Data Mining for Diabetes Readmission Rate Prediction

Siyuan Meng
siyuanme@usc.edu

December 8, 2015

Project Homepage

The link of project repository is https://github.com/siyuan1992/EE660_Project. For more information, please see `README.md`.

Abstract

The management of hyperglycemia in the hospitalized inpatient becomes increasing recognized, due to the morbidity and mortality outcome [1,2]. For most non-ICU (Inensive Care Unit) patients, anecdotal evidence says that inpatient managenent is arbitrary and often leads to either no treatment or wide fluctuations in glucose when traditional management strategies are employed [4,5]. Thus, protocols are recommended. Effective prediction on readmissions enables hospitals to identify and target patients at high risk [slideshare]. Therefore, the goals are discovering major factors contributing to hospitals readmissions as well as finding the effective method to predict the type of readmissions. Top three most important features for prediction are `Number of inpatient visits`, `Admission type` and `Admission source`. The best method for classification is boosting-tree classifier. However, with data pre-mining and QUEST (Quick, Unbiased and Effecient Statistical Tree), there will be 5%~7% further accuray improvement.

Problem Statement and Goals

The data is from the Center for Clinical and Translational Research, Viginia Commonwealth University, which presents 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks [paper]. The goals here are identify the major factors that contribute to hospital readmissions as well as find the best model to predict the readmission type given inpatient information.

The difficulties of this problem are mainly in the following aspects:

- The dataset is relatively large (~100,000 samples, 55 features)
- Several features have large propotional missing data due to the fact that prior to the HITECH legislation of the American Reinvestment and Recovery Act in 2009 hospitals and clinics were not required to capture it in a structured format [paper].
- Significant amounts of preprocessing required:
  - The preliminary dataet contained multiple inpaitent visits for some patients and the observations could not be considered statistically independent [paper].
  - Even though the preliminary dataset was extracted from the database with processing, lots of redundant features still exist.
  - Almost 80% features are nominal. Thus, after feature encoding, data will be expanded into a large dimension, which makes feature selection and classification extremely time consuming.
  - Class labels are hierarchical. Three types of readmission are `NO`, `>30`, `<30`, i.e., whether a patient will have readmission and if the patient have, a hospitalization occurs within 30 days or not.

Literature Review

The research artical [paper] talked about the background information about th dataset and the information of each feature, which as a reference, helps me to deal with redundant features.

The other paper [slide] analyze the influence of different features and use some tricks to improve the performance. The trick they used are listed here:

- Data pre-mining
  - Re-categorize readmission group from 3 groups to 2 groups (Readmission <30 days and Non-within 30 days.)
  - Subsample to make readmission group's ratio be 1:1
  - Review each feature's relationship with readmission
  - Review numeric variables correlation
- Use QUEST model to improve accuracy.

Several gruops use the similar trick, like recategorizing readmission groups to two (Readmission or not). However, data pre-mining is of upmost importance in improving the model accuracy. With data pre-mining and ensemble models, the highest accuracy could reach 80% [2 stage].

Prior and Related Work

This project is exclusively for EE660.

Project Formulation and Setup

I tried several modals and algorithms for this project, e.g. Perceptron, SVM, logistic classifier, etc. I want to list two of them which have best performances.

- The first is gradient tree boosting classifier. The algorithm (pseudocode) is as follows [wiki]:

Input: training set $(\mathbf{x}_i, y_i)_{i=1}^N$, loss function $L(y, f(\mathbf{x}))$ and number of iteration M.
Algorithm:
I. Initialize model with a constant value:

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$$

.
II. for m=1 to M:
1. Compute pesudo-residuals:

$$r_{im} = -[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}]_{f(\mathbf{x})=f_{m-1}(\mathbf{x})} \qquad for\ i = 1, ..., N$$

.
2. Train with training set $(\mathbf{x}_i, r_{im})_{i=1}^N$.

3. Compute γ_m by solving:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(\mathbf{x}_i) + \gamma h_m(\mathbf{x}_i))$$

.

