USE CASE STUDY REPORT

Group No : Group 2

Student Names: Snehal Dahiphale, Sai Raghuram Kothapalli

Topic: Income Prediction

Executive Summary

In this case study, the aim is to build a predictive model to determine income levels (below or above 50k) for people in the US. We are dealing with a binary classification problem for predicting a specific outcome that can only take two distinct values for example 0 and 1 or positive and negative etc using imbalance data.

We obtained the data UCI Machine learning repository here which consists of total 299,285 rows and 41 columns.
The key independent variables of the dataset are -
Age, Marital Status, Income, Family Members, No. of Dependents, Tax Paid, Investment (Mutual Fund, Stock), Return from Investments, Education, Spouse, Education, Nationality, Occupation, Region in US, Race, Occupation category.
The dependent variable is *Income* and we have to predict the income range, whether the income will be less or greater than 50,000 dollars.

We used 3 sampling methods to overcome the **imbalance problem**:

1. Undersampling- by adding more of the minority class so it has more effect on the machine learning algorithm.
2. Oversampling-by removing some of the majority class so it has less effect on the machine learning algorithm.
3. SMOTE-checks n nearest neighbors, measures the distance between them and introduces a new observation at the center of n observations.

Then we apply the following machine learning algorithms on all 3 datasets obtained from different sampling techniques to study the effect and solve **binary classification problem**:

1. Naive Bayes Classifier
2. XGboost

In terms of sampling the imbalanced data, SMOTE dominates undersampling and oversampling because of loss of information and overestimation of minority class respectively. For binary classification based on “income_level”, Naive Bayes results in 79% accuracy whereas XGBoost results in 94.8% accuracy.

I. Background and Introduction

- **The problem:** Machine learning algorithms work best when the number of instances for both classes are equal. The problem arises when the number of instances of one class far exceeds the other one. In this case study, we are dealing with a binary classification problem for imbalanced dataset where 90% data is skewed towards either left or right which makes it difficult for us to derive any conclusions from it. Standard classification algorithms have a bias towards classes which have more number of instances therefore, they tend to predict the majority class data and ignore the minority class features as noise. So, there are high chances of minority class data being misclassified if the accuracy parameter of the confusion matrix is taken into account. For example, if classifier achieves 98% accuracy is not correct if it classifies instances as majority class and eliminates 2% minority class as error.

- **The goal of your study:** We know that the conventional model evaluation techniques will not work accurately when faced with imbalanced datasets. Our aim is to perform sampling methods to balance the classes for either increasing or decreasing the frequency of minority class to obtain same number of instances in both classes and perform machine learning algorithms like Naive Bayes, XGboost and Support Vector Machine.

- **The possible solution:** For data preparation step, first we try different sampling techniques such as oversampling, undersampling and SMOTE techniques to handle the imbalance in data. In oversampling, we add more of minority class to have more effect on the algorithm and in undersampling, we remove some of majority class to have to have less effect on the algorithm implemented while SMOTE looks at n nearest neighbors, measures the distance between them and introduces a new observation at the center of n observations. However, undersampling may lead to loss of important information, oversampling may cause overestimation of minority class and SMOTE uses a hybrid/combined approach which works the best.
  Then we apply machine learning algorithms-
  1. Naives Bayes smoothing techniques which assigns non-zero values to variables that are not present.
  2. XGboost which is an advanced implementation of gradient boosting where the model minimizes the loss function.

II. Data Exploration and Visualization

Data Preparation

We split the data in train and test sets with the following dimensions:

```
> dim(train); str (train); View(train)
[1] 199523     41
```

```
> dim(test); str (test); View(test)
[1] 99762    41
```

The train data has 199523 rows & 41 columns and test data has 99762 rows and 41 columns.The denominations of our target variable is not same which may cause disparity so, we encode the levels as 0 and 1 for <50k and >50k income_level respectively.

```
> round(prop.table(table(train$income_level))*100)

 0  1
94  6
```

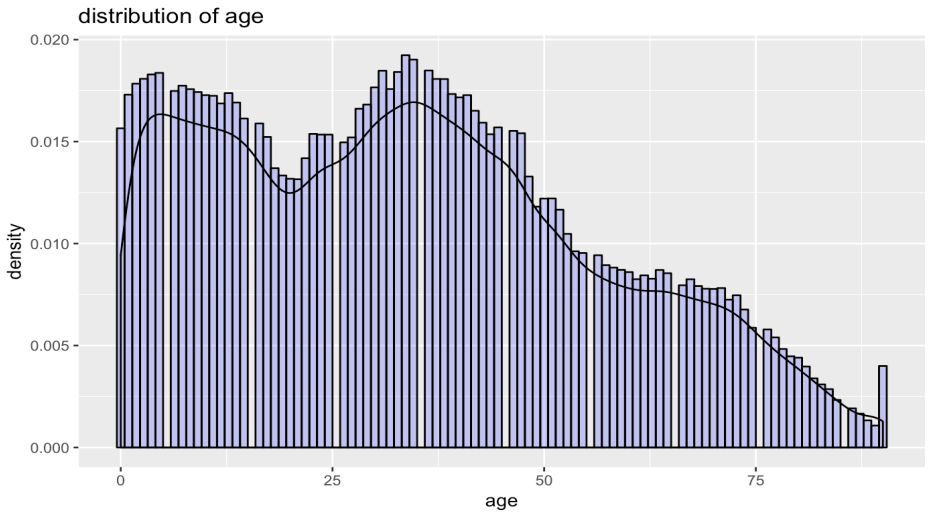
We see that the majority class has a proportion of 94% which is great. But our performance would depend on how good can we predict the minority classes. Next thing, we separate categorical and numerical variables for our further analysis:

```
> #subset categorical variables
> cat_train <- train[,factcols, with=FALSE]
> cat_test <- test[,factcols, with=FALSE]
> #subset numerical variables
> num_train <- train[,numcols, with=FALSE]
> num_test <- test[,numcols, with=FALSE]
```

Data Exploration

Visualizations for different variables in terms of target variable:

