Project 2: Feature Selection with Nearest Neighbor

CS205 Artificial Intelligence (Winter 2019)
Instructor: Dr. Eamonn Keogh

Lokesh Koppaka
ID#: 862123164
lkopp001@ucr.edu
21-March-2019

In completing the project, I have consulted

- Class presentations of Feature Selection provided by Dr. Eamonn Keogh
- The format of the report is taken from sample report provided during Project 1

The code and the documentation are entirely original

- Used Priority Queues in Java to get the feature set with maximum accuracy
- Used Collection package in Java
- The standard compare method is overridden to perform custom sort and comparations

NOTE: As an initiative to save Trees, the report is printed in both sides of the paper

Introduction:
In the field of Artificial Intelligence, features play a vital role in the accuracy of any AI algorithm. Hence it is required to consider the right set of features for a better performance of the algorithm. The report demonstrates the importance of the right feature set and determines them by applying the concept of feature selection on the Nearest Neighbors Classification problem.

Nearest Neighbors Classification Problem:
Nearest Neighbors is a supervised AI classification problem. It classifies the test instances by computing its distance from all the other instances in the training set and classifies it with the class of the nearest instance. The distance measure usually varies with the domain of interest in this case since the instances are numeric values in nature, we consider Euclidean distance. The method "computeNearestNeighbhor()" with parameter "myownAlgo" set to false performs the Nearest Neighbors. The implementation details are provided as comments in the code snippet section

Feature Selection:
Having the right features set, is one of primary goal in Artificial Intelligence to achieve better accuracy of the algorithm. Feature Selection is one such approach that helps in selecting the right features for a given problem. It works by validating every combination of the features and picking up the feature set that has the highest accuracy. This approach tends to increase its complexity exponentially with the number of features since the number of possible features subsets to examine for features of size n = $2^n - 1$. The obvious solution for this challenge is to perform greedy search on the feature set problem space.

The following are the greedy search approaches used

Forward Selection:
In forward Selection, the search starts by computing accuracy for each single feature and pick up the feature with the highest accuracy. The selected feature is then combined with other features to form a feature set of size 2, for each feature set again the accuracy is computed and the feature set with highest accuracy is selected. This process of combining the feature and selecting the feature set with highest accuracy continues till the end of the search space. The method "forwardSelection()" with parameter "myownAlgo" set to false performs forward selection. The implementation details are provided as comments in the code snippet section.

The following graph demonstrates how accuracy performs with various feature sets selected by forward selection

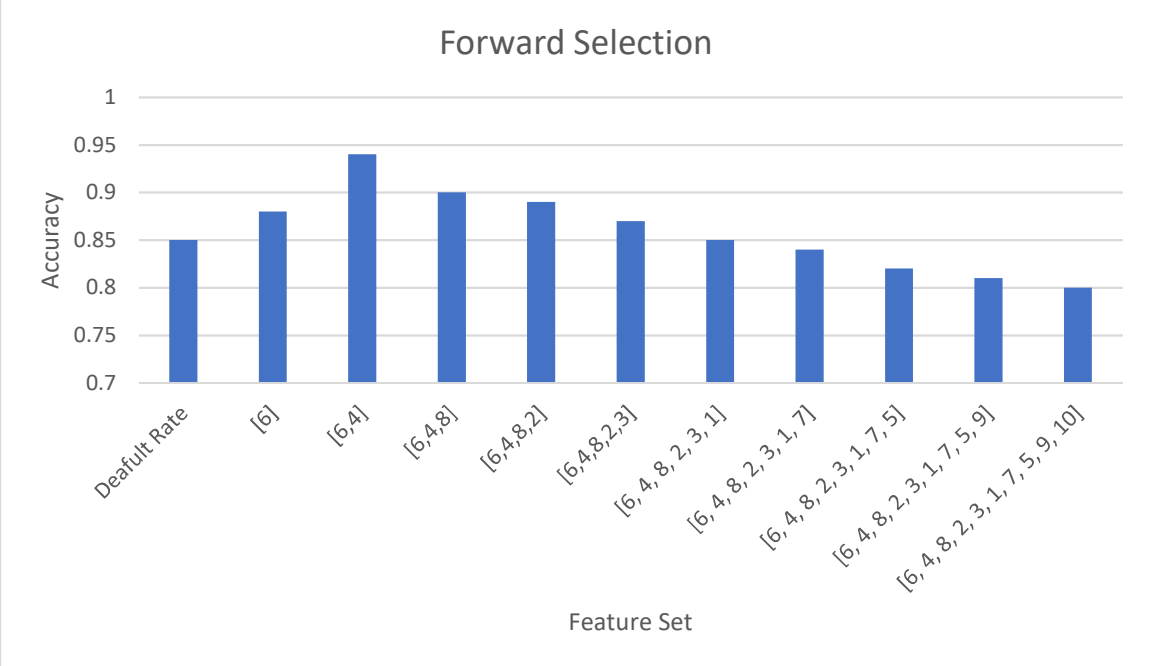

Fig 1.0 Feature Selection with Forward Selection

Backward Elimination:

