Mining the "Diabetes in 130 US Hospitals for Years 1999–2008" Dataset

Dario Gjorgjevski
gjorgjevski.dario@students.finki.ukim.mk

Final Project in Data Mining

September 22, 2016

1 Introduction

The dataset I will be mining is called "Diabetes in 130 US Hospitals for Years 1999–2008" (UCI repository link). The data represents 10 years (1999–2008) of clinical care at 130 US hospitals and integrated delivery networks. It includes over 50 attributes representing patient and hospital outcomes. Based on the insights from the previous exercises, I will:

1. Perform additional preprocessing;
2. Use *thresholding* when classifying;
3. Use (stratified) cross-validation with robust measures (area under ROC curve, F_1, etc.) rather than simple train-test splits whenever possible;
4. Test decision trees for statistically significant difference.

The attributes of the dataset are as follows:

- `encounter_id` (numeric): Unique identifier of an encounter.
- `patient_nbr` (numeric): Unique identifier of a patient.
- `race` (nominal): Values: Caucasian, Asian, African American, Hispanic, and other.
- `gender` (nominal): Values: male, female, and unknown/invalid.
- `age` (nominal): Grouped in 10-year intervals: $[0, 10), [10, 20), \ldots, [90, 100)$.
- `weight` (numeric): Weight in pounds.
- `admission_type_id` (nominal): Integer identifier corresponding to 9 distinct values, e.g., emergency, urgent, elective, newborn, and not available.
- `discharge_disposition_id` (nominal): Integer identifier corresponding to 29 distinct values, e.g., discharged to home, expired, and not available.
- `admission_source_id` (nominal): Integer identifier corresponding to 21 distinct values, e.g., physician referral, emergency room, and transfer from a hospital.
- `time_in_hospital` (numeric): Integer number of days between admission and discharge.
- `payer_code` (nominal): Integer identifier corresponding to 23 distinct values, e.g., Blue Cross/Blue Shield, Medicare, and self-pay.

- `medical_speciality` (nominal): Integer identifier of a specialty of the admitting physician, corresponding to 84 distinct values, e.g., cardiology, internal medicine, family/general practice, and surgeon.
- `num_lab_procedures` (numeric): Number of lab tests performed during the encounter.
- `num_procedures` (numeric): Number of procedures performed during the encounter.
- `num_medications` (numeric): Number of medications administered during the encounter.
- `number_outpatient` (numeric): Number of outpatient visits of the patient in the year preceding the encounter.
- `number_emergency` (numeric): Number of emergency visits of the patient in the year preceding the encounter.
- `number_inpatient` (numeric): Number of inpatient visits of the patient in the year preceding the encounter.
- `diag_1` (nominal): Primary diagnosis (coded as first three digits of ICD9); 848 values.
- `diag_2` (nominal): Secondary diagnosis (coded as first three digits of ICD9); 923 values.
- `diag_3` (nominal): Additional diagnosis (coded as first three digits of ICD9); 954 values.
- `number_diagnoses` (numeric): Number of diagnoses entered to the system.
- `max_glu_serum` (nominal): Indicates the range of the result or if the test was not taken. Values: "200," ">300," "normal," and "none" if not measured.
- `A1Cresult` (nominal): Indicates the range of the result or if the test was not taken. Values: ">8" if the result was greater than 8 %, ">7" if the result was greater than 7 % but less than 8 %, "normal" if the result was less than 7 %, and "none" if not measured.
- 24 attributes for medications (nominal): For the generic names: metformin, repaglinide, nateglinide, chlorpropamide, glimepiride, acetohexamide, glipizide, glyburide, tolbutamide, pioglitazone, rosiglitazone, acarbose, miglitol, troglitazone, tolazamide, examide, sitagliptin, insulin, glyburide-metformin, glipizide-metformin, glimepiride-pioglitazone, metformin-rosiglitazone, and metformin-pioglitazone, the feature indicates whether the drug was prescribed or there was a change in the dosage. Values: "up" if the dosage was increased during the encounter, "down" if the dosage was decreased, "steady" if the dosage did not change, and "no" if the drug was not prescribed.
- `change` (nominal): Indicates if there was a change in diabetic medications (either dosage or generic name). Values: "change" and "no change."
- `diabetesMed` (nominal): Indicates if there was any diabetic medication prescribed. Values: "yes" and "no."
- `readmitted` (nominal): Days to inpatient readmission. Values: "<30" if the patient was readmitted in less than 30 days, ">30" if the patient was readmitted in more than 30 days, and "No" for no record of readmission.

The goal is to predict the `readmitted` attribute, i.e., whether a readmission will occur. In particular, I will focus on *early* readmission (less than 30 days). In order to do so, the following steps will be performed:

1. Preprocessing and visualization (section 2);
2. Classification using decision trees, naïve Bayes, and k-nearest neighbors (section 3);
3. Clustering using k-means and DBSCAN (section 4);
4. Association rule mining using the apriori algorithm (section 5).

2 Preprocessing and visualization

In this section I will take a preliminary look into the data, preprocess it, and provide some insights through various visualizations.

2.1 Reading the data

I will begin by reading the dataset from the provided CSV file. Additionally, the `encounter_id` and `patient_nbr` attributes will be dropped as they are unique among patients. Alternative approaches in literature ([Str+14]) include keeping only *first* encounters. NA values are denoted by question marks (`?`) in the dataset.

```
set.seed(13)  # The seed is needed for reproducibility purposes.
df <- read.csv("diabetic_data.csv", na.strings="?")     # Read the data.
df <- subset(df, select=-c(encounter_id, patient_nbr))  # Drop unique ID's.
```

IDs which have the meaning of nominal attributes, i.e., those present in the `IDs_mapping.csv` file will be converted to factors as that is what they represent.

As already mentioned, I will be focusing on *early* readmission, so a patient will be considered readmitted if and only if he/she was readmitted within 30 days.

```
df <- transform(df, admission_type_id=as.factor(admission_type_id),
                discharge_disposition_id=as.factor(discharge_disposition_id),
                admission_source_id=as.factor(admission_source_id),
                readmitted=as.factor(ifelse(readmitted == "<30", "Readmitted", "Other")))
```

