Predicting Income from Census Data using Multiple Classifiers

CS 6200 Data Mining Project - Spring 2015
Northeastern University

Members:
Monisha Singh
Smitha Bangalore Naresh
Utkarsh Jadhav

Contents

1. Objective

- Analysis of Census Data to determine certain trends.
- Predicting if a person with certain information would be able to earn more than $50,000.
- Use preprocessing techniques to improve the accuracy of prediction on test data set, and avoid bias.
- Evaluate different classification algorithms with and without preprocessing.
- Analyse the accuracy, and run time of different classification algorithms.
- Select a classification algorithm that improves the accuracy of prediction on test data set.

2. Overview of Dataset

The Census Income Data Set dataset is taken from UCI Machine Learning Repository.
Link: https://archive.ics.uci.edu/ml/datasets/Census+Income

Data Set Characteristics:	Multivariate	**Number of Instances:**	48842
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	14
Associated Tasks:	Classification	**Missing Values?**	Yes

Table 1 Overview of dataset

- 48842(train = 32561, test = 16281) instances
- 45222(train = 30162, test = 15060) if instances with unknown values are removed
- Duplicate or conflicting instances : 6
- 2 classes : >50K, <=50K
- 14 attributes : both continuous and discrete-valued.

2.1 Attributes

- **age**: continuous.
- **workclass**: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- **fnlwgt**: continuous.
- **education**: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- education-num: continuous.
- **marital-status**: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- **occupation**: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- **race**: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- **sex**: Female, Male.
- **capital-gain**: continuous.
- **capital-loss**: continuous.
- **hours-per-week**: continuous.
- **native-country**: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

2.2 Data Instances

Sample data instances:

39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K
50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 13, United-States, <=50K
38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners, Not-in-family, White, Male, 0, 0, 40, United-States, <=50K
53, Private, 234721, 11th, 7, Married-civ-spouse, Handlers-cleaners, Husband, Black, Male, 0, 0, 40, United-States, <=50K
28, Private, 338409, Bachelors, 13, Married-civ-spouse, Prof-specialty, Wife, Black, Female, 0, 0, 40, Cuba, <=50K
37, Private, 284582, Masters, 14, Married-civ-spouse, Exec-managerial, Wife, White, Female, 0, 0, 40, United-States, <=50K
49, Private, 160187, 9th, 5, Married-spouse-absent, Other-service, Not-in-family, Black, Female, 0, 0, 16, Jamaica, <=50K
52, Self-emp-not-inc, 209642, HS-grad, 9, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 45, United-States, >50K
31, Private, 45781, Masters, 14, Never-married, Prof-specialty, Not-in-family, White, Female, 14084, 0, 50, United-States, >50K
42, Private, 159449, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 5178, 0, 40, United-States, >50K
37, Private, 280464, Some-college, 10, Married-civ-spouse, Exec-managerial, Husband, Black, Male, 0, 0, 80, United-States, >50K
30, State-gov, 141297, Bachelors, 13, Married-civ-spouse, Prof-specialty, Husband, Asian-Pac-Islander, Male, 0, 0, 40, India, >50K
23, Private, 122272, Bachelors, 13, Never-married, Adm-clerical, Own-child, White, Female, 0, 0, 30, United-States, <=50K
32, Private, 205019, Assoc-acdm, 12, Never-married, Sales, Not-in-family, Black, Male, 0, 0, 50, United-States, <=50K
40, Private, 121772, Assoc-voc, 11, Married-civ-spouse, Craft-repair, Husband, Asian-Pac-Islander, Male, 0, 0, 40, ?, >50K
34, Private, 245487, 7th-8th, 4, Married-civ-spouse, Transport-moving, Husband, Amer-Indian-Eskimo, Male, 0, 0, 45, Mexico, <=50K
25, Self-emp-not-inc, 176756, HS-grad, 9, Never-married, Farming-fishing, Own-child, White, Male, 0, 0, 35, United-States, <=50K
32, Private, 186824, HS-grad, 9, Never-married, Machine-op-inspct, Unmarried, White, Male, 0, 0, 40, United-States, <=50K
38, Private, 28887, 11th, 7, Married-civ-spouse, Sales, Husband, White, Male, 0, 0, 50, United-States, <=50K
43, Self-emp-not-inc, 292175, Masters, 14, Divorced, Exec-managerial, Unmarried, White, Female, 0, 0, 45, United-States, >50K

3. Tools

3.1 Java

Version - jdk 7 with jre1.8
IDE - Eclipse Luna X64

3.2 WEKA

We are building the framework using java along with weka.jar
weka.jar 3.6.12

The Weka workbench is a set of tools for preprocessing data, experimenting with data-mining/machine learning algorithms, and comparing the performance of different methods. Weka also provides a Java class library that enables one to use the Weka filters and classifiers in their own programs.

An ARFF file (Attribute-Relation File Format) is a standard way of representing machine learning data sets as flat files (no relationships among instances). Weka works with ARFF files.

We are given a census-income.data and census-income.test files which contain the training and test instances. Our program internally converts census-income.data and census-income.test files to census-income-data.arff and census-income-test.arff files respectively.

4. Preprocessing

4.1 Missing Values

Missing data, or missing values, occur when no data value is stored for the variable in an observation. Missing data are a common occurrence, and can have a significant effect on the conclusions that can be drawn from the data. Missing data can occur because of nonresponse: no information is provided for several items or no information is provided for a whole unit.

In the given census-income.data, and census-income.test, missing values were found in attributes work-class(2), occupation(7), native-country(14). The missing values can be missing at random (accidentally omitting an answer on a questionnaire), or missing not at random (a questionnaire tend to be skipped deliberately by participants with certain characteristics).

Number of instances given::

census-income-data.arff : 32561
census-income-test.arff : 16281

For census-income-data.arff the number of missing values are as follows:
Missing values in work-class attribute : 1836(6%) Type: Nominal
Missing values in occupation attribute : 1843(6%) Type: Nominal
Missing values in native-country attribute : 583(2%)Type: Nominal

For census-income-test.arff the number of missing values are as follows:
Missing values in work-class attribute : 963(6%) Type: Nominal
Missing values in occupation attribute : 966(6%) Type: Nominal
Missing values in native-country attribute : 274(2%)Type: Nominal

4.1.1 Handling of Missing values

We handled missing values, and experimented various classification algorithms in two ways:

1. Completely removing instances with missing data both in data, and test files and checking the accuracy.
   Number of instances after removing missing value instances:
   a. census-income-data.arff : 30152
   b. census-income-test.arff : 15060

2. Imputing the data with mean(numeric attributes) and mode(nominal attributes) values in both data file and using same mean and mode to impute in test file.
   Number of instances remains the same and are as follows:
   a. census-income-data.arff : 32561
   b. census-income-test.arff : 16281

Accuracy of various classification methods using above two ways is recorded below in Table 1.

<table>
<tr><th>Methods</th><th colspan="3">Remove Missing Values</th><th colspan="3">Replace Missing Values(Mean & Mode)</th></tr>
<tr><th></th><th>Correct</th><th>Incorrect</th><th>Confusion Matrix</th><th>Correct</th><th>Incorrect</th><th>Confusion Matrix</th></tr>
<tr><td>kNN with k=5</td><td>12332
81.8858%</td><td>2728
18.1142 %</td><td>10196 1164 | a = <=50K
1564 2136 | b = >50K</td><td>13417
82.4089 %</td><td>2864
17.5911 %</td><td>11237 1198 | a = <=50K
1666 2180 | b = >50K</td></tr>
<tr><td>J48</td><td>12848
85.3121 %</td><td>2212
14.6879 %</td><td>10547 813 | a = <=50K
1399 2301 | b = >50K</td><td>13987
85.91 %</td><td>2294
14.09 %</td><td>11611 824 | a = <=50K
1470 2376 | b = >50K</td></tr>
<tr><td>Naive Bayes</td><td>12429
82.5299 %</td><td>2631
17.4701 %</td><td>10549 811 | a = <=50K
1820 1880 | b = >50K</td><td>13515
83.0109 %</td><td>2766
16.9891 %</td><td>11582 853 | a = <=50K
1913 1933 | b = >50K</td></tr>
<tr><td>NBTree</td><td>12905
85.6906 %</td><td>2155
14.3094 %</td><td>10507 853 | a = <=50K
1302 2398 | b = >50K</td><td>14004
86.0144 %</td><td>2277
13.9856 %</td><td>11545 890 | a = <=50K
1387 2459 | b = >50K</td></tr>
<tr><td>Logistic</td><td>12766
84.7676 %</td><td>2294
15.2324 %</td><td>10530 830 | a = <=50K
1464 2236 | b = >50K</td><td>13848
85.0562 %</td><td>2433
14.9438 %</td><td>11592 843 | a = <=50K
1590 2256 | b = >50K</td></tr>
<tr><td>LibSVM</td><td>11070
73.5304 %</td><td>3985
26.4696 %</td><td>10570 788 | a = <=50K
3197 500 | b = >50K</td><td>12377
76.0211 %</td><td>3904
23.9789 %</td><td>12149 286 | a = <=50K
3618 228 | b = >50K</td></tr>
</table>

Table 1 Accuracy of various Classification methods using replacing, and removing missing values

Observation:

From the above table, we can infer that replacing missing values gives us better results than removing instances with missing values.

