Classification Algorithms

Meghana Ananth Gad (50182335)
Ladan Golshanara (50093954)
Duc Thanh Anh Luong (50094977)

December 2016

1 Introduction

In this report, we explain our implemetations of K-nearest neighbor, Naïve Bayes, Decision Tree, Random forest and Adaboost classifiers. All algorithms are implemented using R language.

2 Data sets

Dataset1: This dataset has 30 features and contains 569 records. The class lables are either 0 or 1. All features are numeric (continous type).
Dataset2: Dataset2 has 9 features and contains 462 records. The class labels are either 0 or 1. All features (except one) are numberic. One feature is discrete: "absent" and "present".

3 10-fold Cross Validation

The test_data and train_data is given by 10-fold cross validation algorithm which is implemented on top of the classifier algorithms. 10-fold cross validation, divides the data into 10 parts, and gives 1-part as the test_data and remaining 9-part as train_data. In the next iteration, it will choose another part as the test_data so that every part is exactly used once as the test_data.

4 k-Nearest Neighbor Classification

KNN is a *lazy* algorithm. KNN uses k nearest neighbor (by distance) for classifing new records. The psuedo code of our implemnetation is given in Algorithm 1

Algorithm 1 K-Nearest Neighbor

Input: train_data, test_data , K
for Each point p in the test_data **do**
    dis_lst := Calculate the Euclidean distance of P to every point in train_data.
    Sort dis_lst
    knn := Choose the k top points from dis_lst
    $Zeros$:=Count the number of points in knn with label 0
    $Ones$:=Count the number of points in knn with label 1
    Assign $max(Zeros, Ones)$ as the label to np
end for
return predicted labels for test_data

	Dataset1	Dataset2
Average Accuracy	0.9279	0.6755
Average Precision	0.9544	0.5584
Average Recall	0.8465	0.3067
Average F1 measure	0.8953	0.3915

Table 1: KNN algorithm on the given Datasets

4.1 Choosing K

One challenge in implementing the KNN algorithm is choosing k. As suggested in [1] we have experimented from $k = 1$ to square root of the number of records in the training set and calculate the accuracy and F-1 measure. The best value we got was when k is equal to square root of the number of records in the training set.

4.2 Continuous/Categorical Data

In Dataset1 all the feature values are continous and calculating distance is straightforward. In Dataset2 there is a feature with binary values that inorder to use it in the algorithm 1 we change the values of "present" to 1 and "absent" to 0.

4.3 Experiments

In Table 1 we report the values obtained by running 10-fold cross validation with knn.

4.4 Pros and cons of KNN

Pros

1. Simple in implementation.
2. Works well when data is continous

Cons

1. Need to determine parameter k
2. The notion of *distance* is not clear.
3. Slow in run time when k is large and dataset is big
4. not scalable

5 Naïve Bayes Classification

Naïve Bayes classiofication is based on *Bayes Theorem* with an assumption of independence among features. To classify according to the Bayes Theorem means to determine the highset $P(C_i|\mathbf{X})$ among all classes $C_1, ..., C_m$ as follows:

$$P(C_i|\mathbf{X}) = \frac{P(C_i) * P(\mathbf{X}|C_i)}{P(\mathbf{X})}$$

where

- $P(C_i|\mathbf{X})$ is the posterior probability of class C_i given features $\mathbf{X}$
- $P(C_i)$ prioir probability of class C_i
- $P(X|C_i)$ is the likelihood which is the probability of features $\mathbf{X}$ given class C_i
- $P(X)$ prioir probbaility of the features.

Algorithm 2 Naïve Bayes

Input: train_data, test_data
Calculate $P(Y = 0)$ and $P(Y = 1)$
for Each point p in the test_data **do**
    for Each X_i in p **do**

        if X_i is categorical **then**
            Calculate $P(X = X_i|Y = 0)$ based on counting on train_data
            Calculate $P(X = X_i|Y = 1)$ based on counting on train_data
        else if X_i is continuous **then**
            Calculate the conditional mean for feature i
            Calculate the conditional Standard deviation for feature i
            Calculate *pdf* of $P(X = X_i|Y = 0)$
            Calculate *pdf* of $P(X = X_i|Y = 1)$
        end if
    end for
    Calculate the posterior probability $P(Y = 0|\bar{X})$
    Calculate the posterior probability $P(Y = 1|\bar{X})$
    Assign the label to p based on maximum of the above probabilities
end for
return predicted labels for test_data

5.1 Continous/Categorical Data

For the categorical data we use counting to calculate the likelihood. For the continous data we assume $P(X_i|C_k)$ follows a Guassian distribution and use the following formula(*probability density function (pdf)*):

$$P(X_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}} e^{-\frac{1}{2}(\frac{x-\mu_{ik}}{\sigma_{ik}})^2}$$

5.2 Handling Zero probability

In the given datasets, since we do not use counting for continous values and instead find the probability density function, we do not face with the issue of zero probability. For the categorical data we use counting to calculate the likelihood, however for the datasets given to us in this project, we had to only one categorical attribute in *project3_dataset2* which had value "present" or "absent" and with this attribute we did not face the issue of zero-probabilities. However, if all data were categorical and all probabilities are calculated by counting, then one needs to use Laplacian Zero Correction, which adds one to each feature value.

