Department of Statistics
STATS 784: Data Mining

Zhi Zhang, 708439475, zzha822
The University of Auckland

August 3, 2017

1 Question 1

1.1 Application1

In "Applied Predictive Modelling" (Kuhn, M. and Johnson, K. (2013).), chapter 3, there is a case that decribes the application of data pre-processing technology.

Data pre-processing techniques generally refer to the addition, deletion, or transformation of training set data. This case is about Cell Segmentation in High-Content Screening. Using High-Content Screening, the medical researchers can measure the cell characteristics from the kinds of samples of number of cells in a living organism or plant. These samples are regarded as the training set that can be used for data pre-processing techniques.

The first process is to do data transformations for individual predictors, which including centering, scaling, and resolving distributional skewness. Centering and scaling are generally used to improve the numerical stability of some calculations. And after skewness transformation, the distribution is not entirely symmetric but these data are better behaved than when they were in the natural units.

Another pre-process is to do data transformations for multiple predictors, using methods to resolve outliers and reduce the dimension of the data. Usually we an identify the outliers on a figure and there are some predictive models resistant to outliers. Data reduction techniques are another class of predictor transformations. These methods reduce the data by generating a smaller set of predictors that seek to capture a majority of the information in the original variables.

Dealing with Missing Values is also important for pre-processing data. It is important to understand why the values are missing. First and foremost, it is important to know if the pattern of missing data is related to the outcome. Missing data can be imputed and imputation has been extensively studied in the statistical literature. One popular technique for imputation is a K-nearest neighbor model. A new sample is imputed by finding the samples in the training set closest to it and averages these nearby points to fill in the value.

Removing predictors is also used to get potential advantages for the data modeling. A rule of thumb for detecting near-zero variance predictors is:

• The fraction of unique values over the sample size is low (say 10%).

• The ratio of the frequency of the most prevalent value to the frequency of the second most prevalent value is large (say around 20).

In addition, collinearity is the technical term for the situation where a pair of predictor variables have a substantial correlation with each other. It is also possible to have relationships between multiple predictors at once (called multicollinearity).

Also adding predictors and binning predictors are also general techniques we used for pre-processing data before modeling. More details are discussing with the case of Cell Segmentation.

1.2 Application2

In "Applied Predictive Modelling" (Kuhn, M. and Johnson, K. (2013).), chapter 12, there is a case that decribes the application of Discriminant Analysis technology.

Discriminant Analysis is one of the techniques of Classification Analysis for Data Mining. Generally it includes the techniques for linear and nonlinear classification models. In this chapter, the case is about Predicting Successful Grant Applications. The data of these applications are from a 2011 Kaggle competition sponsored by the University of Melbourne where there was interest in predicting whether or not a grant application would be accepted. In addition to predicting grant success, the university sought to understand factors that were important in predicting success.

Logistic regression is a very popular model due to its simplicity and ability to make inferential statements about model terms. It is linear in the parameters, and these parameters are obtained by minimizing the sum of the squared residuals. It turns out that the model that minimizes the sum of the squared residuals also produces maximum likelihood estimates of the parameters when it is reasonable to assume that the model residuals follow a normal (i.e., Gaussian) distribution.

Another important technique is called Linear Discriminant Analysis (LDA). This method define a linear discriminant function (may find an optimal discriminant vector) to do analysis and estimation. Examining the coefficients of the linear discriminant function can provide an understanding of the relative importance of predictors. Due to the inherent problem with LDA, as well as its other fundamental requirements, it is recommended that LDA be used on data sets that have at least 510 times more samples than predictors.

The third technique is called Partial Least Squares Discriminant Analysis (PLS-DA). For retrospectively or prospectively, measured predictors for any particular problem can be highly correlated or can exceed the number of samples collected. If either of these conditions is true, then the usual LDA approach cannot be directly used to find the optimal discriminant function. So we use PLS for the purpose of discrimination. Applying PLS in the classification setting with a multivariate response has strong mathematical connections to both canonical correlation analysis and LDA.

In addition, Many classification models utilize penalties (or regularization) to improve the fit to the data, such as the lasso. And penalization strategies can be applied to LDA models. The penalized LDA model was applied to the grant data. The software for this model allows the user to specify the number of retained predictors as a tuning parameter. As the penalty increases and predictors are eliminated, performance improves and remains relatively constant until important factors are removed. At this point, performance falls dramatically. As a result of the tuning process, six predictors were used in the model which is competitive to other models.