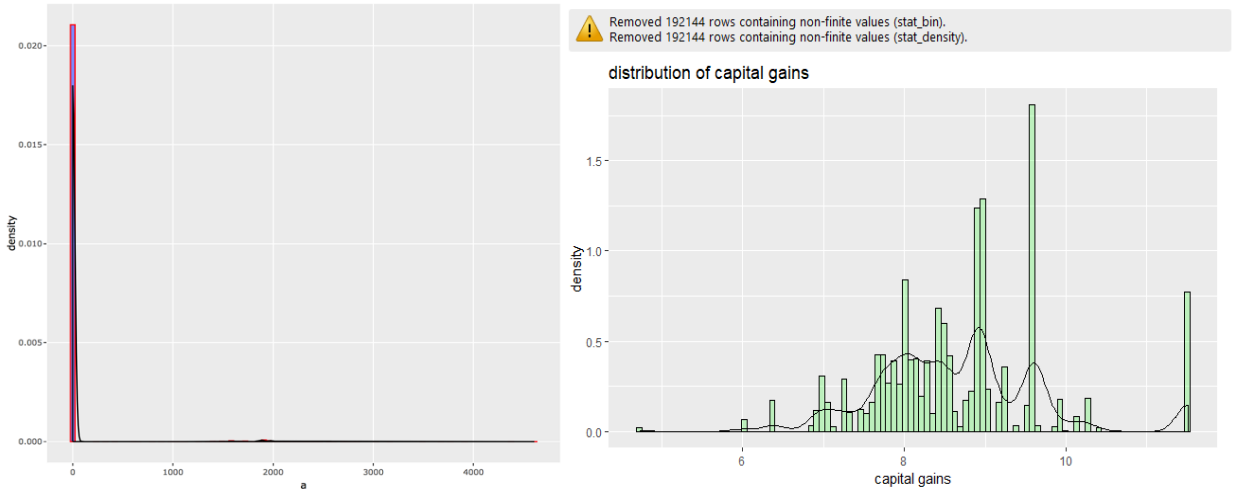
Age



Here we see that age is distributed in **bi-modal** where it has two peaks at about 10-12 years and 35-40 years. Here we have used the density as well as histogram with 100 bins. We considered age as an important factor is categorizing the other features in coming sections.

Capital Gains

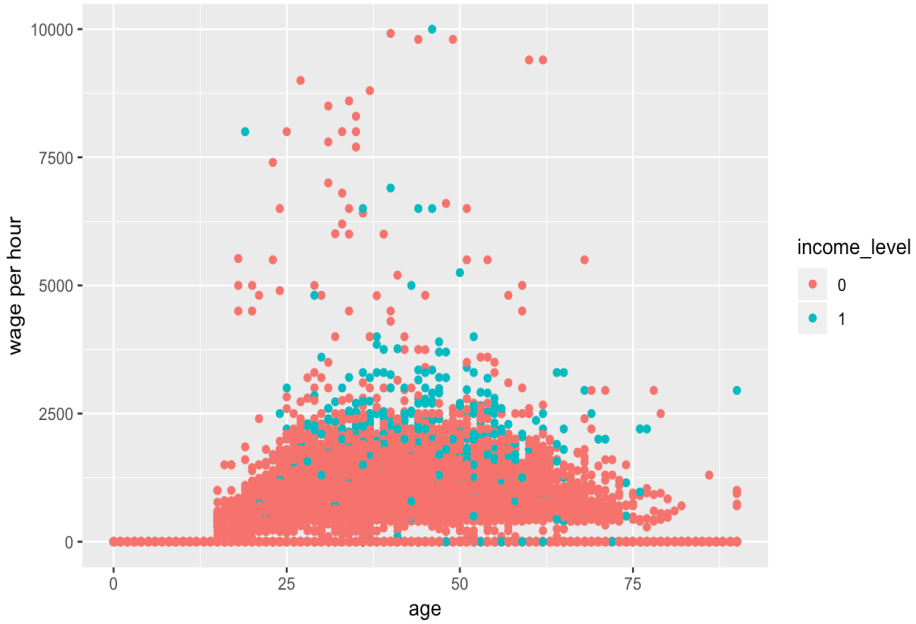
Capital gains refer to the amount of gains the person received in that particular year in dollars. This column has many zeros. The left side of the 2 plots shows the skewed distribution of capital gains which makes no sense. So, we used the logarithmic on the feature to see that it almost follows normal distribution.



There are many numerical variables in this dataset which have more than 95% of the values as zeros.

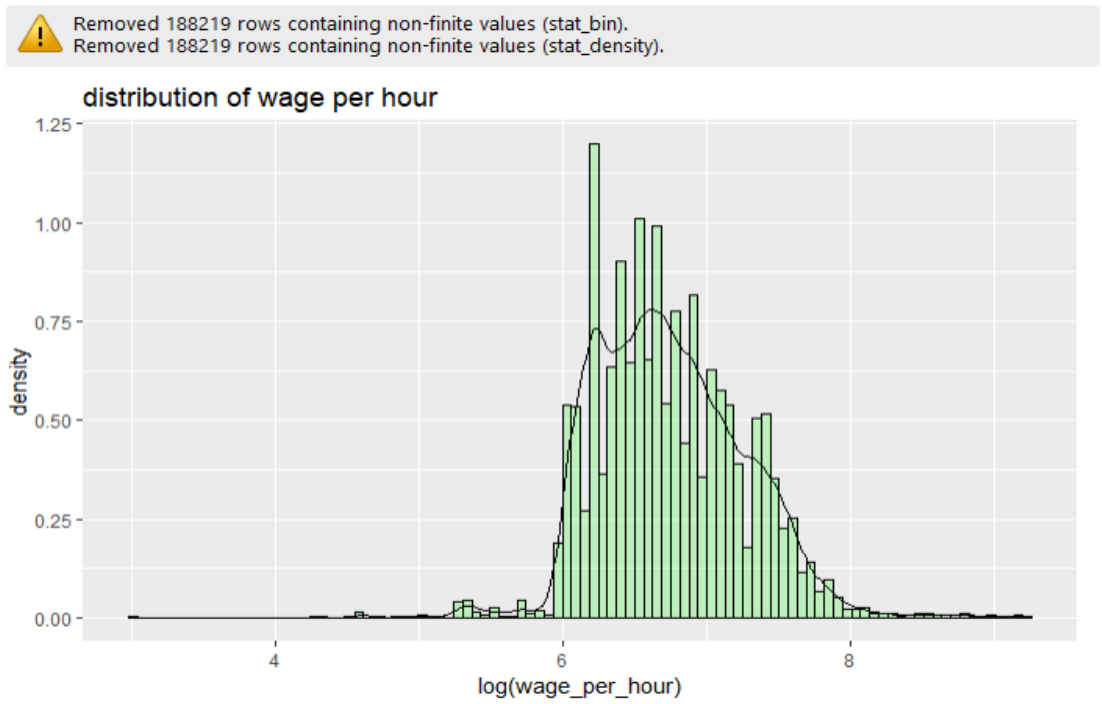
Age vs Wage_per_hour

This has a very non-linear relationship. Major age values which are having income level as 1 are populated between 30-60 years which conforms that we can start binning out age variable in the data manipulation techniques.



