COMPUTER & INFORMATION SCIENCE & ENGINEERING

UF Herbert Wertheim College of Engineering UNIVERSITY *of* FLORIDA

CLASSIFICATION ANALYSIS ON VEHICLE AND PEDESTRIAN STOPS

Introduction to Data Mining

Aditya Kant, UFID: 25498901
Geetanjli Chugh, UFID: 21885647
Mugdha Mathkar, UFID: 54147979
Vagisha Tyagi, UFID: 04289808
Yash Sinha, UFID: 15618171

TABLE OF CONTENTS

1. INTRODUCTION

The projects deals with applying data mining techniques to the Stanford Open Policing Datasets. The dataset contains data on vehicle and pedestrian stops from law enforcement departments across the country. The outcome of the project is predicting the stop outcome based on various attributes like Police Department, Driver Gender, Driver Age, Driver Race, Search Conducted, Contraband Found, Ethnicity, Latitude, Longitude and a host of other feature attributes. The Stanford Open Policing Project has made data from 30 states available on Kaggle. Due to data size pre-requisite of the project, we have used data from 3 states - North Carolina, South Carolina and Washington which add up to be 5 gb.

We have used 5 multi-class classification algorithms to predict the outcomes. These are: K-Nearest Neighbor, Naive Bayes, ID3 Decision Trees, CART Decision Trees and C4.5 Decision Trees. The project report starts with a brief literature survey done for the project. This is followed by dataset description and data source information. Then we discuss the data cleaning and data preprocessing step. We then explain the various algorithms we have used for classification. This is followed by conclusion and a brief discussion on limitations of the project and future improvements.

2. LITERATURE SURVEY

A study of the existing research shows that data mining methods help in improving the decision making process in the field of crime detection and analysis. Studies have been carried out to understand the relation between the outcomes given by police based on various attributes[4].This will help in analyzing the patterns in the outcomes of the traffic related crimes and aid in more efficient policing.

Our project is based on multi-class classification problem. The datasets from the three states each contain multiple classes to be predicted. We studied research papers on various data mining algorithms to understand which methods would be best suited for our crime dataset.The selected features are independent and thus we can use Naive Bayes. Most of the features are categorical variables so we have used decision trees. We also explored KNN which gave good predictions on smaller sample (~64000 records).

According to Yin Lu[5],decision tree algorithms can efficiently deal with large data without imposing any complicated parametric structure.Thus,we implemented ID3,C4.5 and CART to predict the stop outcomes in the dataset.

3. DATA SOURCE & DATASET DESCRIPTION

On a typical day in the United States, police officers make more than 50,000 traffic stops. The Stanford Open Policing project team is gathering, analyzing, and releasing records from millions of traffic stops by law enforcement agencies across the country. Their goal is to help researchers, journalists, and policymakers investigate and improve interactions between police and the public.

The Stanford Open Policing Project is collecting and standardizing data on vehicle and pedestrian stops from law enforcement departments across the country. They have gathered 130 million records from 31 state police agencies and have begun collecting data on stops from law enforcement agencies in major cities, as well.

We have used the data available on Kaggle. The Stanford Open Policing Project is available here: https://www.kaggle.com/stanford-open-policing. The attribute to be predicted on is **stop_outcome**.

The data was obtained from 3 different states. The datasets can be accessed here:
Washington: https://www.kaggle.com/stanford-open-policing/stanford-open-policing-project-washington-state/data
Number of records: 8624032
Attribute columns with their unique value counts

x id 8624032	x search_type_raw 7
state 1	? search_type 6
x stop_date 2647	contraband_found 2
x stop_time 24	**stop_outcome** 3
x location_raw 38	? is_arrested 0
x county_name 38	x violations 168058
x county_fips 38	x officer_id 1484
x fine_grained_location 8241	x officer_gender 2
police_department 0	x officer_race 5
driver_gender 2	x highway_type 3
x driver_age_raw 110	x road_number 210
driver_age 85	x milepost 573
x driver_race_raw 8	x lat 6208
driver_race 5	x lon 6209
x violation_raw 90011	x contact_type 13
? violation 2112	x enforcements 1750
search_conducted 2	x drugs_related_stop 2

Class Labels	Number of records
Arrest or Citation	3118621
Verbal Warning	3010614
Written Warning	139188

North Carolina: https://www.kaggle.com/stanford-open-policing/stanford-open-policing-project-north-carolina/data
Number of records: 9558084
Attribute columns with their unique value counts

x id 9558084 x state 1 x stop_date 5844 x stop_time 1438 x location_raw 157 x county_name 100 x county_fips 100 x fine_grained_location1165 police_department 2 driver_gender 2 x driver_age_raw 119 driver_age 85 x driver_race_raw 11 driver_race 5	x violation_raw 10 ? violation 8 search_conducted 2 x search_type_raw 5 ? search_type 5 contraband_found 2 **stop_outcome** 5 is_arrested 2 x search_basis 37 x officer_id 10992 x drugs_related_stop 1 x ethnicity 2 x district 57

Class Label	Number of records
Citation	7556627
Written Warning	1450986
No Action	280406
Verbal Warning	139988
Arrest	130077

South Carolina: https://www.kaggle.com/stanford-open-policing/stanford-open-policing-project-south-carolina/data
Number of records: 8440934
Attribute columns with their unique value counts

x id 8440934 state 1 x stop_date 4228 x stop_time 0 x location_raw 47 x county_name 44 x county_fips 44 x fine_grained_location 0 police_department 4 driver_gender 2 x driver_age_raw 137 driver_age 85 x driver_race_raw 6 driver_race 4 x violation_raw 48	? violation 13 search_conducted 2 x search_type_raw 0 ? search_type 0 contraband_found 2 **stop_outcome** 4 ? is_arrested 2 x lat 163152 x lon 216755 x highway_type 10 x road_number 45057 x stop_purpose 9 x officer_id 1378 x officer_race 7 x officer_age 67

Class Labels	**Number of Records**
Citation	5297948
Warning	2943935
Arrest	192183
Felony Arrest	6863

4. DATA CLEANING AND PREPROCESSING

The major steps we took for data cleaning and preprocessing were:

4.1 Feature Selection

All of the three datasets contained around 30 features. Most of the features were common. However, there were some features which were present in only one of the datasets but helped in getting better predictions. There were also some attributes in some of the datasets which contained all null values.

Feature selection was done based on following criterias:

a) **Value Counts :**We observed that some of the attributes had a large number of unique values. While these were making the algorithms slower (especially the decision trees), these attributes didn't add much value to the accuracy.

b) **Single valued attributes :**There were also some attributes which had a single unique value and rest of the values were NA. These were also removed. Eg. drugs_related_stop in the NC dataset.

c) **Raw Values :**There were some attributes which contained the non-processed raw values for its complementary attribute. Eg. driver_age_raw, driver_race_raw etc.

d) **Non-useful attributes :**We also identified some attributes which didn't provide any useful information for prediction. Eg. id, officer_id etc.

1. **Filling NA values -- method='ffill'**

We identified the attributes which had maximum values as null. These attributes were removed. Additionally we filled up the NA values with the method ffill which fills the cells with the last non-NA value.

2. **Converting to categorical columns**

We converted the non-float and non-boolean columns to categorical variables. These were used in decision tree algorithms.

3. **Converting to 1-hot encoding**

KNN and Naive Bayes work best with numerical attributes. So we converted the categorical variables to 1-hot encoded series object and passed to KNN and Naive Bayes classifiers.

4. **Boolean to numerical values**
We observed that KNN and Naive Bayes worked faster when we specifically mapped the boolean attributes to numerical values of 1 and 0.

5. **Converting class label to class codes**
We also observed that the performance were better when used categorical codes for the class labels.

6. **Train and test data splitting**
We used sklearn.model_selection.train_test_split function to split the training and test dataset. The function shuffles the dataset by default and returns the training and testing set. We have divided the dataset into 80% training and 20% testing sets.

4.2. Post Processing

In addition to the feature selection step we also explored further on which attributes were actually adding value to the prediction. We did a detailed analysis for both the categorical variables and 1-hot encoded series variables. We found that there were some attributes like *'highway_type', 'lat', 'lon'* which were actually decreasing the accuracy of the prediction. On the other hand there were some attributes like *search_type* which made the algorithms slower and weren't adding to the predictions.

4.3. Selected Features

These were the selected features for the three datasets.

North Carolina:
['police_department', 'driver_gender', 'driver_age', 'driver_race', 'search_conducted', 'contraband_found', 'is_arrested', 'ethnicity']

South Carolina:
['police_department', 'driver_gender', 'driver_age', 'driver_race', 'search_conducted', 'contraband_found', 'is_arrested', 'lat', 'lon', 'highway_type', 'stop_purpose', 'officer_age']

Washington:
['police_department', 'driver_gender', 'driver_age', 'driver_race', 'search_conducted', 'contraband_found', 'is_arrested', 'lat', 'lon', 'highway_type', 'stop_purpose', 'officer_age']

5. ALGORITHM IMPLEMENTATION

5.1 K-Nearest Neighbors

K-Nearest Neighbor algorithm is a lazy learning, parametric algorithm that is used for classification. **K-nearest Neighbors** are the data points that have k smallest distances to sample input.

Euclidean Distance:
Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j.

$$d = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \, .$$

Optimal value of k:
It is important that we select an optimal value of k for the algorithm to work efficiently. We ran the KNN method for various values of k ranging from k=5 to k=20 and found that the accuracy remained constant after k=10.Thus, we selected 10 as the value of k.

5.1.1. Algorithm

1. Calculate “d(x, x_i)” i =1, 2, ….., n; where d denotes the Euclidean distance between the points.
2. Arrange the calculated n Euclidean distances in non-decreasing order.
3. Let k be a positive integer, take the first k distances from this sorted list.
4. Find those k-points corresponding to these k-distances.
5. Let k_i denotes the number of points belonging to the i^{th} class among k points i.e. k ≥ 0
6. If $k_i > k_j \; \forall \; i \neq j$ then put x in class i.

5.1.2. Pros and Cons

KNN does not require any retraining if the new training pattern is added to the existing training set. It works well on large datasets. Thus, it gave faster and better results for our huge crime dataset. It is robust to noisy training data. Also, no learning of the model is required, and all of the work happens at the time a prediction is requested.

However, KNN also has some disadvantages. It takes up a lot of space for larger training data. The computation cost of KNN is also very high as we must find the distance between all training instances. Apart from that it is very important that we select an appropriate value for k. Otherwise, the classifier will perform poorly.

Time Complexity: The time complexity of KNN is O(nk+nd) where n is the number of records, k is the number of nearest neighbors, d is the number of attributes in the dataset.

