22%	41.63%	0.437
5	65.38%	50.92%	36.93%	0.403
10	66.91%	53.47%	38.62%	0.43
30	67.99%	54.98%	30.32%	0.379
50	70.38%	73.22%	27.43%	0.38
100	68.42%	65.76%	17.19%	0.26

E) Result Analysis:

- We observe that k-NN provides very high accuracy as well as precision, recall and f-measure in dataset1. Despite the data being large which means k-NN works on large data scale as well.
- We also observe that dataset1 contains only continuous data and dataset2 contains both continuous as well as categorical data, so k-NN works much better on continuous data than on data which contains categorical data or both continuous and categorical data.
- Also we can clearly see that the performance metrics increase with increasing value of ‘k’ reach certain peek values for some value of ‘k’ and then decrease as we keep on increasing ‘k’.
- This suggests that ‘k’ should neither be too high or too low and depends on the dataset.

F) Pros:

- Easy to implement.
- Robust to noise in training data (especially on usage of inverse square weighted distance).
- It works on very large data sets as well.

G) Cons:

- Choosing ‘k’ is tricky, may lead to misclassification.
- Small value of ‘k’, model is sensitive to noise points.
- Large value of ‘k’, model unnecessary picks points from different classes.
- Normalization of data needs to be carried out, to avoid domination of distance measures because of one or two attributes (scaling is carried out).

Part 2: Decision Trees

- A decision tree is a graph that uses a branching method to illustrate every possible outcome of a decision.

- Decision Tree algorithm belongs to the family of supervised learning algorithms.

- Unlike other supervised learning algorithms, decision tree algorithm can be used for solving regression and classification problems too.

- There are many specific decision-tree algorithms. Notable ones include:
  - ✓ ID3 (Iterative Dichotomiser 3)
  - ✓ C4.5 (successor of ID3)
  - ✓ CART (Classification And Regression Tree)

- We have used CART for our implementation of decision tree.

A) Steps for Decision Tree:

1. Select the best attribute out of the input dataset by iterating over all the attributes by calculating the best split using measures such as Gini Index and Information Gain.

2. If the current Gini Index on the dataset is 0, create a leaf or else continue to the next steps.

3. Split the dataset set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute for categorical data.

4. For continuous data, attributes must be greater than or equal to in one subset and lesser than the splitting attribute in the other subset.

5. Repeat the steps on each subset until you find leaf nodes in all the branches of the tree recursively.

B) Choice Description:

- **Categorical Data:** We have used binary split for categorical data. If the splitting attribute's value is equal to the current attribute, it is put in the "right" set or else, it is put in the "left" set.

- **Continuous data:** Attribute value greater than splitting attribute value are put in the "right" set, or else they are put in the "left" set.
- **Best feature:** Best feature is the one which minimizes Gini Index the most.

Post Processing:

Post processing can be done by following techniques:

- For a fully grown tree, trim the nodes from in a bottom - up fashion. If generalization error improves after trimming, we replace the sub tree by leaf node. Class label is then determined majority basis.
- Another way is to keep aside a separate set of data called the “pruning - set" which is different from test set and can be used to perform technique 1.

Post processing techniques help avoid overfitting and also reduce time in prediction.

C) Results:

1. **dataset1.txt :** (10-fold Cross Validation)

Average Accuracy: **92.32%**	**Average Precision**: **91.75%**
Average Recall: **88.16%**	**Average F-Measure**: **0.897**

2. **dataset2.txt :** (10-fold Cross Validation)

Average Accuracy: **61.46%**	**Average Precision**: **44.41%**
Average Recall: **50.43%**	**Average F-Measure**: **0.465**

D) Pros:

- Very fast classification seeing new data
- Can handle both categorical and continuous attributes
- Easy to interpret, since they give a visual representation
- Helps identifying the most important attributes

E) Cons:

- Susceptible to small variations. Even a small variation in data would result in a different tree.
- Creation of large decision trees is complex and can take time
- Not a very good predictor for data with continuous attributes
- Needs the use of pruning techniques to avoid overfitting.