The nearest-shrunken centroid model (also known as PAM, for predictive analysis for microarrays) is a linear classification model that is well suited for high-dimensional problems. The nearest shrunken centroid method has one tuning parameter: shrinkage. This model works well for problems with a large number of predictors since it has built-in feature selection that is controlled by the shrinkage tuning parameter. Nearest shrunken centroids were originally developed for RNA profiling data, where the number of predictors is large (in the many thousands) and the number of samples is small.

2 Question 2

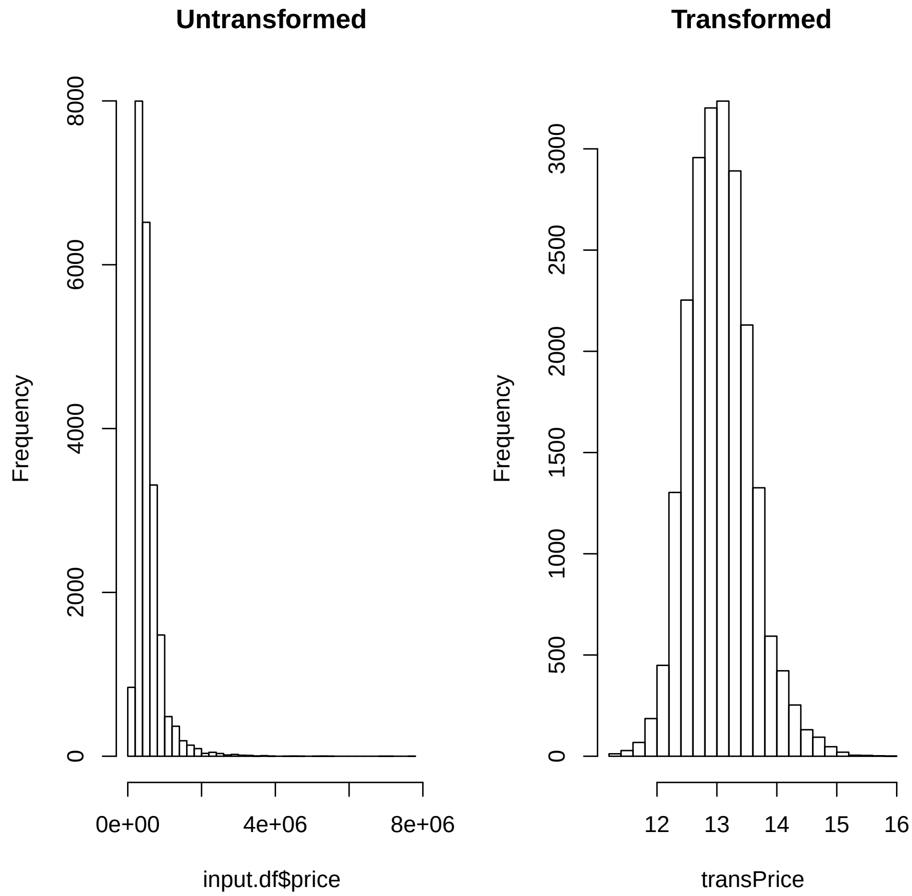
First we load the Housing Price data from the URL and change variables ***view***, ***waterf***, and ***yrr*** to be used for R functions.

```
> URL = "https://www.stat.auckland.ac.nz/~lee/784/Assignments/kc_house_data.csv"
> input.df = read.csv(URL)
> names(input.df) = c("id", "date", "price", "bedrooms", "bathrooms",
+ "sqftlv", "sqftlot", "floors", "waterfr", "view",
+ "cond", "grade", "sqfta", "sqftb", "yrb", "yrr",
+ "zip", "lat", "long", "sqftlv15", "sqftlot15")
> head(input.df)
          id            date   price bedrooms bathrooms sqftlv sqftlot
1 7129300520 20141013T000000  221900        3      1.00   1180    5650
2 6414100192 20141209T000000  538000        3      2.25   2570    7242
3 5631500400 20150225T000000  180000        2      1.00    770   10000
4 2487200875 20141209T000000  604000        4      3.00   1960    5000
5 1954400510 20150218T000000  510000        3      2.00   1680    8080
6 7237550310 20140512T000000 1225000        4      4.50   5420  101930
  floors waterfr view cond grade sqfta sqftb  yrb  yrr   zip     lat
1      1       0    0    3     7  1180     0 1955    0 98178 47.5112
2      2       0    0    3     7  2170   400 1951 1991 98125 47.7210
3      1       0    0    3     6   770     0 1933    0 98028 47.7379
4      1       0    0    5     7  1050   910 1965    0 98136 47.5208
5      1       0    0    3     8  1680     0 1987    0 98074 47.6168
6      1       0    0    3    11  3890  1530 2001    0 98053 47.6561
     long sqftlv15 sqftlot15
1 -122.257     1340      5650
2 -122.319     1690      7639
3 -122.233     2720      8062
4 -122.393     1360      5000
5 -122.045     1800      7503
6 -122.005     4760    101930
>
> input.df$waterfr = factor(input.df$waterfr)
> input.df$view = factor(input.df$view)
> input.df$yrr = factor(input.df$yrr)
> str(input.df)
'data.frame':   21613 obs. of  21 variables:
 $ id       : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
 $ date     : Factor w/ 372 levels "20140502T000000",..: 165 221 291 221 284
              11 57 252 340 306 ...
 $ price    : num  221900 538000 180000 604000 510000 ...
```

```
 $ bedrooms : int  3 3 2 4 3 4 3 3 3 3 ...
 $ bathrooms: num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
 $ sqftlv   : int  1180 2570 770 1960 1680 5420 1715 1060 1780 1890 ...
 $ sqftlot  : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
 $ floors   : num  1 2 1 1 1 1 2 1 1 2 ...
 $ waterfr  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ view     : Factor w/ 5 levels "0","1","2","3",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ cond     : int  3 3 3 5 3 3 3 3 3 3 ...
 $ grade    : int  7 7 6 7 8 11 7 7 7 7 ...
 $ sqfta    : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
 $ sqftb    : int  0 400 0 910 0 1530 0 0 730 0 ...
 $ yrb      : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
 $ yrr      : Factor w/ 70 levels "0","1934","1940",..: 1 46 1 1 1 1 1 1 1 1 ...
 $ zip      : int  98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 ...
 $ lat      : num  47.5 47.7 47.7 47.5 47.6 ...
 $ long     : num  -122 -122 -122 -122 -122 ...
 $ sqftlv15 : int  1340 1690 2720 1360 1800 4760 2238 1650 1780 2390 ...
 $ sqftlot15: int  5650 7639 8062 5000 7503 101930 6819 9711 8113 7570 ...
```

