Data Processing Assignment

Logistic Regression with Data Cleaning and Processing

NUS ISS EBAC 2017

Gaelan Gu, Sunil Prakash, Wang Ruoshi, Yu Yue

Table of Contents

[1. Introduction 3](#_Toc477563040)

[2. Analysis using R 4](#_Toc477563041)

[2.1. Model 1 – Basic Regression 4](#_Toc477563042)

[2.2. Model 2- dropping race column 6](#_Toc477563043)

[2.3. Model 3 – Removing race and Relationship 7](#_Toc477563044)

[2.4. Model 4 – removing race, capital.gain and capital.loss 8](#_Toc477563045)

[2.5. Model 5 – applying PCA 9](#_Toc477563046)

[3. Analysis in Python 12](#_Toc477563047)

[3.1. Model 1 – Data Cleaning after Split 12](#_Toc477563048)

[3.2. Model 2 – Data Cleaning Before Splitting 16](#_Toc477563049)

[4. Summary & Conclusion 17](#_Toc477563050)

# Introduction

We will be preparing the salary dataset, extracted from the 1994 US Census, for a logistic regression.

We will determine whether a person makes over 50k a year; class will be the dependent variable.

# Analysis using R

## Model 1 – Basic Regression

train = read.csv('salary-train.csv')  
test = read.csv('salary-test.csv')  
str(train)

## 'data.frame': 32561 obs. of 14 variables:  
## $ age : int 39 50 38 53 28 37 49 52 31 42 ...  
## $ workclass : Factor w/ 9 levels " ?"," Federal-gov",..: 8 7 5 5 5 5 5 7 5 5 ...  
## $ fnlwgt : int 77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...  
## $ education : Factor w/ 16 levels " 10th"," 11th",..: 10 10 12 2 10 13 7 12 13 10 ...  
## $ marital : Factor w/ 7 levels " Divorced"," Married-AF-spouse",..: 5 3 1 3 3 3 4 3 5 3 ...  
## $ occupation : Factor w/ 15 levels " ?"," Adm-clerical",..: 2 5 7 7 11 5 9 5 11 5 ...  
## $ relationship : Factor w/ 6 levels " Husband"," Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...  
## $ race : Factor w/ 5 levels " Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...  
## $ sex : Factor w/ 2 levels " Female"," Male": 2 2 2 2 1 1 1 2 1 2 ...  
## $ capital.gain : int 2174 0 0 0 0 0 0 0 14084 5178 ...  
## $ capital.loss : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ hours.per.week: int 40 13 40 40 40 40 16 45 50 40 ...  
## $ native.country: Factor w/ 42 levels " ?"," Cambodia",..: 40 40 40 40 6 40 24 40 40 40 ...  
## $ class : Factor w/ 2 levels " <=50K"," >50K": 1 1 1 1 1 1 1 2 2 2 ...

str(test)

## 'data.frame': 16281 obs. of 14 variables:  
## $ age : int 25 38 28 44 18 34 29 63 24 55 ...  
## $ workclass : Factor w/ 9 levels " ?"," Federal-gov",..: 5 5 3 5 1 5 1 7 5 5 ...  
## $ fnlwgt : int 226802 89814 336951 160323 103497 198693 227026 104626 369667 104996 ...  
## $ education : Factor w/ 16 levels " 10th"," 11th",..: 2 12 8 16 16 1 12 15 16 6 ...  
## $ marital : Factor w/ 7 levels " Divorced"," Married-AF-spouse",..: 5 3 3 3 5 5 5 3 5 3 ...  
## $ occupation : Factor w/ 15 levels " ?"," Adm-clerical",..: 8 6 12 8 1 9 1 11 9 4 ...  
## $ relationship : Factor w/ 6 levels " Husband"," Not-in-family",..: 4 1 1 1 4 2 5 1 5 1 ...  
## $ race : Factor w/ 5 levels " Amer-Indian-Eskimo",..: 3 5 5 3 5 5 3 5 5 5 ...  
## $ sex : Factor w/ 2 levels " Female"," Male": 2 2 2 2 1 2 2 2 1 2 ...  
## $ capital.gain : int 0 0 0 7688 0 0 0 3103 0 0 ...  
## $ capital.loss : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ hours.per.week: int 40 50 40 40 30 30 40 32 40 10 ...  
## $ native.country: Factor w/ 41 levels " ?"," Cambodia",..: 39 39 39 39 39 39 39 39 39 39 ...  
## $ class : Factor w/ 2 levels " <=50K."," >50K.": 1 1 2 2 1 1 1 2 1 1 ...