4. Update the model:

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \gamma_m h_m(\mathbf{x})$$

III. Output $F_M(\mathbf{x})$.

Reproducibility

In order to get the same results, need certain set of packages, as well as setting a pseudo-random seed equal the one I used.

- The following libraries were used for this project:

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(pander)
library(psych)
```

```
##
## Attaching package: 'psych'
##
## The following object is masked from 'package:randomForest':
##
##     outlier
##
## The following object is masked from 'package:ggplot2':
##
##     %+%
```

```
library(FactoMineR)
library(ggplot2)
```

- Here is the seed I set to generate pseudo-random numbers for spliting training and test dataset. (see `Preprocessing` section)

```r
set.seed(12345)
```

Getting data

```r
setwd("~/Documents/2015Fall/EE660/EE660_Project")
data <- read.csv("~/Documents/2015Fall/EE660/EE660_Project/diabetic_data.csv",
                 stringsAsFactors=T,na.strings = '?')
dim(data)
```

```
## [1] 101766     50
```

Cleaning data

The original missing value information can be found in http://www.hindawi.com/journals/bmri/2014/781670/tab1/. For simplicity, only show features which have missing values. (cite)

Feature_name	Type	Discription	Propotional_missing
Race	Nominal	Values: Caucasian, Asian, African American, Hispanic, and other	2%
Weight	Numeric	Weight in pounds	97%
Payer code	Nominal	Integer identifier corresponding to 23 distinct values, for example, Blue Cross/Blue Shield, Medicare, and self-pay	40%
Medical specialty	Nominal	Integer identifier of a specialty of the admitting physician, corresponding to 84 distinct values	50%
Diagnosis 3	Nominal	Additional secondary diagnosis (coded as first three digits of ICD9), corresponding to 954 distinct values	1%

The way I deal with missing values is to delete `Weight`, `Payer code` feaures (columns), since both features have more than 50% missing values and they are not relevant to classification. However, `Medical speciality` was maintained, adding the value "missing" in order to account for missing values. (cite)

```r
data <- data[,c(-6,-11)]
data$medical_specialty <- factor(data$medical_specialty,
                levels=c(levels(data$medical_specialty),'Missing'))
data[,10][is.na(data$medical_specialty)] <- 'Missing'
data <- na.omit(data)
dim(data)
```

```
## [1] 98053    48
```

In general, should split the original data into training and testing first and then deal with the missing values. However, both methods will yield same dimension of `data`. For simplicity, deal with NAs first here.

Preprocessing

(Preprocessing is more crucial when using model based algorithms, e.g. Linear Discrimant Analysis, Naive Bayes, Linear Regression. . . than using non-parametrical algorithms.)

- There are also other ways to impute data, like knnimpute. . . For convenience, try omitting NA rows first.
- The data has been preprocessed to ensure each encounter has a unique id. However, each patient may have more than one id.

```
length(unique(data$encounter_id))
```

```
## [1] 98053
```

```
length(unique(data$patient_nbr))
```

```
## [1] 68630
```

We thus used only one encounter per patient; in particular, we considered only the first encounter for each patient as the primary admission and determined whether or not they were readmitted within 30 days. (cite)