5.1.3. Analysis of Results

5.1.3.1. Analysis of Precision, Recall and Matthew's Coefficient:

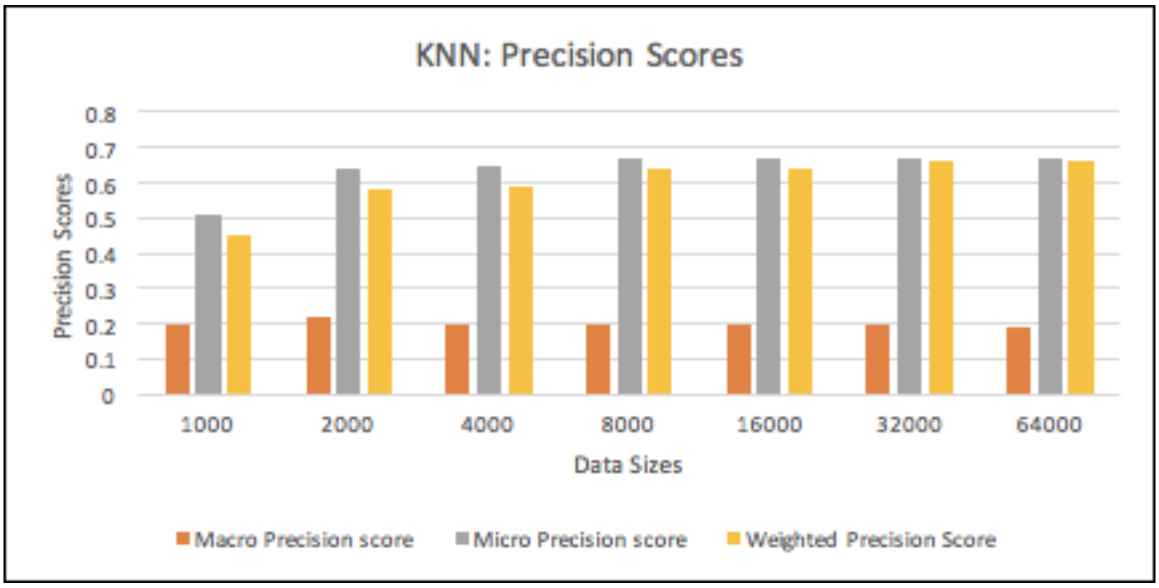


Figure 1 KNN: Precision Scores

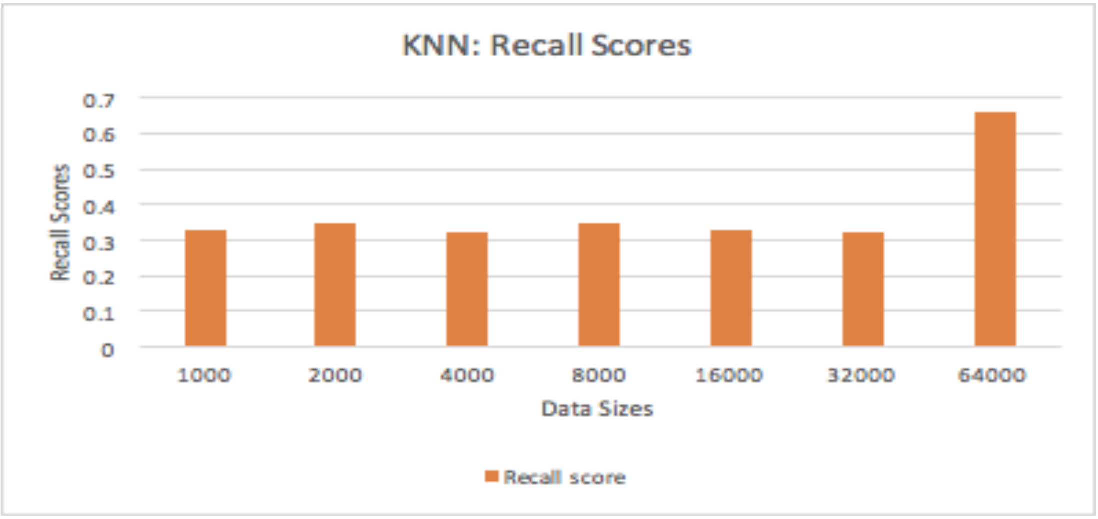


Figure 2 KNN: Recall Scores

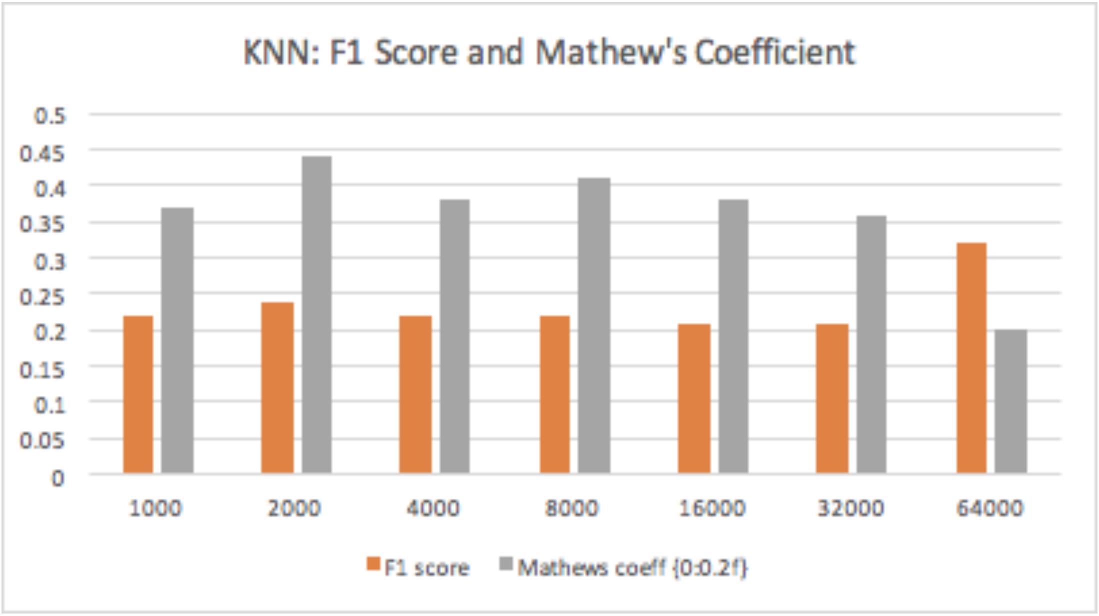


Figure 3 KNN: F1 Scores and Matthew's Coefficient

From Figure 1, we observe that the Micro Precision scores for KNN lie in the range of 50% to 70%. Thus, KNN classifier has comparatively less false positives. The micro precision value gives a proper estimate of the precision measure since our dataset has classes of variable sizes. The classifier has lower recall scores indicating that there are more false negatives. However, this is compensated by higher precision values.

F-measure for KNN classifier is in the range of 40% to 45%. We also calculated the Matthew's coefficient for multi-class data and it lied between 0.2 to 0.3 for datasets of varying sizes.

5.1.3.2. Analysis of Accuracy Measure

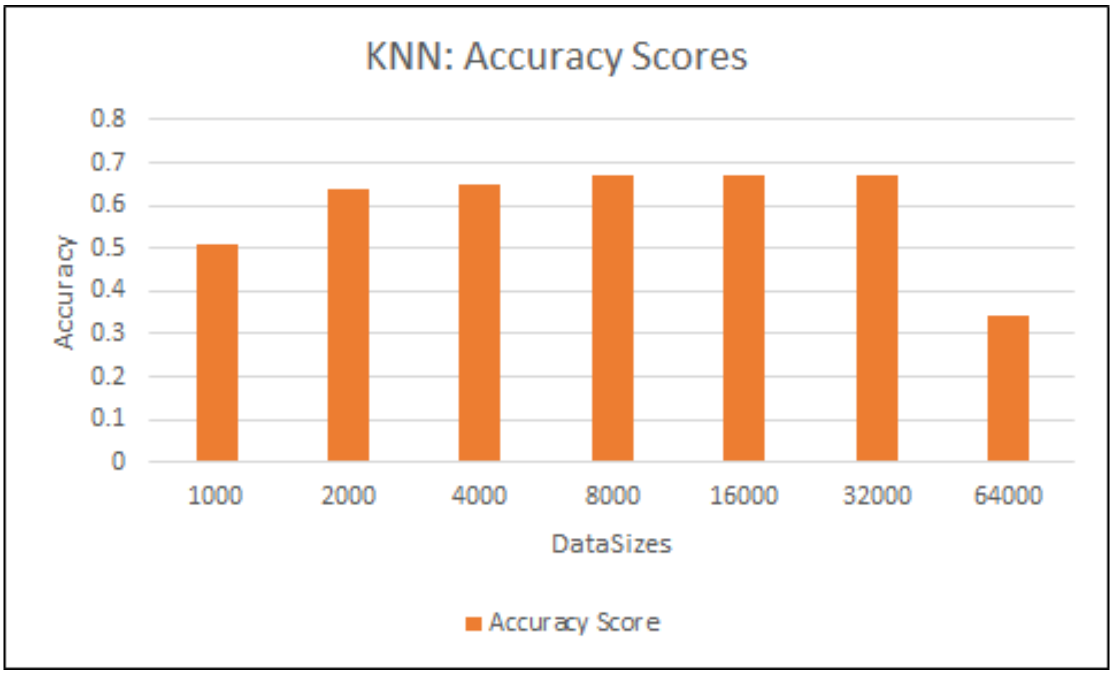


Figure 4 KNN: Accuracy Scores

The graph for accuracy measure shows us that the average accuracy is around 50% to 70% which indicates that the classifier is performing well. However, the accuracy value drops as the number of records increases beyond 50000.This is because the classifier has to calculate more distances between the points.

5.1.3.3. Analysis of Running Time

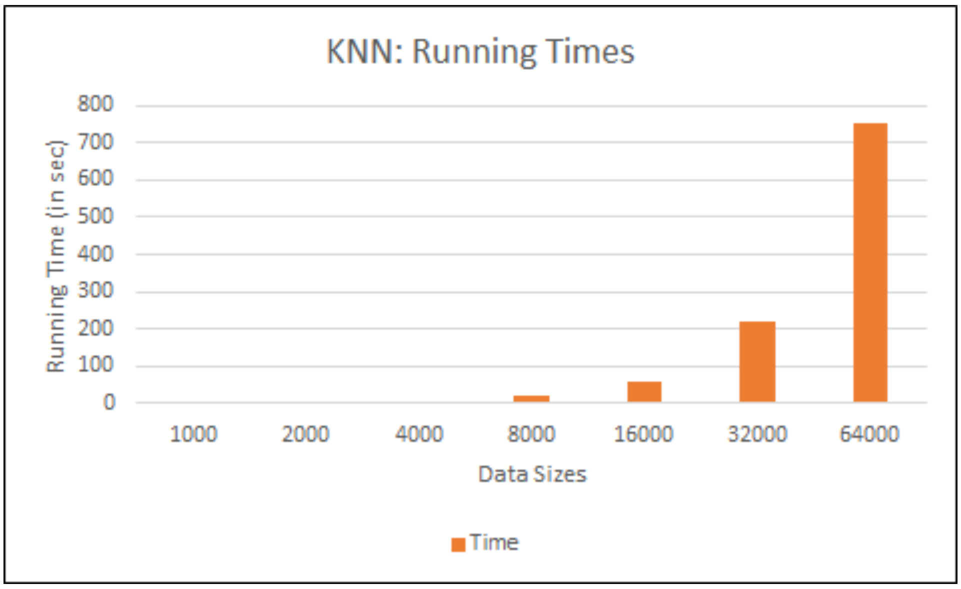


Figure 5 KNN: Running Times

We ran the classifier for various data sizes and observed that the running time for smaller datasets is very less. However, as the size increases beyond 32000 records, the running time deteriorates. This can be attributed to the fact that the time complexity of KNN is directly proportional to the number of records.