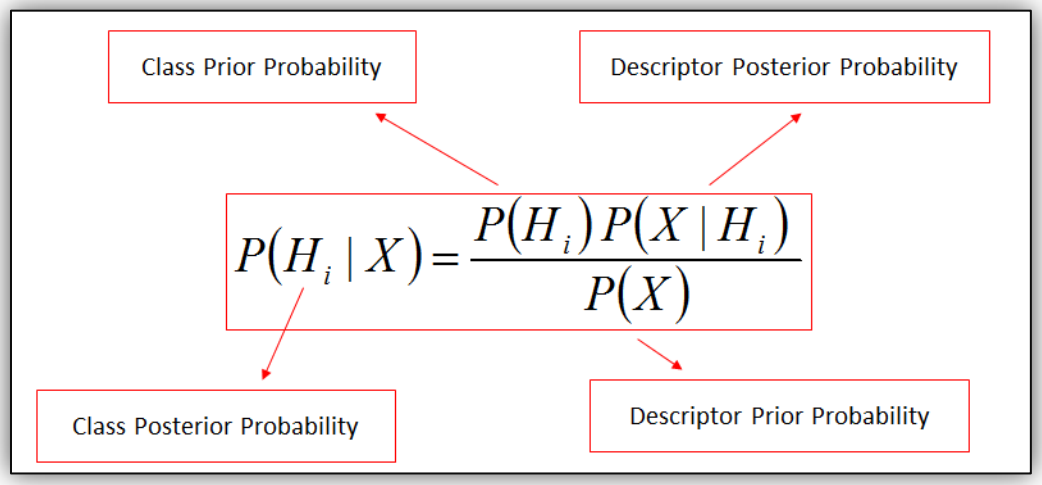
Part 3: Naïve Bayes Classifier

- A Naïve Bayes classifier is a probabilistic classifier which uses the Bayes Theorem to estimate the posterior probability.
- It assumes independence of attributes, that attributes are independent of the values of other attributes.
- It is simple to implement and works very well especially when applied to large databases.

Bayes Theorem:

- Bayes Theorem predicts the Class posterior probability for a certain row of data with independent attributes, and classifies the data into the class with highest P(Hi|X).

- Since P(X) would be constant throughout the dataset, we only need to compare values of P(Hi) * P(X|Hi) and pick the one with the highest value and classify the data with that label.



A) Steps for Naïve Bayes Classifier:

We have implemented the Naïve Bayes classifier in Java, which takes in as input a dataset and performs Naïve Bayes classification on it, predicts the labels and performs a 10-fold cross validation, choosing different set of rows of test data for each of the 10 runs, and outputs the average Accuracy, Precision, Recall and F-Measure values.

1. Take as input the filename for the dataset to be used. Eg: dataSet1.txt.

2. Read the data and store each line in a List of strings and also store the truth values (last column values) in a separate list.

3. Put each row of data in a HashMap with row number as key and the List of attributes of that row as the value.

4. Using 10-fold cross validation, separate the data into Training and Test data, with Test data being changed with each iteration, and remaining rows selected as Training data.

5. Train the classifier. For categorical attributes, calculate the probabilities of occurrence of each label with a specific attribute and store them in a Map. For continuous attributes, find out the mean and variance of each attribute, separated by labels (0 and 1) and store them separately.

6. Loop over the Test data and then loop over each attribute of a particular row of test data. Check if the current attribute is Continuous or Categorical.

7. If the attribute is continuous, then using the Probability Density Function (PDF), find the probability of that specific attribute value. The PDF is calculated using the formula:

$$P(X_i = x_i | Y = y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp^{-\frac{(x_i - \mu_{ij})^2}{2\sigma_{ij}^2}}.$$

   Here σ is the Standard Deviation of an attribute in the training set, and μ is the mean of the attribute in the training set, and χ is the value of the continuous attribute in the test data row.

8. The probabilities of each attribute of a given row of test data are multiplied to get P(X/H0) and P(X/H1), each of which are multiplied by P(H0) and P(H1) respectively. Since P(X) is constant for the whole dataset, it is not necessary to divide by P(X) to compare the values and classify. So we compare the values of P(X/H1) * P(H1) and P(X/H0) * P(H0), and the larger value is classified as the label for that row of data. For example if P(X/H1) * P(H1) >P(X/H0) * P(H0), then the row of data would be classified as "1", and P(X/H1) * P(H1) <P(X/H0) * P(H0), the row is classified as "0".

B) Dealing with Continous Features:

There are two ways of handling continuous values when using Naïve Bayes:

1. **Discretization:**

We can discretize the continuous attributes and replace the original values with the corresponding discrete interval, thus converting continuous attributes to ordinal attributes. However, this method depends a lot on how we discretize the data. If the number of intervals is too large then there would be very few training records for each interval, and if the number of intervals is too small then some intervals may combine records from different classes and miss the correct decision boundary.