We first import our datasets and determine which columns contain missing values.

From a glance, we can tell that the workclass, occupation and native.country columns contain missing values, indicated by question marks.

#### Setting Entries with Question Marks as NA Values

# train set  
train$workclass = as.factor(gsub('?', NA, train$workclass, fixed = T))  
train$native.country = as.factor(gsub('?', NA, train$native.country, fixed = T))  
train$occupation = as.factor(gsub('?', NA, train$occupation, fixed = T))  
  
# test set  
test$workclass = as.factor(gsub('?', NA, test$workclass, fixed = T))  
test$native.country = as.factor(gsub('?', NA, test$native.country, fixed = T))  
test$occupation = as.factor(gsub('?', NA, test$occupation, fixed = T))

Since the missing values exist in both the training and testing datasets, therefore we have to indicate them as NA values before we may exclude them.

#### Removing Incomplete Cases

train = train[complete.cases(train), ]  
test = test[complete.cases(test), ]  
str(train)

## 'data.frame': 30162 obs. of 14 variables:  
## $ age : int 39 50 38 53 28 37 49 52 31 42 ...  
## $ workclass : Factor w/ 8 levels " Federal-gov",..: 7 6 4 4 4 4 4 6 4 4 ...  
## $ fnlwgt : int 77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...  
## $ education : Factor w/ 16 levels " 10th"," 11th",..: 10 10 12 2 10 13 7 12 13 10 ...  
## $ marital : Factor w/ 7 levels " Divorced"," Married-AF-spouse",..: 5 3 1 3 3 3 4 3 5 3 ...  
## $ occupation : Factor w/ 14 levels " Adm-clerical",..: 1 4 6 6 10 4 8 4 10 4 ...  
## $ relationship : Factor w/ 6 levels " Husband"," Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...  
## $ race : Factor w/ 5 levels " Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...  
## $ sex : Factor w/ 2 levels " Female"," Male": 2 2 2 2 1 1 1 2 1 2 ...  
## $ capital.gain : int 2174 0 0 0 0 0 0 0 14084 5178 ...  
## $ capital.loss : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ hours.per.week: int 40 13 40 40 40 40 16 45 50 40 ...  
## $ native.country: Factor w/ 41 levels " Cambodia"," Canada",..: 39 39 39 39 5 39 23 39 39 39 ...  
## $ class : Factor w/ 2 levels " <=50K"," >50K": 1 1 1 1 1 1 1 2 2 2 ...

str(test)

## 'data.frame': 15060 obs. of 14 variables:  
## $ age : int 25 38 28 44 34 63 24 55 65 36 ...  
## $ workclass : Factor w/ 8 levels " Federal-gov",..: 4 4 2 4 4 6 4 4 4 1 ...  
## $ fnlwgt : int 226802 89814 336951 160323 198693 104626 369667 104996 184454 212465 ...  
## $ education : Factor w/ 16 levels " 10th"," 11th",..: 2 12 8 16 1 15 16 6 12 10 ...  
## $ marital : Factor w/ 7 levels " Divorced"," Married-AF-spouse",..: 5 3 3 3 5 3 5 3 3 3 ...  
## $ occupation : Factor w/ 14 levels " Adm-clerical",..: 7 5 11 7 8 10 8 3 7 1 ...  
## $ relationship : Factor w/ 6 levels " Husband"," Not-in-family",..: 4 1 1 1 2 1 5 1 1 1 ...  
## $ race : Factor w/ 5 levels " Amer-Indian-Eskimo",..: 3 5 5 3 5 5 5 5 5 5 ...  
## $ sex : Factor w/ 2 levels " Female"," Male": 2 2 2 2 2 2 1 2 2 2 ...  
## $ capital.gain : int 0 0 0 7688 0 3103 0 0 6418 0 ...  
## $ capital.loss : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ hours.per.week: int 40 50 40 40 30 32 40 10 40 40 ...  
## $ native.country: Factor w/ 40 levels " Cambodia"," Canada",..: 38 38 38 38 38 38 38 38 38 38 ...  
## $ class : Factor w/ 2 levels " <=50K."," >50K.": 1 1 2 2 1 2 1 1 2 1 ...