5.2. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Given a class variable y and a dependent feature vector x_1 through x_n, Bayes' theorem states the following relationship:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

We can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.
The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

In Spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

5.2.1. Algorithm

1. Calculate the mean and variance of all the features for every class
2. Calculate the prior for each class
    a. n_class_instances / float(n_total_instances)
3. For every class:
    a. Calculate the likelihood of each feature
    b. Total posterior of the class is the multiplication of the prior and the likelihood of each feature
4. Assign the class label to the class with the highest posterior value

5.2.2. Pros and Cons

Naive Bayes is computationally fast and simple to implement. It also works well with high dimensions. This made Naive Bayes one of the algorithms we implemented on our dataset.

However, there are a few limitations of the algorithm. If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”. Naive Bayes also relies on independence assumption and will perform badly if this assumption is not met. Our selected features, though, meet this requirement.

5.2.3. Analysis of results

5.2.2.1 Analysis of Precision, Recall and Matthew’s Coefficient

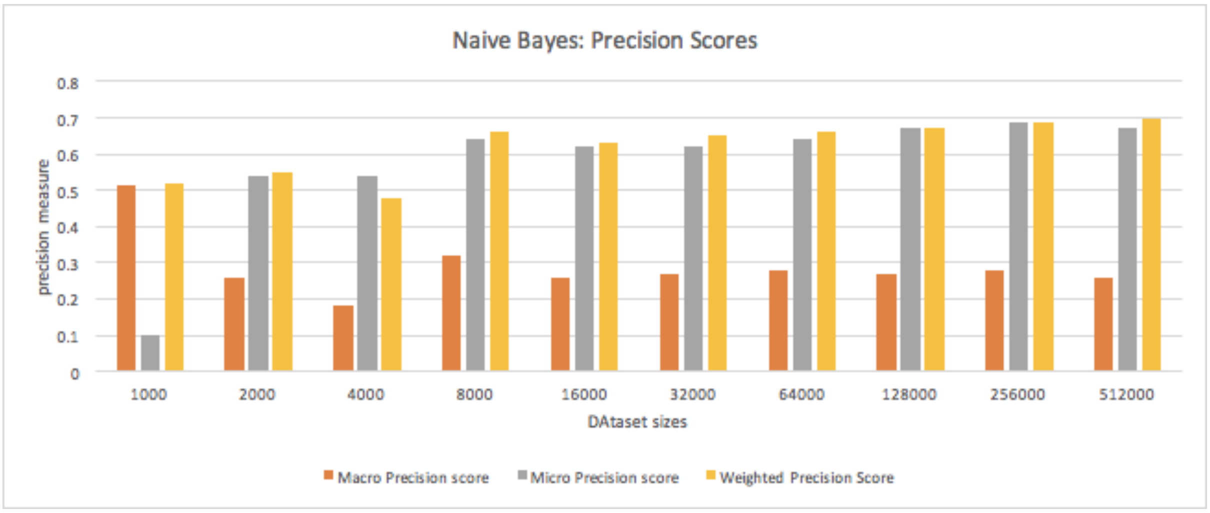


Figure 6 Naive Bayes: Precision Scores

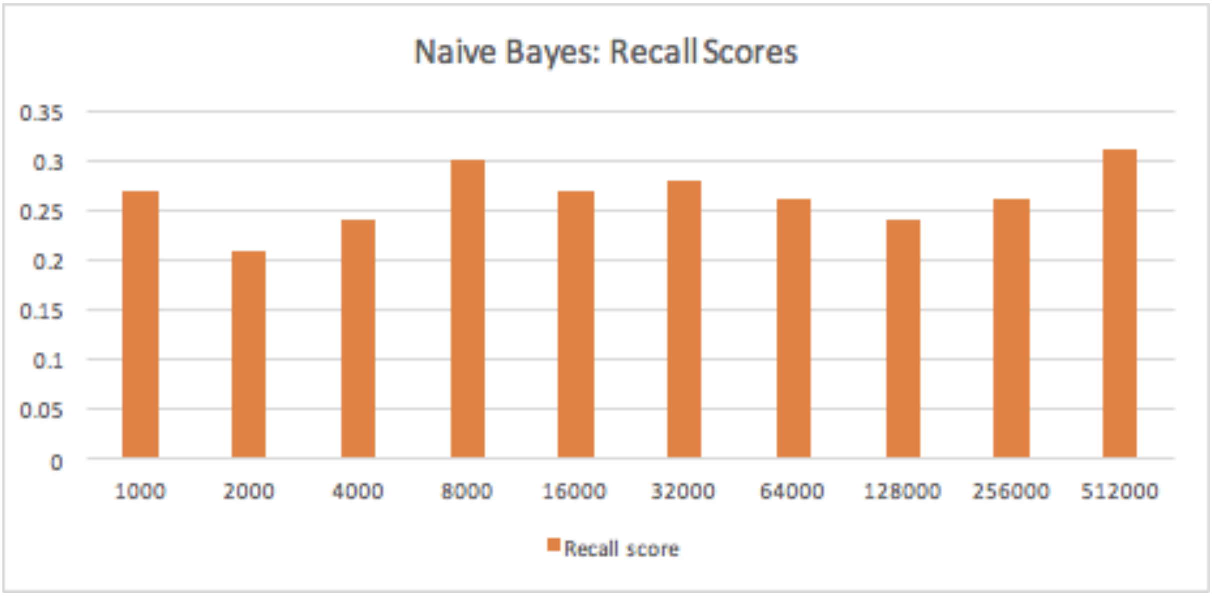


Figure 7 Naive Bayes: Recall Scores

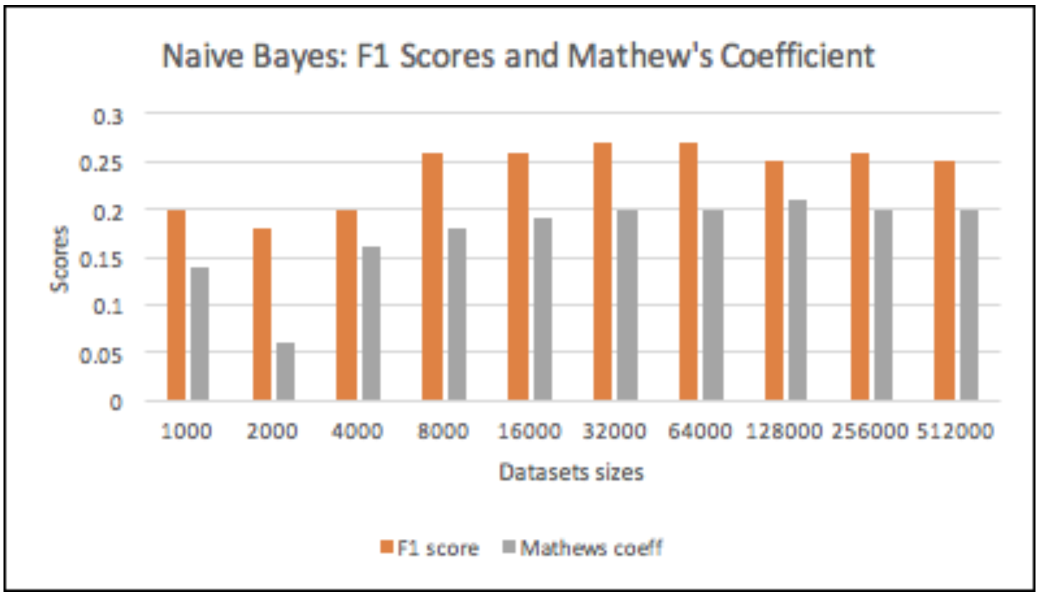


Figure 8 Naive Bayes:: F1 Score and Matthew's Coefficient

We observe that the micro Precision scores lie in the range of 50% to 70%. Naive Bayes classifier has less number of false positives. Similar to the KNN, the micro precision value gives a proper estimate of the precision measure since our dataset has classes of variable sizes. Also, similar to the KNN, the Naive Bayes classifier has lower recall scores and thus there are more false negatives but this again, is compensated by higher precision values.

F-measure for Naive Bayes classifier is in the range of 0.2 to 0.25. The Matthew's coefficient for multi-class data lied between 0.15 to 0.2 for datasets of varying sizes.

5.2.2.2 Analysis of Accuracy

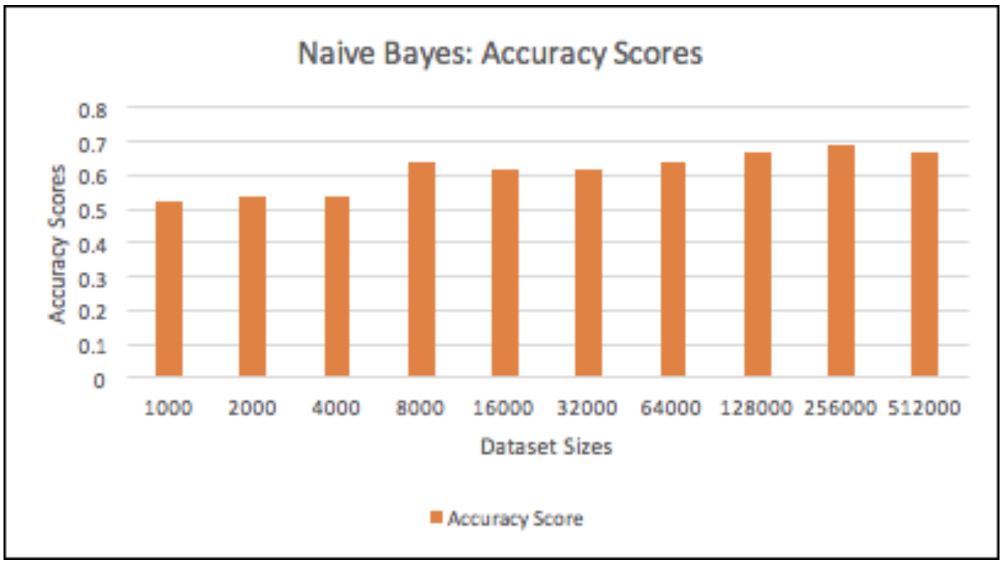


Figure 9 Naive Bayes: Accuracy Scores

Naive Bayes performs really well and the accuracy scores are close to the Decision Tree accuracies. We calculated the accuracy for different dataset sizes. The accuracy is in the range of 50% - 70%.