Let us now summarize the dataset in its present form:

```
str(df, width=75, strict.width="cut")
```

```
## 'data.frame': 101766 obs. of  48 variables:
##  $ race                    : Factor w/ 5 levels "AfricanAmerican",..: 3 3..
##  $ gender                  : Factor w/ 3 levels "Female","Male",..: 1 1 1..
##  $ age                     : Factor w/ 10 levels "[0-10)","[10-20)",..: 1..
##  $ weight                  : Factor w/ 9 levels "[0-25)","[100-125)",..: ..
##  $ admission_type_id       : Factor w/ 8 levels "1","2","3","4",..: 6 1 1..
##  $ discharge_disposition_id: Factor w/ 26 levels "1","2","3","4",..: 24 1..
##  $ admission_source_id     : Factor w/ 17 levels "1","2","3","4",..: 1 7 ..
##  $ time_in_hospital        : int  1 3 2 2 1 3 4 5 13 12 ...
##  $ payer_code              : Factor w/ 17 levels "BC","CH","CM",..: NA NA..
##  $ medical_specialty       : Factor w/ 72 levels "AllergyandImmunology",...
##  $ num_lab_procedures      : int  41 59 11 44 51 31 70 73 68 33 ...
##  $ num_procedures          : int  0 0 5 1 0 6 1 0 2 3 ...
##  $ num_medications         : int  1 18 13 16 8 16 21 12 28 18 ...
##  $ number_outpatient       : int  0 0 2 0 0 0 0 0 0 0 ...
##  $ number_emergency        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ number_inpatient        : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ diag_1                  : Factor w/ 716 levels "10","11","110",..: 125..
##  $ diag_2                  : Factor w/ 748 levels "11","110","111",..: NA..
##  $ diag_3                  : Factor w/ 789 levels "11","110","111",..: NA..
```

```
##  $ number_diagnoses        : int  1 9 6 7 5 9 7 8 8 8 ...
##  $ max_glu_serum           : Factor w/ 4 levels ">200",">300",..: 3 3 3 3..
##  $ A1Cresult               : Factor w/ 4 levels ">7",">8","None",..: 3 3 ..
##  $ metformin               : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ repaglinide             : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ nateglinide             : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ chlorpropamide          : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ glimepiride             : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ acetohexamide           : Factor w/ 2 levels "No","Steady": 1 1 1 1 1 ..
##  $ glipizide               : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ glyburide               : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ tolbutamide             : Factor w/ 2 levels "No","Steady": 1 1 1 1 1 ..
##  $ pioglitazone            : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ rosiglitazone           : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ acarbose                : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ miglitol                : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ troglitazone            : Factor w/ 2 levels "No","Steady": 1 1 1 1 1 ..
##  $ tolazamide              : Factor w/ 3 levels "No","Steady",..: 1 1 1 1..
##  $ examide                 : Factor w/ 1 level "No": 1 1 1 1 1 1 1 1 1 1 ..
##  $ citoglipton             : Factor w/ 1 level "No": 1 1 1 1 1 1 1 1 1 1 ..
##  $ insulin                 : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ glyburide.metformin     : Factor w/ 4 levels "Down","No","Steady",..: ..
##  $ glipizide.metformin     : Factor w/ 2 levels "No","Steady": 1 1 1 1 1 ..
##  $ glimepiride.pioglitazone: Factor w/ 2 levels "No","Steady": 1 1 1 1 1 ..
##  $ metformin.rosiglitazone : Factor w/ 2 levels "No","Steady": 1 1 1 1 1 ..
##  $ metformin.pioglitazone  : Factor w/ 2 levels "No","Steady": 1 1 1 1 1 ..
##  $ change                  : Factor w/ 2 levels "Ch","No": 2 1 2 1 1 2 1 ..
##  $ diabetesMed             : Factor w/ 2 levels "No","Yes": 1 2 2 2 2 2 2..
##  $ readmitted              : Factor w/ 2 levels "Other","Readmitted": 1 1..
```

2.2 Imputation

Since there are attributes with missing values, the first preprocessing step I'll do is to impute those attributes. Let's look at the percentages of values that are missing:

```
percent.missing <- colMeans(is.na(df))
names(percent.missing) <- colnames(df)
percent.missing
```

```
##                     race                   gender                      age
##             0.0223355541             0.0000000000             0.0000000000
##                   weight        admission_type_id discharge_disposition_id
##             0.9685847926             0.0000000000             0.0000000000
##      admission_source_id         time_in_hospital               payer_code
##             0.0000000000             0.0000000000             0.3955741603
##        medical_specialty       num_lab_procedures           num_procedures
##             0.4908220820             0.0000000000             0.0000000000
##          num_medications        number_outpatient         number_emergency
```

```
##             0.0000000000             0.0000000000             0.0000000000
##         number_inpatient                   diag_1                   diag_2
##             0.0000000000             0.0002063558             0.0035178743
##                   diag_3         number_diagnoses            max_glu_serum
##             0.0139830592             0.0000000000             0.0000000000
##                A1Cresult                metformin              repaglinide
##             0.0000000000             0.0000000000             0.0000000000
##              nateglinide           chlorpropamide              glimepiride
##             0.0000000000             0.0000000000             0.0000000000
##            acetohexamide                glipizide                glyburide
##             0.0000000000             0.0000000000             0.0000000000
##              tolbutamide             pioglitazone            rosiglitazone
##             0.0000000000             0.0000000000             0.0000000000
##                 acarbose                 miglitol             troglitazone
##             0.0000000000             0.0000000000             0.0000000000
##               tolazamide                  examide              citoglipton
##             0.0000000000             0.0000000000             0.0000000000
##                   insulin      glyburide.metformin      glipizide.metformin
##             0.0000000000             0.0000000000             0.0000000000
## glimepiride.pioglitazone  metformin.rosiglitazone   metformin.pioglitazone
##             0.0000000000             0.0000000000             0.0000000000
##                  change              diabetesMed               readmitted
##             0.0000000000             0.0000000000             0.0000000000
```

The `weight` attribute is not present in more than 95 % of the observation, whereas the medical specialty and payer code attributes are not present in around 40 % of the observations each. Such attributes will not be of much use to our data mining tasks, and imputing them can lead to meaningless values. Because of this, they will be dropped.

```
df <- subset(df, select=-c(weight, medical_specialty, payer_code))
```

For the remaining attributes, the imputation strategy will be:

- Numeric values will be imputed by their means;
- Non-numeric values will be imputed by random sampling with replacement from the available values.

```
#' Imputation of numeric attributes by mean and of non-numeric by SWR.
impute.missing <- function (attr) {
    if (is.numeric(attr))  # Numeric.
        attr[is.na(attr)] <- mean(attr, na.rm=TRUE)
    else  # Non-numeric.
        attr[is.na(attr)] <- sample(attr[!is.na(attr)],
                                    size=length(attr[is.na(attr)]), replace=TRUE)
    attr
}

df <- data.frame(lapply(df, impute.missing))
```