In backward elimination, initially the accuracy of the algorithm with respect to all the features are computed. Later at each level of the tree the accuracy is computed by eliminating other features and selecting the feature set with highest accuracy. This process of eliminating the feature and selecting the feature set with highest accuracy continues till the features set is empty. The method "backwardElimination()" with parameter "myownAlgo" set to false performs backward elimination. The implementation details are provided as comments in the code snippet section.

The following graph demonstrates how accuracy performs with various features set selected by backward elimination

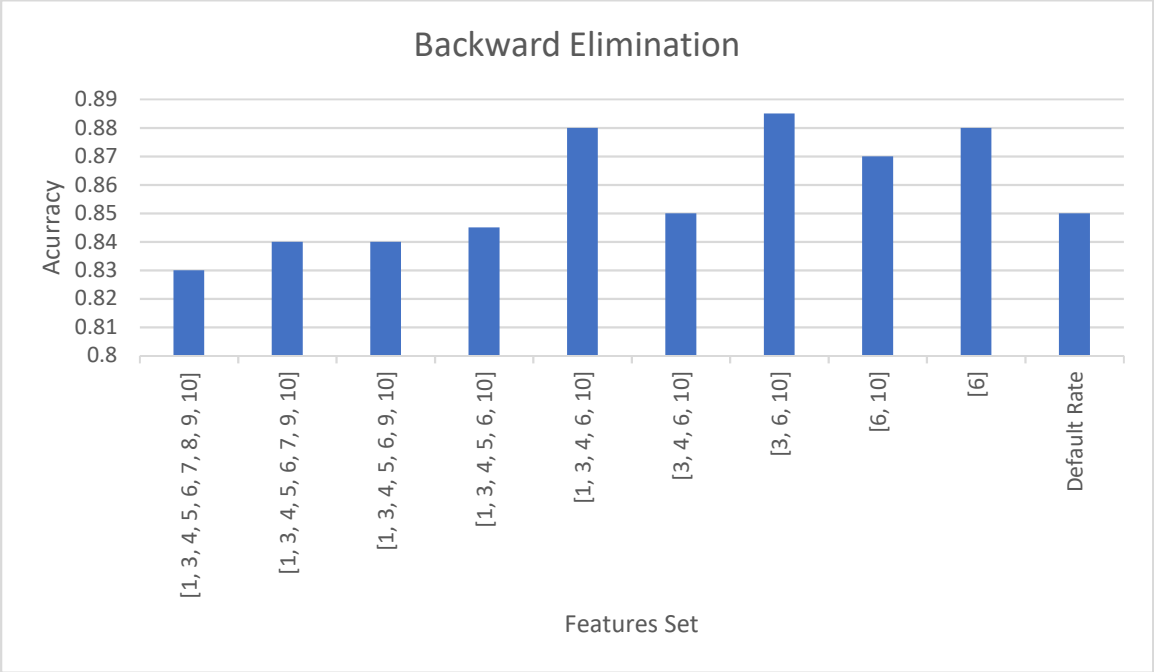

Fig 2.0 Feature Selection with Backward Elimination

Improved Search Algorithm (Special Algorithm):

The above mentioned approaches are further been optimized by using the concept like Alpha – beta pruning. In order to increase the speed of the searching process using the concept of Alpha- beta pruning, a global variable holding the max accuracy is maintained and while computing accuracy w.r.t to current feature set if the number of misclassified exceeds the max misclassified then the algorithm prunes the process of further classifying w.r.t current feature set since any further would yield an accuracy not better than the current max accuracy. The algorithm is further made more faster by placing the misclassified instances at the top of the training set. This improved search by around "92%" on an average which is demonstrated in the comparison graphs section. The method "computeNearestNeighbhor()" with parameter "myownAlgo" set to true performs the above optimization techniques. The implementation details are provided as comments in the code snippet section.