Next we transform the target to make it more symmetric, using the Box-Cox transformation.

```
> library(caret)
> trans = BoxCoxTrans(input.df$price)
> trans
Box-Cox Transformation

21613 data points used to estimate Lambda

Input data summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  75000  321950  450000  540088  645000 7700000

Largest/Smallest: 103
Sample Skewness: 4.02

Estimated Lambda: -0.2
With fudge factor, Lambda = 0 will be used for transformations

>
> transPrice = predict(trans, input.df$price)
> par(mfrow = c(1, 2))
> hist(input.df$price, main = "Untransformed", nclass = 30)
> hist(transPrice, main = "Transformed", nclass = 30)
```



From the plot we see that the transformed data is more symmetric, which will get more adavantage to the prediction. We will use transformed ***price*** to do the prediction.

Then we build a linear regression model to predict the house price. We will use half part of the data as the training set and another half part as the test set. And use the training set to make the linear regression model. Pay attention that we remove variables of ***id***, ***date***, and ***zip*** for they make no sense to the regression. And ***yrr*** and ***sqftb*** are also removed for their values variation range is large and lots of values are zero, which will add more prediction error of the regression model.

```
> used = sample(21613, 10000)
> kingCounty.df = input.df[used,]
> rownames(kingCounty.df) = 1:10000
>
> kingCounty = lm(log(transPrice[1:10000])~ bedrooms + bathrooms
+ + sqftlv + sqftlot
+ + floors + cond + grade
+ + sqfta + yrb + lat + long
+ + sqftlv15 + sqftlot15,
```

```
+ data = kingCounty.df)>
```

We get apparent error and the test set error as below:

```
> mean(residuals(kingCounty)^2)
[1] 0.001614353
> newKingCounty.df = input.df[-used,]
> rownames(newKingCounty.df) = 1:11613
> predictions = predict(kingCounty, newdata = newKingCounty.df)
> actuals = log(transPrice[10001:21613])
> mean((predictions - actuals)^2)
[1] 0.001605501
```