5.3 Experiments

In Table 2 we report the values obtained by running 10-fold cross validation with NB.

	Dataset1	Dataset2
Average Accuracy	0.9348	0.7035
Average Precision	0.9178	0.5729
Average Recall	0.9044	0.6208
Average F1 measure	0.9100	0.5875

Table 2: Naïve Bayes

5.4 Pros and Cons of Naïve Bayes classifier

Pros

1. Simple to implement.
2. Works well with datasets which have conditionally independent attributes.
3. Works well on small datasets

Cons

1. The assumption of conditional independence
2. Cannot find interaction between features
3. Highly biased on small datasets

6 Decision Trees

Unlike KNN, decision tree is an eager classifier. It first builds the tree based on the the training data and then traverse it for the test data. The psuedo code of our implementation is given in Algorithm 3. We use Gini Index to choose the next feature to be splitted.

6.1 Continous/Categorical Data

The decision tree that was explained in the class works well for discrete data. For the continous data we first categorize them into five groups based on their quantiles.

6.2 Best Features

We use Gini index as measurement for impurity of nodes. Gini Index for a given node t is as follows :

$$Gini = 1 - \Sigma_j p[j|t]^2$$

The Gini Index is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions.

Algorithm 3 Decision Tree Classifier

Input: train_data, test_data
while There are features to split in train_data **do**
    Calculate the gini index of all the remaining features $\bar{X}_r$
    Sort the gini indexes and choose the best feature to split X_b
    if X_b is continous **then**
        Group the values into five bins based on quantile
        Calculate the gini index of each quantile and choose the best one Val
        The left child nodes will be for the records with $Val <=$
        The right child will be for the records with $Val >$
        if all the records on the left (resp. right) branch has one class label **then**
            make a leaf node with that class label of the records
        end if
    else if X_b is categorical **then**
        for each unique value of that feature calculate the gini index and choose the best value Val
        The left child is for the records where Val is true
        The right child is for the records where Val is false
        if all the records on the left (resp. right) branch has one class label **then**
            make a leaf node with that class label of the records
        end if
    end if
end while
for Each point p in test_data **do**
    Traverse the tree until reaching a leaf node
    Assign the value of the leaf node as the class label for p
end for
return predicted labels for test_data

6.3 Experiments

Figure 1 shows the decision tree that is built for Dataset1 considering all the records as training data. Note that the features are names as $V_1, ..., V_N$ where V_i is the i^{th} column in the dataset. Figure 2 shows the decision tree that is built for Dataset2.