Comparison Graphs:

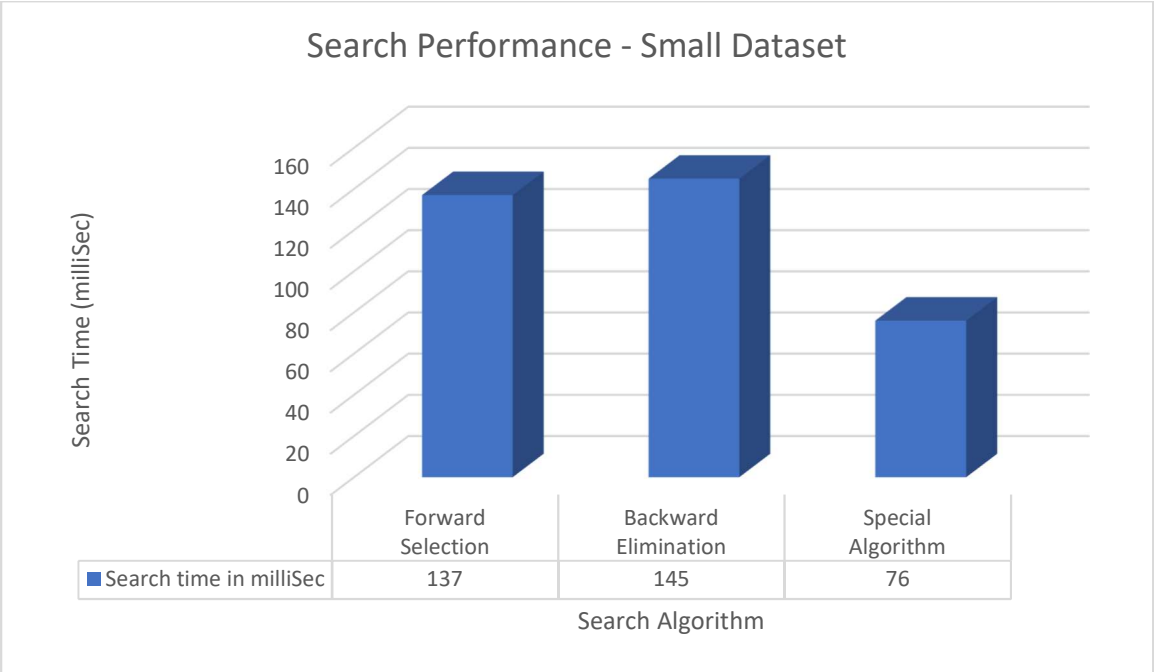

Fig 3.0 Search Performance-small DataSet

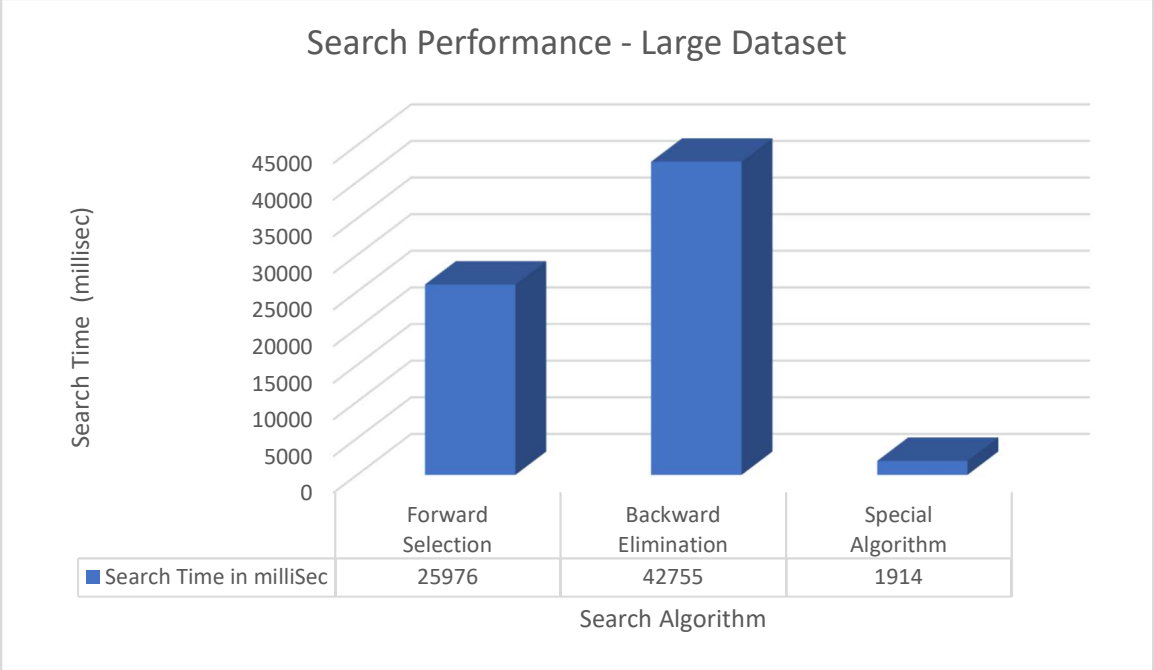

Fig 4.0 Search Performance – Large DataSet

From the above comparison graphs it can be observed that special algorithm has a performance of about 90% higher than Forward Selection and Backward Elimination