We run the complete.cases function to remove the NA values from both datasets. After that we use the str function again to ascertain that the variables are in the formats we need, without anymore missing entries.

#### Full Model

fit = suppressWarnings(glm(formula = class ~ .,  
 family = binomial,  
 data = train))

We start to train our training set using a logistic classifier, with class as our target variable. We use the rest of the variables as input.

#### Prediction the Test Set Results

prob\_pred = predict(fit, type = 'response', newdata = test[-14])  
y\_pred = ifelse(prob\_pred > 0.5, '>50K', '<=50K')  
  
# Confusion Matrix  
cm = table(test[, 14], y\_pred)  
cm

## y\_pred  
## <=50K >50K  
## <=50K. 10530 830  
## >50K. 1465 2235

#### Computing the Accuracy and Error Rates

acc = sum(diag(cm)) / sum(cm)  
acc

## [1] 0.8476096

err = 1 - acc  
err

## [1] 0.1523904

Model has a **84.76%** accuracy rate / **15.24%** error rate.

Let us see if we can improve the error rate through feature selection.

summary(fit)

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 33851 on 30161 degrees of freedom  
## Residual deviance: 19486 on 30066 degrees of freedom  
## AIC: 19678  
##   
## Number of Fisher Scoring iterations: 13

## Model 2- dropping race column

We will try dropping the race variable as it does not appear to be significant from the p-values (mostly > 0.05).

fit\_1 = suppressWarnings(glm(formula = class ~ . - race,  
 family = binomial,  
 data = train))  
  
summary(fit\_1)

##   
## Call:  
## glm(formula = class ~ . - race, family = binomial, data = train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -5.1125 -0.5152 -0.1898 0.0000 3.7969   
##   
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 33851 on 30161 degrees of freedom  
## Residual deviance: 19501 on 30070 degrees of freedom  
## AIC: 19685  
##   
## Number of Fisher Scoring iterations: 13

#### Prediction the Test Set Results

prob\_pred\_1 = predict(fit\_1, type = 'response', newdata = test[-14])  
y\_pred\_1 = ifelse(prob\_pred\_1 > 0.5, '>50K', '<=50K')  
  
# Confusion Matrix 1  
cm\_1 = table(test[, 14], y\_pred\_1)  
cm\_1

## y\_pred\_1  
## <=50K >50K  
## <=50K. 10537 823  
## >50K. 1470 2230

#### Computing the Accuracy and Error Rates

acc\_1 = sum(diag(cm\_1)) / sum(cm\_1)  
acc\_1

## [1] 0.8477424

err\_1 = 1 - acc\_1  
err\_1

## [1] 0.1522576

This model has an accuracy rate of **84.77%**, which is only very slightly improved.

Model 2 is our best model so far.

## Model 3 – Removing race and Relationship

We remove the relationship variable as well as it appears to be a less significant variable.

fit\_2 = suppressWarnings(glm(formula = class ~ . - race - relationship,  
 family = binomial,  
 data = train))  
  
summary(fit\_2)

##   
## Call:  
## glm(formula = class ~ . - race - relationship, family = binomial,   
## data = train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -5.1465 -0.5077 -0.2119 0.0000 3.7692   
##   
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 33851 on 30161 degrees of freedom  
## Residual deviance: 19767 on 30075 degrees of freedom  
## AIC: 19941  
##   
## Number of Fisher Scoring iterations: 13

#### Prediction the Test Set Results

prob\_pred\_2 = predict(fit\_2, type = 'response', newdata = test[-14])  
y\_pred\_2 = ifelse(prob\_pred\_2 > 0.5, '>50K', '<=50K')  
  
cm\_2 = table(test[, 14], y\_pred\_2)  
cm\_2

## y\_pred\_2  
## <=50K >50K  
## <=50K. 10554 806  
## >50K. 1504 2196

#### Computing the Accuracy and Error Rates

acc\_2 = sum(diag(cm\_2)) / sum(cm\_2)  
acc\_2

## [1] 0.8466135

err\_2 = 1 - acc\_2  
err\_2

## [1] 0.1533865

However, accuracy rate has decreased to **84.66**.