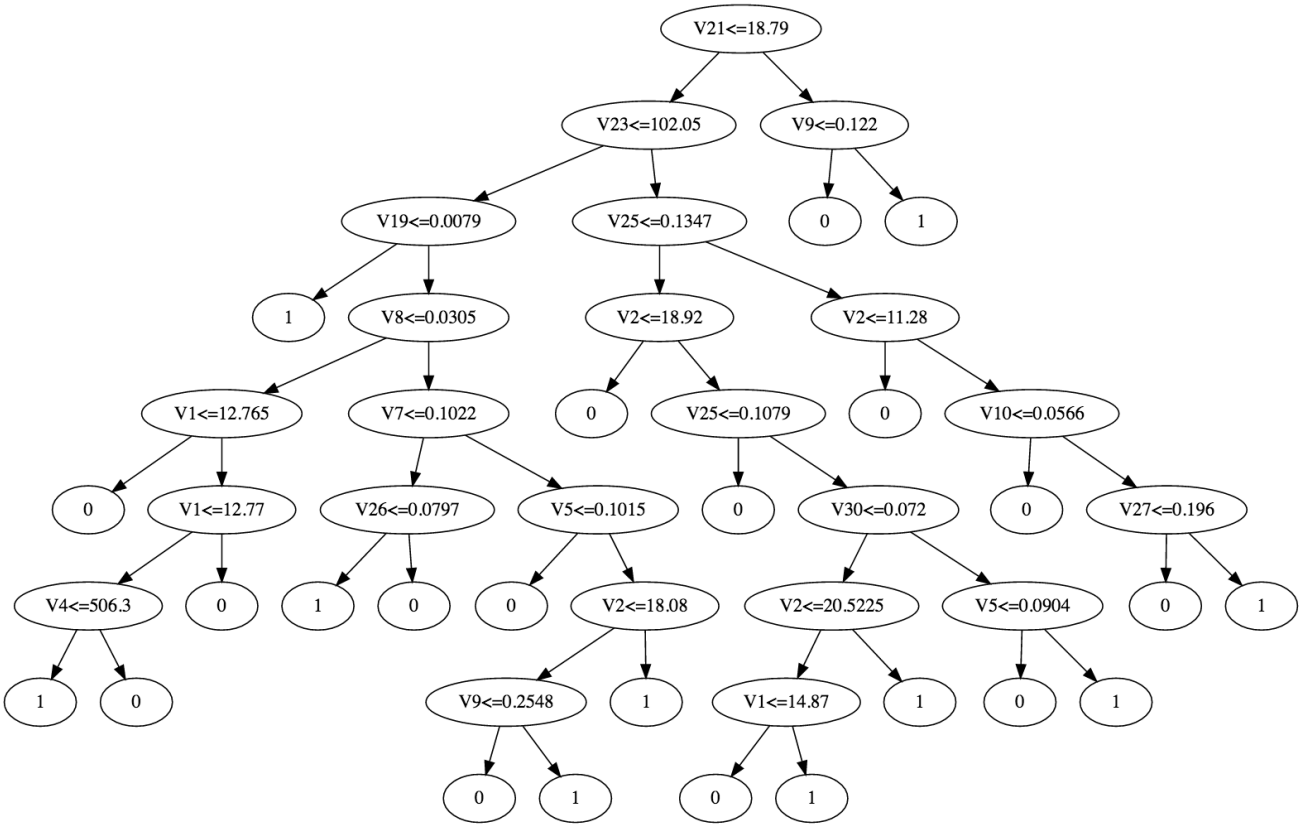

Figure 1: Decision tree on Dataset1

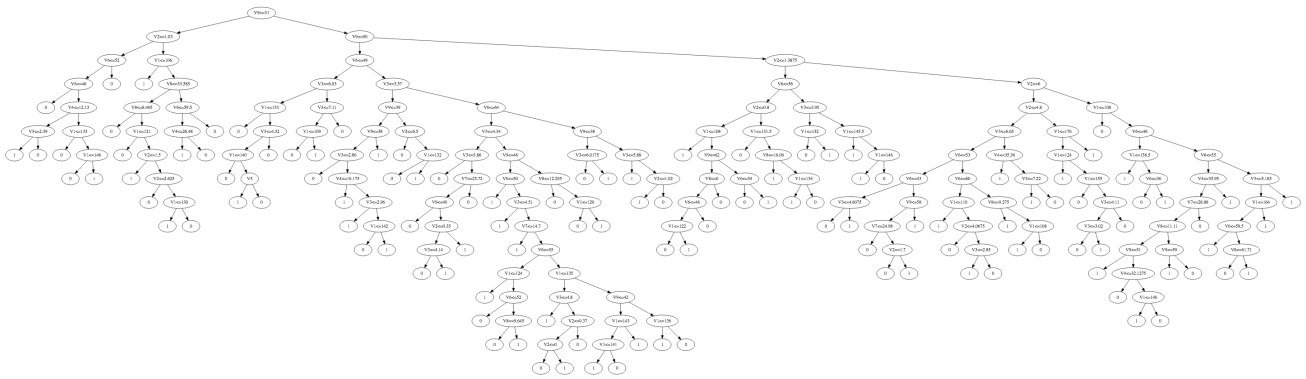

Figure 2: Decision tree on Dataset2

Table 6.3 is reporting the value of the parameters we got by running 10-fold cross validation with decision tree.

	Dataset1	Dataset2
Average Accuracy	0.9472	0.6127
Average Precision	0.9193	0.4483
Average Recall	0.9308	0.4973
Average F1 measure	0.9239	0.4641

Table 3: Decision Tree on the given datasets

6.4 Pros and Cons of Decision Trees

Pros

1. The tree obtained by decision tree algorithm is very comprehensive and easy to interpret.
2. There are no parameters to tune.
3. Can handle both categorical and continous data

Cons

1. One of the disadvantages is that the decision tree algorihtm overfits the data.
2. The accuracy of decision tree highly depends on the data presented in training phase.
3. Pruning takes time.

7 Random Forest

7.1 Algorithm

Algorithm 4 Random Forest

1: **Input:** train_data, Number_of_trees T, Number_of_columns(Percentage rounded to nearest number) m, Number_of_training_records(Percentage rounded to nearest number) N
2: We create T un-pruned trees with using the parmeters m and N, to select the number of columns(number of features used to split at each node) to be used and number of training records to be used. (The decision trees are built using Algorithm) 3
3: The labels for Test_data are predicted by getting a mojority vote on the output of the T trees in the forest.

7.2 Experiments

We carried out some experiments to see how varying the parmeters Number_of_trees T, Number_of_columns(Percentage rounded to nearest number) m, Number_of_training_records(Percentage rounded to nearest number) N affects the performance measures Average Accuracy, Average Precision, Average Recall, Average F1_measure with 10-ffold cross validation.