5.2.2.3. Analysis of Running Time

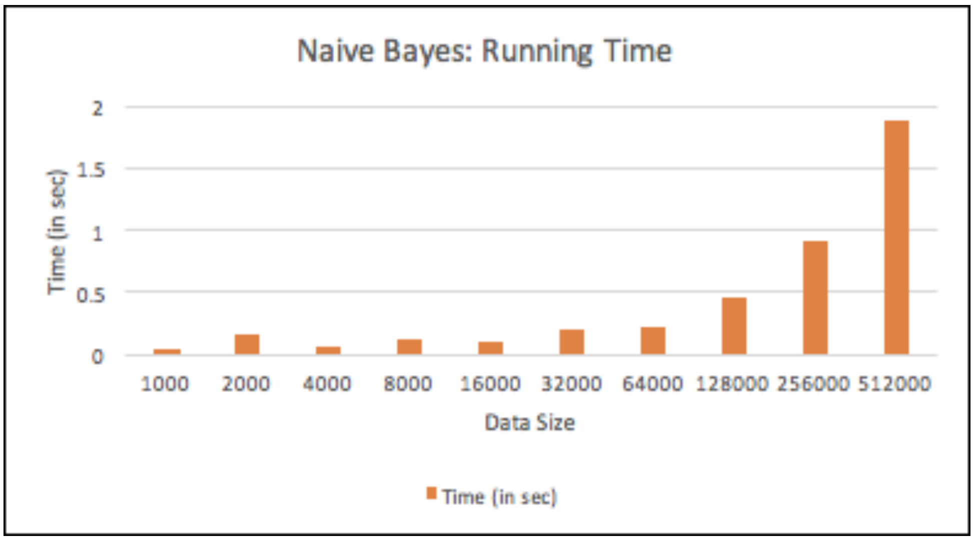


Figure 10 Naive Bayes::Running Times

Naive Bayes is computationally very fast. We can see that as the dataset size increased to almost half a million, yet it was able to predict the values within a couple of seconds. Naive Bayes took around 81 seconds to run on around a million records.

5.3. ID3 (Iterative Dichotomizer)

ID3 (Iterative Dichotomiser 3) is a decision tree algorithm [3] which was invented by Ross Quinlan for the purpose of generating a decision tree from a dataset. ID3 is the precursor to the C4.5 algorithm, and has its applications in the natural language processing and machine learning domains.

The ID3 algorithm is used by training on a dataset in order to produce a decision tree which is stored in memory. This decision tree is used at runtime to classify new unseen test cases by working down the decision tree using the values of this test case to arrive at a terminal node that tells you what class this test case belongs to.

Throughout this algorithm, the decision tree is built with each non-terminal node representing the selected attribute on which splitting was done, and terminal nodes representing the class label of the branch’s final subset.

5.3.1. Algorithm

1. Calculate entropy of the target.
2. The entropy for each branch is calculated which is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split to calculate information gain.
3. Choose the best attribute to split on by selecting largest information gain as decision node.
4. Split dataset on best attribute and on each split, recursively call the algorithm. The empty tree is populated with subtrees which are the result of recursive call.
5. The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

5.3.2. Pros and Cons

ID3 builds the fastest tree out of all the decision tree algorithms that we implemented. In ID3, the algorithm only needs to test enough attributes until all data gets classified. Since leaf nodes have to be found, this enables the test data to be pruned, hence reducing the number of tests.ID3 also utilizes entire dataset to create the tree, hence no part is left untouched.

However, this algorithm has its own limitations. Only one attribute at a time is tested for making a decision. Classifying continuous data may be computationally expensive, as many trees must be generated to see where to break the continuum. Data may be over-fitted or over-classified, if a small sample is tested.

5.3.3. Analysis of results

5.3.3.1 Analysis of Precision, Recall and Matthew's Coefficient:

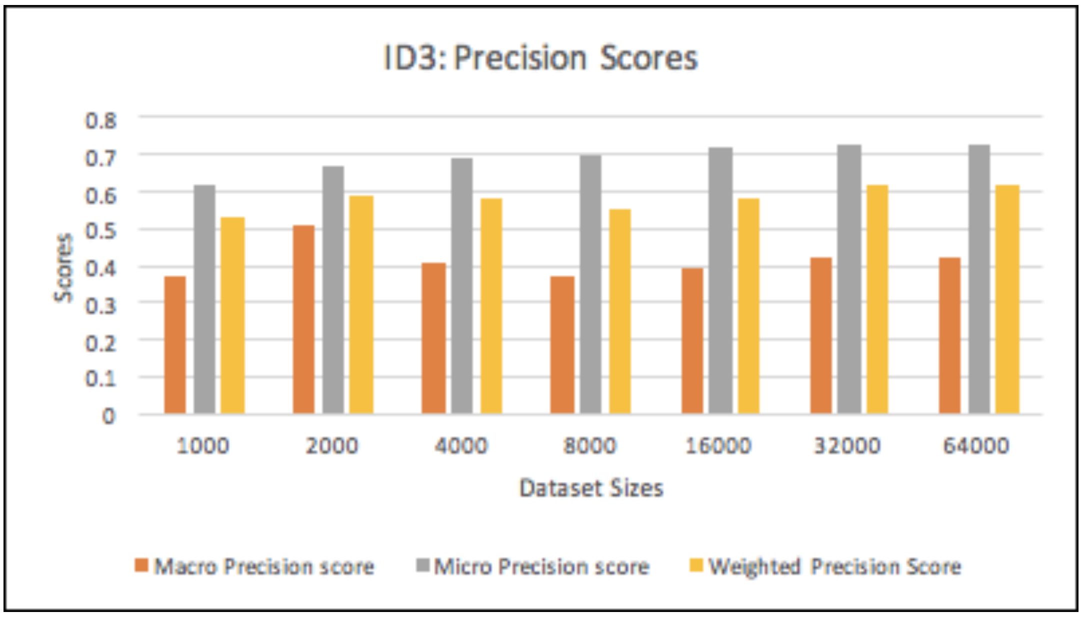


Figure 11 ID3: Precision Scores

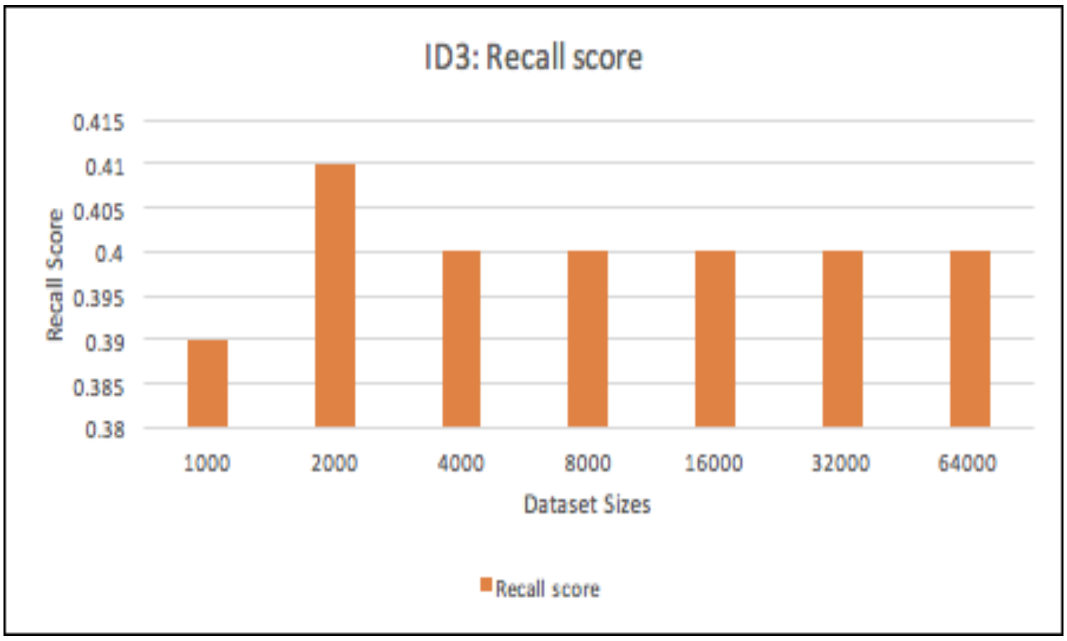


Figure 12 ID3: Recall Scores

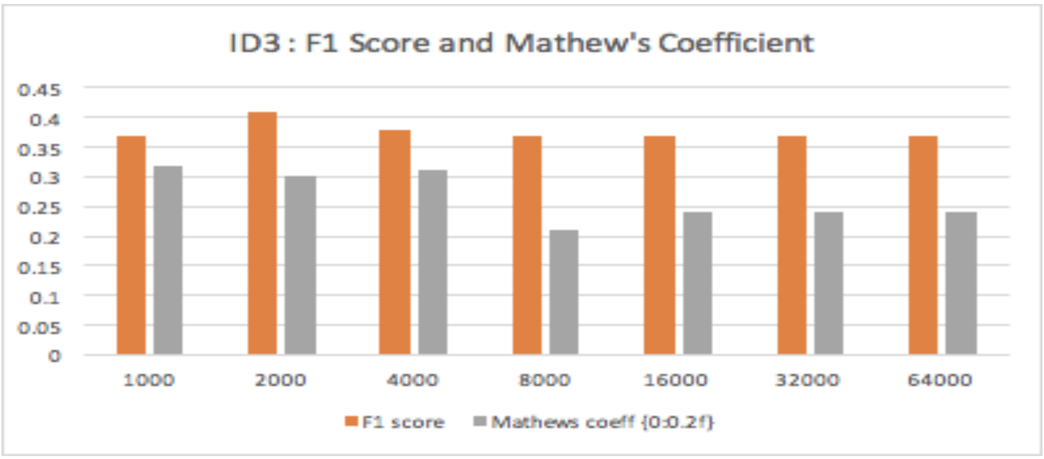
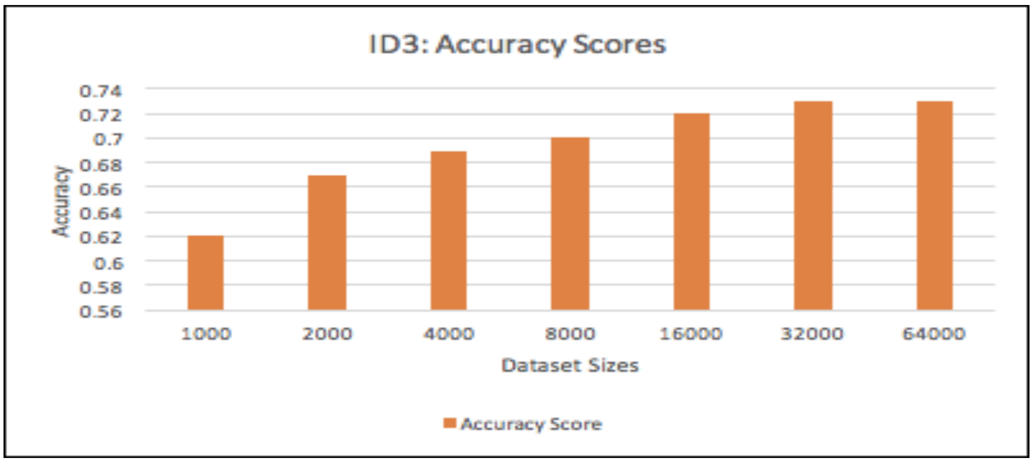


Figure 13 ID3: F1 Scores and Matthew's Coefficient

We observe that the micro Precision scores lie in the range of 62% to 71%. The micro precision value gives a proper estimate of the precision measure since our dataset has classes of variable sizes. Also, ID3 classifier has lower recall scores and thus there are more false negatives but this again, is compensated by higher precision values.

F-measure for ID3 classifier is in the range of 0.36 to 0.42. The Matthew's coefficient for multi-class data lied between 0.21 to 0.31 for datasets of varying sizes.

5.3.3.2 Analysis of Accuracy



ID3 performs really well in terms of accuracy. We calculated the accuracy for different dataset sizes and observed that the accuracy is in the range of 62% - 73%.