Solutions:

The following table holds the feature set computed for the assigned datasets

Data Set Name	Feature Set	Accuracy
CS205_LARGEtestdata__36	{69, 44}	95%
CS205_SMALLtestdata__59	{6,4}	94.5%

Conclusions:

- Nearest Neighbor classifier performs better in terms of accuracy with strong feature set
- Forward Selection performs better than backward elimination in terms of feature selection
- Forward Selection is faster than backward elimination
- Forward Selection and Backward elimination can further made faster using pruning.
- The process of pruning, movement of misclassified instances to top of the training list can significantly improve the search speed by about “92%”

Example Trace on 10 feature problem:

```
Welcome to CS205 Feature Selection Algorithm.
Type in the name of the file to test :
D:\\UCR\\UCR_SecondQuater_Winter2019\\ArtificialIntelligence\\Project_2\\datasets\\CS205_SMALLtestdata__59.txt
Type the number of the algorithm you want to run.
        1. Forward Selection
        2. Backward Elimination
        3. Improved performance Algorithm
1
This dataset has 10 features (not including the class attribute) with 200 instances

Running nearest neighbor with all 10 features, using "leaving-one-out" evaluation, I get an accuracy of 85.5%

Beginning search.

        Using feature(s) [1] accuracy is 78.0%
        Using feature(s) [2] accuracy is 74.0%
        Using feature(s) [3] accuracy is 74.0%
        Using feature(s) [4] accuracy is 76.5%
        Using feature(s) [5] accuracy is 75.5%
        Using feature(s) [6] accuracy is 88.5%
        Using feature(s) [7] accuracy is 75.0%
        Using feature(s) [8] accuracy is 75.0%
        Using feature(s) [9] accuracy is 78.0%
        Using feature(s) [10] accuracy is 74.5%

Feature Set [6] was best, accuracy is 88.5%

        Using feature(s) [6, 1] accuracy is 84.0%
        Using feature(s) [6, 2] accuracy is 86.0%
        Using feature(s) [6, 3] accuracy is 86.0%
        Using feature(s) [6, 4] accuracy is 94.5%
        Using feature(s) [6, 5] accuracy is 86.5%
        Using feature(s) [6, 7] accuracy is 84.5%
        Using feature(s) [6, 8] accuracy is 85.0%
        Using feature(s) [6, 9] accuracy is 85.5%
        Using feature(s) [6, 10] accuracy is 87.5%

Feature Set [6, 4] was best, accuracy is 94.5%

        Using feature(s) [6, 4, 1] accuracy is 88.0%
        Using feature(s) [6, 4, 2] accuracy is 84.5%
        Using feature(s) [6, 4, 3] accuracy is 87.5%
        Using feature(s) [6, 4, 5] accuracy is 85.0%
        Using feature(s) [6, 4, 7] accuracy is 89.0%
        Using feature(s) [6, 4, 8] accuracy is 90.0%
        Using feature(s) [6, 4, 9] accuracy is 88.5%
        Using feature(s) [6, 4, 10] accuracy is 87.5%
```

```
Feature Set [6, 4, 8] was best, accuracy is 90.0%

        Using feature(s) [6, 4, 8, 1] accuracy is 86.5%
        Using feature(s) [6, 4, 8, 2] accuracy is 89.5%
        Using feature(s) [6, 4, 8, 3] accuracy is 89.5%
        Using feature(s) [6, 4, 8, 5] accuracy is 86.5%
        Using feature(s) [6, 4, 8, 7] accuracy is 87.0%
        Using feature(s) [6, 4, 8, 9] accuracy is 84.5%
        Using feature(s) [6, 4, 8, 10] accuracy is 86.5%

Feature Set [6, 4, 8, 2] was best, accuracy is 89.5%

        Using feature(s) [6, 4, 8, 2, 1] accuracy is 82.5%
        Using feature(s) [6, 4, 8, 2, 3] accuracy is 87.5%
        Using feature(s) [6, 4, 8, 2, 5] accuracy is 84.0%
        Using feature(s) [6, 4, 8, 2, 7] accuracy is 86.0%
        Using feature(s) [6, 4, 8, 2, 9] accuracy is 81.0%
        Using feature(s) [6, 4, 8, 2, 10] accuracy is 83.5%

Feature Set [6, 4, 8, 2, 3] was best, accuracy is 87.5%

        Using feature(s) [6, 4, 8, 2, 3, 1] accuracy is 85.0%
        Using feature(s) [6, 4, 8, 2, 3, 5] accuracy is 83.5%
        Using feature(s) [6, 4, 8, 2, 3, 7] accuracy is 81.0%
        Using feature(s) [6, 4, 8, 2, 3, 9] accuracy is 82.5%
        Using feature(s) [6, 4, 8, 2, 3, 10] accuracy is 83.5%

Feature Set [6, 4, 8, 2, 3, 1] was best, accuracy is 85.0%

        Using feature(s) [6, 4, 8, 2, 3, 1, 5] accuracy is 83.0%
        Using feature(s) [6, 4, 8, 2, 3, 1, 7] accuracy is 84.0%
        Using feature(s) [6, 4, 8, 2, 3, 1, 9] accuracy is 84.0%
        Using feature(s) [6, 4, 8, 2, 3, 1, 10] accuracy is 83.5%

Feature Set [6, 4, 8, 2, 3, 1, 7] was best, accuracy is 84.0%

        Using feature(s) [6, 4, 8, 2, 3, 1, 7, 5] accuracy is 82.0%
        Using feature(s) [6, 4, 8, 2, 3, 1, 7, 9] accuracy is 79.5%
        Using feature(s) [6, 4, 8, 2, 3, 1, 7, 10] accuracy is 81.5%


Feature Set [6, 4, 8, 2, 3, 1, 7, 5] was best, accuracy is 82.0%

        Using feature(s) [6, 4, 8, 2, 3, 1, 7, 5, 9] accuracy is 81.0%
        Using feature(s) [6, 4, 8, 2, 3, 1, 7, 5, 10] accuracy is 79.5%

Feature Set [6, 4, 8, 2, 3, 1, 7, 5, 9] was best, accuracy is 81.0%

        Using feature(s) [6, 4, 8, 2, 3, 1, 7, 5, 9, 10] accuracy is 80.5%

Feature Set [6, 4, 8, 2, 3, 1, 7, 5, 9, 10] was best, accuracy is 80.5%

Finished search!! The best feature subset is [6, 4], which has accuracy of 94.5%
Time Taken = 138 ms
```