7.2.1 Varying the number of Trees T

Varying the number of Trees T with Number_of_columns(Percentage rounded to nearest number) $m = 50\%$ and Number_of_training_records(Percentage rounded to nearest number) $N = 40\%$. Below are the results

project3_dataset1

Vary no_of_trees(m= 40% and n=50%)	Average Accuracy	Average Precision	Average Recall	Average f1 measure
3	0.9156328	0.8678378	0.90962	0.8852921
5	0.9191416	0.9014209	0.8779828	0.8860531
7	0.9331767	0.9076657	0.9069852	0.906058
9	0.9243734	0.9059994	0.9009971	0.9017582
11	0.9296679	0.9105521	0.9072735	0.9070542
15	0.9384398	0.9172615	0.9294861	0.9218622
21	0.9578634	0.954433	0.9311056	0.9418948

Figure 3: Varying the number of trees T

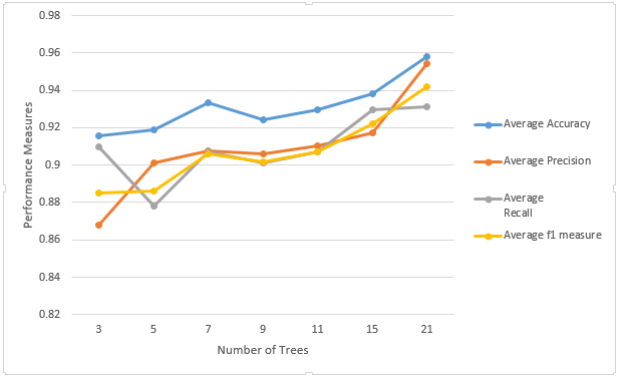

Figure 4: Varying the number of trees T

project3_dataset2

Vary no_of_treesno_of_trees(m= 40% and n=50%)	Average Accuracy	Average Precision	Average Recall	Average f1 measure
3	0.6043478	0.4379742	0.4458516	0.4212488
5	0.6576993	0.4960804	0.4112994	0.4366247
7	0.6384964	0.5277165	0.3554236	0.3937973
9	0.6466486	0.524881	0.4098931	0.433311
11	0.6604167	0.520646	0.3584725	0.414431
15	0.6663949	0.5529759	0.3828038	0.4240614
21	0.6576993	0.5355128	0.3781475	0.4126988

Figure 5: Varying the number of trees T

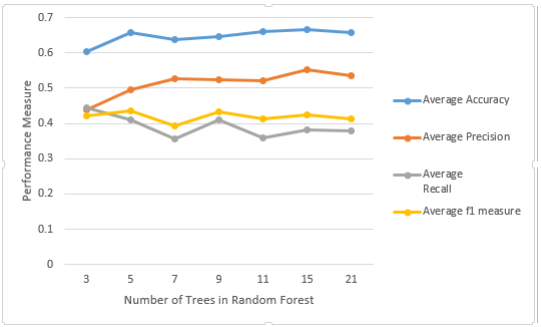

Figure 6: Varying the number of trees T

From the above results we inferred that as the number of trees that form the random forest increase the performance of random forest increases.

7.2.2 Varying the *number of columns* (Percentage rounded to nearest number) m

Varying the with number of columns with Trees $T = 7$ for *project3_dataset1* and $T = 9$ for *project3_dataset2* and *Number of training records* (Percentage rounded to nearest number) $N = 50\%$. Below are the results

project3_dataset1

Vary percentage of columns(no_of_trees = 7 and n=50%)	Average Accuracy	Average Precision	Average Recall	Average f1 measure
10	0.9367168	0.9172034	0.9134802	0.9133953
15	0.922713	0.9408033	0.852697	0.8912324
20	0.9207707	0.8931264	0.8999689	0.8945834
25	0.933114	0.9095076	0.9155024	0.9111277
30	0.920896	0.8849358	0.9003575	0.8898963
35	0.9296679	0.929186	0.8813491	0.903051
40	0.9402256	0.934446	0.9131057	0.9203315
45	0.941886	0.9291375	0.910611	0.9180065
50	0.9278195	0.9299603	0.891701	0.906499
55	0.9402256	0.9137572	0.9304301	0.9203046

Figure 7: Varying the number of *number of columns* (Percentage rounded to nearest number) m