## Model 4 – removing race, capital.gain and capital.loss

We will do more data cleaning, for it appears that there are many zero values present in the capital.loss and capital.gain columns. Let's remove these from our best model so far (Model 2) and see if the result improves.

fit\_3 = suppressWarnings(glm(formula = class ~ . - race - capital.gain - capital.loss,  
 family = binomial,  
 data = train))  
  
summary(fit\_3)

##   
## Call:  
## glm(formula = class ~ . - race - capital.gain - capital.loss,   
## family = binomial, data = train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.6754 -0.5672 -0.2165 -0.0005 3.7071   
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 33851 on 30161 degrees of freedom  
## Residual deviance: 21424 on 30072 degrees of freedom  
## AIC: 21604  
##   
## Number of Fisher Scoring iterations: 13

AIC appears to have risen, which is a sign of a worse fit.

#### Prediction the Test Set Results

prob\_pred\_3 = predict(fit\_3, type = 'response', newdata = test[-14])  
y\_pred\_3 = ifelse(prob\_pred\_3 > 0.5, '>50K', '<=50K')  
  
cm\_3 = table(test[, 14], y\_pred\_3)  
cm\_3

## y\_pred\_3  
## <=50K >50K  
## <=50K. 10415 945  
## >50K. 1616 2084

#### Computing the Accuracy and Error Rates

acc\_3 = sum(diag(cm\_3)) / sum(cm\_3)  
acc\_3

## [1] 0.8299469

err\_3 = 1 - acc\_3  
err\_3

## [1] 0.1700531

Accuracy rate has decreased to **82.99%** in this case.

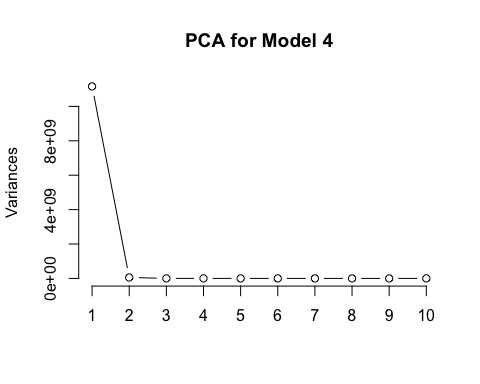
## Model 5 – applying PCA

We will try feature transformation in this last model, by means of Principal Component Analysis (PCA). To prepare our datasets for this, we will need to create dummy variables.

# install.packages('dummies')  
library(dummies)

## dummies-1.5.6 provided by Decision Patterns

train\_4 = dummy.data.frame(train, names = c('workclass','education', 'marital','occupation',  
 'relationship','race','sex','native.country'))  
test\_4 = dummy.data.frame(test, names = c('workclass','education', 'marital','occupation',  
 'relationship','race','sex','native.country'))  
  
train\_4\_pca = prcomp(train\_4[-104])  
test\_4\_pca = prcomp(test\_4[-103])  
  
screeplot(train\_4\_pca, type = 'l', main = 'PCA for Model 4')



summary(train\_4\_pca)