```
data <- data[order(data[,2]),]
unique_list <- array(TRUE,nrow(data))
for (l in 2:nrow(data)){
    if (data[l,'patient_nbr']==data[l-1,'patient_nbr']){
        unique_list[l]=FALSE
    }
}
data <- subset(data,unique_list)
```

- Kill the first two features(`encounter_id` and `patient_nbr`) which are ids for encounted and patients. Besides, extract label column, which is the last column of `data`. For two-stage classification, generate new label `label_gen` which has two levels.

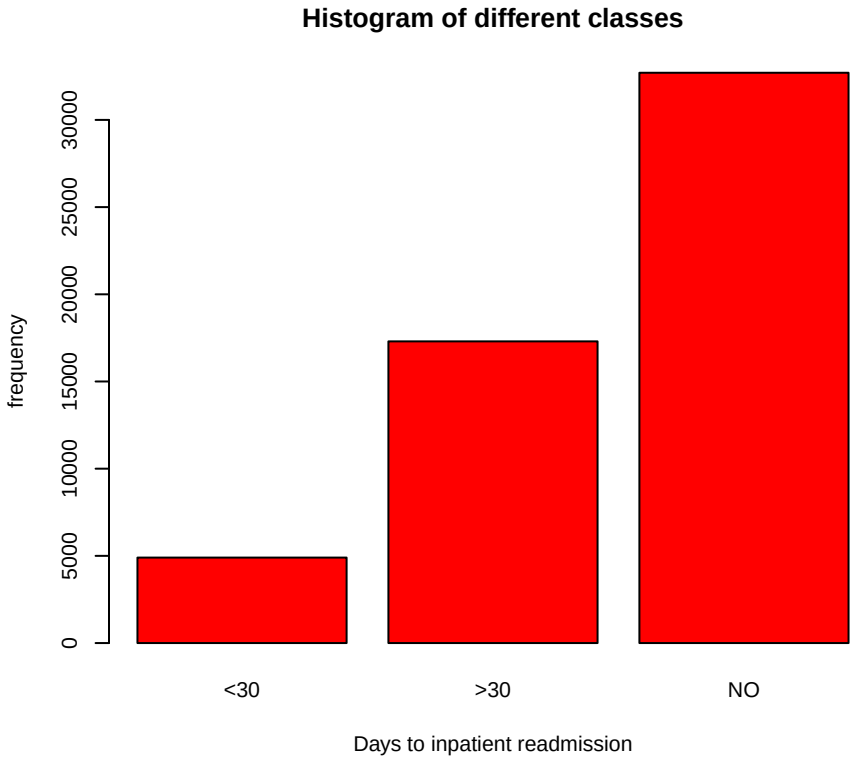
```
data <- data[,c(-1,-2)]
label <- data$readmitted
label_gen <- label
levels(label_gen) <- c('Readmission','Readmission','NO')
```

- Split this train data into training and test dataset according to the ratio 6:4 (set seed to make partioning reproducible) (Here I don't split validation since without looking data, there is possibility that test and training may not have the same distribution of different classes. Using cross validation is better in this case, though the computational complexity is high.)

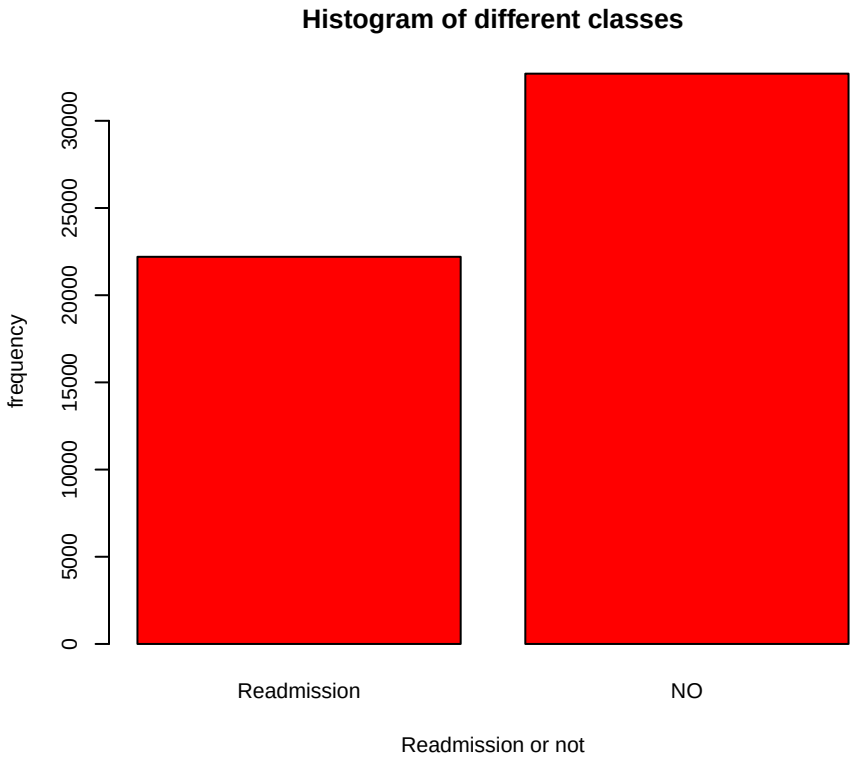
```
inTrain <- createDataPartition(label,p=0.8,list=FALSE)
training <- data[inTrain,]
training_label <- label[inTrain]
training_label_gen <- label_gen[inTrain]
test <- data[-inTrain,]
test_label <- label[-inTrain]
test_label_gen <- label_gen[-inTrain]
```

- Now, let's see if samples of different classes of `training` dataset are unbalanced.