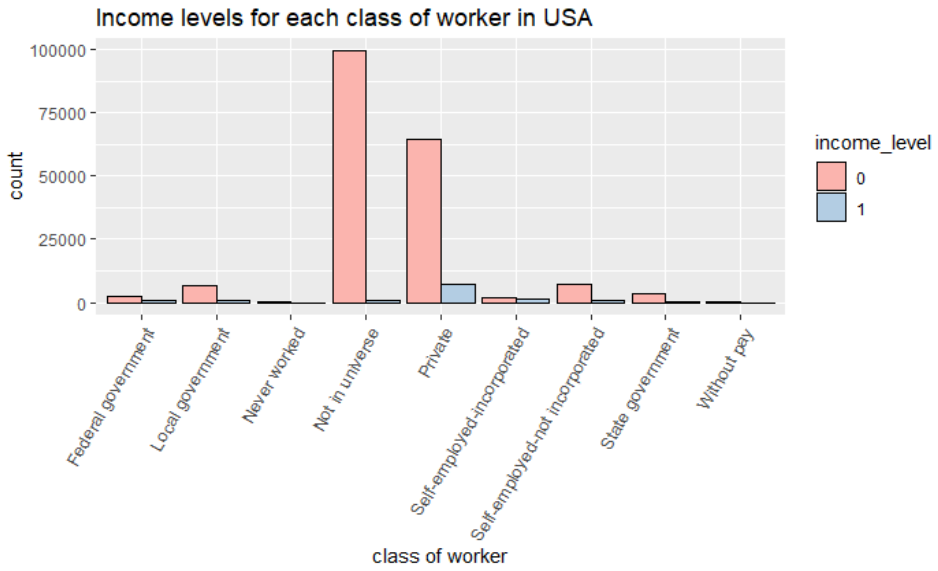
The following shows the distribution of log(wage_per_hour).

The logarithmic values are used because this also has a right skewed distribution containing many zeros in it.



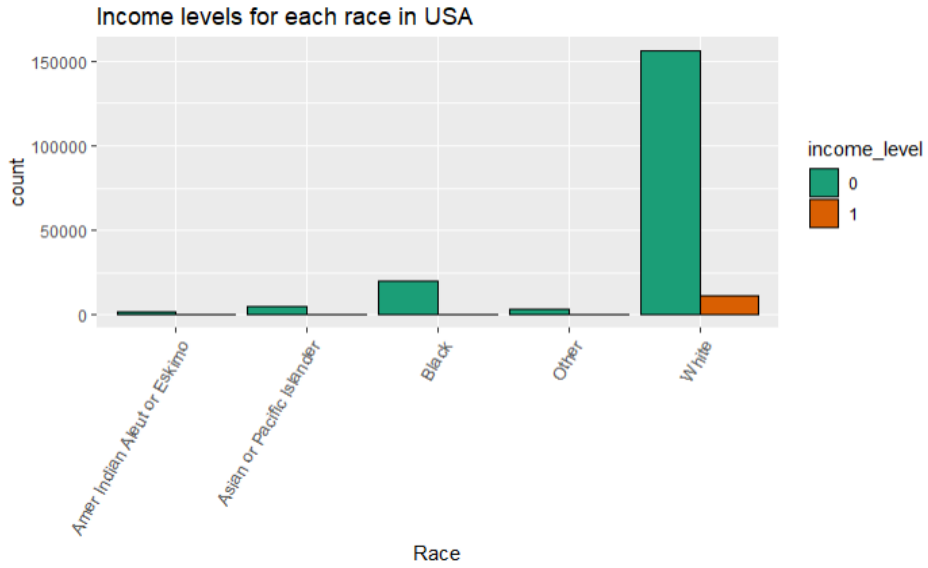
Class of worker

There is no specific information is provided about *Not in universe* category. This variable looks imbalanced i.e. only two category levels seem to dominate. In such situation, a good practice is to combine levels having less than 5% frequency of the total category frequency. We hav done this in data manipulation techniques.



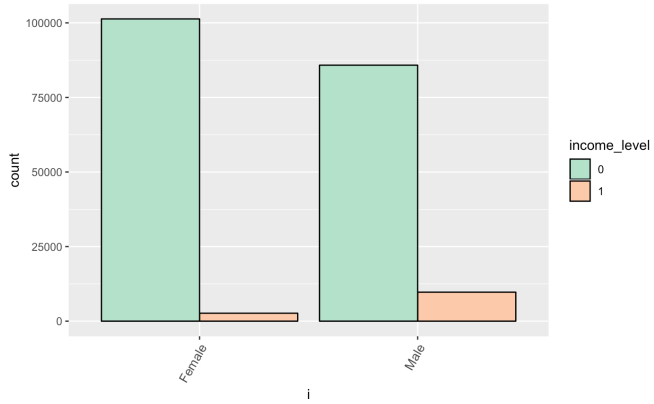
Race

Obviously, there are a lot more white people in United States but the strange thing is that they are also contributing to the maximum <50K income levels here. So we decided to bin the less frequent ones into one and change it into a categorical variable where only 2-3 levels are there.



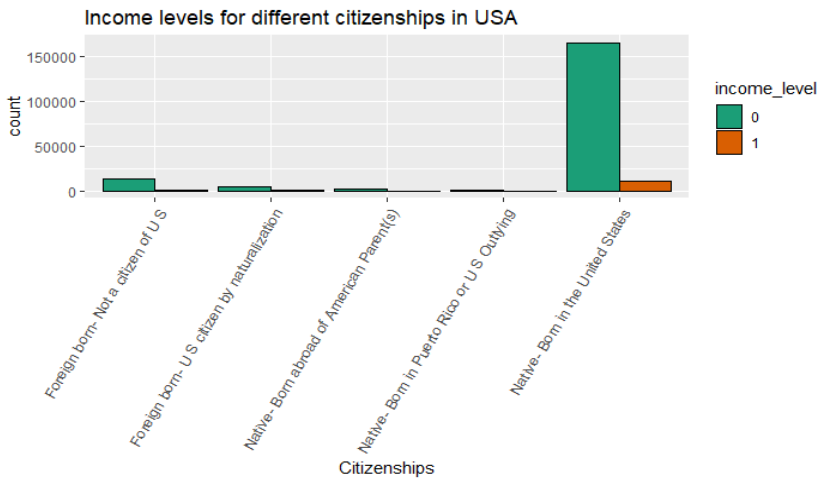
Sex

More males earn >50k salary than females. So, we made a dummy variable where we added two columns saying male and female in the form of one-hot encoded columns.



Citizenships

Citizenship also played a crucial role. We divided this into two classes saying “Native born” and “other”. This will act as a dummy variable and we are hoping that this will play a good role in influencing the prediction of salary.



III. Data Preparation and Preprocessing

Data Cleaning

We check for missing values in numerical variables and get the following output:

```
> #check missing values in numerical data
> table(is.na(num_train))

  FALSE
1396661
> table(is.na(num_test))

 FALSE
698334
```

The numeric variable has no NA values. After filtering out the variables with high correlation, "weeks_worked_in_year" gets removed.

Now, we check for missing values in categorical variables and get the following output:

```
> table(is.na(cat_train))

  FALSE    TRUE
5967883   17807
> #check missing values per columns
> table(is.na(cat_test))

  FALSE    TRUE
2983860    9000
```

Some variables have over 50% missing values so, we remove these category levels using subset() function. For rest of the missing values, we mark them as "unavailable" because imputation may affect the model performance for imbalance dataset.