Analysis:

If we run filtering with evaluation criteria as information gain and search method as attribute ranking, we get the following results

Ranked attributes:

<Information Gain> <Attribute Number> <Name of the attribute>

0.1876	11	capital-gain
0.1165	12	capital-loss
0.0854	6	marital-status
0.0768	8	relationship
0.0406	10	sex
0.0376	5	education-num
0.0319	4	education
0.0297	1	age
0.0268	13	hours-per-week
0.0248	7	occupation
0.0111	2	workclass
0.0105	9	race
0.0102	14	native-country
0	3	fnlwgt

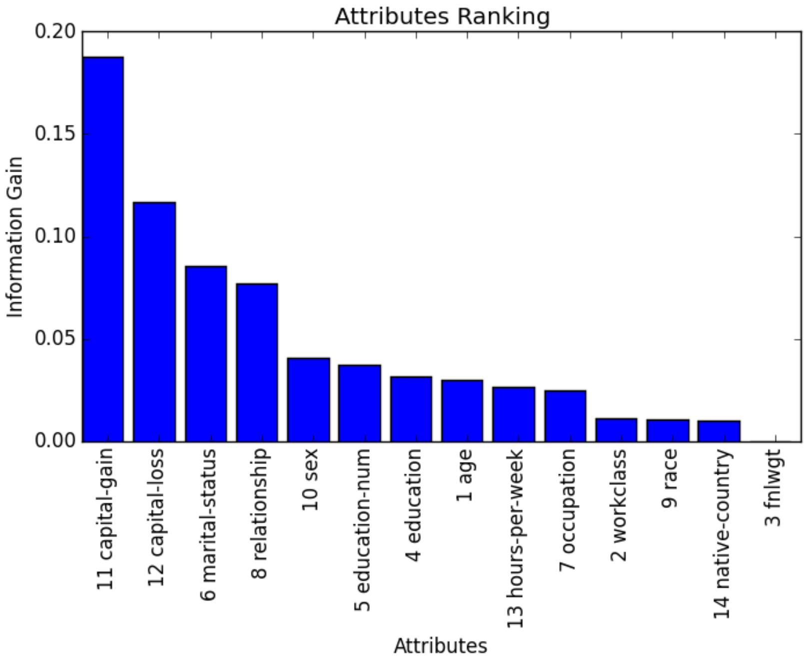

FIGURE 1 Attribute Ranking evaluation criteria as information gain

Missing attributes in data and test: work-class(2), occupation(7), native-country(14)

From the above results, we can infer that the attributes which are not missing plays a more important role in predicting the class of an instance. Thus we cannot remove the instances with missing value all together and by implementing mean and mode imputation the algorithms are able to achieve a better accuracy as they are able to learn much more with mean mode imputation.

4.2 Balancing the data

A dataset is imbalanced if the classification categories are not approximately equally represented. Standard learner algorithms are often biased towards the majority class when dataset is imbalanced. This is because these classifiers attempt to reduce the global error rate, not taking into account the data distribution. As a result examples from the majority class are well-classified whereas examples from the minority class tend to be misclassified

Observation

After running different algorithms with mean mode imputation we observed the more number of test instances with >50K(minority class) are getting misclassified. Thus we can infer methods are more biased towards predicting <=50K (majority class) correctly. This could be due to imbalance in our dataset.

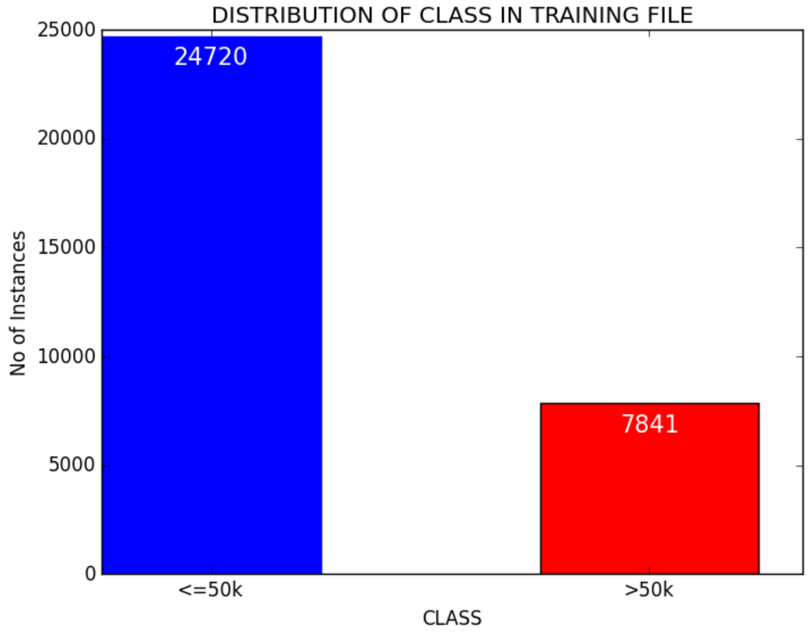

Figure 2 Number of Training Instances in given census.data

The number instances in training file classified as <=50K is much greater than number of instances with class >50K.
To overcome this problem we need to balance our data, using techniques such as oversampling, SMOTE etc. We experimented with three methods for balancing the data:

1. Oversampling
2. Undersampling
3. SMOTE

4.2.2 Oversampling

In random oversampling we select the minority examples multiple times to create a balanced dataset

Oversampling with, minority class/majority class = 0.6 and sampleSizePercent=130.
Now the number of instances now is as follows-

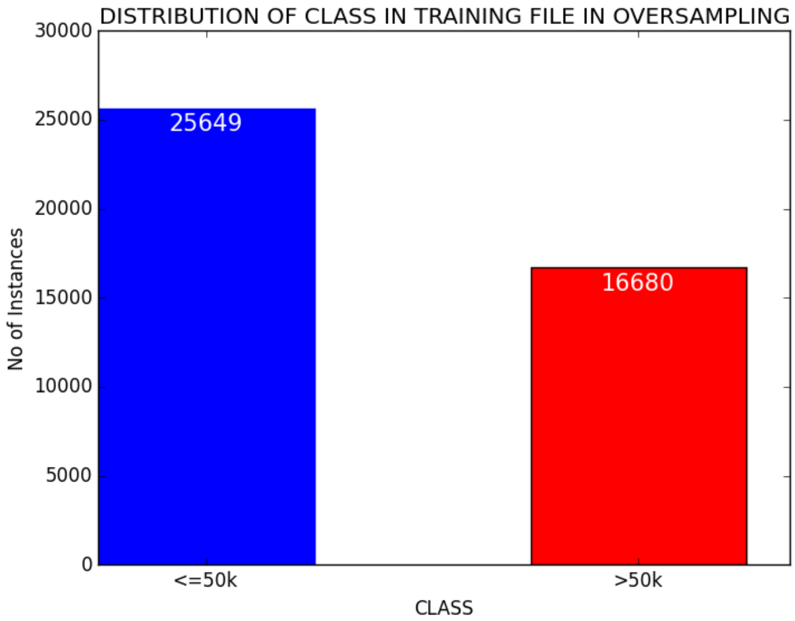

FIGURE 3 Number of Instances after Oversampling

Lets run different algorithms and check the accuracy.