Code:

NearestNeighbor.java

```
/* @Author : Lokesh Koppaka(862123164)
 * Course  : CS 205 Artificial Intelligence
 * Description : This class performs  Nearest Neighbor and measures the accuracy of it across various
feature set using following Algorithms and outputs the feature set (Strong features) with maximum accuracy
 *                  1. Forward Selection
 *                  2. Backward Elimination
 *                  3. Improved performance Algorithm
 */

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Collections;
import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Scanner;


public class NearestNeighbor {
	/* Class Name : FeatureSet
	 * Description : Inner class to hold the feature set and accuracy together. The instance of this
object is later used to retrieve Feature set with maximum accuracy
	 */
	class FeatureSet implements Comparable<FeatureSet>{
		List<Integer> features; // List - holds the list of features
		Double accuracy; // Double - holds the accuracy
		// Constructor performs Initialization
		FeatureSet(List<Integer> features, Double accuracy){
			this.features = features;
			this.accuracy = accuracy;
		}
		// Custom compare to sort the elements based on accuracy
		@Override
		public int compareTo(FeatureSet fs) {
			return accuracy.compareTo(fs.accuracy);
		}

	}
	/* Class Name : DistanceToPoint
	 * Description : Inner class to hold the distance and point together. The instance of this object
is later used to retrieve data point with minimum Euclidean distance
	 */
	class DistanceToPoint{
		Double distance; // Double - holds the Euclidean distance to the current point
		DataPoint point; // DataPoint Instance - holds the data point
		// Constructor performs Initialization
		DistanceToPoint(Double distance, DataPoint point){
			this.distance = distance;
			this.point = point;
		}
	}
	/* Class Name : DataPoint
	 * Description : Inner class represents a  single data point
	 */
	class DataPoint{
		Double classLabel; // Double classLabel - holds the data point class label
		List<Double> featureList; // List featureList - holds data point features
		// Constructor performs Initialization
		DataPoint(Double classLabel, List<Double> featureList){
			this.classLabel = classLabel;
			this.featureList = featureList;
		}
	}
```

```
	List<DataPoint> trainingDataSet; // List trainingDataSet - holds all the dataPoints
	private Integer numberOfFeatures; // Integer - holds no of features
	public Double defaultRate;        // Double - holds the default rate
	double maxAccuracy;   // Double - holds max Accuracy at that current instance of time
	int maxWrongGuess; // Double - holds max Wrong guesses can made, this is later used to improve the performance
	// Constructor performs Initialization
	public NearestNeighbor() {
		trainingDataSet = new ArrayList<DataPoint>();
		numberOfFeatures = 0;
		maxAccuracy = 0.0;
		maxWrongGuess = 0;
	}
	/* Method Name: computeNearestNeighbhor
	 * Parameters  :
	 * featureList - the list of features to consider while computing nearestNeighbhor
	 * myOwnAlgo - Boolean flag to run improved version of the algorithm for fast search using Alpha Beta pruning
	 * Description: Computes Nearest Neighbors by performing leave one out cross validation and returns the accuracy achieved using the featureSet. Additionally, the method holds logic to perform Alpha Beta pruning to improve search
	 */
	public double computeNearestNeighbhor(List<Integer> featureList, Boolean myOwnAlgo) {
		double accuracy = 0.0;
		int correctDataPoints =0;
		int size = trainingDataSet.size();
		int wrongDataPoints = 0; // Int - holds the number of data points classified wrong
		/* List pointsGuessedWrong - holds the wronging classified data points, later these are pushed to the top of training set to make Search faster
		 */
		List<DataPoint> pointsGuessedWrong = new ArrayList<DataPoint>();
		Boolean flag =  false;
		// Logic to perform NearestNeighbhor using K fold cross validation where K = 1
		for(int i = 0 ; i < size ; i++) {
			DistanceToPoint minDist = new DistanceToPoint(Double.MAX_VALUE,null);
			for(int j = 0; j < size; j++) {
				if(i != j) {
					double sum = 0.0;
					// Logic to compute Euclidean distance , consider only the features in the featureList parameter
					for(int k  = 0; k < featureList.size() ; k ++) {
						sum += Math.pow((trainingDataSet.get(j).featureList.get(featureList.get(k) - 1) - trainingDataSet.get(i).featureList.get(featureList.get(k) - 1)),2);
					}
					sum = Math.sqrt(sum);
					if(sum < minDist.distance) {
						minDist = new DistanceToPoint(sum,trainingDataSet.get(j));
					}
				}
			}
			// Logic to keep track correctly classified data points and wrongly classified data points