Data Manipulation

Many categorical variables have several levels with low frequencies which will not help as they may not be available in the test test at all. We can combine all these levels and name that category as "other".

```
> #check columns with unequal levels
> library(mlr)
> summarizeColumns(cat_train)[,"nlevs"]
 [1] 3 3 2 5 2 5 3 8 3 2 2 3 2 4 4 2 2 6 5 3 4 3 2 2 3 3 2 3 2 1
> summarizeColumns(cat_test)[,"nlevs"]
 [1] 3 3 2 5 2 5 3 8 3 2 2 3 2 4 4 2 2 6 5 3 4 3 2 2 3 3 2 3 2 1
```

After a quick hygiene check of summarizing columns, we obtain unique number of levels for the given set of variables for train and test set. Next, we bin the "age" variable from 0-30,31-60 and 61-90 and numeric variables with "zero" and "morethanzero" to understand that more than 70% of values in these variables are zeroes.

Machine Learning

Dealing with imbalanced datasets are a big blow during the performance of the metrics because they seem to classify with more than 97% accuracy, but the actual minority class goes unnoticed.

Below are the reasons which leads to manipulation in accuracy of ML algorithms on imbalanced data sets:

1. The unequal distribution in dependent variable causes discrepancy in accuracy.
2. This causes the performance of existing classifiers to get biased towards majority class.
3. The minority class contributes very little as the algorithms always tend to focus on reducing errors and increasing accuracy.
4. They also assume that errors obtained from different classes have same cost.

Methods to deal with them are many and two of the most effective are

a. Sampling methods - Uses different ways to sample out the data of the majority or the minority class from the dependent variable and make a balanced class from it.
b. Cost function method - Assigns 1 false negative for every 100 false positive and then tries to minimize the cost of each algorithm used based on this criterion.

Types of Sampling methods performed in this case study are:

Undersampling - It reduces the number of observations from majority class to make the data set balanced. This method is best to use when the data set is huge and reducing the number of training samples helps to improve run time and storage troubles. (Works with majority class)

```
> #undersampling
> train.under <- undersample(train.task,rate = 0.1) #keep only 10% of majority class
> table(getTaskTargets(train.under))

    0     1
18714 12382
```

The problem with this approach is that removing observations may cause the training data to lose important information pertaining to majority class

.

Oversampling - It replicates the observations from minority to balance the data.The advantage is that there is no information loss and the disadvantage is that as there are replications of the same type of data our dataset which leads to overfitting while predictions.

```
> #oversampling
> train.over <- oversample(train.task,rate=15) #make minority class 15 times
> table(getTaskTargets(train.over))

     0      1
187141 185730
```

SMOTE - stands for Synthetic Minority Over-sampling Technique which creates artificial data based on feature space (rather than data space) similarities from minority samples. To generate artificial data, it uses bootstrapping and k-nearest neighbors. Precisely, it works this way:

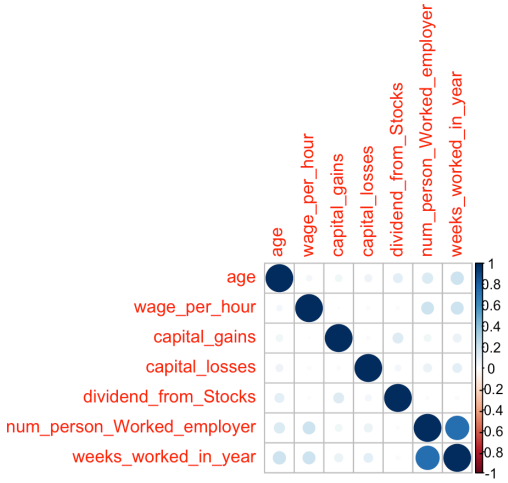
1. Take the difference between the feature vector (sample) under consideration and its nearest neighbor.
2. Multiply this difference by a random number between 0 and 1

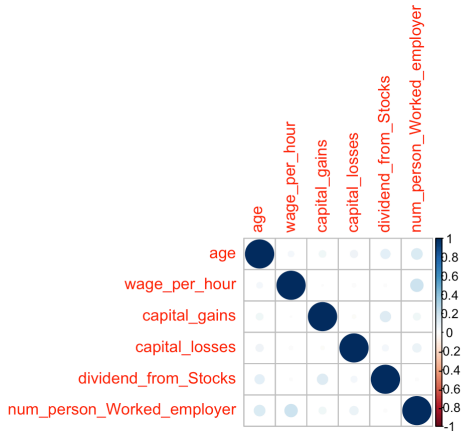
3. Add it to the feature vector under consideration
4. This causes the selection of a random point along the line segment between two specific features

```
> table(getTaskTargets(train.smote))

     0      1
187141 123820
```

Correlation analysis

Age being the most continuous one did not create any issues but the as we see here, the weeks_worked_in_year is directly related to "num_person_worked_employer" with a correlation value of over 0.8. The second graph states the correlation plot where we removed one of the correlated variables from it.





IV. Data Mining Techniques and Implementation

Naive Bayes

Naive Bayes is a classification that is based on the Bayes theorem with assumption that the presence of one feature in a class is completely unrelated to the presence of all other features. It is recommended to be used for categorical variables than continuous variables.