2.3 Grouping of nominal attributes

[Str+14] suggest that nominal attributes with many levels be grouped together. This greatly reduces the complexity of the task without inflicting a huge loss on the learning algorithms.

```
#' Grouping of diagnoses.
#' Diagnoses related to similar organs are grouped together (IC9 codes).
#' See http://www.hindawi.com/journals/bmri/2014/781670/tab3/
group.diags <- function (diags) {
    diags <- as.character(diags)
    factor(sapply(diags, function (diag) {
        if (!is.na(suppressWarnings(as.numeric(diag)))) {
            diag <- as.numeric(diag)
            if ((diag >= 390 && diag <= 459) || diag == 785)
                "Circulatory"
            else if ((diag >= 460 && diag <= 519) || diag == 786)
                "Respiratory"
            else if ((diag >= 520 && diag <= 579) || diag == 787)
                "Digestive"
            else if (diag >= 250 && diag < 251)
                "Diabetes"
            else if (diag >= 800 && diag <= 999)
                "Injury"
            else if (diag >= 710 && diag <= 739)
                "Musculoskeletal"
            else if ((diag >= 580 && diag <= 629) || diag == 788)
                "Genitourinary"
            else if ((diag >= 140 && diag <= 239) ||
                     diag == 780 || diag == 781 || diag == 784 ||
                     (diag >= 240 && diag <= 279) ||
                     (diag >= 680 && diag <= 709) || diag == 782 ||
                     (diag >= 1 && diag <= 139))
                "Neoplasms"
            else
                "Other"
        } else {
            "Other"
        }
    }),
    levels=c("Circulatory", "Respiratory", "Digestive", "Diabetes", "Injury",
             "Musculoskeletal", "Genitourinary", "Neoplasms", "Other"),
    ordered=FALSE)
}

#' Grouping of ages, discharges, and admissions.
#' The grouping is done based on distributional properties of readmission.
#' See http://www.hindawi.com/journals/bmri/2014/781670/tab3/
group.ages <- function (ages) {
    levels(ages) <- c(rep("[0-30)", 3), rep("[30-60)", 3), rep("[60-100)", 4))
```

```
    ages
}

group.discharges <- function (discharges) {
    levels(discharges) <- c("To home", rep("Other", 29)); discharges
}

group.admissions <- function (admission.sources) {
    levels(admission.sources) <-
        c("Because of physician/clinic referral", rep("Other", 4),
          "From emergency room", rep("Other", 18)); admission.sources
}

df <- transform(df, diag_1=group.diags(diag_1), diag_2=group.diags(diag_2),
                diag_3=group.diags(diag_3), age=group.ages(age),
                discharge_disposition_id=group.discharges(discharge_disposition_id),
                admission_source_id=group.admissions(admission_source_id))
```

2.4 Outlier removal

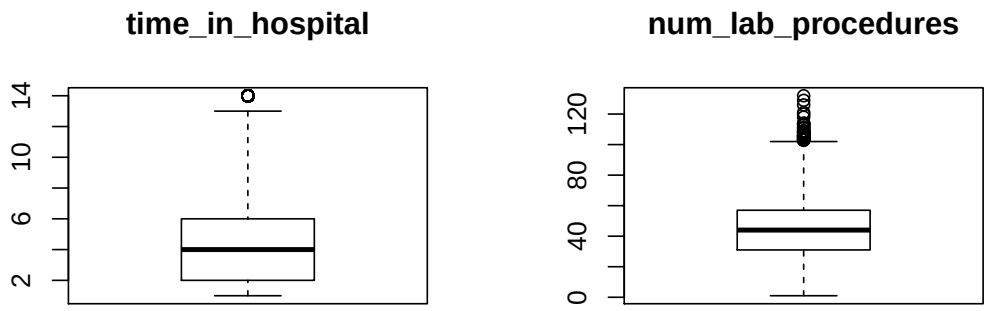
Outliers are associated to values outside interquartile ranges. More specifically, I will consider as an outlier everything outside of the

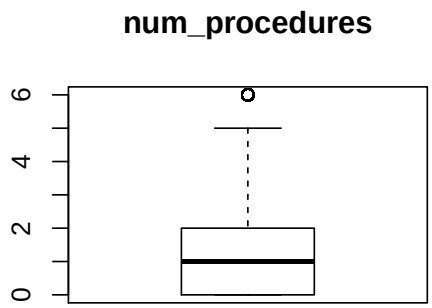
$$[\text{Q1} - 1.75 \cdot \text{IQR}, \text{Q3} + 1.75 \cdot \text{IQR}] \tag{1}$$

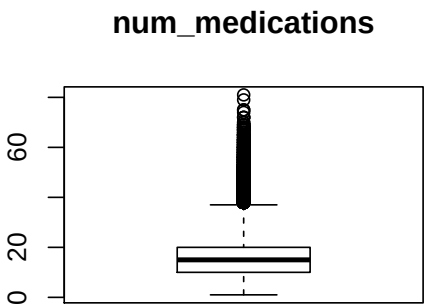
range, where Q1 is the first, Q3 the third quartile, and IQR the interquartile range. Note that many people use 1.5 instead of 1.75, but I will relax that condition.

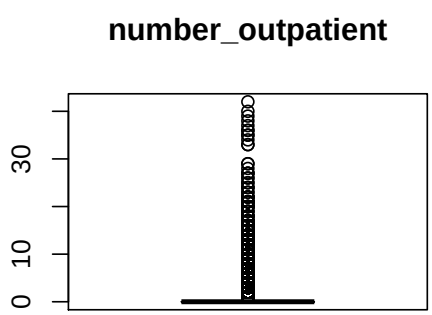
Boxplots are associated to quartiles, so let's analyze the boxplots of numeric attributes.

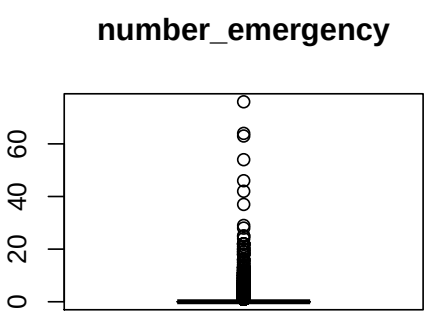
```
par(mfrow=c(1, 2))
for (name in colnames(df[sapply(df, is.numeric)]))
    boxplot(df[[name]], range=1.75, main=name)
```

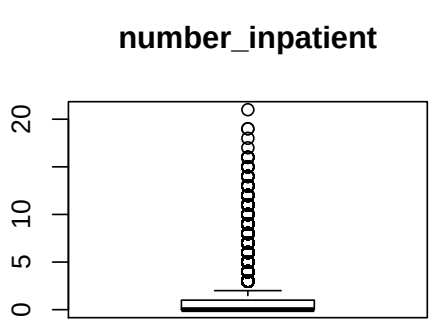


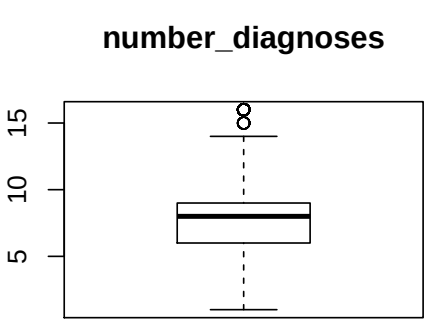

num_procedures
0
2
4
6


num_medications
0
20
60


number_outpatient
0
10
30


number_emergency
0
20
40
60


number_inpatient
0
5
10
20


number_diagnoses
5
10
15

The boxplots shows the presence of outliers – values far outside of the whiskers. Let's drop them.

```
for (name in colnames(df[sapply(df, is.numeric)])) {
    stats <- boxplot.stats(df[[name]], coef=1.75, do.conf=FALSE)
    # stats$out contains outlier values
    df <- df[!(df[[name]] %in% stats$out), ]
}
```