			if(minDist.point.classLabel.equals(trainingDataSet.get(i).classLabel)) {
				correctDataPoints++;
			}else {
				pointsGuessedWrong.add(trainingDataSet.get(i));
				wrongDataPoints ++;
			}
			// if wrongly classified data points exceeded the maxWrongGuess limit set accuracy as zero - Alpha beta pruning
			if(myOwnAlgo && wrongDataPoints > maxWrongGuess) {
				flag = true;
				break;
			}
		}
		if(myOwnAlgo) {
```

```
                        // Logic to push wrongly guessed data points to the top of training set.
                        for(DataPoint p : pointsGuessedWrong) {
                                trainingDataSet.remove(p);
                        }
                        pointsGuessedWrong.addAll(trainingDataSet);
                        trainingDataSet = pointsGuessedWrong;
                        // Logic to return accuracy as 0 if wrongly classified data points exceeded the
maxWrongGuess
                        if(flag) {
                                return 0.0;
                        }
                }
                // Logic to compute accuracy
                accuracy = (double)correctDataPoints/trainingDataSet.size();
                // Logic to set the threshold maxWrongGuess, this is used during pruning
                if(accuracy > maxAccuracy) {
                        maxAccuracy = accuracy;
                        maxWrongGuess = size - correctDataPoints;
                }
                return accuracy;
        }
        /* Method Name : forwardSelection
         * Parameters  : myOwnAlgo -Boolean flag to run improved version of the algorithm for fast search
using Alpha Beta pruning, the value is passed to compute NearestNeighbhor
         * Description : performs Forward Feed Search and output the feature set with max accuracy
         */
        public void forwardSelection(Boolean myown) {
                List<Integer> featureSet = new ArrayList<Integer>(); // List featureSet - holds the list of
features
                List<Integer> bestFeatureSet = new ArrayList<Integer>();// List bestFestureSet - holds best
features
                int i = 1;
                /* PriorityQueue featurePriorityQueue - holds the instances of featureSet at each level,
this is used to retrieve the best feature set at a particular level
                 * PriorityQueue topFeatures - holds the instances of featureSet at entire tree level, this
is used to retrieve the best feature set across the tree
                 */
                PriorityQueue<FeatureSet> featurePriorityQueue = new
PriorityQueue<FeatureSet>(Collections.reverseOrder());
                PriorityQueue<FeatureSet> topFeatures = new
PriorityQueue<FeatureSet>(Collections.reverseOrder());
                // Logic - generates feature set at each level and computes Nearest Neighbors for the
generated feature set
                while(bestFeatureSet.size() < numberOfFeatures) {
                        featureSet = new ArrayList<Integer>();
                        if(!bestFeatureSet.isEmpty()) {
                                featureSet.addAll(bestFeatureSet);
                        }
                        FeatureSet fs;
                        if(!featureSet.contains(i)) {
                                featureSet.add(i);
                                // Logic to call NearestNeighbhot method to retrieve the accuracy for the
generated feature Set
                                Double accuracy = computeNearestNeighbhor(featureSet, myown);
                                System.out.println("\tUsing feature(s) " + featureSet + " accuracy is " +
(accuracy*100) + "%");
                                fs = new FeatureSet(featureSet,accuracy);
                                featurePriorityQueue.add(fs);
                        }
                        i ++;
                        // Logic to get the best feature set at each level
                        if(i % (numberOfFeatures + 1) == 0) {
                                bestFeatureSet = new ArrayList<Integer>();
                                fs = featurePriorityQueue.poll();
                                System.out.println();
                                System.out.println("Feature Set " + fs.features + " was best, accuracy is " +
(fs.accuracy * 100) +"%");
                                System.out.println();
```

```
                                topFeatures.add(fs);
                                bestFeatureSet.addAll(fs.features);
                                i = 1;
                                featurePriorityQueue = new
PriorityQueue<FeatureSet>(Collections.reverseOrder());
                                if(fs.accuracy > maxAccuracy) {
                                        maxAccuracy = fs.accuracy;