## Importance of components:  
## PC1 PC2 PC3 PC4 PC5 PC6  
## Standard deviation 1.057e+05 7.406e+03 404.06991 13.26 11.67 0.8597  
## Proportion of Variance 9.951e-01 4.890e-03 0.00001 0.00 0.00 0.0000  
## Cumulative Proportion 9.951e-01 1.000e+00 1.00000 1.00 1.00 1.0000  
## PC7 PC8 PC9 PC10 PC11 PC12 PC13  
## Standard deviation 0.5582 0.5482 0.4821 0.4571 0.439 0.4211 0.3845  
## Proportion of Variance 0.0000 0.0000 0.0000 0.0000 0.000 0.0000 0.0000  
## Cumulative Proportion 1.0000 1.0000 1.0000 1.0000 1.000 1.0000 1.0000  
## PC14 PC15 PC16 PC17 PC18 PC19 PC20  
## Standard deviation 0.369 0.3481 0.3437 0.3354 0.3145 0.303 0.2918  
## Proportion of Variance 0.000 0.0000 0.0000 0.0000 0.0000 0.000 0.0000  
## Cumulative Proportion 1.000 1.0000 1.0000 1.0000 1.0000 1.000 1.0000  
## PC21 PC22 PC23 PC24 PC25 PC26 PC27  
## Standard deviation 0.2841 0.2703 0.2598 0.2397 0.2318 0.2254 0.2149  
## Proportion of Variance 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000  
## Cumulative Proportion 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000  
## PC28 PC29 PC30 PC31 PC32 PC33 PC34  
## Standard deviation 0.2107 0.2083 0.1985 0.1899 0.1886 0.1854 0.1814  
## Proportion of Variance 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000  
## Cumulative Proportion 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000  
## PC35 PC36 PC37 PC38 PC39 PC40 PC41  
## Standard deviation 0.1797 0.1776 0.1698 0.1642 0.1619 0.142 0.1404  
## Proportion of Variance 0.0000 0.0000 0.0000 0.0000 0.0000 0.000 0.0000  
## Cumulative Proportion 1.0000 1.0000 1.0000 1.0000 1.0000 1.000 1.0000  
## PC42 PC43 PC44 PC45 PC46 PC47 PC48  
## Standard deviation 0.1293 0.1237 0.1221 0.1178 0.115 0.1116 0.1041  
## Proportion of Variance 0.0000 0.0000 0.0000 0.0000 0.000 0.0000 0.0000  
## Cumulative Proportion 1.0000 1.0000 1.0000 1.0000 1.000 1.0000 1.0000  
## PC49 PC50 PC51 PC52 PC53 PC54  
## Standard deviation 0.09095 0.09028 0.07307 0.07207 0.06837 0.06381  
## Proportion of Variance 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000  
## PC55 PC56 PC57 PC58 PC59 PC60  
## Standard deviation 0.05941 0.05804 0.05754 0.05604 0.05528 0.0541  
## Proportion of Variance 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.00000 1.00000 1.0000  
## PC61 PC62 PC63 PC64 PC65 PC66  
## Standard deviation 0.05193 0.04921 0.04794 0.04681 0.04656 0.04582  
## Proportion of Variance 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000  
## PC67 PC68 PC69 PC70 PC71 PC72  
## Standard deviation 0.04451 0.04386 0.04301 0.0404 0.03941 0.03868  
## Proportion of Variance 0.00000 0.00000 0.00000 0.0000 0.00000 0.00000  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.0000 1.00000 1.00000  
## PC73 PC74 PC75 PC76 PC77 PC78  
## Standard deviation 0.03712 0.03559 0.03334 0.03298 0.03187 0.03117  
## Proportion of Variance 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000  
## PC79 PC80 PC81 PC82 PC83 PC84  
## Standard deviation 0.03024 0.02974 0.02841 0.0273 0.02519 0.02477  
## Proportion of Variance 0.00000 0.00000 0.00000 0.0000 0.00000 0.00000  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.0000 1.00000 1.00000  
## PC85 PC86 PC87 PC88 PC89 PC90  
## Standard deviation 0.02417 0.02372 0.02364 0.02319 0.02266 0.02149  
## Proportion of Variance 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000  
## PC91 PC92 PC93 PC94 PC95 PC96  
## Standard deviation 0.02079 0.02 0.01917 0.01781 0.005818 1.05e-11  
## Proportion of Variance 0.00000 0.00 0.00000 0.00000 0.000000 0.00e+00  
## Cumulative Proportion 1.00000 1.00 1.00000 1.00000 1.000000 1.00e+00  
## PC97 PC98 PC99 PC100 PC101  
## Standard deviation 1.05e-11 1.05e-11 1.05e-11 1.05e-11 1.05e-11  
## Proportion of Variance 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.00e+00  
## Cumulative Proportion 1.00e+00 1.00e+00 1.00e+00 1.00e+00 1.00e+00  
## PC102 PC103  
## Standard deviation 1.05e-11 7.278e-14  
## Proportion of Variance 0.00e+00 0.000e+00  
## Cumulative Proportion 1.00e+00 1.000e+00

The first 2 principal components are sufficient to explain most, if not all of the variation in the variables. So we will use them for the fit.

# Joining PC1 and PC2 columns to the train\_4 dataset  
train\_4\_pca\_df = as.data.frame(train\_4\_pca$x)  
train\_4$PC1 = train\_4\_pca\_df$PC1  
train\_4$PC2 = train\_4\_pca\_df$PC2  
  
# Joining PC1 and PC2 columns to the test\_4 dataset  
test\_4\_pca\_df = as.data.frame(test\_4\_pca$x)  
test\_4$PC1 = test\_4\_pca\_df$PC1  
test\_4$PC2 = test\_4\_pca\_df$PC2  
  
fit\_4 = suppressWarnings(glm(class ~ PC1 + PC2,  
 family = 'binomial',  
 data = train\_4))  
  
summary(fit\_4)