5.3.3.3. Analysis of Running Time

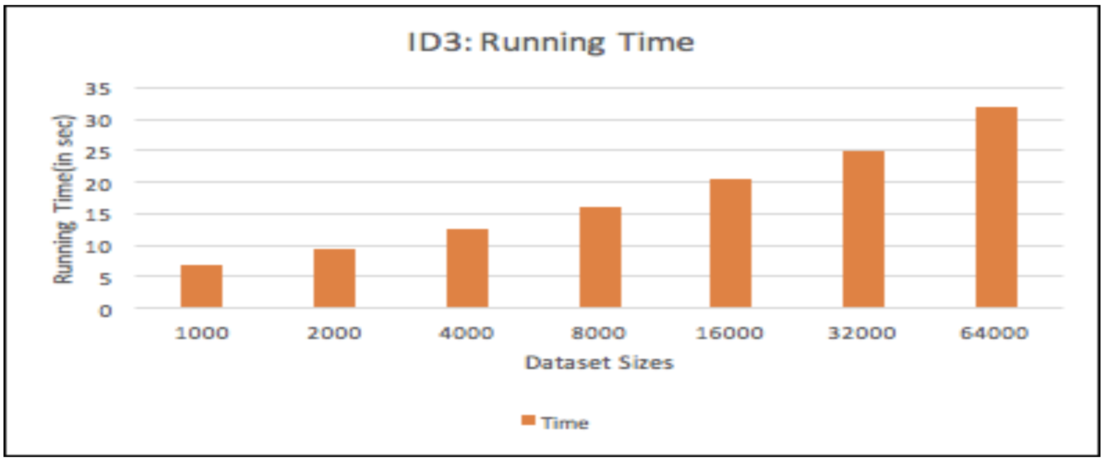


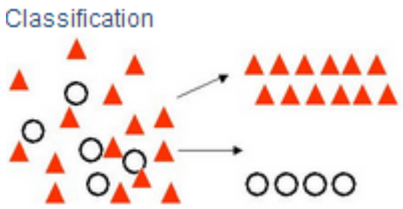
Figure 14 ID3: Running Times

ID3 is computationally very fast and the fastest among all Decision Trees algorithms. We can see that as the dataset size increased to almost 64000, yet it was able to predict the values within a couple of seconds. ID3 took around 32 seconds to run on 64000 records.

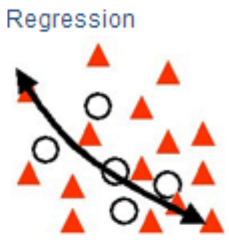
5.4. CART - Classification And Regression Trees

The CART or Classification & Regression Trees methodology was introduced in 1984 by Leo Breiman, Jerome Friedman, Richard Olshen and Charles Stone as an umbrella term to refer to the following types of decision trees:

- **Classification Trees**: where the target variable is categorical and the tree is used to identify the "class" within which a target variable would likely fall into.



- **Regression Trees:** where the target variable is continuous and tree is used to predict its value.



The CART algorithm is structured as a sequence of questions, the answers to which determine what the next question, if any should be. The result of these questions is a tree like structure where the ends are terminal nodes at which point there are no more questions.

The main elements of CART (and any decision tree algorithm) are:

- Specifying the criteria for predictive accuracy
- Selecting splits
- Determining when to stop splitting
- Selecting the "right-sized" tree (Pruning)

CART algorithm can be used for building both Classification and Regression Decision Trees. The impurity (or purity) measure used in building decision tree in CART is Gini Index. The decision tree built by CART algorithm is always a binary decision tree (each node will have only two child nodes).

The Gini Index is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions. Information Gain multiplies the probability of the class times the log (base=2) of that class probability. Information Gain favors smaller partitions with many distinct values. Ultimately, you have to experiment with your data and the splitting criterion.

5.4.1. GINI Index

- Favors larger partitions.
- Uses squared proportion of classes.
- Perfectly classified, Gini Index would be zero.
- Evenly distributed would be 1 – (1/# Classes).
- We want a variable split that has a low Gini Index.

GINI Index:

$$i(t) = \sum_{i,j} C(i|j)p(i|t)p(j|t), \qquad \Delta i(s,t) = i(t) - p_L i(t_L) - p_R i(t_R),$$

Pseudo Algorithm for calculating Gini Index:

```
Gini Index:
    for each branch in split:
        Calculate percent branch represents #Used for weighting
        for each class in branch:
            Calculate probability of class in the given branch.
            Square the class probability.
        Sum the squared class probabilities.
        Subtract the sum from 1. #This is the Gini Index for branch
    Weight each branch based on the baseline probability.
    Sum the weighted gini index for each split.
```

5.4.2. Algorithm

1. Take all of your data.
2. Consider all possible values of all variables.
3. Select the variable/value (X=t1) that produces the greatest “separation” in the target. (Gini has been used as evaluation criteria)
4. (X=t1) is called a “split”.
5. If X< t1 then send the data to the “left”; otherwise, send data point to the “right”.
6. Now repeat same process on these two “nodes” to build the decision tree until stopping criteria
7. Use this tree to predict outcome on training data

5.4.3. Pros and Cons

CART can be used to characterize outcomes as a function of many predictors. It can handle continuous data and can be used in regression problems. It is nonparametric and thus requires no probabilistic assumptions. It uses any combination of continuous/discrete variables and used to discover interactions between variables. This makes CART one of the simplest yet most powerful algorithm for our dataset.

There are some limitations while using CART. The trees tend to overfit data. Additionally, small changes to input data could result in major changes to tree structure. CART also struggles to capture a linear relationship.

5.4.4. Analysis of Results

5.4.4.1. Analysis of Precision, Recall and Matthew's Coefficient

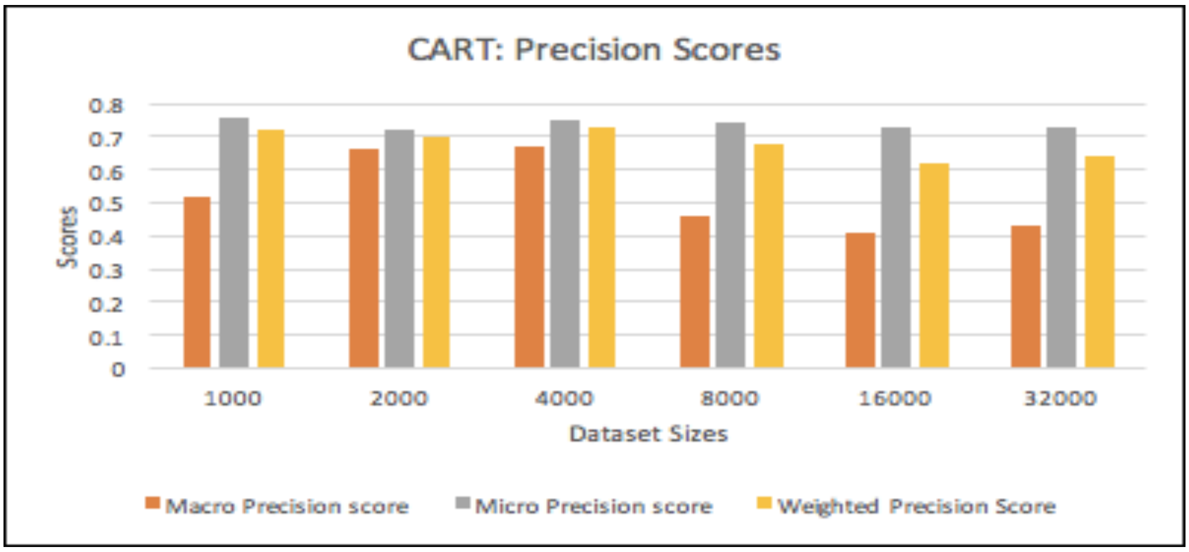


Figure 15 CART: Precision Scores

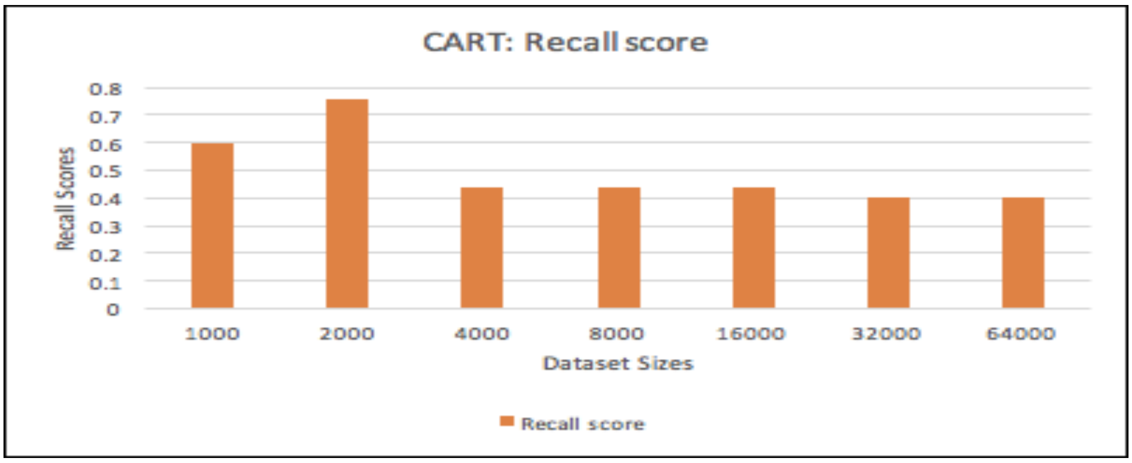


Figure 16 CART: Recall Scores

The micro precision score for CART lies in the range of 70% to 80% which emphasizes the fact that CART performs really well even on larger datasets. The recall score is in the range of 0.4 and 0.75.

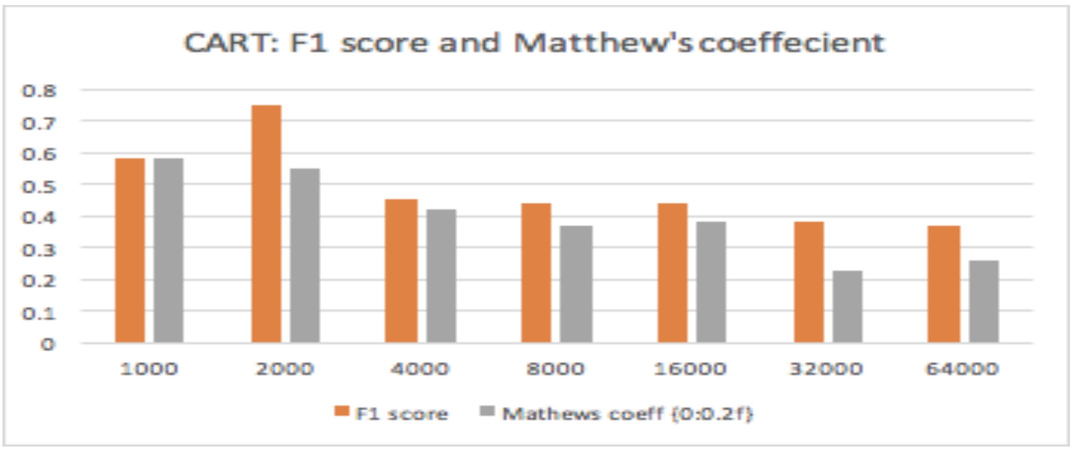


Figure 17 : F1 Scores and Matthew's Coefficient

F-measure for CART classifier is in the range of 0.35 to 0.75. The Matthew's coefficient for multi-class data lied between 0.21 to 0.58 for datasets of varying sizes.