Methods	Replace Missing Values(Mean & Mode)			Replace Missing Values and Oversample		
	Correct	Incorrect	Confusion Matrix	Correctly	Incorrectly	Confusion Matrix
kNN with k=5	13417 82.4089 %	2864 17.5911 %	11237 1198 | a = <=50K **1666** 2180 | b = >50K	12783 78.5148 %	3498 21.4852 %	10178 2257 | a = <=50K **1241** 2605 | b = >50K
J48	13987 85.91 %	2294 14.09 %	11611 824 | a = <=50K **1470** 2376 | b = >50K	13397 82.2861 %	2884 17.7139 %	10672 1763 | a = <=50K **1121** 2725 | b = >50K
Naive Bayes	13515 83.0109 %	2766 16.9891 %	11582 853 | a = <=50K **1913** 1933 | b = >50K	13579 83.404 %	2702 16.596 %	11410 1025 | a = <=50K **1677** 2169 | b = >50K
NBTree	14004 86.0144 %	2277 13.9856 %	11545 890 | a = <=50K **1387** 2459 | b = >50K	13419 82.4212 %	2862 17.5788 %	10152 2283 | a = <=50K **579** 3267 | b = >50K
Logistic	13848 85.0562 %	2433 14.9438 %	11592 843 | a = <=50K **1590** 2256 | b = >50K	13547 83.2074 %	2734 16.7926 %	10641 1794 | a = <=50K **940** 2906 | b = >50K

Table 2 Performance of classification algorithms with respect to Oversampling

Observation:

The number of correctly classified instances for minority class increased, whereas the number of correctly classified instances for the majority class decreased.

Analysis:

From above we can conclude that just by duplicating the instances of minority class we are compromising the predictions of the majority class. If the goal is to improve the accuracy for predicting the minority class then oversampling will be helpful. In general oversampling results in overfitting of the data.

4.2.3 Undersampling

In undersampling we randomly select a subset of the majority examples to match the number of minority examples to create a balanced dataset

Undersampling the data by 48% bias to minority class, gives us the following results.

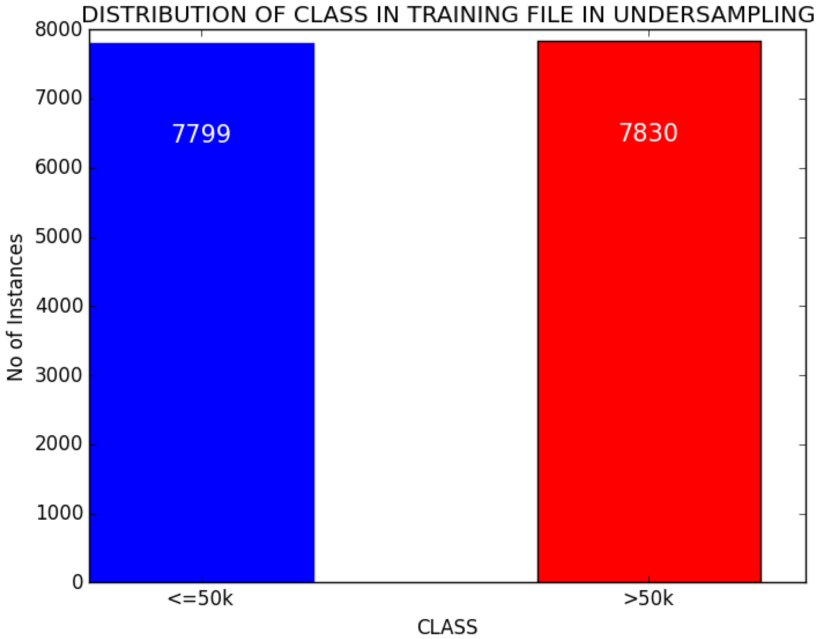

FIGURE 4 Number of Instances after Undersampling

Lets run different algorithms and check the accuracy. Table3 shows the performance of different algorithms.

Methods	Replace Missing Values(Mean & Mode)			Replace Missing Values and UnderSample		
	Correct	Incorrectly	Confusion Matrix	Correctly	Incorrectly	Confusion Matrix
kNN with k=5	13417 82.4089 %	2864 17.5911 %	11237 1198 | a = <=50K **1666** 2180 | b = >50K	12451 76.4756 %	3830 23.5244 %	9490 2945 | a = <=50K **885** 2961 | b = >50K
J48	13987 85.91 %	2294 14.09 %	11611 824 | a = <=50K **1470** 2376 | b = >50K	13004 79.8722 %	3277 20.1278 %	9887 2548 | a = <=50K **729** 3117 | b = >50K
Naive Bayes	13515 83.0109 %	2766 16.9891 %	11582 853 | a = <=50K **1913** 1933 | b = >50K	13557 83.2688 %	2724 16.7312 %	11294 1141| a = <=50K **1583** 2263 | b = >50K
NBTree	14004 86.0144 %	2277 13.9856 %	11545 890 | a = <=50K **1387** 2459 | b = >50K	13108 80.511 %	3173 19.489 %	9735 2700 | a = <=50K **473** 3373 | b = >50K
Logistic	13848 85.0562 %	2433 14.9438 %	11592 843 | a = <=50K **1590** 2256 | b = >50K	13115 80.554 %	3166 19.446 %	9883 2552 | a = <=50K **614** 3232 | b = >50K

Table 3 Performance of classification algorithms with respect to undersampling

Observation:

The number of correctly classified instances for minority class increased, whereas the number of correctly classified instances for the majority class decreased. And overall accuracy decreased.

Analysis:

From above we can conclude that by removing the instances of majority class although we are able to improve the accuracy for predicting the minority class, we are compromising the predictions of the majority class by potentially removing certain important instances thus loosing the information and thus the overall accuracy is also decreasing.

4.2.4. SMOTE(Synthetic Minority Oversampling TEchnique)

SMOTE combines directed oversampling of the minority class with random undersampling of the majority class. It uses nearest neighbor approach to generate synthetic samples of the minority class.

To select k we use the following formula :
Assume the ratio r = (# majority examples /# of minority examples) = 24720/7841 = 3.15
Choose k such that $k \geq r - 1$, we choose k=5

Evaluating the results using SMOTE by - “Replace missing values and SMOTE = 130% k = 5”

4.2.4.1 Replace missing values and SMOTE

With SMOTE percentage= 130% k = 5

Number of Train Instances after SMOTE : 42754
Number of Instances Test Instances is same as before: 16281

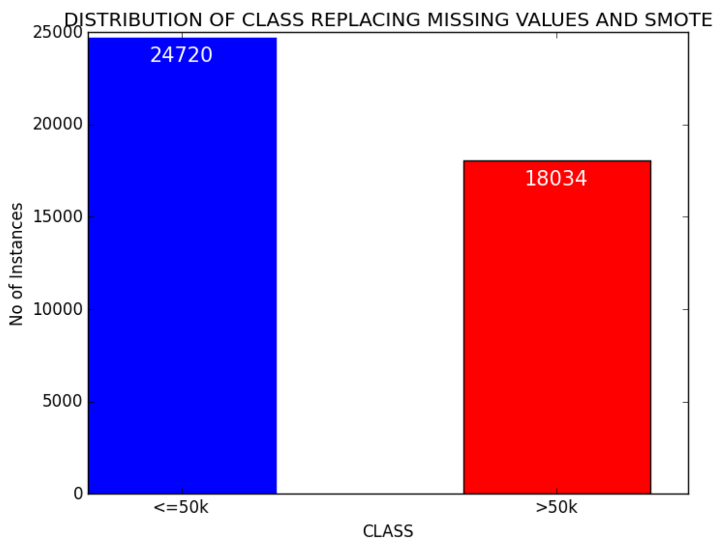

FIGURE 5 Number of Instances after using SMOTE with 130%

With SMOTE percentage= 65% k = 5

Number of Train Instances after SMOTE : 37657

Number of Instances Test Instances is same as before: 16281

Lets run methods and check the accuracy:

Methods	Replace Missing Values(Mean & Mode)			Replace Missing Values and SMOTE with percentage =65% k=5			Replace Missing Values and SMOTE with percentage =130% k=5		
	Correctly	Incorrectly	Confusion	Correctly	Incorrectly	Confusion	Correctly	Incorrectly	Confusion
kNN with k=5	13417 82.4089 %	2864 17.5911 %	11237 1198 | a = <=50K **1666** 2180 | b = >50K	13387 82.2247 %	2894 17.7753 %	11059 1376 **1518** 2328	13355 82.0281 %	2926 17.9719 %	10927 1508 **1418** 2428
J48	13987 85.91 %	2294 14.09 %	11611 824 | a = <=50K **1470** 2376 | b = >50K	13877 85.2343 %	2404 14.7657 %	11418 1017 **1387** 2459	13854 85.0931 %	2427 14.9069 %	11368 1067 **1360** 2486
Naive Bayes	13515 83.0109 %	2766 16.9891 %	11582 853 | a = <=50K **1913** 1933 | b = >50K	13601 83.5391 %	2680 16.4609 %	11499 936 **1744** 2102	13632 83.7295 %	2649 16.2705 %	11432 1003 **1646** 2200
NBTree	14004 86.0144 %	2277 13.9856 %	11545 890 | a = <=50K **1387** 2459 | b = >50K	13991 85.9345 %	2290 14.0655 %	11477 958 **1332** 2514	13801 84.7675 %	2480 15.2325 %	11268 1167 **1313** 2533
Logistic	13848 85.0562 %	2433 14.9438 %	11592 843 | a = <=50K **1590** 2256 | b = >50K	13721 84.2762 %	2560 15.7238 %	11121 1314 **1246** 2600	13546 83.2013 %	2735 16.7987 %	10754 1681 **1054** 2792

Table 4 Performance of classification algorithms after applying SMOTE

Observation:

As we see above with SMOTE our accuracy is slightly decreased but again ratio of minority class(>50K) getting classified correctly has improved. And overall accuracy decreased. But 65% increase in minority samples slightly gave a better prediction accuracy when compared to 130% increase in minority samples.

Analysis:

SMOTE is predicting the minority class better because the instances of the minority class has increased. The new instances that were added are not just the duplicates of already existing instances, but are more close to real time data and thus we are able to achieve better accuracy than oversampling, and undersampling. And also noise might have been generated from the new synthetic samples which could have lead to overall decrease in accuracy.

4.3 Attribute Selection

The selection of attributes is critically important in building a good model. Not All Attributes Are Equal, the consequences of irrelevant attributes are explained below:

Misleading

Including redundant attributes can be misleading to modeling algorithms. Instance-based methods such as k-nearest neighbor use small neighborhoods in the attribute space to determine classification and regression predictions. These predictions can be greatly skewed by redundant attributes.

Overfitting

Keeping irrelevant attributes in dataset can result in overfitting. Decision tree algorithms like C4.5 seek to make optimal splits in attribute values. Those attributes that are more correlated with the prediction are split on first. Deeper in the tree less relevant and irrelevant attributes are used to make prediction decisions that may only be beneficial by chance in the training dataset. This overfitting of the training data can negatively affect the modeling power of the method and cripple the predictive accuracy.

It is important to remove redundant and irrelevant attributes from the dataset before evaluating algorithms.

4.3.1 Selecting features/attributes

Feature Selection or attribute selection is a process by which we automatically search for the best subset of attributes in our dataset.

Three key benefits of performing feature selection on data are:

1. Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.
2. Improves Accuracy: Less misleading data means modeling accuracy improves.
3. Reduces Training Time: Less data means that algorithms train faster.

We are experimenting with Filtering method and Wrapper methods to evaluate attributes using following evaluation criteria and search methods

Filter methods used:

a. **CfsSubsetEval as evaluation criteria with Greedy Stepwise search method.**

CfsSubsetEval: The Correlation Feature Selection (CFS) measure evaluates subsets of features on the basis of the following hypothesis: "Good feature subsets contain features highly correlated with the class value, yet uncorrelated to each other".

GreedyStepWise: Uses a forward (additive) or backward (subtractive) step-wise strategy to navigate attribute subsets.

Results obtained are as follows for CfsSubsetEval and Greedy Stepwise (forwards).

Start set: no attributes
Merit of best subset found: 0.211

Attribute Subset Evaluator (supervised, Class (nominal): 15 Outcome):
Selected attributes: 5,6,8,11,12 : 5
education-num
marital-status
relationship
capital-gain
capital-loss

b. **GainRatioAttributeEval**

Gain Ratio attribute evaluation is a refinement to Information Gain. While Information Gain favors features that have a large number of values, The Gain Ratio approach is to maximize the feature information gain while minimizing the number of its values. The gain ratio of X is thus defined as the information gain of X divided by its intrinsic value:

Gain Ratio (X) = IG(X)/IV (X)

where,

IV(X) = Sum of all (i=1 to r) (|Xt|/N) log(|Xt|/N)

from which |Xi| is the number of instances where attribute X takes the value of Xi; r is the number of distinct values of X; and N is the total number of instances in the dataset

Results obtained using GainRatioAttributeEval with Ranker Search Method are as follows.

Ranked attributes:
0.1876 11 capital-gain
0.1165 12 capital-loss
0.0854 6 marital-status
0.0768 8 relationship

0.0406 10 sex
0.0376   5 education-num
0.0319   4 education
0.0297   1 age
0.0268 13 hours-per-week
0.0248   7 occupation
0.0111   2 workclass
0.0105   9 race
0.0102 14 native-country
0   3 fnlwgt

Wrapper Methods Used:

Assess subsets using a classifier that we specify and n-fold cross validation.

1) WrapperSubsetEval using Naive Bayes and GreedyStepWise(Forward Search)

Number of Iterations :5
Results :
Selected attributes: 1,2,3,4,5,6,7,8,10,11,12,13,14 : 13
age
workclass
fnlwgt
education
education-num
marital-status
occupation
relationship
sex
capital-gain
capital-loss
hours-per-week
native-country

Observations : Attribute 9 race is removed.

2) WrapperSubsetEval using Logistic and GreedyStepWise(Forward Search)

Number of Iterations :5
Results:
Selected attributes: 11,12 : 2
capital-gain
capital-loss

3) WrapperSubsetEval using J48 and GreedyStepWise(Backward Search)

Number of Iterations :5
Results:
Selected attributes: 1,2,4,5,6,7,8,10,11,12,13,14 : 12
age
workclass
education
education-num
marital-status
occupation
relationship
sex
capital-gain
capital-loss
hours-per-week
native-country

Now lets run methods and check the accuracy:

<table>
<tr><th>Methods</th><th>Without Removing any attributes(with Replacing missing values)</th><th>Removing attributes(with Replacing missing values)
CfsSubsetEval & GreedyStepwise
Selected attributes: 5,6,8,11,12 : 5
education-num
marital-status
relationship
capital-gain
capital-loss</th><th>Removing attributes(with Replacing missing values and SMOTE)
CfsSubsetEval & GreedyStepwise
Selected attributes: 1,5,6,8,9,10,11,12,13 : 9
age
education-num
marital-status
relationship
race
sex
capital-gain
capital-loss
hours-per-week</th></tr>
<tr><td>REPTree</td><td>13780 84.6385 %
2501 15.3615 %
11474 961 | a = <=50K
1540 2306 | b = >50K</td><td>13965 85.7748 %
2316 14.2252 %
11792 643 | a = <=50K
1673 2173 | b = >50K</td><td>13735 84.3621 %
2546 15.6379 %
11300 1135 | a = <=50K
1411 2435 | b = >50K</td></tr>
<tr><td>Logistic</td><td>13848 85.0562 %
2433 14.9438 %
11592 843 | a = <=50K
1590 2256 | b = >50K</td><td>13719 84.2639 %
2562 15.7361 %
11507 928 | a = <=50K
1634 2212 | b = >50K</td><td>13305 81.721 %
2976 18.279 %
10400 2035 | a = <=50K
941 2905 | b = >50K</td></tr>
<tr><td>RandomForest</td><td>13766 84.5525 %
2515 15.4475 %
11435 1000 | a = <=50K
1515 2331 | b = >50K</td><td>FilteredSubsetEval With RandomSampling and CfsSubsetEval
RankSearch With GainRatioAttributeEval

Selected attributes: 6,8,11,12 : 4
marital-status
relationship
capital-gain
capital-loss

13562 83.2996 %
2719 16.7004 %

12393 42 | a = <=50K</td><td>FilteredSubsetEval With RandomSampling and CfsSubsetEval
RankSearch With GainRatioAttributeEval

Selected attributes: 1,5,6,8,10,11,12,13 : 8
age
education-num
marital-status
relationship
sex
capital-gain
capital-loss
hours-per-week