2. **Using Probability Density Function:**

- Assuming a Gaussian distribution of the continuous values, we can estimate the class-conditional probability using the Probability Density Function(PDF), which needs the use of the training data mean and standard deviation (or variance, since variance is SD^2).

$$P(X_i = x_i | Y = y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp^{-\frac{(x_i - \mu_{ij})^2}{2\sigma_{ij}^2}}.$$

- The PDF is calculated using the above formula where σ is the Standard Deviation of an attribute in the training set, and μ is the mean of the attribute in the training set, and χ is the value of the continuous attribute in the test data row.

- We have used the probability density function to handle the continuous attributes in our implementation of the Naïve Bayes Classifier.

C) Dealing with Zero-Probability:

- Since the Descriptor posterior probability is a product of probabilities of each attribute for a given data, if even one of the probabilities is 0, then the whole Descriptor posterior probability is evaluated as 0.

- To deal with such a situation, we need to use the Laplacian Correction. In such a case as described above, we would add 1 to each of the count of labels when calculating the probability, so that none of the probabilities is estimated as 0.
- For example, If we have a dataset with 100 tuples for a class C, height = short (count 0), height = average (count 85) and height = tall (count 15), applying the Laplacian correction, the probabilities become as follows:

  ✓ Prob(height = short | H) = 1/103
  ✓ Prob(height = average| H) = 86/103
  ✓ Prob(height = tall | H) = 16/103

D) Results:

1. **dataset1.txt :** (10-fold Cross Validation)

Average Accuracy: **93.57%**	**Average Precision**: **92.41%**
Average Recall: **90.53%**	**Average F-Measure**: **0.9132**

2. **dataset2.txt :** (10-fold Cross Validation)

Average Accuracy: **70.22%**	**Average Precision**: **57.08%**
Average Recall: **61.75%**	**Average F-Measure**: **0.5842**

E) Result Analysis:

- On comparing the average performance metrics for Dataset 1 and 2, we can see that the Naïve Bayes classifier works substantially better for Dataset 1 than Dataset 2, however even for Dataset 2, it achieves a 70.2% accuracy over a 10-fold cross validation.
- We think this might have something to do with the fact that Dataset 1 contains all continuous values whereas Dataset 2 contains some categorical attributes along with continuous data.
- Also, since Naïve Bayes classifier has the drawback of assuming the independence of attributes in a given dataset, it might very well be the case that attributes in Dataset 2 are correlated, and hence score poorly on the performance metrics like accuracy, precision, recall and F-measure, all of which are substantially lower for Dataset 2.

F) Pros:

- Very simple and easy to implement.
- It is a fast learner, and trains on seeing the prior data immediately.

- Works very well for large databases.
- Comparable in performance to decision trees.
- Can handle both categorical and continuous attributes.

G) Cons:

- It assumes that attributes are independent of each other.
- Will not work well on data with attributes that are correlated.
- Will not work for labels that are new and not present in the training set.

Part 4: Random Forests

- Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

- 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.

- Random decision forests correct for decision trees' habit of overfitting to their training set.

A) Steps for Random Forests:

1. We take the parameter “T” from the user for the number of decision trees in the random forest.

2. We use “bagging” for training data, wherein if our training dataset contains N items, we pick N items randomly from the training data but with replacement. We have used “numpy.random.choice()” method to implement bagging.

3. For building a tree, at every node we select “m” number of features (20% of total features) for selecting the best feature for the split.

4. We repeat steps 2, 3 “T” times to build “T” trees.

5. For testing data, for every item in the testing dataset we predict the class “T” number of times for “T” trees.

6. We then select the majority class out of all trees as the final predicted class.

B) Results:

1. dataset1.txt

T	Accuracy	Precision	Recall	F-measure
3	93.87%	95.23%	87.74%	0.912
5	94.41%	94.04%	90.34%	0.920
7	95.48%	95.36%	91.83%	0.933

2. dataset2.txt

T	Accuracy	Precision	Recall	F-measure
3	65.13%	50.65%	42.13%	0.447
5	66.20%	50.27%	40.83%	0.441
7	61.06%	42.24%	36.00%	0.381

C) Result Analysis:

1. We have tested our algorithm for number of trees "T" varying from 3 – 7 and we have observed that in most of the cases, for both datasets the accuracy is increasing. This shows that as the number of trees increase the algorithm works better in predicting correctly.

2. We see that recall is also increasing in every case which tells us the increase in true positives in every case with increasing "T".