5.4.4.2. Analysis of Accuracy

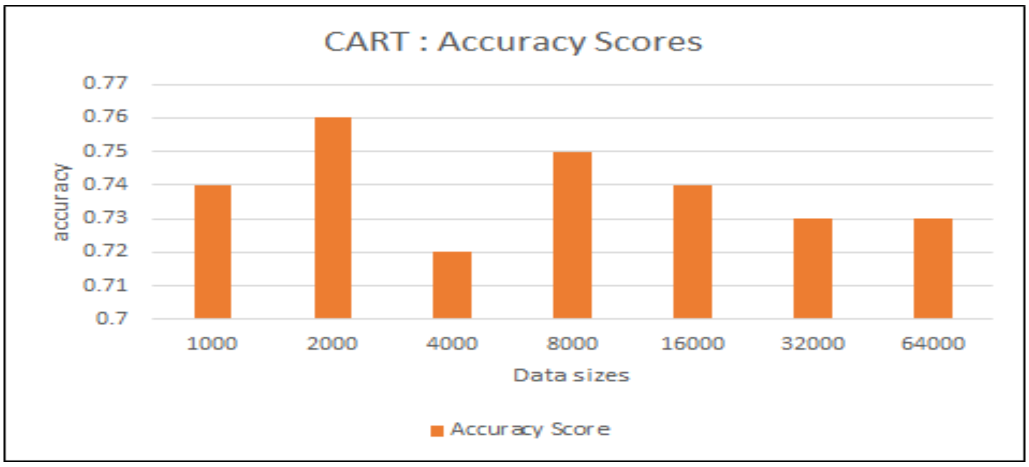


Figure 18 CART: Accuracy Scores

CART algorithm achieved one of the best accuracies across all the algorithms. The accuracy lies between 72% to 76% over different sizes of datasets.

5.4.4.2. Analysis of Running Time

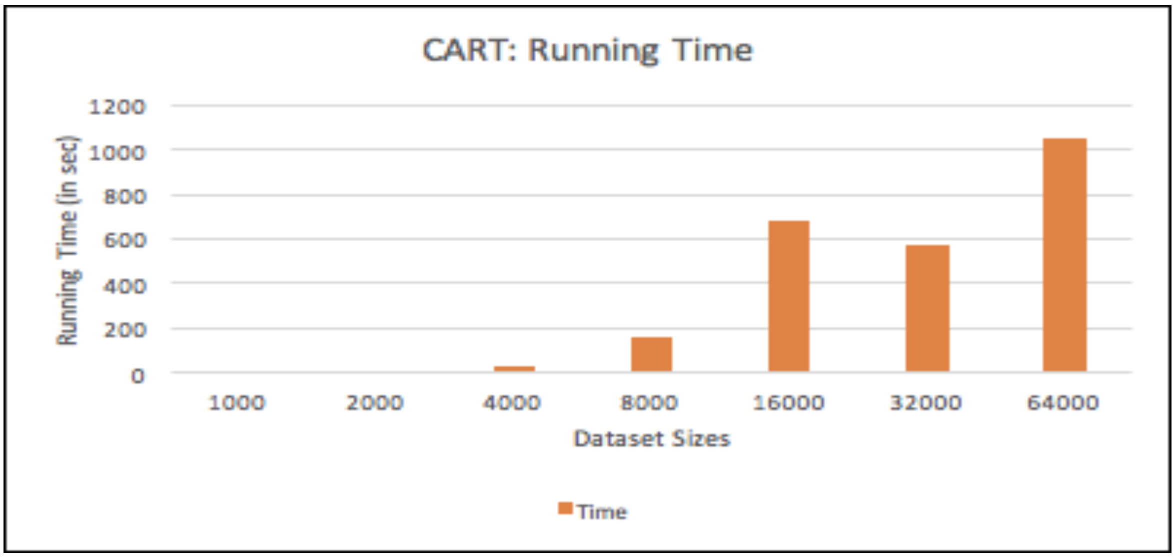


Figure 19 CART: Running Times

The running time of CART increased drastically when the dataset sizes were increased. CART performed very badly after 64000 records.

5.5. C4.5

C4.5 is a classification algorithm used to generate a decision tree developed by Ross Quinlan. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier.

5.5.1. Algorithm

1. C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy.
2. The training data is a set **$S= s_1,s_2,...$** of already classified samples.
3. Each sample **s_i** consists of a p-dimensional vector **$(x_{1,i},x_{2,i},....x_{p,i})$**, where the **$x_j$** represent attribute values or features of the sample, as well as the class in which **s_i** falls.
   At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other.
4. The splitting criterion is the normalized information gain (difference in entropy).
5. The attribute with the highest normalized information gain is chosen to make the decision. The algorithm then recurs on the smaller sublists.

Entropy : Entropy indicates the diversity of the set element values. The smaller the entropy is, the purer the set is. If the set has only one value, its entropy should be 0.

For distributed set:

$$\text{Entropy(X)} = -\sum_{i \in X, i \neq "?"} \text{P}(\text{x} = \text{i})\log(\text{P}(\text{x} = \text{i}))$$

For continuous set:

$$\text{Entropy\_c(X, i)} = -P(x < i)\log(P(x < i)) - P(x \geq i)\log(P(x \geq i))$$

Gain:

The gain of the attribute indicates the contribution to the classification by the attribute.

For distributed set:

$$\text{Gain(attribute)} = Entropy(category) - \sum_{k} P(attribute = k) Entropy(category_{attribute=k})$$

For continuous set:

$$\text{Gain\_c(attribute)} = Entropy(category) - \min_{i \in \text{attribute}, i \neq "?"} \{E \mid E = -P(\text{attribute} < i) Entropy(category_{attribute<k}) - P(\text{attribute} \geq i) Entropy(category_{attribute \geq k})\}$$

Gain Ratio:

If the selection of the attribute is only based the gain of attribute, it may often prefer the attributes which has more values, which is unreasonable. Thus we introduce Gain ratio.

For distributed set:

Ratio = Gain(attribute) /$Entropy$(attribute)

For continuous set:

Ratio = Gain_c(attribute) /$Entropy\_c$(attribute, i)

$$i = \min_{i \in \text{attribute}, i \neq "?"} \{i \mid E = -P(\text{attribute} < i) Entropy(category_{attribute<k}) - P(\text{attribute} \geq i) Entropy(category_{attribute \geq k})\}$$

5.5.2. Pros and Cons

Following are the pros c4.5 algorithm has over its precursor ID3:

- Avoiding overfitting the data
    - Determining how deeply to grow a decision tree.
- Reduced error pruning.
- Handling continuous and discrete attributes.
    - e.g., Age and Gender respectively
- Choosing an appropriate attribute selection measure.
- Handling training data with missing attribute values.
- Improving computational efficiency.

Some of the disadvantages of this algorithm are:

- Small variation in data can lead to different decision trees (especially when the variables are close to each other in value).
- Does not work very well on a small training set.

5.5.3. Analysis of Results

5.5.3.1. Analysis of Precision, Recall

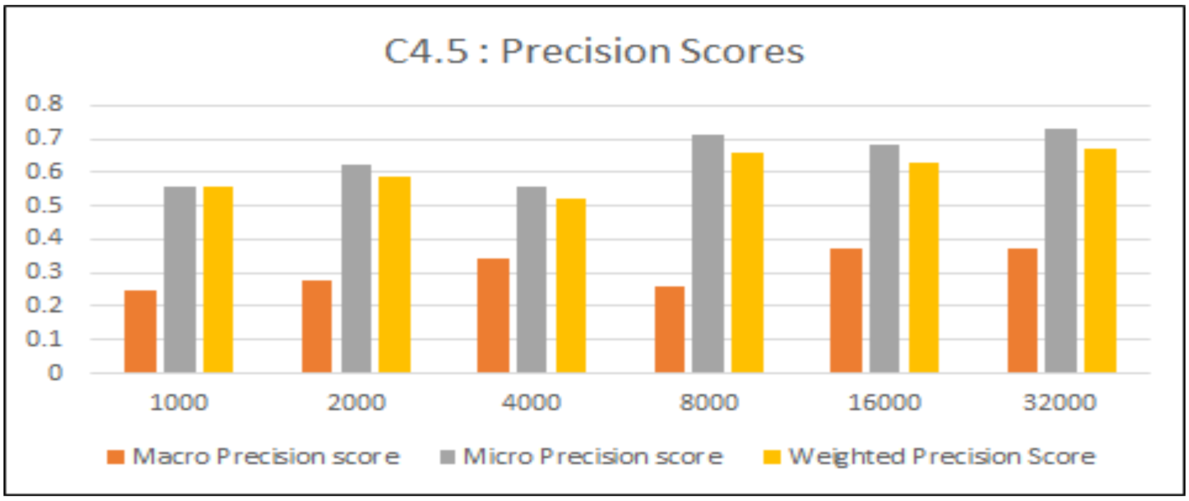


Figure 20 C4.5 :Precision Scores

The micro precision score for C4.5 lies in the range of 55% to 75% which gives a proper estimate of the precision measure since our dataset has classes of variable sizes.

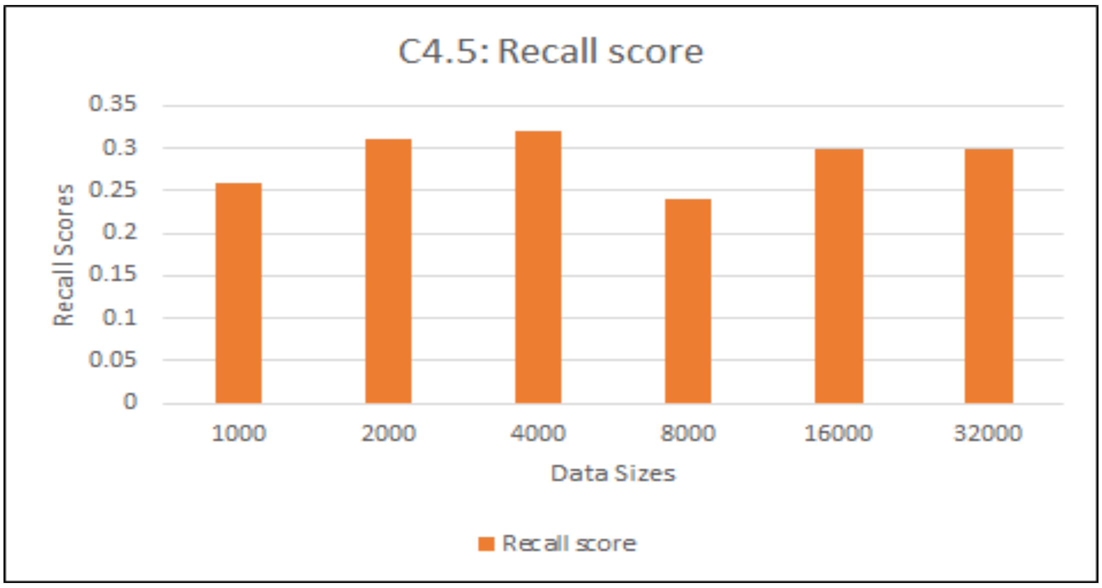


Figure 21 : Recall Scores

Also, C4.5 classifier has lower recall scores and thus there are more false negatives but this again, is compensated by relatively higher precision values.