##   
## Call:  
## glm(formula = class ~ PC1 + PC2, family = "binomial", data = train\_4)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -4.9964 -0.6879 -0.6832 -0.6521 1.8616   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -9.670e-01 1.543e-02 -62.689 <2e-16 \*\*\*  
## PC1 -1.888e-07 1.353e-07 -1.395 0.163   
## PC2 3.345e-04 8.916e-06 37.519 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 33851 on 30161 degrees of freedom  
## Residual deviance: 30698 on 30159 degrees of freedom  
## AIC: 30704  
##   
## Number of Fisher Scoring iterations: 6

#### Prediction the Test Set Results

prob\_pred\_4 = predict(fit\_4, type = 'response', newdata = test\_4[c(104, 105)])  
y\_pred\_4 = ifelse(prob\_pred\_4 > 0.5, '>50K', '<=50K')  
  
cm\_4 = table(test\_4[, 103], y\_pred\_4)  
cm\_4

## y\_pred\_4  
## <=50K >50K  
## <=50K. 11200 160  
## >50K. 2970 730

#### Computing the Accuracy and Error Rates

acc\_4 = sum(diag(cm\_4)) / sum(cm\_4)  
acc\_4

## [1] 0.7921647

err\_4 = 1 - acc\_4  
err\_4

## [1] 0.2078353

Accuracy rate has decreased to **79.22%** for this model, which shows that feature transformation does not improve our results.

Let’s do some the same analysis with data cleaning in python

# Analysis in Python

## Model 1 – Data Cleaning after Split

Provided training and test data in messy format, lots of clean-up required

**import** numpy **as** np

**import** pandas **as** pd

**import** statsmodels.api **as** sm

**import** matplotlib.pyplot **as** plt

**from** patsy **import** dmatrices

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.cross\_validation **import** train\_test\_split

**from** sklearn **import** metrics

**from** sklearn.cross\_validation **import** cross\_val\_score

%matplotlib inline

Reading CSV with special params

1. Treating ‘?’ as NA , as lots of places ? is present, with spaces appended also

2. Skipping initial space of all columns and values

train = pd.read\_csv("salary-train.csv", sep=',',na\_values=['?',' ?', '? '],skipinitialspace=True)

test = pd.read\_csv("salary-test.csv", sep=',',na\_values=['?',' ?', '? '],skipinitialspace=True)

**print**(train.shape)

**print**(test.shape)

train = train.dropna()

test = test.dropna()

**print**(train.shape)

**print**(test.shape)

(32561, 14)

(16281, 14)

(30162, 14)

(15060, 14)

So the size of train before cleaning was 32,561 and after cleaning becomes 30,162 with 14 variables and the size of test before cleaning was 16,281 and after cleaning becomes 15,060 with 14 variables

train.describe()

train.head(5)

As Observed, the class variable of test are appended with ‘.’, so, need to remove them first to make train and test class similar

test['class'] = test['class'].replace('<=50K.','<=50K')

test['class'] = test['class'].replace('>50K.','>50K')

test.head(5)

**print**(train.columns.tolist())

['age', 'workclass', 'fnlwgt', 'education', 'marital', 'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'class']

Defining a common method, for doing all pre-processing like,

1. One hard coding or dummy encoding for all categorical variables

2. replacing the existing columns with encoded columns

3. returning the features and class variables

**def** preprocessing(data):

**from** sklearn **import** preprocessing

X=data[data.columns.difference(['class'])]

y=data['class']

#print(preprocessing.LabelEncoder())

label\_encoder = preprocessing.LabelEncoder()

encoded\_workclass = label\_encoder.fit\_transform(data["workclass"])

encoded\_education = label\_encoder.fit\_transform(data["education"])

encoded\_marital = label\_encoder.fit\_transform(data["marital"])

encoded\_occupation = label\_encoder.fit\_transform(data["occupation"])

encoded\_relationship = label\_encoder.fit\_transform(data["relationship"])

encoded\_race = label\_encoder.fit\_transform(data["race"])

encoded\_sex = label\_encoder.fit\_transform(data["sex"])

encoded\_native\_country = label\_encoder.fit\_transform(data["native-country"])

X = X.drop('workclass', axis=1)

X['workclass'] = encoded\_workclass

X = X.drop('education', axis=1)

X['education'] = encoded\_education

X = X.drop('marital', axis=1)