13667 83.9445 %</td></tr>
</table>

		2677 1169 | b = >50K	2614 16.0555 % 11317 1118 | a = <=50K 1496 2350 | b = >50K
J48	13987 85.91 % 2294 14.09 % 11611 824 | a = <=50K 1470 2376 | b = >50K	13973 85.824 % 2308 14.176 % 11820 615 | a = <=50K 1693 2153 | b = >50K	13875 85.222 % 2406 14.778 % 11536 899 | a = <=50K 1507 2339 | b = >50K
NBTree	14030 86.1741 % 2251 13.8259 % 11565 870 | a = <=50K 1381 2465 | b = >50K	14004 86.0144 % 2277 13.9856 % 11804 631 | a = <=50K 1646 2200 | b = >50K	13928 85.5476 % 2353 14.4524 % 11802 633 | a = <=50K 1720 2126 | b = >50K
Naive Bayes	13534 83.1276 % 2747 16.8724 % 11575 860 | a = <=50K 1887 1959 | b = >50K	13016 79.9459 % 3265 20.0541 % 11832 603 | a = <=50K 2662 1184 | b = >50K	13489 82.8512 % 2792 17.1488 % 11574 861 | a = <=50K 1931 1915 | b = >50K

Table 5 Accuracy of classification methods after attribute selection

Observation:

We see in methods such as Logistic Regression, accuracy slightly decreased after removing attributes and with SMOTE the accuracy still further decreased. J48 method did not have much effect after removing attributes.

In REPTree with attributes removed on replace missing values we saw an increase in accuracy of 1.1363%.

We saw a very interesting observation after running RandomForest after removing attributes. Though the overall accuracy decreased, the number of <=50K predictions was very close to accurate and >50K prediction performed very badly.

In general with SMOTE we saw a decrease in overall accuracy in all methods.

Analysis:

In general, the overall accuracy after removing the attributes decreased, owing to the reason that the attributes that were removed might have played a significant role in correctly classifying the instances.
We can also conclude that for some algorithms the feature selection works because some algorithms scaled well as the complexity of their respective models was reduced, and a simpler model is simpler to understand and explain.

Also, in general SMOTE didn't increase accuracy because due to the addition of synthetic instances noise was added to data. And also we ran experiments with oversampling and removing attributes (Readings not recorded in report) which lead to decrease in overall accuracy in almost all classification methods.

And also we can infer from above experiments feature selection is not always necessary to achieve good performance.

Use of preprocessing in our final algorithm:

Preprocessing is very important step which helps in improving the prediction accuracy for almost all classification methods. In our final algorithm for training dataset we are imputing the missing values with mean and mode and using SMOTE over oversampling or undersampling because synthetic minority instances generated by SMOTE is more close to real world data, and is thus better than using duplicates. And in SMOTE we choose to increase minority samples by 65% as it gave better results in overall prediction accuracy and also improving minority class predictions, than using 130%. And we did not remove any features explicitly as we did not see a huge improvement in prediction accuracy instead we chose to use algorithms like Random forest (explained in detail below). And for test dataset we are imputing the missing values with mean and mode same as training dataset.

5. Classification Methods

Brief Overview of various Classification methods Experimented:

Naive Bayes:

Naive Bayes methods is a set of supervised learning algorithm based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features.

We ran the Naive Bayes algorithm on the train and test data set, and observed the following values

Methods	Replace Missing Values(Mean & Mode)			Replace Missing Values and SMOTE SMOTE = 65% k=5		
	Correct	**Incorrect**	**Confusion Matrix**	**Correct**	**Incorrect**	**Confusion Matrix**
Naive Bayes	13515 83.0109 %	2766 16.9891 %	11582 853 | a = <=50K 1913 1933 | b = >50K	13601 83.5391 %	2680 16.4609 %	11499 936 | a = <=50K 1744 2102 | b = >50K

Observation:

The Naive Bayes classifier performed better with imputed data. However, the overall accuracy so achieved was still not significant.

Analysis:

Naive Bayes assumes that the features are independent of each other, which might not be the case in reality. This leads to a poor performance on data sets in which features are correlated. Also, Naive Bayes is not sensitive to irrelevant features, which can also be one of the reasons for it not performing well on the given data set.

kNN:

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).
We ran the kNN algorithm, with different k values, which corresponds to the number of neighbors.

K	Correct	Incorrect	Confusion Matrix
3	13281 81.5736 %	3000 18.4264 %	11134 1301 | a = <=50K 1699 2147 | b = >50K
5	13417 82.4089 %	2864 17.5911 %	11237 1198 | a = <=50K 1666 2180 | b = >50K
7	13478 82.7836 %	2803 17.2164 %	11279 1156 | a = <=50K 1647 2199 | b = >50K

9	13522 83.0539 %	2759 16.9461 %	11314 1121 | a = <=50K 1638 2208 | b = >50K
11	13567 83.3303 %	2714 16.6697 %	11335 1100 | a = <=50K 1614 2232 | b = >50K
13	13558 83.275 %	2723 16.725 %	11335 1100 | a = <=50K 1623 2223 | b = >50K
15	13564 83.3118 %	2717 16.6882 %	11341 1094 | a = <=50K 1623 2223 | b = >50K

Table 6 kNN with different k values

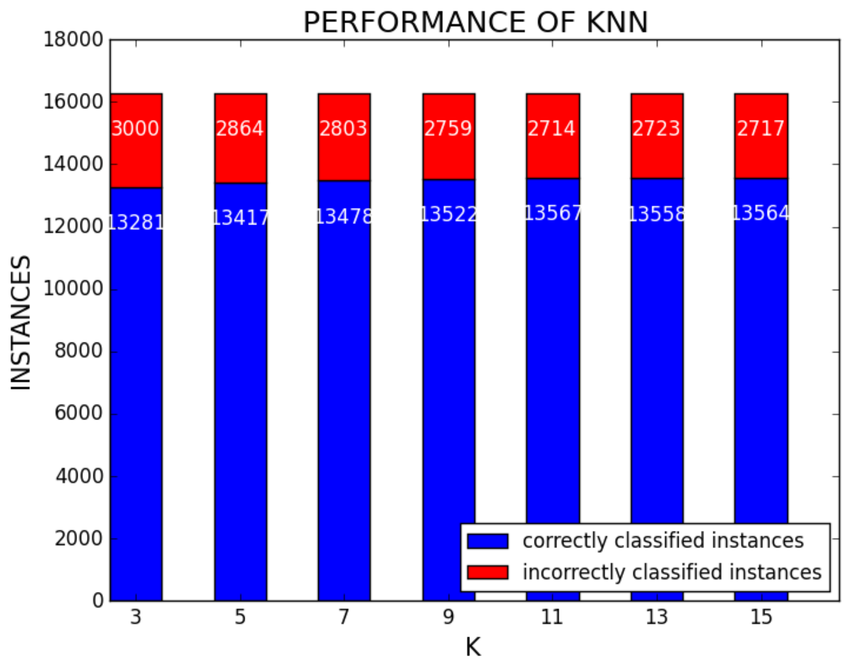

Figure 5 Performance of kNN with different with different k values

Observation:

The different k values didn't contribute much in achieving a better accuracy.

Analysis:

kNN didn't perform well because it just uses a similarity measure to predict the classes. It is a lazy learner, i.e. it does not learn anything from the training data and simply uses the training data itself for classification. Also the train instances needs to be stored, and time taken to get output is more.

SMO(Sequential Minimal Optimization):

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. Sequential minimal optimization (SMO) is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support vector machines.

We ran the SMO algorithm using PolyKernel, with different C values. C is the complexity parameter used.