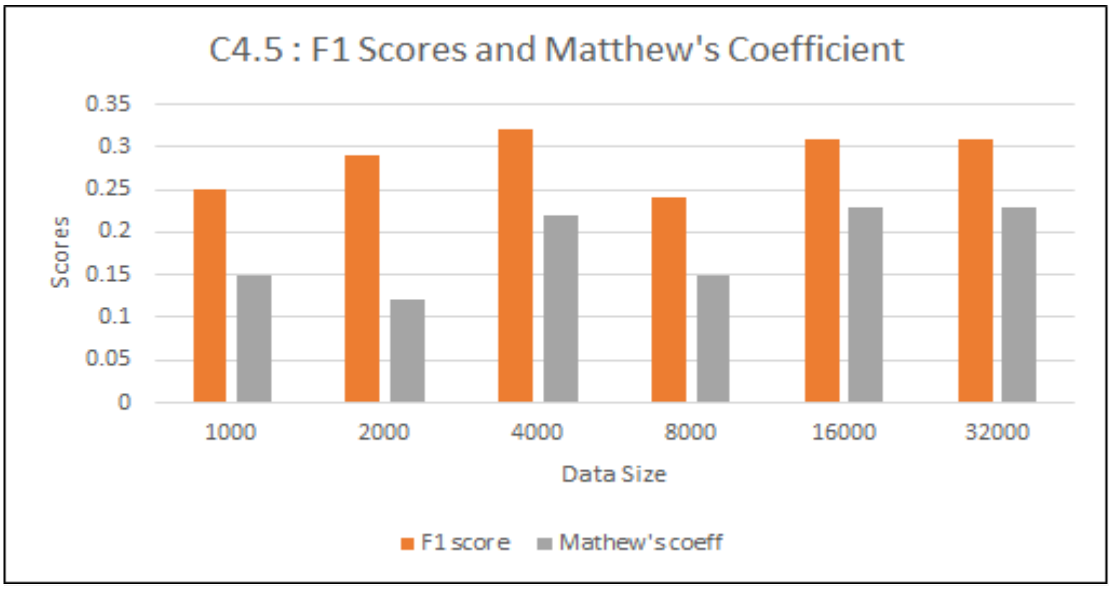
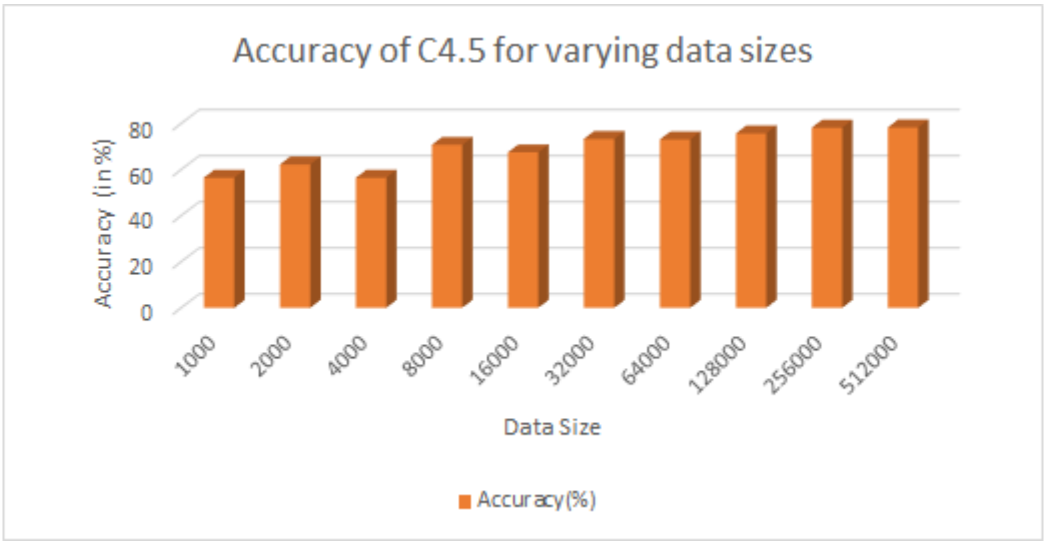


Figure 22 C4.5 :F1 Scores and Matthew's Coefficient

5.5.3.2. Analysis of Accuracy



C4.5 performs really well with a larger training dataset. We calculated the accuracy for different dataset sizes. The accuracy is in the range of 56% - 79%.

5.5.3.3. Analysis of Running Time

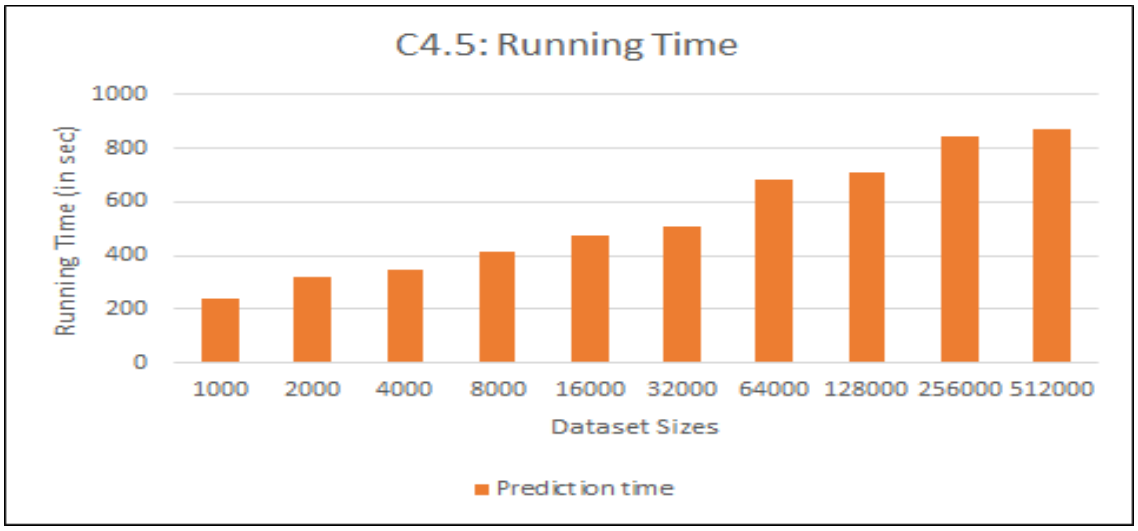


Figure 23 C4.5: Running Times

The running time of C4.5 increased linearly when the dataset sizes were increased. C4.5 performed very badly after 512000 records.

6. PERFORMANCE EVALUATION

We did an overall comparison of the classification methods in terms of accuracy, precision-recall, running time and made some observations based on the graphs obtained.

6.1. Overall Comparison of Accuracies

We did an overall comparison of the accuracies for C4.5, Naïve Bayes, KNN, CART and ID3 for datasets having varying number of records and generated the graph shown in Figure 24.

Dataset Sizes	C4.5	Naïve Bayes	KNN	CART	ID3
1000	56	52	51	74	62
2000	62	54	64	76	67
4000	56	54	65	72	69
8000	71	64	67	75	70
16000	68	62	67	74	72
32000	73	62	67	73	73
64000	72	64	34	73	73

Figure 24 Table for Overall comparison of Accuracies (in Percentage)

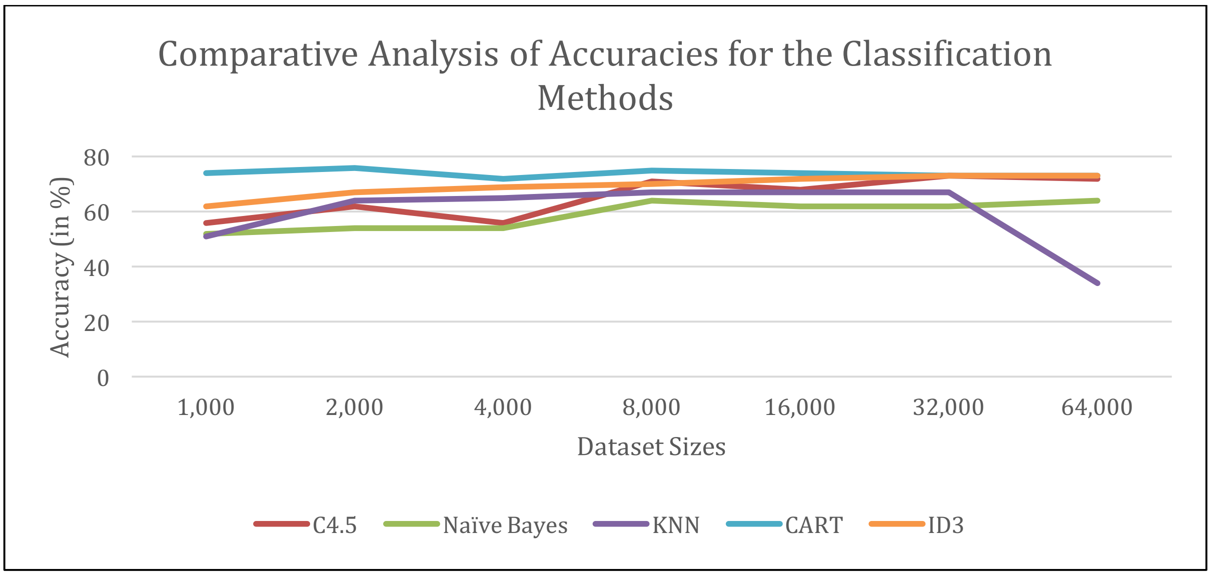


Figure 25 Graph for Comparative Analysis of Accuracies for Classification Methods

Following observations based on the graph obtained:

- It was observed that the decision Tree algorithms – C4.5, ID3, CART, have much better accuracies (in the range of 55% to 75%) as compared to KNN and Naïve Bayes.
- Their accuracies remain constant even with the increase in data sizes.
- CART has the best overall accuracy measure of 70% to 75%.

- This is followed by ID3 whose accuracy increases with the increase in the size of records.
- C4.5 has lesser accuracy score as compared to the other two decision tree algorithms. However, it's performance also increases with the increase in data size.
- KNN has an accuracy in the range 50% to 65%. Its accuracy increases linearly with the increase in data size, However, the accuracy drops significantly for data size of more than 32000.
- Finally, Naïve Bayes algorithm has the lowest accuracy as compared to other algorithms. It's accuracy also increases linearly with the data size.

6.2. Overall Comparison of Running Times

We obtained the running times for the algorithms for varying data sizes as shown in Figure 26 and did an overall analysis of the classifiers based on the graph obtained.