X['marital'] = encoded\_marital

X = X.drop('occupation', axis=1)

X['occupation'] = encoded\_occupation

X = X.drop('relationship', axis=1)

X['relationship'] = encoded\_relationship

X = X.drop('race', axis=1)

X['race'] = encoded\_race

X = X.drop('sex', axis=1)

X['sex'] = encoded\_sex

X = X.drop('native-country', axis=1)

X['native-country'] = encoded\_native\_country

**return**(X,y)

# Initialize label encoder

X\_train, y\_train = preprocessing(train)

X\_test, y\_test = preprocessing(test)

**print**(X\_train.head(3))

y\_train.head(3)

age capital-gain capital-loss fnlwgt hours-per-week workclass \

0 39 2174 0 77516 40 5

1 50 0 0 83311 13 4

2 38 0 0 215646 40 2

education marital occupation relationship race sex native-country

0 9 4 0 1 4 1 38

1 9 2 3 0 4 1 38

2 11 0 5 1 4 1 38

0 <=50K

1 <=50K

2 <=50K

Name: class, dtype: object

X\_test.head(3)

|  |
| --- |

Initializing the Logistic Regression and fitting the training data

model = LogisticRegression()

model = model.fit(X\_train, y\_train)

model.score(X\_train,y\_train)

0.78486174656852992

y\_predicted = model.predict(X\_test)

y\_predicted

array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '>50K', '<=50K'], dtype=object)

# generate evaluation metrics

**print**(y\_test[1])

**print**(y\_predicted[:5])

metrics.accuracy\_score(y\_test, y\_predicted)

<=50K

['<=50K' '<=50K' '<=50K' '>50K' '<=50K']

0.78326693227091637

metrics.confusion\_matrix(y\_test, y\_predicted)

array([[10670, 690],

[ 2574, 1126]])

**print**(metrics.classification\_report(y\_test, y\_predicted))

precision recall f1-score support

<=50K 0.81 0.94 0.87 11360

>50K 0.62 0.30 0.41 3700

avg / total 0.76 0.78 0.75 15060

So Accuracy score is **78.3%**

## Model 2 – Data Cleaning Before Splitting

The case could be because of test and train split before cleaning, so let’s try to combine clean and then split

data\_combined = pd.concat([train,test])

data\_combined.shape

(45222, 14)

X,y = preprocessing(data\_combined)

X\_train2, X\_test2, y\_train2, y\_test2 = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

model2 = LogisticRegression()

model2.fit(X\_train2, y\_train2)

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,

penalty='l2', random\_state=None, solver='liblinear', tol=0.0001,

verbose=0, warm\_start=False)

predicted\_y2 = model2.predict(X\_test2)

**print**(metrics.accuracy\_score(y\_test2, predicted\_y2))

0.788088744748

**print**(metrics.confusion\_matrix(y\_test2, predicted\_y2))

**print**(metrics.classification\_report(y\_test2, predicted\_y2))

[[9746 512]

[2363 946]]

precision recall f1-score support

<=50K 0.80 0.95 0.87 10258

>50K 0.65 0.29 0.40 3309

avg / total 0.77 0.79 0.76 13567

there is slightly increase in precision and recall, and very little increase in accuracy also to **78.8%**

# evaluate the model using 10-fold cross-validation

scores = cross\_val\_score(LogisticRegression(), X, y, scoring='accuracy', cv=10)

**print**(scores)

**print**(scores.mean())

[ 0.78509839 0.78156091 0.79703736 0.79482644 0.78969483 0.78482972

0.79787705 0.79035825 0.77770405 0.79495687]

0.789394386064

Nice, as it still performs accuracy with **78.9%**

# Summary & Conclusion

We have used the following models in our analysis to predict the class variable:

* Model 1 (fit) - using **all** variables 84.76%
* Model 2 (fit\_1) - using all **except** race 84.77%
* Model 3 (fit\_2) - using all **except** race and relationship 84.66%
* Model 4 (fit\_3) - using all **except** race, capital.gain and capital.loss 82.99 %
* Model 5 (fit\_4) - using 1st 2 components of PCA 79.22 %
* Python Model 1 – Data cleaning before split 78.3 %
* Python Model 2 – Data Cleaning after split 78.9 %

We can conclude that **Model 2** produces the best result, with the race variable excluded.

This model has the highest accuracy rate of **84.77%**.