Next, we will use function cross.val in library R330 to caculate estimates of prediction error for the predictor. The cross validation results and standard error are below:

```
> library(R330)
> cross.val(kingCounty, nfold = 10, nrep = 20)
Cross-validated estimate of root
mean square prediction error =  0.04023651
> 0.04023651^2
[1] 0.001618977
> ####################################################################
> cross.val.mod <- function (f, nfold = 10, nrep = 20, ...){
+     X <- model.matrix(f$terms, model.frame(f))
+     y = fitted.values(f) + residuals(f)
+     n <- dim(X)[1]
+     CV <- numeric(nrep)
+     pred.error <- numeric(nfold)
+     m <- n%/%nfold
+     for (k in 1:nrep) {
+         rand.order <- order(runif(n))
+         yr <- y[rand.order]
+         Xr <- X[rand.order, ]
+         sample <- 1:m
+         for (i in 1:nfold) {
+               use.mat <- as.matrix(Xr[-sample,])
+               test.mat <- as.matrix(Xr[sample,])
+               y.use = yr[-sample]
+               new.data <- data.frame(test.mat)
+               fit <- lm(y.use ~ -1+use.mat)
+               my.predict = test.mat%*%coefficients(fit)
+               pred.error[i] <- sum((yr[sample] - my.predict)^2)/m
+               sample <- if(i==nfold) (max(sample)+1):n else sample + m
+
+             }
+             CV[k] <- mean(pred.error)
+         }
+ mean(CV)
+ }
```

```
> ##############################################
> cvvec = 1:20
> for (i in 1:20) {
+ cvvec[i] = cross.val.mod(kingCounty, nfold = 10, nrep = 1)
+ }
> mean(cvvec)
[1] 0.001619168
> sd(cvvec)
[1] 6.037923e-07
```

We can also use err.boot function to caculate the estimates of the error.

```
> err.boot(kingCounty, B = 50)
$err
[1] 0.001614353

$Err
[1] 0.001622295
```

Now we will try using bootstrap library to estimate the prediction error of our model. The function we will use is crossval() and bootpred().

```
> y = log(kingCounty.df[,1])
> x = kingCounty.df[,-1]
> theta.fit = function(x, y){lsfit(x, y)}
> theta.predict = function(fit, x){cbind(1, x) %*% fit$coef}
> sq.err = function(y, yhat) {(y - yhat)^2}
> cv10err = crossval(x, y, theta.fit, theta.predict, ngroup = 10)
Error in lsfit(x, y) : NA/NaN/Inf in 'x'

> boot = bootpred(x,y,nboot=200, theta.fit, theta.predict,
+           err.meas=sq.err)
Error in lsfit(x, y) : NA/NaN/Inf in 'x'
```

Finally, we will use caret libary to caculate the estimates of the predictor error by using function train() to call CV and bootstrap methods.

```
> library(caret)
>
> CV10 = train(log(price)~ bedrooms + bathrooms
+ + sqftlv + sqftlot
+ + floors + cond + grade
+ + sqfta + yrb + lat + long
+ + sqftlv15 + sqftlot15,
+ data = kingCounty.df,
+ method = "lm",
+ trControl = trainControl(method = "cv", number = 10,
+ repeats = 20))
> CV10
Linear Regression

10000 samples
```

```
   13 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 8999, 9000, 9000, 9001, 9000, 9001, ...
Resampling results:

  RMSE       Rsquared
  0.2633016  0.7486457

Tuning parameter 'intercept' was held constant at a value of TRUE
>
> boot = train(log(price)~ bedrooms + bathrooms
+ + sqftlv + sqftlot
+ + floors + cond + grade
+ + sqfta + yrb + lat + long
+ + sqftlv15 + sqftlot15,
+ data = kingCounty.df,
+ method = "lm",
+ trControl = trainControl(method = "boot",
+ repeats = 200))
> boot
Linear Regression

10000 samples
   13 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 10000, 10000, 10000, 10000, 10000, 10000, ...
Resampling results:

  RMSE      Rsquared
  0.263882  0.7483226

Tuning parameter 'intercept' was held constant at a value of TRUE
```