C	Correct	Incorrect	Confusion Matrix
0.1	13755 84.485 %	2526 15.515 %	11693 742 | a = <=50K 1784 2062 | b = >50K
0.3	13816 84.8597 %	2465 15.1403 %	11684 751 | a = <=50K 1714 2132 | b = >50K
0.5	13855 85.0992 %	2426 14.9008 %	11689 746 | a = <=50K 1680 2166 | b = >50K
0.8	13863 85.1483 %	2418 14.8517 %	11680 755 | a = <=50K 1663 2183 | b = >50K
1.0	13860 85.1299 %	2421 14.8701 %	11678 757 | a = <=50K 1664 2182 | b = >50K

Table 7 SMO with different C values

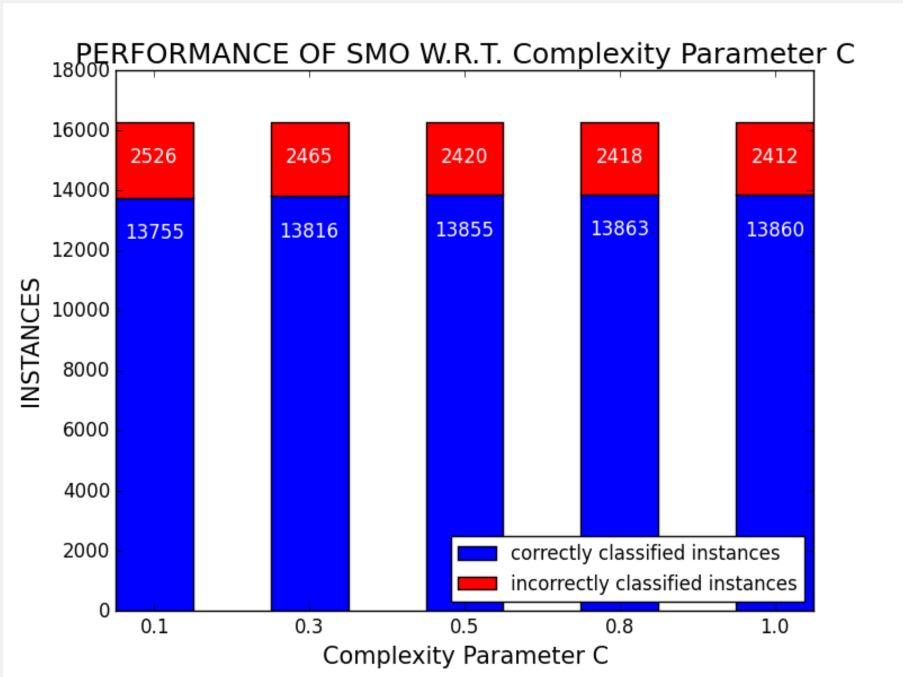

Figure 6 Performance of SMO with different C values

Observation:

From the above table we can observe that by increasing the value of C(complexity parameter), we were able to achieve better accuracy. However the accuracy achieved was lesser than compared to other algorithms, and the time taken to build the model significantly increased with the increase in C values.

Analysis:

The higher is C, the higher is the weight given to in-sample misclassifications, the lower is the generalization of the machine. Low generalisation means that the machine may work well on the training set but would perform miserably on a new sample. Bad generalisation may be a result of overfitting on the training sample, for example, in the case that this sample shows some untypical and non-repeating data structure. By choosing a low C, the risk of overfitting an SVM on the training sample is reduced, however the model so built will be more general and might not properly classify the data.
Also, SVMs have high algorithmic complexity and extensive memory requirements in large-scale tasks. Hence, this is not beneficial for our data set.

Boosting with Decision tree:

Boosting is an ensemble technique. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. Adaptive Boosting is adaptive in the sense that subsequent weak learners are improved in favor of those instances misclassified by previous classifiers. At each iteration of the training process, a weight is assigned to each sample in the training set equal to the current error on that sample. These weights can be used to inform the training of the weak learner. The classic weak learner is a decision tree.

We ran AdaptiveBoosting with J48, with different weight threshold values, and the results are as shown below.

Weight Threshold	**Correct**	**Incorrect**	**Confusion Matrix**
010	13987 85.91 %	2294 14.09 %	11611 824 | a = <=50K 1470 2376 | b = >50K
100	13534 83.1276 %	2747 16.8724 %	11136 1299 | a = <=50K 1448 2398 | b = >50K

Observation:

By decreasing the weight threshold values, we were able to achieve better accuracy. However the accuracy achieved was lesser compared to other algorithms.

Analysis:

Higher weight threshold values resulted in lesser accuracy, because the model so built might have overfitted the training data and thus performed poorly on the testing data. We were not able to achieve a better accuracy with boosting because boosting in general will overfit with many weak learners.

Majority voting

It is an ensemble technique. In Majority Voting, a classification of an unlabeled instance is made according to the class that obtains the highest number of votes

$\sum Tt{=}1dt,J(\mathbf{x})=\max j{=}1,\cdots,C\sum Tt{=}1dt,j$

Under the condition that the classifier outputs are independent, it can be shown the majority voting combination will always lead to a performance improvement. If there are a total of T classifiers for a two-class problem, the ensemble decision will be correct if at least $\lfloor T/2+1 \rfloor$ classifiers choose the correct class.

Classifiers used for Majority Voting:

1)Logistic Regression

It models the relationship between a dependent and one or more independent variables, and allows us to look at the fit of the model as well as at the significance of the relationships (between dependent and independent variables) that we are modelling. Logistic regression models P(y|x) directly without assuming any particular distribution of P(x|y).

Parameters set for running Logistic regression in weka:

Logistic -R 1.0E-8 -M -1 where,
-R Sets the ridge in the log-likelihood.
-M Sets the maximum number of iterations (default -1, until convergence)

The following table depicts how logistic regression performed.

Methods	**Replace Missing Values(Mean & Mode)**			**Replace Missing Values and SMOTE** **SMOTE = 65% k=5**		
	Correctly	**Incorrectly**	**Confusion**	**Correctly**	**Incorrectly**	**Confusion**
Logistic	13848 85.0562 %	2433 14.9438 %	11592 843 | a = <=50K 1590 2256 | b = >50K	13721 84.2762 %	2560 15.7238 %	11121 1314 | a = <=50K 1246 2600 | b = >50K

Observation:

The overall accuracy achieved by logistic is **85.05%** and was slightly lower when SMOTE was used.

Analysis:

The decrease in accuracy owes to the fact that in SMOTE the new instances added are synthetic and not actual data. However the overall accuracy achieved using SMOTE and Logistic regression was better than other algorithms and also Logistic regression with replace missing values is giving good accuracy, and so we chose logistic regression as one of the classifiers in Majority Voting. The reason for Logistic regression performing better is because it takes in consideration the relationship between features.

2) J48 (Decision Tree)

Generates a pruned or unpruned C4.5 decision tree. C4.5 builds decision trees from a set of training data, using the concept of information entropy. The training data is a set S = s1, s2, s3,...., si of already classified samples. Each sample si consists of a p-dimensional vector (x1i, x2i,...., xpi), , where the xj represent attributes or features of the sample, as well as the class in which si 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. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sublists.

Parameters set for running J48 in weka:

J48 -C 0.1 -M 2

-C <pruning confidence> Sets confidence threshold for pruning.(default 0.25) (The confidence is used to compute a pessimistic upper bound on the error rate at a leaf/node.)

- M <minimum number of instances> Sets minimum number of instances per leaf. (default 2) (minimum instances per leaf guarantees that at each split, at least 2 of the branches)

Pruning confidence is set to 0.1

The following table depicts how J48 performed.

Methods	**Replace Missing Values(Mean & Mode)**			**Replace Missing Values and SMOTE** **SMOTE = 65% k=5**		
	Correctly	**Incorrectly**	**Confusion**	**Correctly**	**Incorrectly**	**Confusion**
J48	13987 85.91 %	2294 14.09 %	11611 824 | a = <=50K 1470 2376 | b = >50K	13877 85.2343 %	2404 14.7657 %	11418 1017 | a = <=50K 1387 2459 | b = >50K

Observation:

The overall accuracy achieved by J48 is **85.91%** and was slightly lower when SMOTE was used. However the overall accuracy achieved by J48 in general was good.

Analysis:

The decrease in accuracy owes to the fact that in SMOTE the new instances added are synthetic and not actual data.
The accuracy achieved by J48 was good because it uses decision trees which in turn uses entropy values(information gain) to choose the best attribute to make the decision. Information gain is usually a

good measure for deciding the relevance of an attribute.In decesion trees nonlinear relationships between parameters do not affect tree performance and its easy to build and use. A decision tree also allows for partitioning data in a much deeper level, not as easily achieved with other decision-making classifiers such as logistic regression or support of vector machines. Hence we chose J48 as one of the classifiers in Majority Voting.

3) NBTree:

The NBTree algorithm is a hybrid between C4.5 Decision Tree classifiers and Naive Bayes classifiers. The NBTree algorithm is written below with input of T sets of labeled instances and a decision-tree with Naive Bayes category at the output (leaves):

1. For each attribute Xi, evaluate the utility, u(Xi), of a split on attribute Xi. For continuous attributes, a threshold is also evaluated at this stage.
2. Let J = AttMax(Ui). The attribute with highest utility (Maximum utility).
3. If Uj is not significantly better than the utility of the current node, create a Naïve Bayes classifier for the current node and return.
4. Partition T according to the test on Xj. If Xj is continuous, a threshold split is used; if Xj is discrete, a multi-way split is made for all possible values.
5. For each child, call the algorithm recursively on the portion of T that matches the test leading to the child.