We'll be implementing Naive Bayes on all 4 data sets i.e. imbalanced, oversample, under sample and SMOTE and compare the prediction accuracy using cross validation.

```
> #naive Bayes
> naive_learner <- makeLearner("classif.naiveBayes",predict.type = "response")
> naive_learner$par.vals <- list(laplace = 1)
>
> #10fold CV - stratified
> folds <- makeResampleDesc("CV",iters=10,stratify = TRUE)
>
> #cross validation function
> fun_cv <- function(a){
+   crv_val <- resample(naive_learner,a,folds,measures = list(acc,tpr,tnr,fpr,fp,fn))
+   crv_val$aggr
+ }
>
> fun_cv(train.task)
```

```
> fun_cv(train.task)
Resampling: cross-validation
Measures:             acc       tpr       tnr       fpr       fp        fn
[Resample] iter 1:    0.7240878   0.7127284   0.8957997   0.1042003   129.0000000 5376.0000000
[Resample] iter 2:    0.7225341   0.7116063   0.8877221   0.1122779   139.0000000 5397.0000000
[Resample] iter 3:    0.7269219   0.7153086   0.9023406   0.0976594   121.0000000 5328.0000000
[Resample] iter 4:    0.7262931   0.7148659   0.8990307   0.1009693   125.0000000 5336.0000000
[Resample] iter 5:    0.7249536   0.7131025   0.9039548   0.0960452   119.0000000 5369.0000000
[Resample] iter 6:    0.7232358   0.7117666   0.8966074   0.1033926   128.0000000 5394.0000000
[Resample] iter 7:    0.7242382   0.7125147   0.9014540   0.0985460   122.0000000 5380.0000000
[Resample] iter 8:    0.7273456   0.7159880   0.8990307   0.1009693   125.0000000 5315.0000000
[Resample] iter 9:    0.7297514   0.7180720   0.9063005   0.0936995   116.0000000 5276.0000000
[Resample] iter 10:   0.7217823   0.7103238   0.8949919   0.1050081   130.0000000 5421.0000000


Aggregated Result: acc.test.mean=0.7251144,tpr.test.mean=0.7136277,tnr.test.mean=0.8987232,fpr.test.mean=0.1012768,fp.test.
mean=125.4000000,fn.test.mean=5359.2000000
                    lapply(X, FUN, ...)

acc.test.mean tpr.test.mean tnr.test.mean fpr.test.mean  fp.test.mean  fn.test.mean
    0.7251144     0.7136277     0.8987232     0.1012768   125.4000000  5359.2000000
> fun_cv(train.under)
Resampling: cross-validation
Measures:             acc       tpr       tnr       fpr       fp        fn
[Resample] iter 1:    0.7507237  0.6488509  0.9046850  0.0953150  118.0000000657.0000000
[Resample] iter 2:    0.7590865  0.6595404  0.9095315  0.0904685  112.0000000637.0000000
[Resample] iter 3:    0.7603731  0.6552646  0.9192246  0.0807754  100.0000000645.0000000
[Resample] iter 4:    0.7614915  0.6693376  0.9007264  0.0992736  123.0000000619.0000000
[Resample] iter 5:    0.7590865  0.6520577  0.9208401  0.0791599  98.0000000 651.0000000
[Resample] iter 6:    0.7500804  0.6451096  0.9087237  0.0912763  113.0000000664.0000000
[Resample] iter 7:    0.7610932  0.6538462  0.9232633  0.0767367  95.0000000 648.0000000
[Resample] iter 8:    0.7594082  0.6504543  0.9240711  0.0759289  94.0000000 654.0000000
[Resample] iter 9:    0.7707395  0.6720085  0.9200323  0.0799677  99.0000000 614.0000000
[Resample] iter 10:   0.7534555  0.6415598  0.9225182  0.0774818  96.0000000 671.0000000


Aggregated Result: acc.test.mean=0.7585538,tpr.test.mean=0.6548030,tnr.test.mean=0.9153616,fpr.test.mean=0.0846384,fp.test.
mean=104.8000000,fn.test.mean=646.0000000


acc.test.mean tpr.test.mean tnr.test.mean fpr.test.mean  fp.test.mean  fn.test.mean
   0.75855382    0.65480295    0.91536161    0.08463839  104.80000000  646.00000000
> fun_cv(train.over)
Resampling: cross-validation
Measures:             acc       tpr       tnr       fpr       fp        fn
[Resample] iter 1:    0.7807815   0.6453457   0.9172455   0.0827545   1537.00000006637.0000000
[Resample] iter 2:    0.7816934   0.6475900   0.9168147   0.0831853   1545.00000006595.0000000
[Resample] iter 3:    0.7837316   0.6512771   0.9171916   0.0828084   1538.00000006526.0000000
[Resample] iter 4:    0.7830343   0.6496206   0.9174608   0.0825392   1533.00000006557.0000000
[Resample] iter 5:    0.7844557   0.6512237   0.9186992   0.0813008   1510.00000006527.0000000
[Resample] iter 6:    0.7892888   0.6617152   0.9178377   0.0821623   1526.00000006331.0000000
[Resample] iter 7:    0.7837852   0.6479107   0.9206913   0.0793087   1473.00000006589.0000000
[Resample] iter 8:    0.7834366   0.6541092   0.9137458   0.0862542   1602.00000006473.0000000
[Resample] iter 9:    0.7815593   0.6513306   0.9127766   0.0872234   1620.00000006525.0000000
[Resample] iter 10:   0.7878349   0.6582238   0.9184300   0.0815700   1515.00000006396.0000000


Aggregated Result: acc.test.mean=0.7839601,tpr.test.mean=0.6518347,tnr.test.mean=0.9170893,fpr.test.mean=0.0829107,fp.test.
mean=1539.9000000,fn.test.mean=6515.6000000


acc.test.mean tpr.test.mean tnr.test.mean fpr.test.mean  fp.test.mean  fn.test.mean
 7.839601e-01  6.518347e-01  9.170893e-01  8.291068e-02  1.539900e+03  6.515600e+03
> fun_cv(train.smote)
Resampling: cross-validation
Measures:             acc       tpr       tnr       fpr       fp        fn
[Resample] iter 1:    0.7660792   0.6685904   0.9134227   0.0865773   1072.00000006202.0000000
[Resample] iter 2:    0.7665691   0.6660967   0.9184300   0.0815700   1010.00000006249.0000000
[Resample] iter 3:    0.7693272   0.6693919   0.9203683   0.0796317   986.0000000 6187.0000000
[Resample] iter 4:    0.7660792   0.6644758   0.9196414   0.0803586   995.0000000 6279.0000000
[Resample] iter 5:    0.7706779   0.6715828   0.9204490   0.0795510   985.0000000 6146.0000000
[Resample] iter 6:    0.7667867   0.6680560   0.9160071   0.0839929   1040.00000006212.0000000
[Resample] iter 7:    0.7708065   0.6735599   0.9177839   0.0822161   1018.00000006109.0000000
[Resample] iter 8:    0.7673013   0.6659186   0.9205298   0.0794702   984.0000000 6252.0000000
[Resample] iter 9:    0.7691665   0.6716362   0.9165724   0.0834276   1033.00000006145.0000000
[Resample] iter 10:   0.7621237   0.6626055   0.9125343   0.0874657   1083.00000006314.0000000


Aggregated Result: acc.test.mean=0.7674917,tpr.test.mean=0.6681914,tnr.test.mean=0.9175739,fpr.test.mean=0.0824261,fp.test.
mean=1020.6000000,fn.test.mean=6209.5000000


acc.test.mean tpr.test.mean tnr.test.mean fpr.test.mean  fp.test.mean  fn.test.mean
    0.7674917     0.6681914     0.9175739     0.0824261  1020.6000000  6209.5000000
```

The confusion matrix shows that Naive Bayes has a good 85.6% accuracy and the f-measure is close to 80%.

XGBoost

XGBoost stands for Extreme Gradient Boosting which originates which leverages boosting, hardware design and model penalties to perform portable and accurate large scale tree boosting. It's a good alternative to random forest and recommended to use in quick predictive applications. Let us break down the implementation of this training code :

1. Deciding the parameters for the xgboost learner can be set in **params**
2. We have used k-fold cross validation where k = 5 in this.
3. Using the training set, trained the xgboost for 100 bootstrapping iterations without repetition. Notice, we used early stopping at round value of 20.
4. Notice that we have also used error rate as our metric.