2.5 Attribute cleaning

The first cleaning step I'll perform is to drop attributes whose variance is *close to zero*. Such attributes will not be of much use to our data mining tasks, as they provide little to no information. Upon inspection it can be seen that such attributes will include `glyburide.metformin` and similar nominal attributes.

```
nzv.info <- nzv(df, saveMetrics=TRUE)  # Check for near-zero variance.
nzv.info
```

```
##                             freqRatio percentUnique zeroVar   nzv
## race                         3.756572   0.007295223   FALSE FALSE
## gender                       1.161646   0.004377134   FALSE FALSE
## age                          2.264239   0.004377134   FALSE FALSE
## admission_type_id            2.929000   0.011672357   FALSE FALSE
## discharge_disposition_id     1.559202   0.002918089   FALSE FALSE
## admission_source_id          2.391497   0.004377134   FALSE FALSE
## time_in_hospital             1.024570   0.018967580   FALSE FALSE
## num_lab_procedures           1.122239   0.148822551   FALSE FALSE
## num_procedures               2.288034   0.008754268   FALSE FALSE
## num_medications              1.017878   0.053984651   FALSE FALSE
## number_outpatient            0.000000   0.001459045    TRUE  TRUE
## number_emergency             0.000000   0.001459045    TRUE  TRUE
## number_inpatient             4.022469   0.004377134   FALSE FALSE
## diag_1                       1.909288   0.013131402   FALSE FALSE
## diag_2                       1.678901   0.013131402   FALSE FALSE
## diag_3                       1.645912   0.013131402   FALSE FALSE
## number_diagnoses             3.277051   0.020426625   FALSE FALSE
## max_glu_serum               45.360996   0.005836178   FALSE  TRUE
## A1Cresult                    9.610363   0.005836178   FALSE FALSE
## metformin                    4.260428   0.005836178   FALSE FALSE
## repaglinide                 79.709906   0.005836178   FALSE  TRUE
## nateglinide                161.736342   0.005836178   FALSE  TRUE
## chlorpropamide            1069.828125   0.005836178   FALSE  TRUE
## glimepiride                 20.501104   0.005836178   FALSE  TRUE
## acetohexamide                0.000000   0.001459045    TRUE  TRUE
## glipizide                    7.875246   0.005836178   FALSE FALSE
## glyburide                    9.381318   0.005836178   FALSE FALSE
## tolbutamide               3606.263158   0.002918089   FALSE  TRUE
## pioglitazone                14.134943   0.005836178   FALSE FALSE
```

```
## rosiglitazone                15.743571   0.005836178   FALSE FALSE
## acarbose                    402.111765   0.004377134   FALSE  TRUE
## miglitol                   3114.318182   0.004377134   FALSE  TRUE
## troglitazone              22845.000000   0.002918089   FALSE  TRUE
## tolazamide                 2014.794118   0.004377134   FALSE  TRUE
## examide                       0.000000   0.001459045    TRUE  TRUE
## citoglipton                   0.000000   0.001459045    TRUE  TRUE
## insulin                       1.562386   0.005836178   FALSE FALSE
## glyburide.metformin         140.881988   0.005836178   FALSE  TRUE
## glipizide.metformin       11422.000000   0.002918089   FALSE  TRUE
## glimepiride.pioglitazone  68537.000000   0.002918089   FALSE  TRUE
## metformin.rosiglitazone   68537.000000   0.002918089   FALSE  TRUE
## metformin.pioglitazone    68537.000000   0.002918089   FALSE  TRUE
## change                        1.281102   0.002918089   FALSE FALSE
## diabetesMed                   3.095733   0.002918089   FALSE FALSE
## readmitted                    9.449459   0.002918089   FALSE FALSE
```

```
df <- df[, !nzv.info$nzv]   # Drop any attributes with near-zero variance.
```

Next, I will analyze the correlations (Pearson's r coefficient) between numeric attributes. High correlation means linear dependence between attributes, or, in other words, redundancy. Such redundancy will only slow our algorithms down without making them any better (sometimes it might even make them worse).

The correlations will be tested for statistical significance.

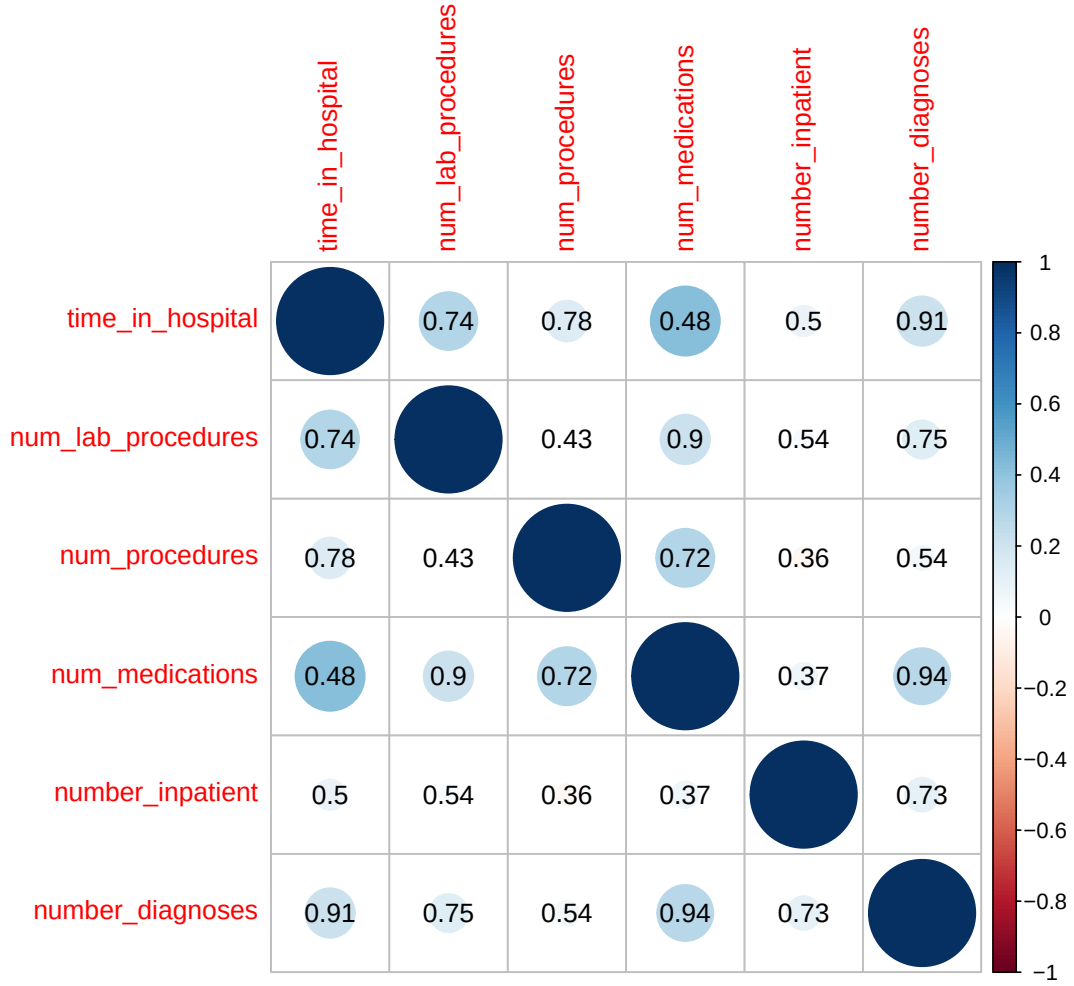
```
#' p-value and confidence interval estimation for correlation matrices.
cor.mat.test <- function (cor.mat, conf.level=0.95) {
    cor.mat <- as.matrix(cor.mat)
    n <- ncol(cor.mat)
    p.mat <- lowCI.mat <- uppCI.mat <- matrix(NA, n, n)
    diag(p.mat) <- 0
    diag(lowCI.mat) <- diag(uppCI.mat) <- 1
    for (i in 1:(n - 1)) {
        for (j in (i + 1):n) {
            tmp <- cor.test(cor.mat[, i], cor.mat[, j], conf.level=conf.level)
            p.mat[i, j] <- p.mat[j, i] <- tmp$p.value
            lowCI.mat[i, j] <- lowCI.mat[j, i] <- tmp$conf.int[1]
            uppCI.mat[i, j] <- uppCI.mat[j, i] <- tmp$conf.int[2]
        }
    }
    list(p.mat=p.mat, lowCI.mat=lowCI.mat, uppCI.mat=uppCI.mat)
}

cor.mat <- cor(df[sapply(df, is.numeric)], method="pearson"); cor.mat
```

```
##                    time_in_hospital num_lab_procedures num_procedures
## time_in_hospital          1.0000000        0.296154442    0.149418746
## num_lab_procedures        0.2961544        1.000000000    0.004343638
## num_procedures            0.1494187        0.004343638    1.000000000
## num_medications           0.4228224        0.217371384    0.299407591
```

```
## number_inpatient          0.0838182        0.032548571   -0.040495053
## number_diagnoses          0.2174980        0.130352844    0.053274185
##                     num_medications number_inpatient number_diagnoses
## time_in_hospital         0.42282243       0.08381820       0.21749801
## num_lab_procedures       0.21737138       0.03254857       0.13035284
## num_procedures           0.29940759      -0.04049505       0.05327419
## num_medications          1.00000000       0.06448034       0.27846171
## number_inpatient         0.06448034       1.00000000       0.10289994
## number_diagnoses         0.27846171       0.10289994       1.00000000
```

```
corrplot(cor.mat, p.mat=cor.mat.test(cor.mat)$p.mat, insig="p-value", sig.level=0.05)
```