The following table depicts how NBTree performed.

Methods	**Replace Missing Values(Mean & Mode)**			**Replace Missing Values and SMOTE** **SMOTE = 65% k=5**		
	Correctly	**Incorrectly**	**Confusion**	**Correctly**	**Incorrectly**	**Confusion**
NBTree	14004 86.0144 %	2277 13.9856 %	11545 890 | a = <=50K 1387 2459 | b = >50K	13991 85.9345 %	2290 14.0655 %	11477 958 | a = <=50K 1332 2514 | b = >50K

Observation:

The overall accuracy achieved by NBTree was **86.0144%** and very slightly lower when SMOTE was used. However the overall accuracy achieved by NBTree in general was good.

Analysis:

For discrete valued attributes, the Naive Bayes method performs quite well. With the increase in data size, the performance also improves. But in case of continuous valued attributes, Naive Bayes method does not take into account the attribute interactions. Whereas, the decision trees do not give good performance when the data size is very large. These shortcomings are overcome by the NBTree algorithm. Hence we chose NBTree as one of the classifiers in Majority Voting.

4) RandomForest :

Random Forests are an ensemble learning method for classification and regression that construct a number of decision trees at training time and outputting the class that is the mode of the classes output by individual trees.

Random Forests are a combination of tree predictors where each tree depends on the values of a random vector sampled independently with the same distribution for all trees in the forest. The basic principle is that a group of “weak learners” can come together to form a “strong learner”.

The Random Forests algorithm was developed by Leo Breiman and Adele Cutler. Random Forests grows many classification trees. Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.

2. If there are M input variables, a number mM is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.

3. Each tree is grown to the largest extent possible. There is no pruning.

FastRandomForest is a re-implementation of the Random Forest classifier (RF) for the Weka environment that brings speed and memory use improvements over the original Weka RF.

Parameters set for running Random Forest in weka:

RandomForest -I 100 -K 0 -S 1 where
-I <number of trees> Sets Number of trees to build. (default 100)
-K <number of features> Sets Number of features to consider (<1=int(log_2(#predictors)+1)). default (0)
-S Sets Seed for random number generator. (default 1)

No. of variables randomly selected at node and used to find best split(s) = 4

The following table depicts how RandomForest performed.

Methods	**Replace Missing Values(Mean & Mode)**			**Replace Missing Values and SMOTE SMOTE = 65% k=5**		
	Correctly	**Incorrectly**	**Confusion**	**Correctly**	**Incorrectly**	**Confusion**
Random Forest	13766 84.5525 %	2515 15.4475 %	11435 1000 | a = <=50K 1515 2331 | b = >50K	13714 84.2332 %	2567 15.7668 %	11314 1121 | a = <=50K 1446 2400 | b = >50K

Observation:

The overall accuracy achieved by Random Forest is **84.55%** and slightly lower when SMOTE was used. However the overall accuracy achieved by RandomForest in general was good.

Analysis:

Random Forests are a wonderful tool for making predictions considering they do not overfit because of the law of large numbers. Introducing the right kind of randomness makes them accurate classifiers and regressors.Single decision trees often have high variance or high bias. Random Forests attempts to mitigate the problems of high variance and high bias by averaging to find a natural balance between the two extremes. And also produces highly accurate classifier and learning is fast. Thus the accuracy achieved by using RamdomForest in general was good. So we chose RandomForest as one of the classifiers in Majority Voting.

5) Bagging with J48(Decision Tree)

Bagging stands for Bootstrap Aggregation. It is a type of ensemble learning. The algorithm used for Bagging is as follows

(i)Decide the number of bags, , e.g. n = 3, 5, or 10. We take odd number of bags so that we don't face a tie situation during majority voting.
(ii)For each bag, create a classifier by combining the minority examples and a random sample of same number of majority examples
(iii) Decide the class label by taking a majority vote among the n classifiers
(iv) Create ensembles by "bootstrap aggregation", i.e., repeatedly randomly re-sampling training data

Parameters set for running Bagging in weka:

Bagging -P 25 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
-P sets Size of each bag, as a percentage of the training set size.
-S sets Random number seed.(default 1)
-I sets Number of iterations.(default 10)
-W sets base classifier
Pruning confidence is set to 0.25 while running J48.

The following table depicts how Bagging with J48 performed with different bag size percentages.

Bag Size Percentage	Correct	Incorrect	Confusion Matrix
100	13872 85.2036 %	2409 14.7964 %	11547 888 | a = <=50K 1521 2325 | b = >50K
50	14009 86.0451 %	2272 13.9549 %	11664 771 | a = <=50K 1501 2345 | b = >50K

75	13977 85.8485 %	2304 14.1515 %	11619 816 | a = <=50K 1488 2358 | b = >50K
25	14040 86.2355 %	2241 13.7645 %	11726 709 | a = <=50K 1532 2314 | b = >50K
20	13991 85.9345 %	2290 14.0655 %	11718 717 | a = <=50K 1573 2273 | b = >50K

Table 8 Bagging with different bag sizes

Observation:

In general, the performance of Bagging with J48 was good. The best accuracy **86.23%** was achieved when the bag size was 25. And with bag size 25 and SMOTE we achieved an accuracy of 85.32%.

Analysis:

Bagging reduces the model variance. Decision trees are an ideal choice for bagging because they have low bias and high variance when they grow sufficiently deep. The majority vote of the classifiers built during bagging samples perform better than the model constructed on the entire data set because of the reduction in the variance. The bag size percentage 25 means the size of each bag is 25% of the training data. With Bagging we were able to achieve a very good accuracy, so we chose Bagging with J48 as one of the classifiers in Majority Voting.

6. Final Implemented Algorithm

We have implemented our final algorithm as follows:

1) Replace Missing Values with mean mode imputation followed by SMOTE(65%) as preprocessing step and running Majority Vote with base learners as Logistic, J48, NBTree, RandomForest, Bagging(with J48)

Individual accuracy achieved in all the 5 algorithms with using replace Missing Values with mean mode imputation followed by SMOTE(65%) as preprocessing step:

1) Logistic Regression: **84.2762 %**
2) J48(Decision Tree): **85.2343 %**
3) NBTree: **85.9345 %**
4) Random Forest: **84.2332 %**
5) Bagging with J48(Decision Tree) (Bag Size: 25): **85.32%**

The results are shown below.

Some other implementations that we have included for reference and result comparison with the standard output

1) Replace Missing Values with mean mode imputation and running Majority Vote with base learners as Logistic, J48, NBTree, RandomForest, Bagging(with J48)
2) Remove Missing Values as preprocessing step and running Majority Vote with base learners as Logistic, J48, NBTree, RandomForest, Bagging(with J48)

Lets run majority voting ensemble technique combined with different preprocessing steps:

Ensemble method	**Preprocessing**	**Classifiers used(As Base Learners)**	**Results**
Majority Vote	Replace missing values with mean and mode imputation. Number of Training Instances: 32561 Number of Test Instances: 16281	weka.classifiers.trees.NBTree weka.classifiers.trees.J48 -C 0.1 -M 2 weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 weka.classifiers.trees.RandomForest -I 100 -K 0 -S 1 weka.classifiers.meta.Bagging -P 25 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2	Correctly Classified Instances 14075 86.4505 % Incorrectly Classified Instances 2206 13.5495 % a b <=-- classified as 11698 737 |a = <=50K 1469 2377 |b = >50K
Majority Vote (Our final Algorithm)	**Replace Missing values , SMOTE with 65% increase and k=5** **Number of Training Instances: 37657** **Number of Test Instances: 16281**	**weka.classifiers.trees.NBTree** **weka.classifiers.trees.J48 -C 0.1 -M 2** **weka.classifiers.functions.Logistic -R 1.0E-8 -M -1** **weka.classifiers.trees.RandomForest -I 100 -K 0 -S 1** **weka.classifiers.meta.Bagging -P 25 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2**	**Correctly Classified Instances 13982 85.8792 %** **Incorrectly Classified Instances 2299 14.1208 %** **a b <=-- classified as** **11495 940 |a = <=50K** **1356 2490 |b = >50K**
Majority Vote	Replace Missing values , SMOTE with 130% increase and k=5 Number of Training Instances: 42754	weka.classifiers.trees.NBTree weka.classifiers.trees.J48 -C 0.1 -M 2 weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 weka.classifiers.trees.RandomForest -I 100 -K 0 -S 1 weka.classifiers.meta.Bagging -P 25 -S 1 -I 10	Correctly Classified Instances 13903 85.394 % Incorrectly Classified Instances 2378 14.606 % a b <=-- classified as