Now we can do the subset selection to see how many variables we need to choose for this linear regression model.

```
 > null.model = lm(log(transPrice[1:10000])~1, data = kingCounty.df)
> selected = step(null.model, scope = formula(kingCounty),
+ direction = "forward", trace = 0)
> selected

Call:
lm(formula = log(transPrice[1:10000]) ~ 1, data = kingCounty.df)

Coefficients:
(Intercept)
```

```
     2.567

> selected = step(kingCounty, scope = formula(kingCounty),
+ direction = "backward", trace = 0)
> selected

Call:
lm(formula = log(transPrice[1:10000]) ~ grade + yrb + long, data = kingCounty.df)

Coefficients:
(Intercept)        grade          yrb         long
  1.963e+00   -6.506e-04    2.369e-05   -4.600e-03

> selected = step(kingCounty, scope = formula(kingCounty),
+ direction = "both", trace = 0)
> selected

Call:
lm(formula = log(transPrice[1:10000]) ~ grade + yrb + long, data = kingCounty.df)

Coefficients:
(Intercept)        grade          yrb         long
  1.963e+00   -6.506e-04    2.369e-05   -4.600e-03

> allpossregs(kingCounty)
     rssp sigma2 adjRsq     Cp       AIC      BIC    CV bedrooms
1  16.157  0.002      0 -1.815  9998.185 10012.60 1.616        0
2  16.154  0.002      0 -1.248  9998.752 10020.38 1.616        0
3  16.152  0.002      0 -0.929  9999.071 10027.91 1.617        0
4  16.149  0.002      0 -0.656  9999.344 10035.40 1.617        0
5  16.147  0.002      0  0.334 10000.334 10043.60 1.617        0
6  16.146  0.002      0  1.788 10001.788 10052.26 1.617        0
7  16.146  0.002      0  3.391 10003.391 10061.07 1.617        1
8  16.145  0.002      0  4.856 10004.856 10069.75 1.617        1
9  16.144  0.002      0  6.508 10006.508 10078.61 1.618        1
10 16.144  0.002      0  8.168 10008.168 10087.48 1.618        1
11 16.144  0.002      0 10.025 10010.025 10096.55 1.618        1
12 16.144  0.002      0 12.003 10012.003 10105.74 1.619        1
13 16.144  0.002      0 14.000 10014.000 10114.94 1.619        1
   bathrooms sqftlv sqftlot floors cond grade sqfta yrb lat long
1          0      0       0      0    0     1     0   0   0    0
2          1      0       0      0    0     1     0   0   0    0
3          0      0       0      0    0     1     0   1   0    1
4          0      0       0      0    0     1     0   1   0    1
5          1      0       0      0    0     1     0   1   0    1
6          1      0       1      0    0     1     0   1   0    1
7          0      1       1      0    0     1     0   1   0    1
8          1      1       1      0    0     1     0   1   0    1
9          1      1       1      0    0     1     0   1   0    1
```

```
10          1       1       1       0    0      1     0   1   1     1
11          1       1       1       1    0      1     0   1   1     1
12          1       1       1       1    1      1     0   1   1     1
13          1       1       1       1    1      1     1   1   1     1
   sqftlv15 sqftlot15
1         0         0
2         0         0
3         0         0
4         0         1
5         0         1
6         0         1
7         0         1
8         0         1
9         1         1
10        1         1
11        1         1
12        1         1
13        1         1
```

If we do not use transformed price data (meaning not symmetric), then the forward subset selection from null.model is more complex than the previous one.

```
> null.model = lm(log(price)~1, data = kingCounty.df)
> selected = step(null.model, scope = formula(kingCounty),
+ direction = "forward", trace = 0)
> selected

Call:
lm(formula = log(price) ~ grade + lat + sqftlv + yrb + sqftlv15 +
    bathrooms + cond + floors + bedrooms + sqfta + sqftlot +
    long, data = kingCounty.df)

Coefficients:
(Intercept)        grade          lat       sqftlv          yrb
 -5.610e+01    1.707e-01    1.315e+00    1.743e-04   -3.851e-03
   sqftlv15    bathrooms         cond       floors     bedrooms
  1.232e-04    6.951e-02    6.308e-02    7.687e-02   -2.098e-02
      sqfta      sqftlot         long
 -3.257e-05    3.560e-07   -9.773e-02
```

The accuracy is pretty good when using transformed Price value in the kingCounty data, which is also similar to the result of using cross.val() and err.boot() in R330 library. The error is about 0.0016 and the SD is much more little. But when using train() in caret, for we cannot use transformed price data, so the sqaure error is a little larger, which is about 0.7483.