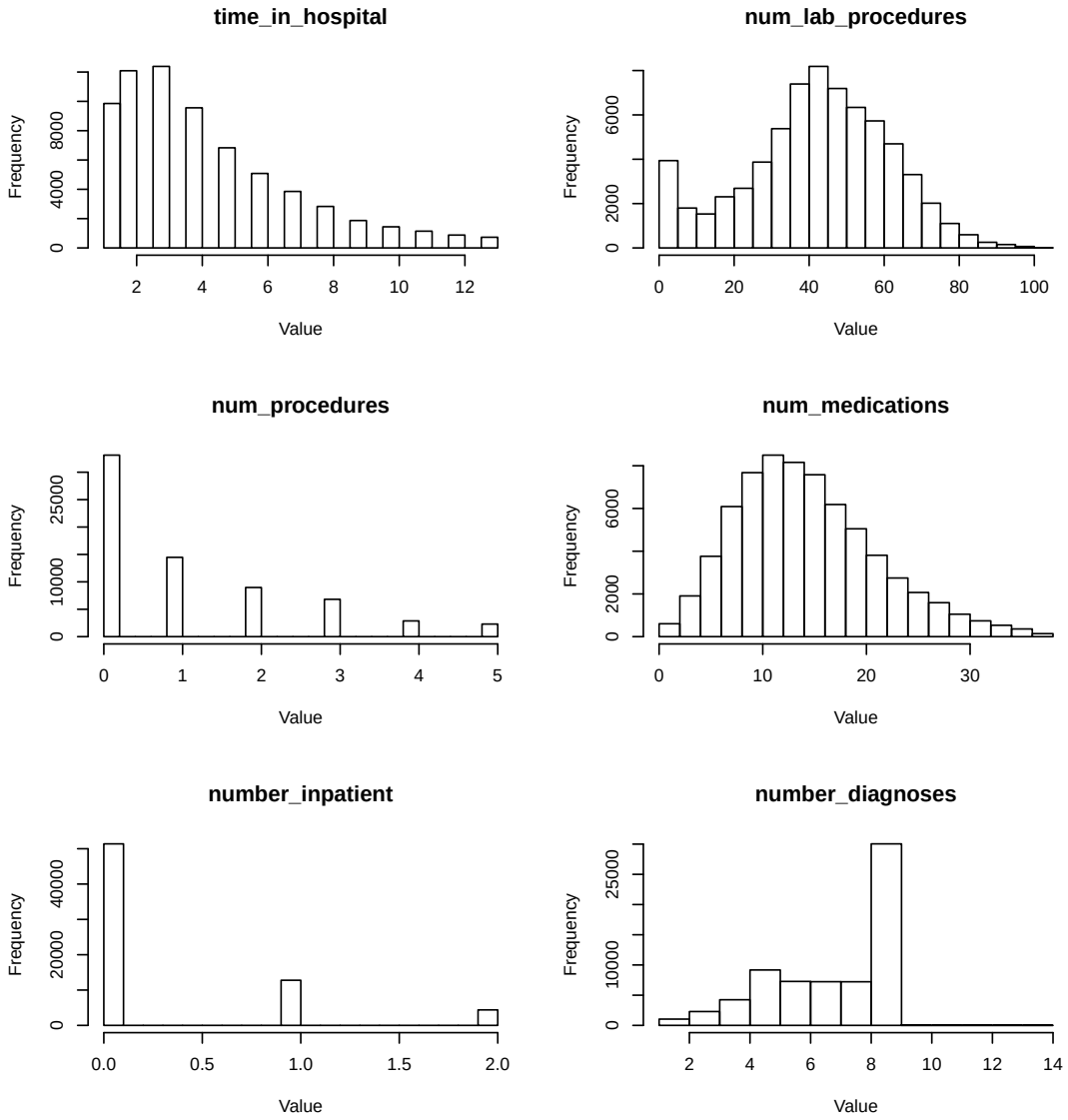
There are not any highly correlated attributes, which is a good sign. Reassuringly, the p-values are high enough and support the null hypothesis that the correlations are 0. The highest correlations

are exhibited by attributes related to hospital stay and medications (as expected – longer stays usually lead to more medication), but both are less than 0.5.

2.6 Transformation

Numeric attributes should be analyzed for *skew* in their distributions. Skews are generally undesirable and should be corrected via numeric transformations.

```
par(mfrow=c(3, 2))
for (name in colnames(df[sapply(df, is.numeric)]))
    hist(df[[name]], main=name, xlab="Value")
```



It can be seen that some attributes have skewed distributions, so in addition to *centering* and *scaling* I will perform a Box–Cox transformation[1] [BC64] of the data, which is useful in correcting skewed distributions. Centering and scaling together are also called z-score normalization.