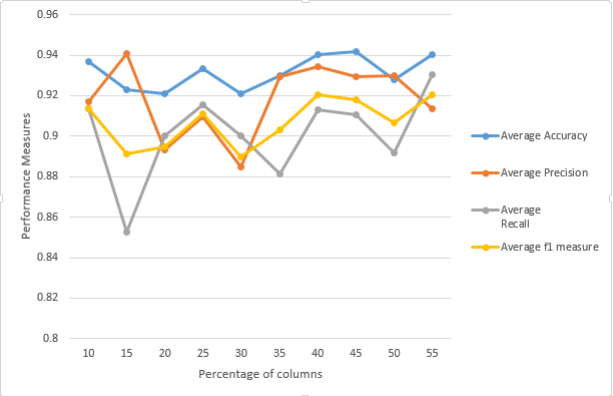

Figure 8: Varying the number of *number of columns* (Percentage rounded to nearest number) m

project3_dataset2

Vary percentage of columns(no_of_trees = 7 and n=50%)	Average Accuracy	Average Precision	Average Recall	Average f1 measure
15	0.6304348	0.6428571	0.4285714	0.5142857
20	0.6521739	0.5	0.3809524	0.4318182
25	0.6440217	0.4893004	0.3544362	0.3993472
30	0.6776268	0.5427305	0.3932785	0.4250761
35	0.6537138	0.5073465	0.412131	0.4407195
40	0.6604167	0.5239982	0.4895697	0.4893639
45	0.6424819	0.4841576	0.3823704	0.4096907
50	0.6403986	0.5019048	0.2896927	0.3450619
55	0.6468297	0.4767241	0.3114595	0.3491825

Figure 9: Varying the number of *number of columns* (Percentage rounded to nearest number) m

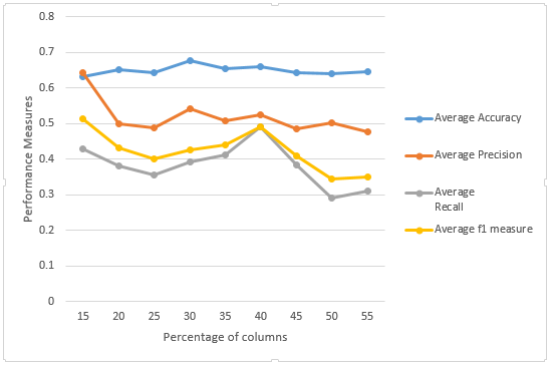

Figure 10: Varying the number of *number of columns* (Percentage rounded to nearest number) m

From the above results, we do not see a pattern in the change of performance measures i.e. there is neither steady increase or decrease of the values of performance measures. But we noticed that the best values for both datasets were noticed to be at $m = 40\%$.

7.2.3 Varying the *number of training records* (Percentage rounded to nearest number) N

Varying the with number of number of training records with Trees $T = 7$ for *project3_dataset1* and $T = 9$ for *project3_dataset2* and *Number of columns* (Percentage rounded to nearest number) $m = 40\%$. Below are the results

project3_dataset1

Vary percentage of training records used to build trees(no_of_trees = 7 and m=40)	Average Accuracy	Average Precision	Average Recall	Average f1 measure
10	0.8751566	0.9147247	0.7364215	0.7806449
15	0.8803258	0.8005576	0.91591	0.8485636
20	0.8927632	0.8473009	0.8717738	0.8500796
25	0.9102757	0.9049669	0.8512624	0.8735539
30	0.9032581	0.8563741	0.9009459	0.8737953
35	0.9137845	0.8770937	0.9110256	0.8901621
40	0.9366855	0.9272956	0.8975239	0.9092937
45	0.9261591	0.9223443	0.8818888	0.8994152
50	0.9173872	0.8946407	0.8903144	0.891285
55	0.9261278	0.9078409	0.9085301	0.9049661

Figure 11: Varying the *number of training records* N

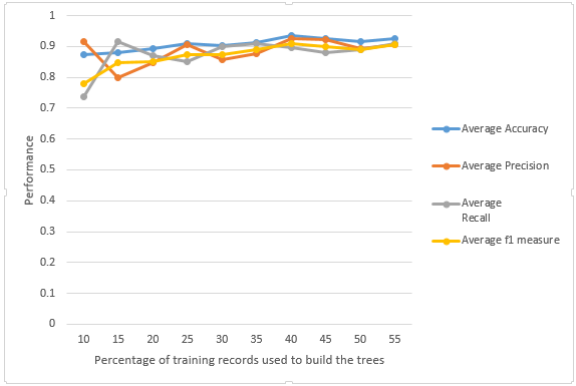

Figure 12: Varying the *number of of training records* (Percentage rounded to nearest number) N

project3_dataset2

Vary percentage of training records used to build trees(no_of_trees = 9 and m=40)	Average Accuracy	Average Precision	Average Recall	Average f1 measure
15	0.5797101	0.3408118	0.3747131	0.3058455
20	0.6407609	0.554551	0.3266466	0.3467361
25	0.6341486	0.4250959	0.2819644	0.309087
30	0.6423913	0.5311596	0.2621877	0.3135627
35	0.6557065	0.4820635	0.3714631	0.4079649
40	0.6256341	0.4062782	0.2795947	0.3184131
45	0.6793478	0.562002	0.5052906	0.5137861
50	0.6686594	0.5455237	0.3993116	0.4329936
55	0.6620471	0.5426768	0.3677932	0.4173115

Figure 13: Varying the *of training records* (Percentage rounded to nearest number) N