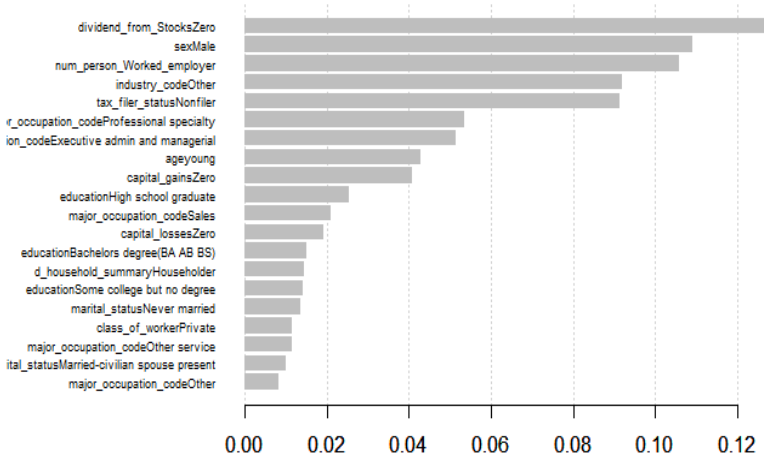
The results are shared in the performance evaluation.

```
dtrain <- xgb.DMatrix(data = new_tr,label = tr_labels)
dtest <- xgb.DMatrix(data = new_ts,label= ts_labels)

params <- list(booster = "gbtree",
               objective = "binary:logistic",
               eta=0.3, gamma=0, max_depth=6,
               min_child_weight=1, subsample=1,
               colsample_bytree=1)
xgbcv <- xgb.cv( params = params,
                 data = dtrain, nrounds = 100,
                 nfold = 5, showsd = T,
                 stratified = T, print.every.n = 10,
                 early.stop.round = 20, maximize = F)
xgb1 <- xgb.train (params = params,
                   data = dtrain, nrounds = 100,
                   watchlist = list(val=dtest,train=dtrain),
                   print.every.n = 10,
                   early.stop.round = 10,
                   maximize = F , eval_metric = "error")
```

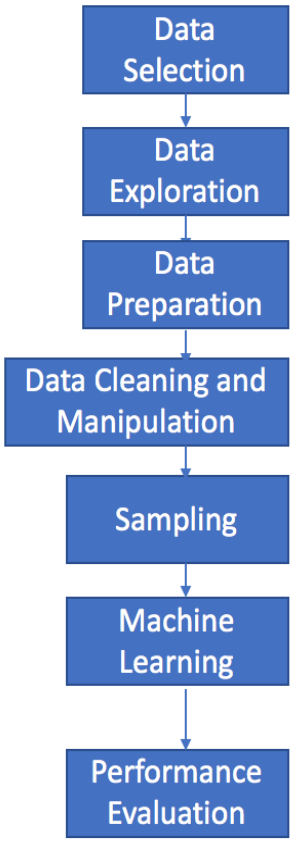
```
'print.every.n' is deprecated.
Use 'print_every_n' instead.
See help("Deprecated") and help("xgboost-deprecated").'early.stop.round' is deprecated.
Use 'early_stopping_rounds' instead.
See help("Deprecated") and help("xgboost-deprecated").[1]	train-error:0.056045+0.001115	test-error:0.056765+0.001345
Multiple eval metrics are present. Will use test_error for early stopping.
Will train until test_error hasn't improved in 20 rounds.

[11]	train-error:0.053055+0.000297	test-error:0.053708+0.001302
[21]	train-error:0.051653+0.000393	test-error:0.052746+0.001011
[31]	train-error:0.050917+0.000313	test-error:0.052465+0.001064
[41]	train-error:0.050229+0.000316	test-error:0.052505+0.001399
[51]	train-error:0.049526+0.000400	test-error:0.052480+0.001483
Stopping. Best iteration:
[34]	train-error:0.050724+0.000280	test-error:0.052370+0.001175

'print.every.n' is deprecated.
Use 'print_every_n' instead.
See help("Deprecated") and help("xgboost-deprecated").'early.stop.round' is deprecated.
Use 'early_stopping_rounds' instead.
See help("Deprecated") and help("xgboost-deprecated").[1]	val-error:0.055632	train-error:0.055227
Multiple eval metrics are present. Will use train_error for early stopping.
Will train until train_error hasn't improved in 10 rounds.

[11]	val-error:0.053357	train-error:0.052951
[21]	val-error:0.052876	train-error:0.051683
[31]	val-error:0.052174	train-error:0.051152
[41]	val-error:0.052355	train-error:0.050435
[51]	val-error:0.052214	train-error:0.049834
[61]	val-error:0.051964	train-error:0.049533
[71]	val-error:0.051994	train-error:0.049062
[81]	val-error:0.051974	train-error:0.048696
[91]	val-error:0.052214	train-error:0.048030
[100]	val-error:0.052194	train-error:0.047864
```

The XGBoost classifier is able to allocate the importance of the each variable.

```r
mat <- xgb.importance (feature_names = colnames(new_tr),model = xgb1)
xgb.plot.importance (importance_matrix = mat[1:20])
```



Case Study Flowchart



UCI repository- Census Data: here

Visualized the effect of other variables on target variable (income)

1) Split the data in 90% train and 10% test set
2) Separate numerical and categorical variables

1) Check for NA values.
2) Remove low frequency variables.

Handle imbalances by performing- Oversampling, Undersampling and SMOTE on train set.

Apply following algorithms:
1) Naïve Bayes 2) XGBoost

1) SMOTE Sampling has better results compared to undersampling and oversampling in handling data imbalance.
2) XGBoost performs better than Naïve bayes for binary classification problem.
3) Support Vector Machines can be applied for improvement.

V. Performance Evaluation

Sampling Methods

The sampling techniques used on Naive Bayes gives these results and we see that SMOTE turns out to be most effective -

fun_cv()	acc.test.mean	tpr.test.mean	tnr.test.mean	fpr.test.mean
train.task	0.7251144	0.7136277	0.8987232	0.1012768
train.under	0.75855382	0.65480295	0.91536161	0.08463839
train.over	7.839601e-01	6.518347e-01	9.170893e-01	8.291068e-02
train.smote	0.8574917	0.8181914	0.9175739	0.0824261

Binary Classification

1)Naive Bayes

```
> table(d_test$income_level,nB_prediction)
   nB_prediction
        0     1
  0 62602 30974
  1   527  5659
> #calculate F measure
> precision <- dCM$byClass['Pos Pred Value']
> recall <- dCM$byClass['Sensitivity']
>
> f_measure <- 2*((precision*recall)/(precision+recall))
> f_measure
Pos Pred Value
     0.798979
```

Prediction accuracy - 79%

2)XGBoost

```
> xg_confused
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 92455  4086
         1  1121  2100

               Accuracy : 0.9478
                 95% CI : (0.9464, 0.9492)
    No Information Rate : 0.938
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4219
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9880
            Specificity : 0.3395
         Pos Pred Value : 0.9577
         Neg Pred Value : 0.6520
             Prevalence : 0.9380
         Detection Rate : 0.9268
   Detection Prevalence : 0.9677
      Balanced Accuracy : 0.6637

       'Positive' Class : 0

> precision <- xg_confused$byClass['Pos Pred Value']
>
> recall <- xg_confused$byClass['Sensitivity']
>
>
> f_measure <- 2*((precision*recall)/(precision+recall))
> f_measure
Pos Pred Value
     0.9726116
>
```