More importantly, note that the class distribution is very imbalanced, i.e., there are very few positive observations compared to negative.

```
table(df$readmitted)
```

```
##
##      Other Readmitted
##      61979       6559
```

To attempt to compensate for this, I will use a hybrid oversampling technique called SMOTE [Cha+02]. SMOTE oversamples the minority class by synthesizing new observations.

```
#' Wrap a mlr learning algorithm in a SMOTE preprocessor that boosts
#' the minority class by a rate of 5 and considers 10 nearest-neighbors
#' when synthesizing observations.
#' Additionally, perform centering, scaling, and Box--Cox transformation.
wrap.learner <- function (base.learner)
    makePreprocWrapperCaret(makeSMOTEWrapper(base.learner, sw.rate=5, sw.nn=10),
                            ppc.center=TRUE, ppc.scale=TRUE, ppc.BoxCox=TRUE)
```

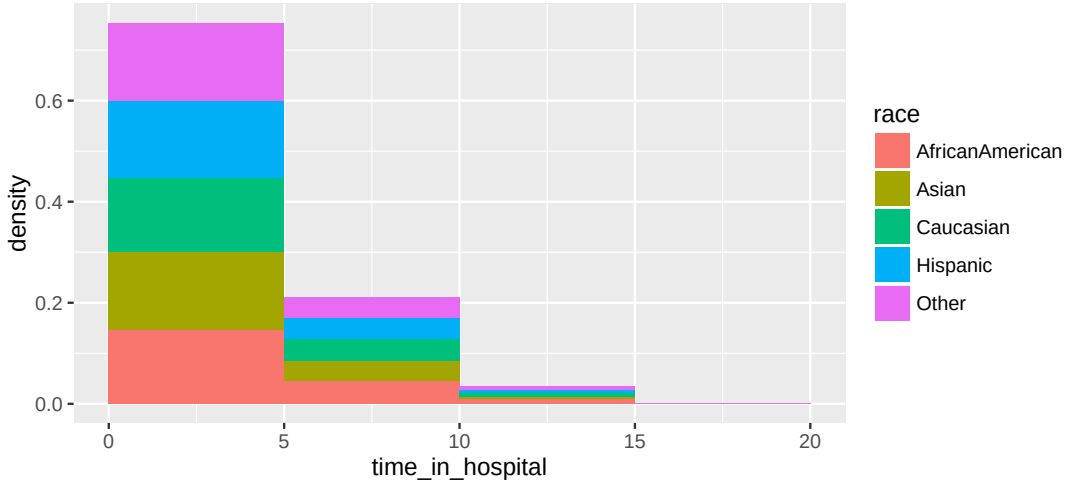
There is no point in further dimensionality reduction as PCA analysis shows that we need to retain all components in order to capture a significant portion of the variance.

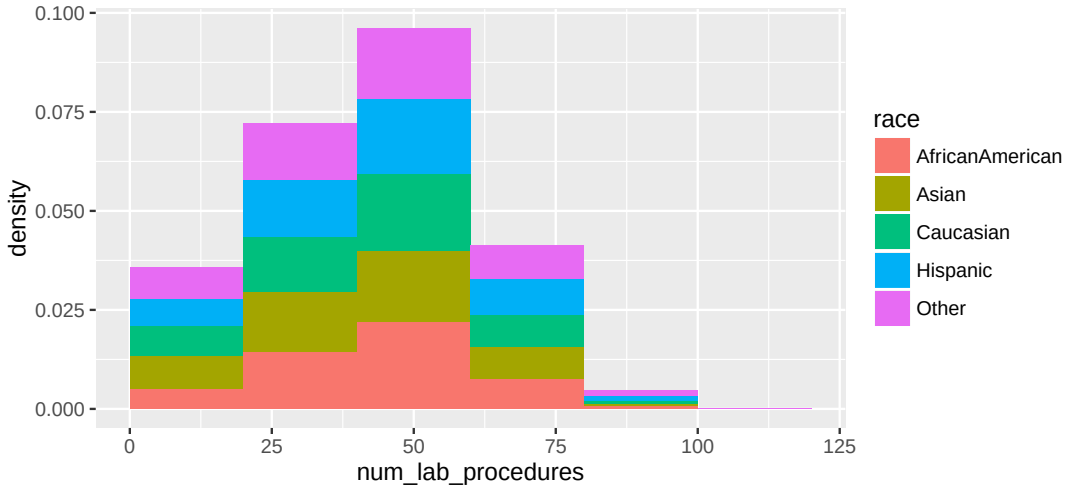
2.7 Visualization

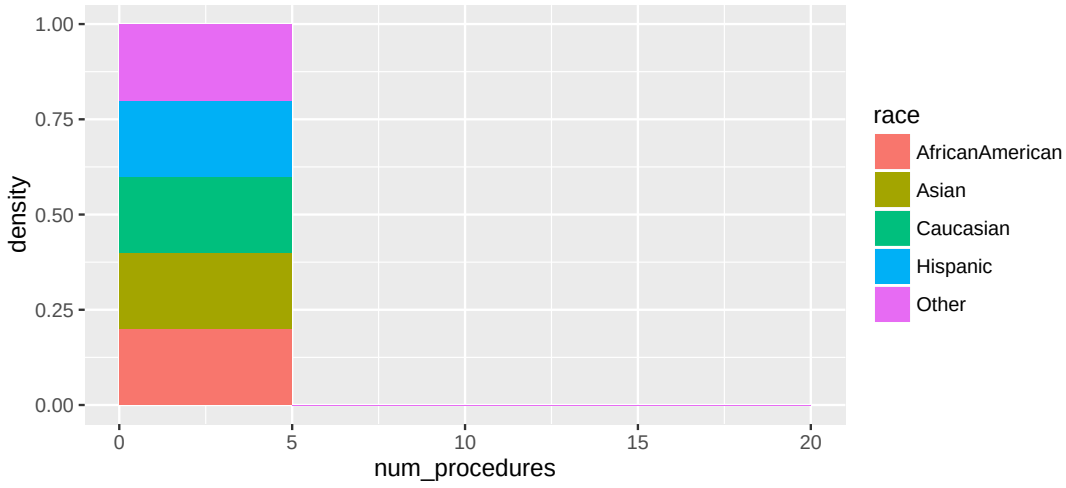
An interesting concept to explore is whether patients receive roughly the same amount of medical care regardless of race. We can do this by overlaying race information on top of histograms.