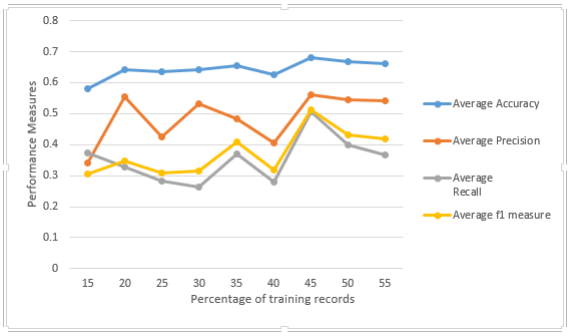

Figure 14: Varying the *of training records* (Percentage rounded to nearest number) N

As the datasets provided for this project are small, it is difficult to generalize the impact of changing the *Number of training records* used to build the trees that comprise the random forests. But we see that the performance measures for both the datasets were good when 40 to 50 % of the training records were chosen.

7.3 Pros and Cons of Random Forests

Pros

1. It runs efficiently on large datasets.
2. It solves the problem of overfitting in decision trees
3. As multiple trees are built selecting random attributes and random subset of records of training data and the final prediction made is a mojority vote of trees, so random forests can handle situations where there is missing or partial training data.

Cons

1. The running time will increase if there are a large number of trees
2. The feature selection process is not explicit.
3. Has weaker performance on small size training data.

8 Adaboost with Decision Trees

The Boosting algorithm we have implemented is Adaboost. Adaboost algorithm learns weak classifiers and takes a weighted average of the classifiers to predict the final class of the test records. The main idea in any boosting algorithm is to boost the performance of the algorithm by improving the accuracy of the weak learners that form the classifiers.

8.1 Algorithm

Our implementation of Adaboost Algorithm is based on the pseudo code in Algorithm 5

Algorithm 5 Adaboost on Decision Tree

Input: train_data, test_data , Bootstrap_sample_size, Number_of_classifiers
Initialize Weights: Declare a list to hold the weights and initialize every element in this list to $1/n$, where n is equal to the number of records in train_data
for ℓ in range 1 to $number\_of\_classifiers$ **do**
    Randomly select $Bootstrap\_sample\_size$ of records from train_data. Let us call this $Boot\_strap\_sample$ and denote the number of records in it as k
    Build ℓth classifier decision tree using $Boot\_strap\_sample$.
    Test this ℓth classifier against all the records in the train_data
    We then compute classifier error as $error_\ell = \frac{\sum_{n=1}^{k} weights_k * I(C_\ell(x_k) \neq y_k)}{\sum_{n=1}^{k} weights_k}$

    if error >0.50 **then**
        Re-initialize the weights to $1/n$ and reset ℓ=0 and start from the beginning.
    else
        We do the following computations
        Compute the importance of Classifier C_ℓ as $\alpha_\ell = \frac{1}{2} * \ln(\frac{1-error_\ell}{error_\ell})$
        $weights_k = \frac{weights_k * (\alpha_\ell * C_\ell(x_k) * y_k)}{\sum_1^k weights_k}$
    end if
end for
return predicted labels for test_data using the formula

$$C^*(x) = argmax_y \sum_1^\ell \alpha_\ell * I(C_\ell)$$

8.2 Experiments

We did some experiments by varying the parameter $Bootstrap\_sample\_size$ and $Number\_classifier$. Below are the results of varying $Number\_of\_classifiers$ (ℓ) for a fixed value of $Bootstrap\_sample\_size$ (k)

Below are the results for $project3\_dataset1$

	Average Accuracy	Average Precision	Average Recall	Average F1_Measure
$\ell = 3$	0.9437	0.9435	0.9059	0.9225
$\ell = 5$	0.9525	0.9455	0.9259	0.9346
$\ell = 7$	0.9560	0.9650	0.9140	0.9377
$\ell = 9$	0.9578	0.9653	0.9213	0.9410
$\ell = 11$	0.9649	0.9686	0.9358	0.9505
$\ell = 15$	0.9665	0.9788	0.9311	0.9536
$\ell = 19$	0.9666	0.9689	0.9417	0.9543
$\ell = 23$	0.9612	0.9718	0.9228	0.9459
$\ell = 25$	0.9648	0.9787	0.9271	0.9507

Table 4: Varying the number of classifiers ℓ for Dataset1 fixing the bootstrap_sample_size $k = 200$ (which is roughly 50% of training data size)