Dataset Sizes	C4.5	Naïve Bayes	KNN	CART	ID3
1000	240	0.044	1.128	1.978	6.825
2000	321	0.157	2.541	9.943	9.525
4000	350	0.057	6.723	30.313	12.662
8000	412	0.118	19.277	162.67	15.897
16000	475	0.101	56.976	685.275	20.518
32000	510	0.194	217.365	772.932	24.944
64000	682	0.231	750.1	1054.156	32.098

Figure 26 Table for Overall comparison of Running Times (in seconds)

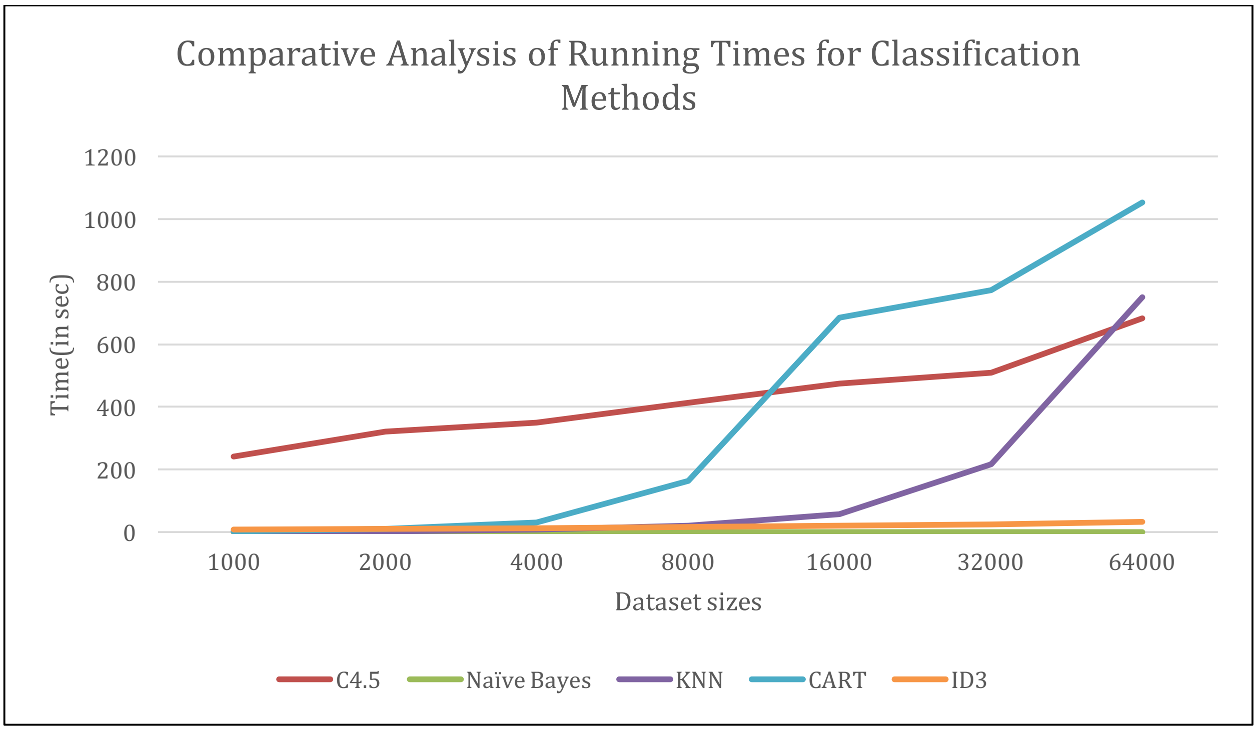


Figure 27 Graph for Comparative Analysis of Accuracies for Classification Methods

The following observations were made based on the graph:

- Naïve Bayes classifier is the fastest of all the algorithms followed by ID3.Their performance remains constant regardless of the size of the dataset.
  Since ID3 algorithm tests attributes only until all the data has been classified, it has the fastest running time.
  The fast running time of Naïve Bayes can be attributed to the fact that it is computationally fast and simple in terms of implementation since it is based on conditional independence.

- Decision Tree algorithms: C4.5 and CART perform poorly in terms of running time. This can be because the size of the decision tree increases with the increase in data leading to increase n running time.

- CART performs well for smaller data sizes, but as the number of records increases beyond 4000, the performance starts deteriorating. It takes the maximum time of 1024s for data size of 64000.

- C4.5 has a higher running time than Naïve Bayes, KNN and ID3 even for smaller data sizes.

- KNN classifier performs extremely well for smaller data sizes (<16000). The running time increases in a linear manner as the size of dataset increases.

6.3. Overall Comparison of Precision Recall Scores

We did an analysis of the precision recall scores since accuracy measure can sometimes be misleading. According to Entezari-Maleki, Rezaei [2] they are an important evaluation measure for understanding how well a classifier performs.
Micro precision score was selected as the evaluation criteria as it gives a clearer picture when data sizes are variable.

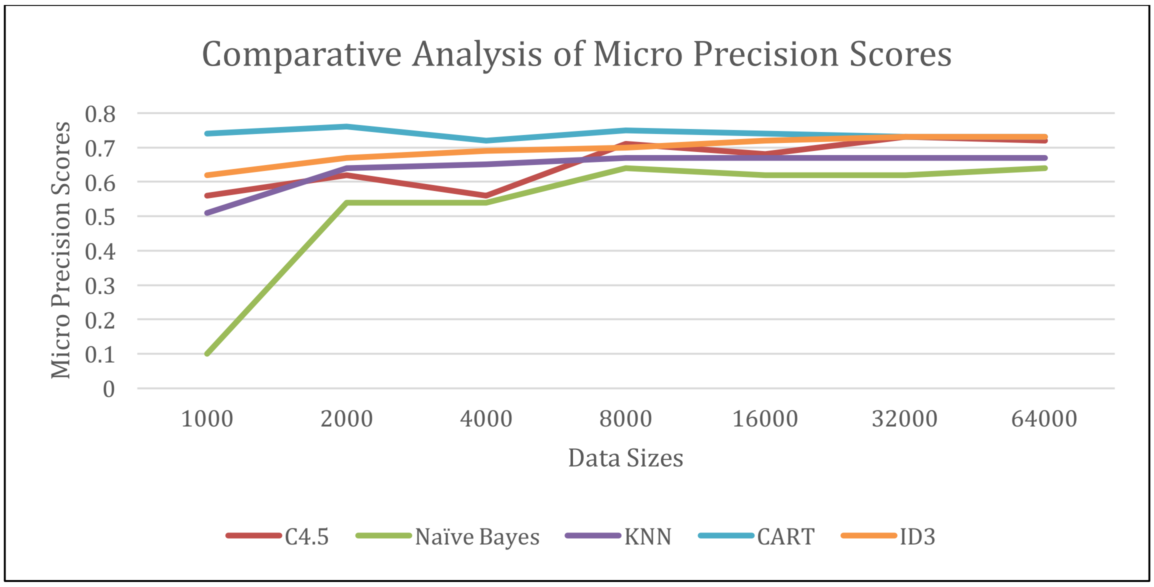


Figure 28 Graph for overall comparison of Precision scores for all classification methods

As observed from Figure 28, the precision scores for the crime dataset lie in the range of 0.55 to 0.77. The precision score of CART is the highest (0.70 to 0.77) followed by ID3, C4.5 and KNN. Naive Bayes has comparatively lesser precision scores for the data sizes. Therefore, we can say that the decision tree algorithms have a better average precision score.

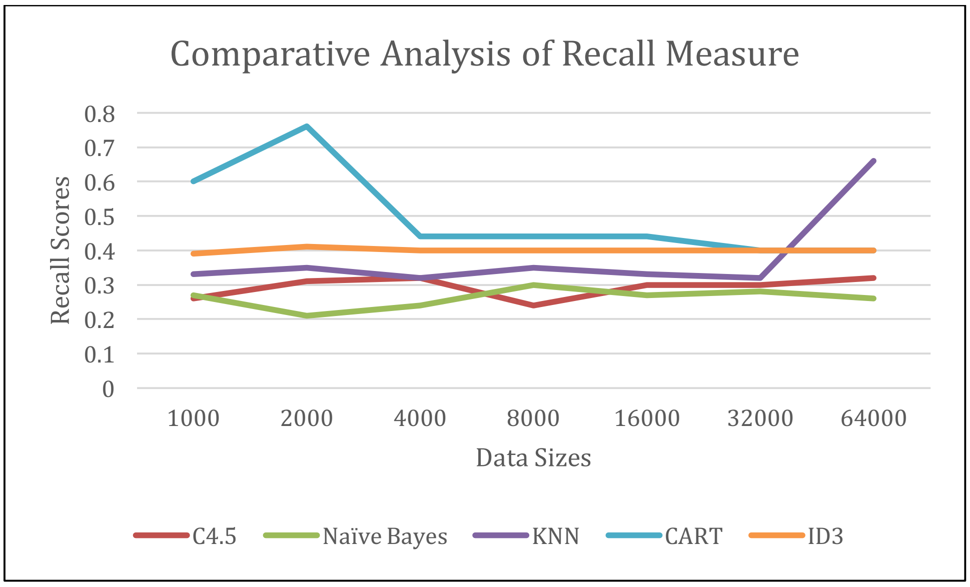


Figure 29 Graph for comparison of Recall scores for all classification methods

From Figure 29, we can observe that the recall scores for the classifiers lie in the range of 0.25 to 0.50. We observed that CART had the best recall score as compared to other algorithms with average recall of 0.50. This is followed by ID3 and C4.5 which have the average recall scores of 0.44 and 0.31 respectively. KNN has an average precision score of 0.35. Naive Bayes has the least average recall score of 0.28 as compared to all other classifiers.

7. CONCLUSION

From the analysis of the evaluation measures, we can conclude that the decision tree algorithms perform well in terms of running times and precision recall measures as compared to Naïve Bayes. Although KNN algorithm was really fast for small datasets, it takes up a lot of space for training larger data. Also, the computation cost of KNN is higher and it takes more time for lager datasets.

The stop outcome by the police could be predicted correctly using the classification methods with an average accuracy of 70%. This would be useful in determining the quality of inspection done by police and improving the rules and regulations.

8. LIMITATIONS & POTENTIAL IMPROVEMENTS

One of the limitations that we faced while classifying the data was the restriction in terms of the computing power available for classifying huge data. A potential future improvement that we can implement to solve this problem would be to do a parallelization of algorithms to make them more scalable. This can further lead to an improved performance for larger data sizes. Secondly, we can use PySpark for handling the large data.

Apart from that, there were some limitations on the data for Crime classification. Some of the columns in the Texas Crime dataset were empty due to which we had to move to different datasets-South Carolina, Washington and North Carolina. If we can obtain more data from other states as well, we can have a broader range of results and a better accuracy.

REFERENCES

[1] E. Pierson, C. Simoiu, J. Overgoor, S. Corbett-Davies, V. Ramachandran, C. Phillips, S. Goel. (2017) “A large-scale analysis of racial disparities in police stops across the United States”.

[2] R. Entezari-Maleki, A. Rezaei, and B. Minaei-Bidgoli, “Comparison of Classification Methods Based on the Type of Attributes and Sample Size,” Available at: http://www4.ncsu.edu/~arezaei2/paper/JCIT4- 184028_Camera%20Ready.pdf

[3]Decision Tree ID3 classification, Available at: http://www.saedsayad.com/decision_tree.htm

[4]Yan-yan Song,Ying Lu , ”Decision tree methods: applications for classification and prediction”.

[5] http://scikit-learn.org/stable/modules/naive_bayes.html

[6] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[7] http://scikit-learn.org/stable/modules/neighbors.html

[8] http://scikit-learn.org/stable/modules/tree.html

[9] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[10] http://www.saedsayad.com/k_nearest_neighbors.htm

[11] https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

[12] https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[13] http://dataaspirant.com/2017/02/06/naive-bayes-classifier-machine-learning/