Prediction accuracy - 97.26%

VI. Discussion and Recommendation

The following are a mixture of discussions and recommendations. Discussions include the advantages and disadvantages of using these methods stated. Here we go -

a. The project started out by training models directly on the given dataset with 95% accuracy, but the minority class was unnoticed. So, we used 3 sampling techniques namely -

1. Undersampling - major disadvantage is it might remove the important informations regarding the features used for prediction
2. Oversampling - major disadvantage is that it causes overfitting due to repetition of data points.
3. SMOTE sampling - this has advantage that it generates synthetic data using k-nearest neighbors.

b. Then we used data visualizations to see the underlying inferences. These turned out to be pretty good because all the methods used in data manipulation were results of inferences drawn in visualizations.

c. The age variable binning used by us involves 0-30, 30-60, 60-90. Furthermore, binning can be done for models to learn the hidden patterns.

d. First models used was the Naive Bayes classifier. It uses Naive Bayes theorem and it assumes that the features follow the normality and linearity assumptions and at the same time that the features are not correlated to each other.

e. XGBoost can be tried out in various other ways like changing these parameters :

1. Increase the number of rounds
2. Do 10 fold CV (time consuming and requires better computational environment)
3. Increase repetitions in random search

e. There are various ways to search for the optimum parameters like Random search and Grid search which are more effective than the way we approached.

f. Support Vector machines with effective tuning of parameters can work better than XGBoost and Naive Bayes.

VII. Summary

The problem refers to the fact that for a large number of real world problems, the number of positive examples is dwarfed by the number of negative examples. For most algorithms, if we give them data that is 99.9% negative and 0.1% positive, they will simply learn to always predict negative.

The problem is not with the data, but rather with the way that we have defined the learning problem therefore, we do not focus on the accuracy. In this case study, we focus on the confusion matrix and f-measure characteristic to completely evaluate the performance.

Resampling to get more balanced data:

1) Down-sampling: majority class
2) Up-sampling: minority class
3) SMOTE: Hybrid

SMOTE performs better than over and under sampling methods.

Binary Classification models:

1) Naive Bayes: 79% accuracy
2) XGBoost: 97.26% accuracy

XGBoost shows 18% better accuracy than Naive Bayes.

This information can be found applicable by census department to form tax reforms based on income levels and for businesses to target the right audience.