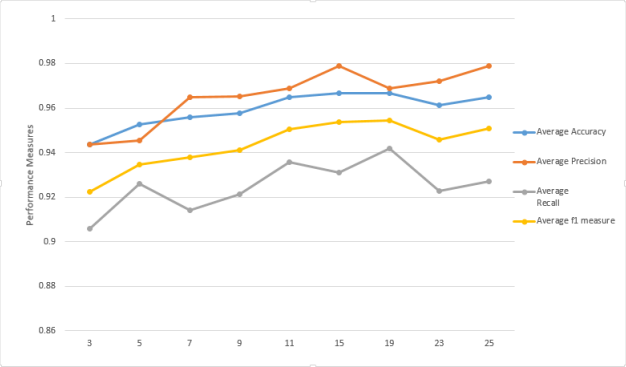

Figure 15: Varying the number of classifiers ℓ for Dataset1 fixing the bootstrap_sample_size $k = 200$

Below are the results for $project3\_dataset2$

	Average Accuracy	Average Precision	Average Recall	Average F1_Measure
$\ell = 3$	0.6403	0.4813	0.4298	0.4441
$\ell = 5$	0.6212	0.4579	0.4221	0.4275
$\ell = 7$	0.6493	0.4912	0.4394	0.4512
$\ell = 9$	0.6474	0.4858	0.4285	0.4381
$\ell = 11$	0.6321	0.4750	0.4179	0.4326
$\ell = 15$	0.6233	0.4405	0.4077	0.4127
$\ell = 19$	0.6623	0.5141	0.4527	0.4713
$\ell = 23$	0.6425	0.4876	0.4067	0.4334
$\ell = 25$	0.6623	0.5199	0.4079	0.4419

Table 5: Varying the number of classifiers ℓ for Dataset2 fixing the bootstrap_sample_size $k = 200$ (which is roughly 50% of training data size)

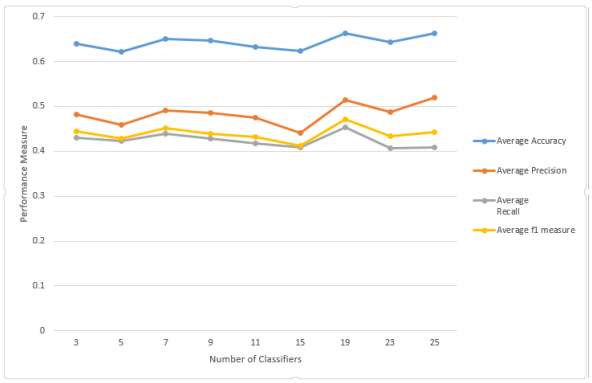

Figure 16: Varying the number of classifiers ℓ for Dataset2 fixing the bootstrap_sample_size $k = 200$

Our inference from the above experiments was that in case of both the datasets, with 10-fold cross validation the values of Average Accuracy, Average Precision, Average Recall, Average F1_Measure increased as the value of $Number\_of\_classifiers$ was increased. We also noticed that the highest value of the above mentioned parmeters was recorded at $Number\_of\_classifiers$ = 19 and so we conducted another set of experiments by keeping $Number\_of\_classifiers = 19$ and varying the $Bootstrap\_sample\_size$ 50-250, with increments of 25(which is roughly 5 percent of training_data)

Below are the results for *project*3_*dataset*1

	Average Accuracy	Average Precision	Average Recall	Average F1_Measure
$k = 50$	0.9419	0.9414	0.9068	0.9225
$k = 75$	0.9454	0.9456	0.9074	0.9246
$k = 100$	0.9471	0.9381	0.9219	0.9286
$k = 125$	0.9472	0.9540	0.9010	0.9260
$k = 150$	0.9559	0.9597	0.9220	0.9396
$k = 175$	0.9560	0.9760	0.9358	0.9374
$k = 200$	0.9630	0.9735	0.9263	0.9483
$k = 225$	0.9665	0.9724	0.9371	0.9535
$k = 250$	0.9578	0.9603	0.9293	0.9435

Table 6: Varying the bootstrap_sample_size k for Dataset1 fixing the number_of_classifier $\ell = 19$

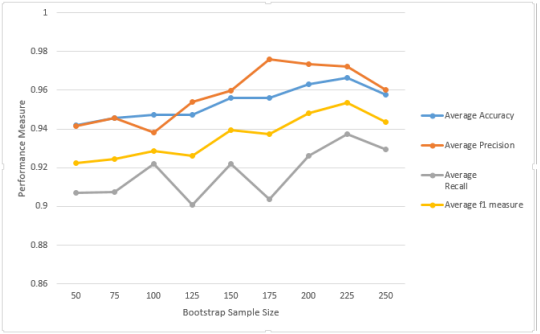

Figure 17: Varying the bootstrap_sample_size k for Dataset1 fixing the number_of_classifier $\ell = 19$

Below are the results for *project*3_*dataset*2

	Average Accuracy	Average Precision	Average Recall	Average F1_Measure
$k = 50$	0.6708	0.5456	0.3616	0.4194
$k = 75$	0.6385	0.4659	0.3864	0.4063
$k = 100$	0.6596	0.5139	0.4151	0.4466
$k = 125$	0.6492	0.4858	0.3844	0.4175
$k = 150$	0.6707	0.5308	0.4247	0.4595
$k = 175$	0.6212	0.4391	0.3505	0.3742
$k = 200$	0.6449	0.4757	0.3892	0.4167
$k = 225$	0.6254	0.4539	0.3800	0.4030
$k = 250$	0.6448	0.4778	0.4040	0.4273