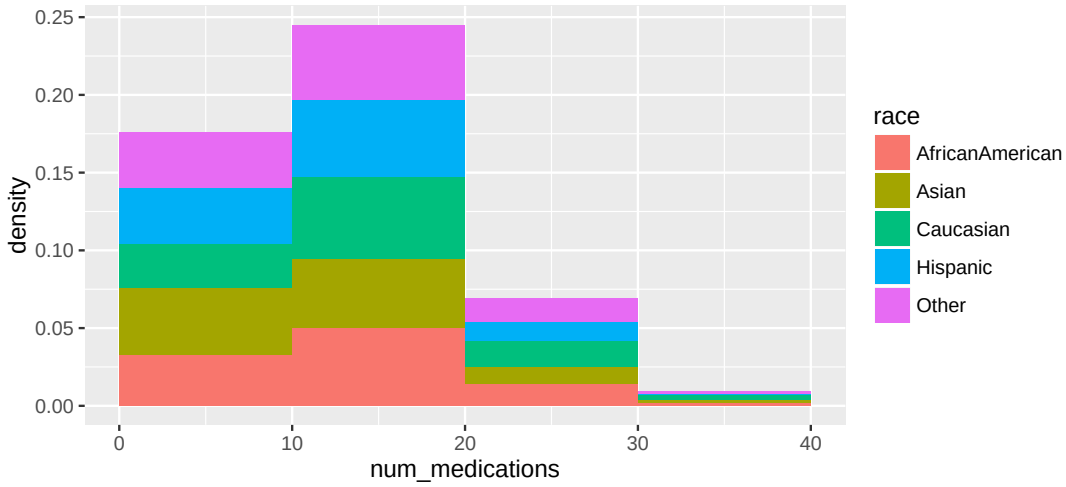
```
for (name in colnames(df[sapply(df, is.numeric)])) {
    attr <- df[[name]]
    breaks <- pretty(range(attr, n=nclass.Sturges(attr), min.n=1))
    print(ggplot(df, aes_q(x=as.name(name))) +
          stat_bin(aes(y=..density.., fill=race), breaks=breaks))
}
```

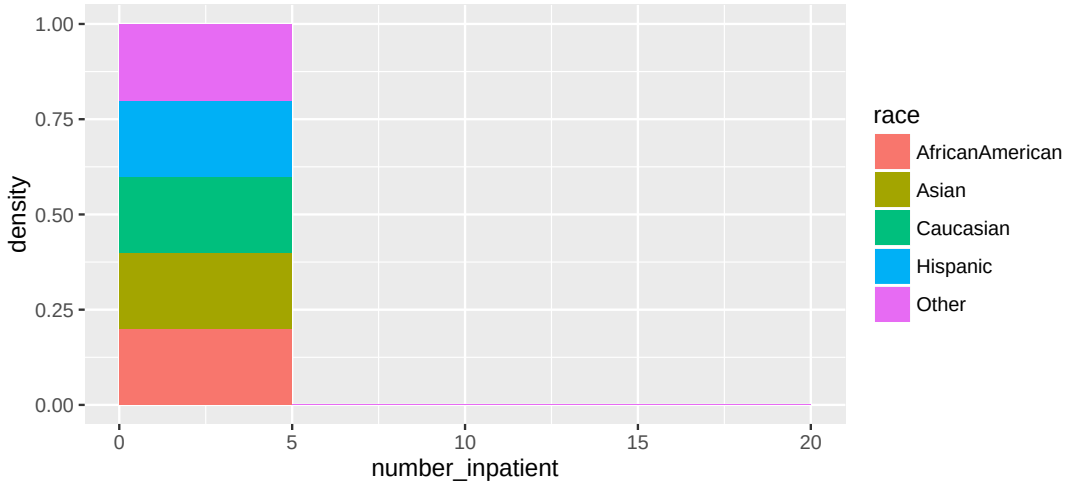
[1] A so-called power transform.

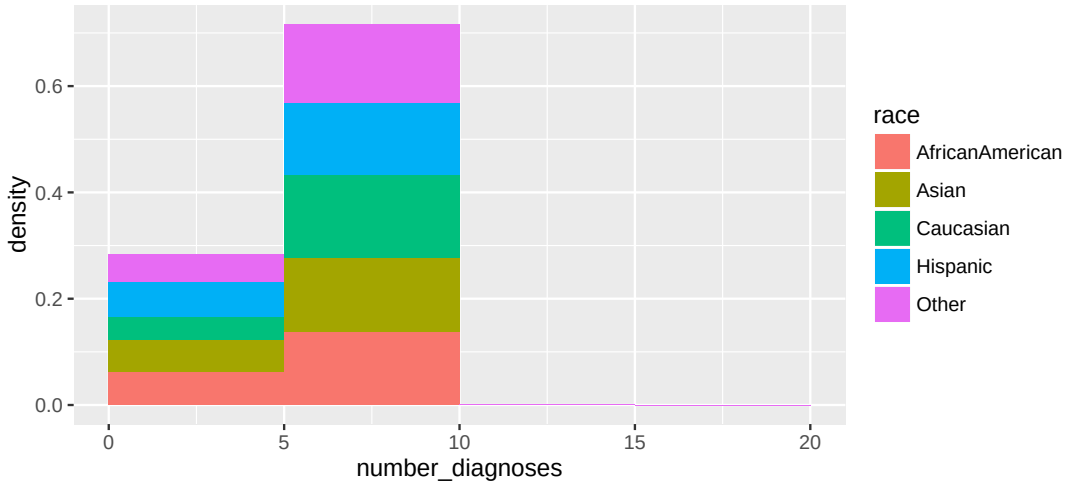

density
0.6
0.4
0.2
0.0
0
5
10
15
20
time_in_hospital
race
AfricanAmerican
Asian
Caucasian
Hispanic
Other


density
0.100
0.075
0.050
0.025
0.000
0
25
50
75
100
125
num_lab_procedures
race
AfricanAmerican
Asian
Caucasian
Hispanic
Other


density
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
num_procedures
race
AfricanAmerican
Asian
Caucasian
Hispanic
Other

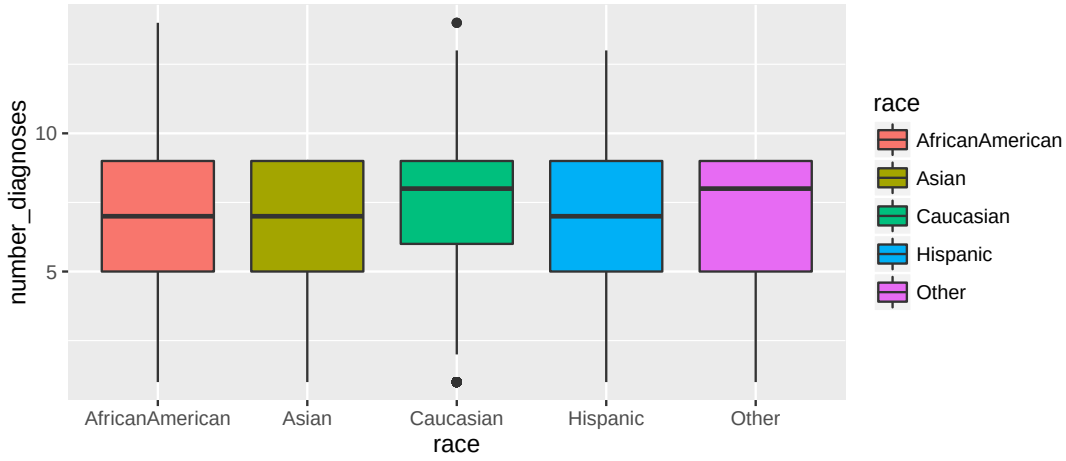

0.25
0.20
0.15
0.10
0.05
0.00
density
0
10
20
30
40
num_medications
race
AfricanAmerican
Asian
Caucasian
Hispanic
Other


1.00
0.75
0.50
0.25
0.00
density
0
5
10
15
20
number_inpatient
race
AfricanAmerican
Asian
Caucasian
Hispanic
Other


0.6
0.4
0.2
0.0
density
0
5
10
15
20
number_diagnoses
race
AfricanAmerican
Asian
Caucasian
Hispanic
Other

It can be seen that the densities are very uniform, i.e., each person gets roughly the same amount of medical care no matter what his/her race is. *However*, a boxplot shows that Caucasian patients receive at the median more diagnoses than do other patients.

```
ggplot(df, aes(x=race, y=number_diagnoses)) + geom_boxplot(aes(fill=race))
```



We are now free to continue with classification, clustering, and association rule mining.