Appendix: R Code for use case study

```
#load packages & data
library(data.table)
train <- fread("train.csv",na.strings = c("","  ","?","NA",NA))
test <- fread("test.csv",na.strings = c("","  ","?","NA",NA))

dim(train); str (train); View(train)
dim(test); str (test); View(test)

#check first few rows of train & test
train[1:5]
test [1:5]

#check target variables
unique(train$income_level)
unique(test$income_level)

#encode target variables
train[,income_level := ifelse(income_level == "-50000",0,1)]
test[,income_level := ifelse(income_level == "-50000",0,1)]
round(prop.table(table(train$income_level))*100)

#set column classes
factcols <- c(2:5,7,8:16,20:29,31:38,40,41)
numcols <- setdiff(1:40,factcols)

train[,(factcols) := lapply(.SD, factor), .SDcols = factcols]
train[,(numcols) := lapply(.SD, as.numeric), .SDcols = numcols]

test[,(factcols) := lapply(.SD, factor), .SDcols = factcols]
test[,(numcols) := lapply(.SD, as.numeric), .SDcols = numcols]

#subset categorical variables
cat_train <- train[,factcols, with=FALSE]
cat_test <- test[,factcols, with=FALSE]

#subset numerical variables
num_train <- train[,numcols, with=FALSE]
num_test <- test[,numcols, with=FALSE]

#load libraries
library(ggplot2)
library(plotly)

#write a plot function
tr <- function(a){
```

```
  ggplot(data = num_train, aes(x= a, y=..density..)) +
geom_histogram(fill="blue",color="red",alpha = 0.5,bins =100) + geom_density()
  ggplotly()
}

#variable age
tr(num_train$age)

#variable capital_losses
tr(num_train$capital_losses)

#add target variable
num_train[,income_level := cat_train$income_level]

#create a scatter plot
ggplot(data=num_train,aes(x = age,
y=wage_per_hour))+geom_point(aes(colour=income_level))+scale_y_continuous("wage per
hour", breaks = seq(0,10000,1000))

#dodged bar chart
all_bar <- function(i){
  ggplot(cat_train,aes(x=i,fill=income_level))+geom_bar(position = "dodge",
color="black")+scale_fill_brewer(palette = "Pastel1")+theme(axis.text.x =element_text(angle  =
60,hjust = 1,size=10))
}

#variable class_of_worker
all_bar(cat_train$class_of_worker)

#variable education
all_bar(cat_train$education)

prop.table(table(cat_train$marital_status,cat_train$income_level),1)
prop.table(table(cat_train$class_of_worker,cat_train$income_level),1)

#check missing values in numerical data
table(is.na(num_train))
table(is.na(num_test))

corrplot(cor(num_train))
cor(num_train)
corrplot(cor(num_train))

library(caret)

#set threshold as 0.7
ax <-findCorrelation(x = cor(num_train), cutoff = 0.7)
```

```
num_train <- num_train[,-ax,with=FALSE]
num_test[,weeks_worked_in_year := NULL]

#check missing values per columns
mvtr <- sapply(cat_train, function(x){sum(is.na(x))/length(x)})*100
mvte <- sapply(cat_test, function(x){sum(is.na(x)/length(x))}*100)
mvtr
mvte


cat_train <- subset(cat_train, select = mvtr < 5 )
cat_test <- subset(cat_test, select = mvte < 5)

#set NA as Unavailable - train data
#convert to characters
cat_train <- cat_train[,names(cat_train) := lapply(.SD, as.character),.SDcols = names(cat_train)]
for (i in seq_along(cat_train)) set(cat_train, i=which(is.na(cat_train[[i]])), j=i, value="Unavailable")
#convert back to factors
cat_train <- cat_train[, names(cat_train) := lapply(.SD,factor), .SDcols = names(cat_train)]

#set NA as Unavailable - test data
cat_test <- cat_test[, (names(cat_test)) := lapply(.SD, as.character), .SDcols = names(cat_test)]
for (i in seq_along(cat_test)) set(cat_test, i=which(is.na(cat_test[[i]])), j=i, value="Unavailable")
#convert back to factors
cat_test <- cat_test[, (names(cat_test)) := lapply(.SD, factor), .SDcols = names(cat_test)]

#write a general plot function to get
hist_plotter <- function(a, fill_color, color, alpha){
        ggplot(data = num_train, aes(x= a, y=..density..)) + geom_histogram(fill=fill_color,color=
color,alpha = alpha,bins =100 ) +                geom_density(color = color)
        # ggtitle(label = sprintf("Distribution of %s w.r.t its density", a)) + xlab(sprintf("%s", a))
}

hist_plotter(a = num_train$age, fill_color = "blue", alpha = 0.2, color = "black") +
ggtitle("distribution of age ") + xlab("age")

hist_plotter(log(num_train$capital_gains), fill_color = "green", alpha = 0.2, color = "black" )

hist_plotter(log(num_train$wage_per_hour), fill_color = "green", alpha = 0.2, color = "black" )

#add target variable
num_train[,income_level := cat_train$income_level]

#create a scatter plot
```

```
ggplot(data=num_train,aes(x = age,
y=wage_per_hour))+geom_point(aes(colour=income_level))+scale_y_continuous("wage per
hour")

ggplot(data=num_train,aes(x = age,
y=dividend_from_Stocks))+geom_point(aes(colour=income_level))+scale_y_continuous("Divide
nd from stocks", breaks = seq(0,100000,15000))

#combine factor levels with less than 5% values
#train
for(i in names(cat_train)){
  p <- 5/100
  ld <- names(which(prop.table(table(cat_train[[i]])) < p))
  levels(cat_train[[i]])[levels(cat_train[[i]]) %in% ld] <- "Other"
}

#test
for(i in names(cat_test)){
  p <- 5/100
  ld <- names(which(prop.table(table(cat_test[[i]])) < p))
  levels(cat_test[[i]])[levels(cat_test[[i]]) %in% ld] <- "Other"
}

#check columns with unequal levels
library(mlr)
summarizeColumns(cat_train)[,"nlevs"]
summarizeColumns(cat_test)[,"nlevs"]

num_train[,.N,age][order(age)]
num_train[,.N,wage_per_hour][order(-N)]

#bin age variable 0-30 31-60 61 - 90
num_train[,age:= cut(x = age,breaks = c(0,30,60,90),include.lowest = TRUE,labels =
c("young","adult","old"))]
num_train[,age := factor(age)]

num_test[,age:= cut(x = age,breaks = c(0,30,60,90),include.lowest = TRUE,labels =
c("young","adult","old"))]
num_test[,age := factor(age)]

#Bin numeric variables with Zero and MoreThanZero
num_train[,wage_per_hour := ifelse(wage_per_hour ==
0,"Zero","MoreThanZero")][,wage_per_hour := as.factor(wage_per_hour)]
num_train[,capital_gains := ifelse(capital_gains == 0,"Zero","MoreThanZero")][,capital_gains :=
as.factor(capital_gains)]
num_train[,capital_losses := ifelse(capital_losses == 0,"Zero","MoreThanZero")][,capital_losses
:= as.factor(capital_losses)]
```

```
num_train[,dividend_from_Stocks := ifelse(dividend_from_Stocks ==
0,"Zero","MoreThanZero")][,dividend_from_Stocks := as.factor(dividend_from_Stocks)]

num_test[,wage_per_hour := ifelse(wage_per_hour ==
0,"Zero","MoreThanZero")][,wage_per_hour := as.factor(wage_per_hour)]
num_test[,capital_gains := ifelse(capital_gains == 0,"Zero","MoreThanZero")][,capital_gains :=
as.factor(capital_gains)]
num_test[,capital_losses := ifelse(capital_losses == 0,"Zero","MoreThanZero")][,capital_losses
:= as.factor(capital_losses)]
num_test[,dividend_from_Stocks := ifelse(dividend_from_Stocks ==
0,"Zero","MoreThanZero")][,dividend_from_Stocks := as.factor(dividend_from_Stocks)]
num_train[,income_level := NULL]


#combine data and make test & train files
d_train <- cbind(num_train,cat_train)
d_test <- cbind(num_test,cat_test)

#remove unwanted files
rm(num_train,num_test,cat_train,cat_test) #save memory

#load library for machine learning
library(mlr)

#create task
train.task <- makeClassifTask(data = d_train,target = "income_level")
test.task <- makeClassifTask(data=d_test,target = "income_level")

#remove zero variance features
train.task <- removeConstantFeatures(train.task)
test.task <- removeConstantFeatures(test.task)

#get variable importance chart
var_imp <- generateFilterValuesData(train.task, method = c("information.gain"))
plotFilterValues(var_imp,feat.type.cols = TRUE)

#undersampling
train.under <- undersample(train.task,rate = 0.1) #keep only 10% of majority class
table(getTaskTargets(train.under))

#oversampling
train.over <- oversample(train.task,rate=15) #make minority class 15 times
table(getTaskTargets(train.over))

#SMOTE
train.smote <- smote(train.task,rate = 15,nn = 5)
```

```
system.time(
  train.smote <- smote(train.task,rate = 10,nn = 3)
)
table(getTaskTargets(train.smote))

#lets see which algorithms are available
listLearners("classif","twoclass")[c("class","package")]

#naive Bayes
naive_learner <- makeLearner("classif.naiveBayes",predict.type = "response")
naive_learner$par.vals <- list(laplace = 1)

#10fold CV - stratified
folds <- makeResampleDesc("CV",iters=10,stratify = TRUE)

#cross validation function
fun_cv <- function(a){
  crv_val <- resample(naive_learner,a,folds,measures = list(acc,tpr,tnr,fpr,fp,fn))
  crv_val$aggr
}

fun_cv(train.task)
fun_cv(train.under)
fun_cv(train.over)
fun_cv(train.smote)

#train and predict
nB_model <- train(naive_learner, train.smote)
nB_predict <- predict(nB_model,test.task)

#evaluate
nB_prediction <- nB_predict$data$response
dCM <- confusionMatrix(d_test$income_level,nB_prediction)
# Accuracy : 0.8174
# Sensitivity : 0.9862
# Specificity : 0.2299

#calculate F measure
precision <- dCM$byClass['Pos Pred Value']
recall <- dCM$byClass['Sensitivity']

f_measure <- 2*((precision*recall)/(precision+recall))
f_measure

table(d_test$income_level,nB_prediction)

#xgboost
```

```
dtrain <- xgb.DMatrix(data = new_tr,label = tr_labels)
dtest <- xgb.DMatrix(data = new_ts,label= ts_labels)

params <- list(booster = "gbtree",
               objective = "binary:logistic",
               eta=0.3, gamma=0, max_depth=6,
               min_child_weight=1, subsample=1,
               colsample_bytree=1)
xgbcv <- xgb.cv( params = params,
                 data = dtrain, nrounds = 100,
                 nfold = 5, showsd = T,
                 stratified = T, print.every.n = 10,
                 early.stop.round = 20, maximize = F)
xgb1 <- xgb.train (params = params,
                   data = dtrain, nrounds = 100,
                   watchlist = list(val=dtest,train=dtrain),
                   print.every.n = 10,
                   early.stop.round = 10,
                   maximize = F , eval_metric = "error")

xgbpred <- predict (xgb1,dtest)
xgbpred <- ifelse (xgbpred > 0.5,1,0)
library(caret)
xg_confused <- confusionMatrix(xgb_predict, td_labels)
mat <- xgb.importance (feature_names = colnames(new_tr),model = xgb1)
xgb.plot.importance (importance_matrix = mat[1:20])

precision <- xg_confused$byClass['Pos Pred Value']
recall <- xg_confused$byClass['Sensitivity']

f_measure <- 2*((precision*recall)/(precision+recall))
f_measure
#0.9726374
```