Table 7: Varying the bootstrap_sample_size k for Dataset2 fixing the number_of_classifier $\ell = 19$

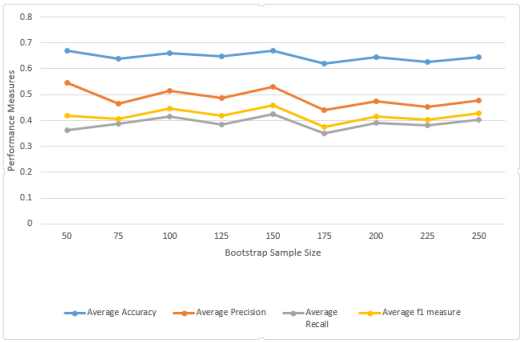

Figure 18: Varying the bootstrap_sample_size k for Dataset1 fixing the number_of_classifier $\ell = 19$

From the above experiments, we were able to infer that for the given datasets with Number_of_classifiers=19, the best value of Average Accuracy, Average Precision, Average Recall, Average F1_Measure were observed when the Bootstrap_sample_size was in the range 35% to 40% of the training dataset. More specifically we observed that best values of above mentioned parameters when Boostrap_sample_size was 35% and 40% of size of training dataset for project_dataset2 and project_dataset1 respectively.

8.3 Pros and Cons of Adaboost with Decision Tree

8.3.1 Pros

1. The biggest advantage of Adaboost or any Boosting Algorithm in general is that they are easy to implement

2. Boosting Decision Trees provides signifincant increase in performance when compared to the performance of a single decision tree. This is because of the fact that Adaboost creates multiple weak classifiers and uses a weighted average of their outputs to predict the class of a new record.

3. There are not many parameters to tune.

8.3.2 Cons

1. The only disadvantage of Adaboost with Decision Trees is that if there is not sufficient training data the performance of the algorithm might be very poor. This is because with less data the weak classifiers (Decision Trees) will overfit the data and hence there will be degradation in the performance.

9 Comparison of the Algorithms

We plotted the performance measures of various algorithms to compare how they performed on *project3_dataset1* and *project3_dataset2*. As shown in Figure 19 Adaboost classifier is the best in terms of all evaluation parameters for *project3_dataset1*. It has the most accuaracy and F1 measure. Observe that Knn has a high precison but a low recall which is due to a lot of False Negatives. Note that both Random Forest and Adaboost improves the result of decision trees.

As depicted in Figure 20 Naïve Bayes algorithm outperforms other algorithms. This is because *project3_dataset2* is small in terms of number of features and number of records. Observe that Knn has high accuracy and precision but it has low recall, this is due to a lot of False Negetives. Note that Naïve Bayes has the highest recall amonthe all classifiers.

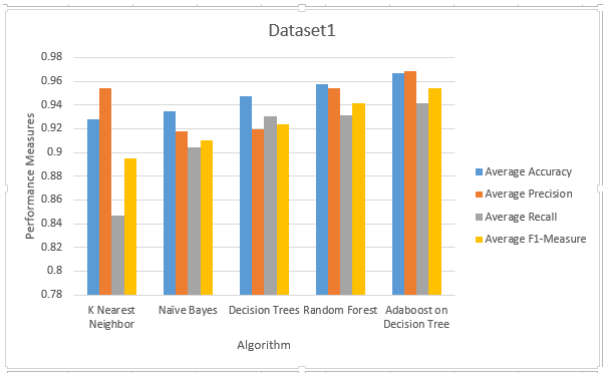

Figure 19: Comparison of Perforamnce of various algorithms on *project3_dataset1*

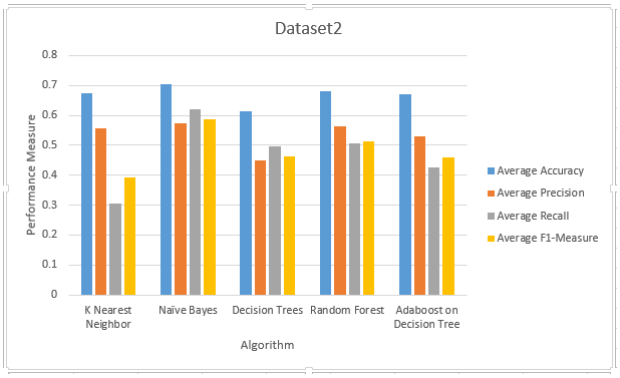

Figure 20: Comparison of Perforamnce of various algorithms on *project3_dataset2*

References

[1] https://www3.nd.edu/ steve/computing_with$_data$/17_$Refining_kNN$/$refining_knn.html$