	Number of Test Instances: 16281	-W weka.classifiers.trees.J48 -- -C 0.25 -M 2	11341 1094 |a = <=50K 1284 2562 |b = >50K
Majority Vote	Removing Missing values completely from train and test Number of Training Instances: 30162 Number of Test Instances: 15060	weka.classifiers.trees.NBTree weka.classifiers.trees.J48 -C 0.1 -M 2 weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 weka.classifiers.trees.RandomForest -I 100 -K 0 -S 1 weka.classifiers.meta.Bagging -P 25 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2	Correctly Classified Instances 12922 85.8035 % Incorrectly Classified Instances 2138 14.1965 % a b <=-- classified as 10624 736 |a = <=50K 402 2298 |b = >50K

Table 6 Accuracy of Majority Voting ensemble method

Final Observation:

As we see from the table above the best prediction accuracy achieved was **85.8977%(error:14.1023%)** using Replace Missing Values with mean mode imputation followed by SMOTE(65% increase in minority class(> 50K)) as preprocessing step and running Majority Vote with base learners as Logistic, J48, NBTree, RandomForest, Bagging(with J48). Our code also outputs the same when run with choice 2 as an option..

If our goal is to improve prediction of both classes in the dataset then we need to balance the data.

Final Analysis:

We saw a significant improvement prediction accuracy by using Majority Voting when compared to individual classification methods. This contributes to the fact that when combining multiple independent decisions, each of which is at least more accurate than random guessing, random errors cancel each other out and correct decisions are reinforced. Restating the same fact again, the main advantage of using Majority vote is of different classifiers is that it is unlikely that all classifiers will make the same mistake. In fact, as long as every error is made by a minority of the classifiers, you will achieve optimal classification!

Code Output of our final Implementation:

With Replace Missing Values and SMOTE percentage=65% k=5 as preprocessing step and running Majority Vote we get accuracy as follows:

Correctly Classified Instances 13985 85.8977 %
Incorrectly Classified Instances 2296 14.1023 %

The output from code:

```
=== Starting to Run Majority Voting with Replacing Missing Values(Mean & Mode Imputation) then SMOTE percentage=65% ===
=== Number of Training Instances ===
37657

=== Classifier model (full training set) ===

Vote combines the probability distributions of these base learners:
        weka.classifiers.trees.NBTree
        weka.classifiers.trees.J48 -C 0.1 -M 2
        weka.classifiers.functions.Logistic -R 1.0E-8 -M -1
        weka.classifiers.trees.RandomForest -I 100 -K 0 -S 1
        weka.classifiers.meta.Bagging -P 25 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
using the 'Majority Voting' combination rule
```

=== Building Model ===
=== Time taken to Build Classifiers & Test ===

107 seconds

Results
===============

Correctly Classified Instances 13985 85.8977 %
Incorrectly Classified Instances 2296 14.1023 %
Kappa statistic 0.594
K&B Relative Info Score 908879.8951 %
K&B Information Score 7168.9086 bits 0.4403 bits/instance
Class complexity | order 0 12840.958 bits 0.7887 bits/instance
Class complexity | scheme 2465904 bits 151.459 bits/instance
Complexity improvement (Sf) -2453063.042 bits -150.6703 bits/instance
Mean absolute error 0.141
Root mean squared error 0.3755
Relative absolute error 39.0793 %
Root relative squared error 88.4095 %
Total Number of Instances 16281

For <50K F-Measure :0.9091987661156371 Precision :0.8944829196171504 Recall :0.9244069159630076
For >50K F-Measure :0.6844420010995051 Precision :0.7259475218658892 Recall :0.6474258970358814
=== Confusion Matrix ===

a	b	
11495	940	|a = <=50K
1356	2490	|b = >50K

Some other outputs of our implemented algorithm for your reference and result comparison with the standard output:

With Replace Missing Values as preprocessing step and running Majority Vote we get accuracy as follows:
Correctly Classified Instances 14075 86.4505 %
Incorrectly Classified Instances 2206 13.5495 %

The output from code:

=== Starting to Run Majority Voting with Replacing Missing Values(Mean & Mode Imputation) ===
=== Number of Training Instances ===
32561

=== Classifier model (full training set) ===

Vote combines the probability distributions of these base learners:
weka.classifiers.trees.NBTree
weka.classifiers.trees.J48 -C 0.1 -M 2
weka.classifiers.functions.Logistic -R 1.0E-8 -M -1
weka.classifiers.trees.RandomForest -I 100 -K 0 -S 1
weka.classifiers.meta.Bagging -P 25 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
using the 'Majority Voting' combination rule

=== Building Model ===
=== Time taken to Build Classifiers & Test ===

74 seconds

Results
===============

Correctly Classified Instances 14075 86.4505 %
Incorrectly Classified Instances 2206 13.5495 %
Kappa statistic 0.5981
K&B Relative Info Score 937067.8836 %
K&B Information Score 7391.245 bits 0.454 bits/instance
Class complexity | order 0 12840.958 bits 0.7887 bits/instance
Class complexity | scheme 2369244 bits 145.522 bits/instance
Complexity improvement (Sf) -2356403.042 bits -144.7333 bits/instance
Mean absolute error 0.1355
Root mean squared error 0.3681
Relative absolute error 37.5475 %
Root relative squared error 86.6594 %
Total Number of Instances 16281

For <50K F-Measure :0.9138348566518241 Precision :0.8884332042226779 Recall :0.9407318053880177
For >50K F-Measure :0.6830459770114943 Precision :0.7633269107257546 Recall :0.6180447217888716
=== Confusion Matrix ===

a	b	
11698	737	|a = <=50K
1469	2377	|b = >50K

With Remove Missing Values as preprocessing step and running Majority Vote we get accuracy as follows:

Correctly Classified Instances 12922 85.8035 %
Incorrectly Classified Instances 2138 14.1965 %

The output from code:

=== Starting to Run Majority Voting with Removing Missing Values ===
=== Number of Training Instances ===
30162

=== Classifier model (full training set) ===

Vote combines the probability distributions of these base learners:
weka.classifiers.trees.NBTree
weka.classifiers.trees.J48 -C 0.1 -M 2
weka.classifiers.functions.Logistic -R 1.0E-8 -M -1
weka.classifiers.trees.RandomForest -I 100 -K 0 -S 1
weka.classifiers.meta.Bagging -P 25 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
using the 'Majority Voting' combination rule

=== Building Model ===
=== Time taken to Build Classifiers & Test ===

39 seconds

Results
===============

Correctly Classified Instances 12922 85.8035 %
Incorrectly Classified Instances 2138 14.1965 %
Kappa statistic 0.5922
K&B Relative Info Score 859583.4598 %
K&B Information Score 6914.6614 bits 0.4591 bits/instance

```
Class complexity | order 0          12113.7426 bits      0.8044 bits/instance
Class complexity | scheme          2296212      bits    152.4709 bits/instance
Complexity improvement     (Sf)    -2284098.2574 bits   -151.6666 bits/instance
Mean absolute error                  0.142
Root mean squared error               0.3768
Relative absolute error             38.3003 %
Root relative squared error           87.5238 %
Total Number of Instances           15060

For <50K F-Measure :0.9085777815787224 Precision :0.8834192582737402 Recall :0.9352112676056338
For >50K F-Measure :0.6825066825066826 Precision :0.7574159525379037 Recall :0.6210810810810811
=== Confusion Matrix ===
a        b
10624    736      |a =  <=50K
1402     2298     |b =  >50K
```

7. Conclusion

- ❏ Missing value imputation is a better alternative than removing instances with missing values as there is no loss of valuable data, which might be useful in class prediction.

- ❏ Balancing is essential to avoid the classifiers to be biased towards majority class.

- ❏ No single algorithm wins all the time. Ensemble methods when used enforces correct decisions by cancelling out random errors.