```
par(cex=0.7,pin=c(4,3))
plot((training_label), xlab = 'Days to inpatient readmission', ylab = 'frequency',col = 'red', main = '
```




```
par(cex=0.7,pin=c(4,3))
plot((training_label_gen), xlab = 'Readmission or not', ylab = 'frequency',
     col = 'red', main = 'Histogram of different classes')
```



Classes are unbalanced distributed in three levels, but not very skewed. Since `<30` and `>30` are subsets of `Readmission`, hierarchical classification (two-stage classification) is straightforward.

```
set.seed(12121)
```

- Kill unimportant features.(*nearZeroVar* diagnoses predictors that have one unique value (i.e. are zero variance predictors) or predictors that are have both of the following characteristics: they have very few unique values relative to the number of samples and the ratio of the frequency of the most common value to the frequency of the second most common value is large.) (cite)

```
NZV <- nearZeroVar(training,saveMetrics = T,freqCut=95/5)
training <- training[,-which(NZV$nzv==TRUE)]
```

- Kill same features for `test` dataset without looking inside.

```
test <- test[,-which(NZV$nzv==TRUE)]
```

- Use background information for all features to analyze which features should be converted to numerical features. (cite)
- Note: Need to combine training and test to a big dataset and then convert to numerical fearutes. If convert seperately, some feature may have some level which has only few points and are all assigned into `training` or `test` dataset. This may lead to different dimensions for `training` and `test` dataset after conversion.
- First, drop large factors `diag2` and `diag3`.

```
training <- training[,c(-16,-17)]
test <- test[,c(-16,-17)]
```

- Second, drop last nine features.

```
training <- training[,-c(17:25)]
test <- test[,-c(17:25)]
```

- Third, convert categorical features to numerical features.

```
source('~/Documents/2015Fall/EE660/EE660_Project/C2N.R')
data <- rbind(training,test)
for (i in c(1:6,8,15)){
    temp <- as.factor(data[,i])
    data <- cbind(data,C2N(temp))
}
data <- data[,-c(1:6,8,15)]
training <- data[1:length(training_label),]
test <- data[(length(training_label)+1):68630,]
```

- Again, kill unimportant features using *nearZeroVar*.

```
NZV_2 <- nearZeroVar(training,saveMetrics = T,freqCut=95/5)
training <- training[,-which(NZV_2$nzv==TRUE)]
test<- test[,-which(NZV_2$nzv==TRUE)]
```

- Convert dataframe to numericl matrix for PCA analysis. (Need to use validation set to check what is the best dimension in classification.)

```
training <- apply(training,2,as.numeric)
test <- apply(test,2,as.numeric) # workspace saved here
PCA_fit <- prcomp(training)
training_std <- PCA_fit$x[,1:3]
test_std <- predict(PCA_fit,test)[,1:3]
```

- Normalize data (There are lots of diffeent methods for normalization,

```
training_std <- apply(training,2,function(x) (x-mean(x))/sd(x))
test_std <- apply(test,2,function(x) (x-mean(x))/sd(x))
```

Save `training` and `test` into csv file for future use

```
write.csv(cbind(training_std,training_label),'training.csv',row.names = FALSE)
write.csv(cbind(test_std,test_label),'test.csv',row.names = FALSE)
```

Purpose for doing that is `R` is good for exploratory research, but not good at dealing with large dataset for clasification. Use `Python` to read csv as `SFrame` for classification will speed up! (cite)