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

2. Analysis using R ... 4

2.1. Model 1 – Basic Regression ... 4

2.2. Model 2- dropping race column ... 6

2.3. Model 3 – Removing race and Relationship ... 7

2.4. Model 4 – removing race, capital.gain and capital.loss ... 8

2.5. Model 5 – applying PCA ... 9

3. Analysis in Python ... 12

3.1. Model 1 – Data Cleaning after Split ... 12

3.2. Model 2 – Data Cleaning Before Splitting ... 16

4. Summary & Conclusion ... 17

1. 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.*

2. Analysis using R

2.1. Model 1 – Basic Regression

```r
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***

```r
# 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 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
```

2.2. 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.

2.3. 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**.*

2.4. 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.*

2.5. 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.

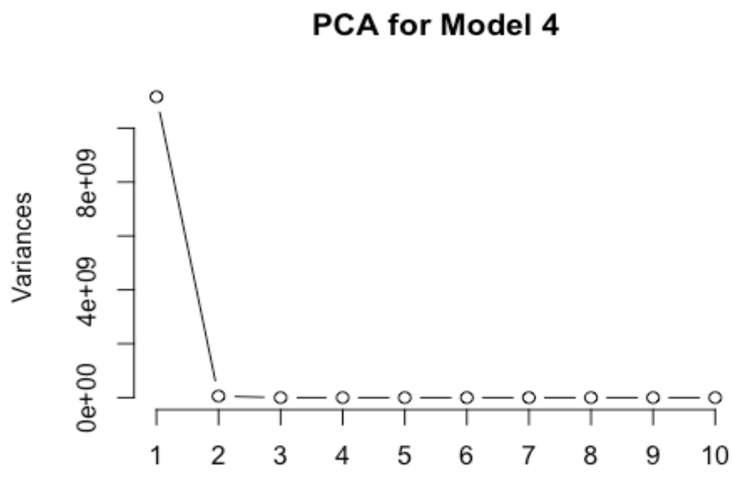
```
# install.packages('dummies')
library(dummies)

## 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

3. *Analysis in* ***Python***

3.1. 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-gai
n', '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%***

3.2. 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%***

4. 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%.***