                                        maxWrongGuess = trainingDataSet.size() - (int)(maxAccuracy *
trainingDataSet.size());
                                }
                        }
                }
                FeatureSet fs = topFeatures.poll();
                System.out.println("Finished search!! The best feature subset is " + fs.features + ", which
has accuracy of "+ (fs.accuracy * 100) +"%");
                maxAccuracy = 0;
                maxWrongGuess = trainingDataSet.size();
        }
        /* Method Name : backwardEllimination
         * Parameters  : myOwnAlgo-Boolean flag to run improved version of the algorithm for fast search
using Alpha Beta pruning the value is passed to compute NearestNeighbhor
         * Description : performs Back propagation Search and output the feature set with max accuracy
         */
        public void backwardElimination(Boolean myown) {
                List<Integer> featureSet = new ArrayList<Integer>(); // List featureSet - holds the list of
features
                List<Integer> bestFeatureSet = new ArrayList<Integer>(); // List bestFestureSet - holds best
features
                for(int i = 1; i <= numberOfFeatures ; i++) {
                        bestFeatureSet.add(i);
                }
                int i = 1;
                /* PriorityQueue featurePriorityQueue - holds the instances of featureSet at each level,
this is used to retrieve the best feature set at a particular level
                 * PriorityQueue topFeatures - holds the instances of featureSet at entire tree level, this
is used to retrieve the feature set across the tree
                 */
                PriorityQueue<FeatureSet> featurePriorityQueue = new
PriorityQueue<FeatureSet>(Collections.reverseOrder());
                PriorityQueue<FeatureSet> topFeatures = new
PriorityQueue<FeatureSet>(Collections.reverseOrder());
                while(bestFeatureSet.size() > 1) {
                        featureSet = new ArrayList<Integer>();
                        if(!bestFeatureSet.isEmpty()) {
                                featureSet.addAll(bestFeatureSet);
                        }
                        FeatureSet fs;
                        if(featureSet.contains(i)) {
                                featureSet.remove(featureSet.indexOf(i));
                                if(!featureSet.equals(bestFeatureSet)) {
                                        // Logic to call NearestNeighbhot method to retrieve the accuracy for
the generated feature Set
                                        Double accuracy = computeNearestNeighbhor(featureSet,myown);
                                        System.out.println("\tUsing feature(s) " + featureSet + " accuracy is
" + (accuracy*100) + "%");
                                        fs = new FeatureSet(featureSet,accuracy);
                                        featurePriorityQueue.add(fs);
                                }
                        }
                        i ++;
                        // Logic to get the best feature set at each level
                        if(i % (numberOfFeatures + 1) == 0) {
                                bestFeatureSet = new ArrayList<Integer>();
                                fs = featurePriorityQueue.poll();
                                System.out.println();
                                System.out.println("Feature Set " + fs.features + " was best, accuracy is " +
(fs.accuracy * 100) +"%");
                                System.out.println();
```

```
                                topFeatures.add(fs);
                                bestFeatureSet.addAll(fs.features);
                                i = 1;
                                featurePriorityQueue = new
PriorityQueue<FeatureSet>(Collections.reverseOrder());
                                if(fs.accuracy > maxAccuracy) {
                                        maxAccuracy = fs.accuracy;
                                        maxWrongGuess = trainingDataSet.size() - (int)(maxAccuracy *
trainingDataSet.size());
                                }
                        }
                }
                FeatureSet fs = topFeatures.poll();
                System.out.println("Finished search!! The best feature subset is " + fs.features + ", which
has accuracy of "+ (fs.accuracy * 100) +"%");
                // Once done setting back maxAccuracy and maxWrongGuess to default values
                maxAccuracy = 0;
                maxWrongGuess = trainingDataSet.size();
        }
        /* Method Name: loadDataPoints
         * parameters : Location - the file location of the training data set
         * Description: Reads the data points and constructs the training DataSet List and computes default
rate
         */