D) Pros:

- Efficient over large databases.
- One of the most accurate classifying algorithms .
- Can estimate missing data and stay accurate even when large portion of data is missing.
- Gives indication of important attributes in the classification.

E) Cons:

- Does not work well with continuous data.
- Slower than methods like Boosting.
- Classification is difficult to interpret, compared to Decision Trees.
- With too many classes, misclassification rate is high.

Part 5: Boosting

- Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones.

- A weak learner is defined to be a classifier which is only slightly correlated with the true classification (it can label examples better than random guessing). In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification.

Adaboost:

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm can be used in conjunction with many other types of learning algorithms to improve performance.

A) Steps for AdaBoost on Decision Tree:

1. Take the parameter "L" from the user that indicates the number of classifiers to be used.

2. Set weights to the training data such that all initial weights are 1/ size of training data.

3. Use weighted - bagging to draw N random items from N sized dataset based on weights. We have used "numpy.random.choice()" method to implement this.

4.Use this bag to train the classifier and build the tree.

5.Now, use the entire training dataset to test the built decision tree.

6.Calculate error by adding all the weights where the items have been misclassified divided by the sum of all weights.

7.Calculate "alpha" by using the formula: alpha = 0.5*log((1.0-error)/error). We save this alpha for future use.

8.Update the weights for all training data by the formula: w[i] = w[i]*exp((-1*alpha)*X) where, X is 1 if that item was classified correctly in step 5 and -1 if it wasn't classified correctly.

9.Repeat steps 3 - 8 for "L" number of classifiers such that you have "L" trees and "L" alphas.

10. Test the testing data by predicting the label on every tree out of "L" and multiply it with the corresponding alpha for that particular tree such that its 1*alpha if predicted 1 or -1*alpha the classifier predicted 0.

11. Add all the values calculated such that the final prediction is 1 if summation value is greater than 0 or else, it's -1 (0 in our case).

Results:

1. dataset1.txt

L	Accuracy	Precision	Recall	F-measure
3	94.36%	94.56%	89.48%	0.92
5	94.97%	93.96%	92.84%	0.931
7	94.23%	94.31%	90.44%	0.92

2. dataset2.txt

L	Accuracy	Precision	Recall	F-measure
3	62.78%	46.6%	44.54%	0.44
5	63.86%	50.46%	42.60%	0.44
7	64.75%	49.34%	45.58%	0.46

B) Result Analysis:

1. We have tested our algorithm for number of learners "L" varying from 3 to 7 and we have observed that boosting didn't have much impact on Dataset 1 as the accuracy is almost same in every case.
2. The results of dataset 2 show an increase in accuracy in every case which tells that boosting does a good job in classifying categorical data.

C) Pros:

- Uses many different weak classifiers to form a strong classifier.
- Gives a better result than compared to individual single classifier.
- Can run efficiently on large datasets.
- Does not require a lot of parameters.

D) Cons:

- Susceptible to noise and outliers.
- Computationally expensive.
- Hard to implement.

Overall Result Analysis for Decision Tree, Random Forests and Boosting:

After running all three algorithms on project3_dataset1.txt and project3_dataset2.txt we have the following analysis:

1. Both, Random Forest and AdaBoost work better than Decision Tree and help classify better.
2. Random Forest works really well by increasing the number of trees but takes more computation time.
3. Boosting works well on categorical data and predicts well on increasing the number of classifiers.

Cross Validation

- For getting a better assessment of how the classifiers would perform on unseen data, we use k-fold cross validation.

- A k-fold Cross validation partitions the data into k disjoint subsets, and uses k-1 subsets for training, and the kth subset as testing data, and this is done k times, such that, each time the testing data is a different subset out of the k subsets.

- For each fold of cross validation, we calculate the performance metrics: Accuracy, Precision, Recall and F-measure, and at the end of k-rounds, we average each metric and display the average results.

Performance Metrics:

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

TP : True Positive | **TN**: True Negative | **FP** : False Positive | **FN**: False Negative

The performance metrics used are defined as follows:

- **Accuracy** : Accuracy can be calculated as (TP+TN) divided by (TP + TN + FP + FN)

- **Precision (p)** : Precision is calculated as TP divided by (TP + FP)

- **Recall (r)** : Recall is calculated as TP divided by (TP + FN)

- **f-Measure** : It is the harmonic mean of Recall and Precision, and can be calculated as 2*r*p divided by (r + p) , and is also equal to 2*TP divided by ((2 * TP) + FN + FP)