        public void loadDataPoints(String location) {
                BufferedReader reader;
                try {
                        reader = new BufferedReader(new FileReader(location));
                        int lines = 0;
                        while (reader.readLine() != null) lines++;
                        int linesRead = 1;
                        reader = new BufferedReader(new FileReader(location));
                        String line = reader.readLine();
                        int classOneCount = 0;
                        int classTwoCount = 0;
                        while (line != null) {
                                DataPoint dp = createDataPointInstance(line);
                                if(dp.classLabel.equals(1.0)) {
                                        classOneCount ++;
                                }else if(dp.classLabel.equals(2.0)) {
                                        classTwoCount ++;
                                }
                                trainingDataSet.add(dp);
                                // read next line
                                line = reader.readLine();
                                linesRead = linesRead + 1;
                        }
                        defaultRate = (double)Math.max(classOneCount, classTwoCount)/lines;
                         maxAccuracy = 0;
                         maxWrongGuess = lines;
                         System.out.println("This dataset has "+ numberOfFeatures + " features (not including
the class attribute) with " + lines + " instances");
                         System.out.println();
                         System.out.println("Running nearest neighbor with all " + numberOfFeatures + "
features, using \"leaving-one-out\" evaluation, I get an accuracy of " + (defaultRate * 100) + "%");
                         System.out.println();
                        reader.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        /* Method : createDataPointInstance
         * Parameter : datapoint - the stringify version of the data point
         * Description : Helper method, converts the stringify version of the data point to DataPoint class
instance
         */
        public DataPoint createDataPointInstance(String datapoint){
                datapoint = datapoint.trim();
```

```
                String [] datapointArr = datapoint.split("\\s+");
                int dpLen = datapointArr.length;
                List<Double> featureList = new ArrayList<Double>();
                for(int i = 1; i < dpLen ; i++) {
                        featureList.add(Double.valueOf(datapointArr[i]));
                }
                if(numberOfFeatures == 0) {
                        numberOfFeatures = featureList.size();
                }
         return new DataPoint(Double.valueOf(datapointArr[0]), featureList);

        }

        public static void main(String args[]) {
                System.out.println("Welcome to CS205 Feature Selection Algorithm.");
                System.out.println("Type in the name of the file to test :");
                Scanner sc = new Scanner(System.in);
                String location = sc.nextLine();
                System.out.println("Type the number of the algorithm you want to run.");
                System.out.println("\t1. Forward Selection");
                System.out.println("\t2. Backward Elimination");
                System.out.println("\t3. Improved performance Algorithm");
                int choice = sc.nextInt();
                NearestNeighbor nn = new NearestNeighbor();
                nn.loadDataPoints(location);
                System.out.println("Beginning search.");
                System.out.println();
                if(choice == 1) {
                        long startTime = System.nanoTime();
                        nn.forwardSelection(false);
                        long endTime = System.nanoTime();
                        System.out.println("Time Taken = "+ (endTime - startTime)/1000000 + " ms");
                }else if(choice == 2) {
                        long startTime = System.nanoTime();
                        nn.backwardElimination(false);
                        long endTime = System.nanoTime();
                        System.out.println("Time Taken = "+ (endTime - startTime)/1000000 + " ms");
                }else if(choice == 3) {
                        long startTime = System.nanoTime();
                        nn.forwardSelection(true);
                        long endTime = System.nanoTime();
                        System.out.println("Time Taken = "+ (endTime - startTime)/1000000 + " ms");
                }else {
                        System.out.println("Please Select valid choice..");
                }
        }

}
```