3 Classification

Classification will be performed using decision trees, naïve Bayes, and k-nearest neighbors.

Whenever possible, algorithms will be set to predict *probabilities* rather than classes, i.e., the probability that a certain observation belongs to a given class. This allows *thresholding* and *ROC analysis* to be performed.

Our go-to measures of goodness will be the area under the ROC curve and the F_1 score. The F_1 score is defined as

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

When thresholding, I will use a personally defined measure (denoted bm) as a "balanced" score between accuracy and F_1:

$$\text{bm} = \frac{\text{accuracy} + 2 \cdot F_1}{3}. \tag{2}$$

The code to create the `mlr` classification task, resampling strategy, and balanced measure follows:

```
# Classification task creation.
classif.task <- makeClassifTask(
    data=df, target="readmitted", positive="Readmitted")

# 5-fold startified x-val for performance measurement.
classif.rdesc <- makeResampleDesc("CV", iters=5, stratify=TRUE)
```

```r
# Train and test sets.
classif.n <- getTaskSize(classif.task)
classif.train <- sample(classif.n, size=round(4/5 * classif.n))
classif.test <- setdiff(seq_len(classif.n), classif.train)

# Balanced measure.
bm <- makeMeasure(id="bm", name="Balanced measure",
                  properties=c("classif", "req.pred", "req.truth"),
                  minimize=FALSE, worst=0, best=1,
                  fun=function (...) (acc$fun(...) + 2 * f1$fun(...)) / 3)
```

3.1 Decision trees

I will present and compare two different methods for building decision trees:

- CART [Bre+84] provided by the rpart package. This algorithm uses the Gini index when deciding which attribute to split by.
- J48 (C4.5) [Qui93] provided by the RWeka package. This algorithm uses information gain when deciding which attribute to split by.

```r
# Create the CART learner.
rpart.lrn <- makeLearner("classif.rpart", predict.type="prob")
rpart.lrn <- setHyperPars(rpart.lrn, par.vals=list(xval=10))  # 10-fold x-val.
rpart.lrn <- wrap.learner(rpart.lrn)

# Create the J48 learner.
j48.lrn <- makeLearner("classif.J48", predict.type="prob")    # 3-fold x-val.
j48.lrn <- wrap.learner(j48.lrn)
```

These decision trees can now be trained and have their performance assessed. The internal cross-validation is used for *pruning* and *parameter tuning*. Let's see what CART can do:

```r
# Run cross-validation using a default threshold of 50%.
rpart.res <- resample(rpart.lrn, classif.task, classif.rdesc,
                      measures=list(f1, auc, acc))
rpart.res$aggr
```

```
##  f1.test.mean auc.test.mean acc.test.mean
##     0.0000000     0.5894739     0.9041845
```

Unfortunately, due to the class imbalance, using a default threshold of 0.5 leads to an F_1 score of almost 0, despite the accuracy being as high as 90 %. In order to remedy that, I will tune the threshold:

```r
# Fit the model on the training and testing sets.
rpart.mod <- train(rpart.lrn, task=classif.task, subset=classif.train)
rpart.pred <- predict(rpart.mod, task=classif.task, subset=classif.test)

# Tune the threshold using the previously-defined balanced measure.
rpart.thresh <- tuneThreshold(rpart.pred, measure=bm)
```

```
rpart.pred <- setThreshold(rpart.pred, rpart.thresh$th)

# Check the results.
performance(rpart.pred, measures=list(tpr, fpr, acc, f1, auc, bm))
```

```
##        tpr        fpr        acc         f1        auc         bm
## 0.10961969 0.06169645 0.85723665 0.13060862 0.60261508 0.37281796
```

```
rpart.pred$time
```

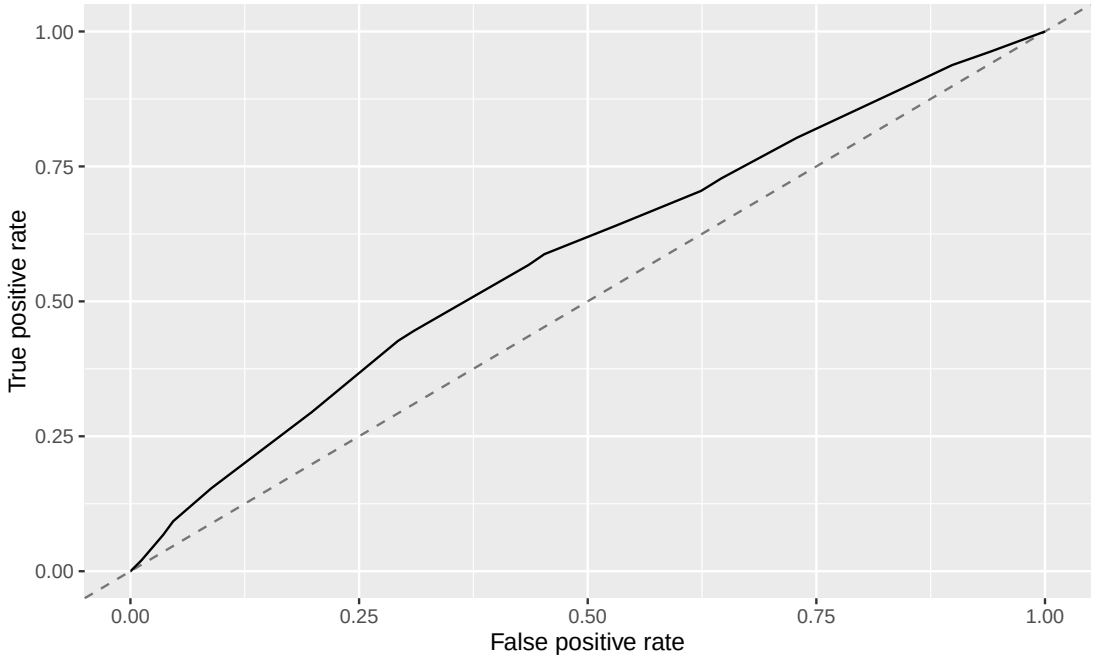
```
## [1] 0.142
```

```
getConfMatrix(rpart.pred)
```

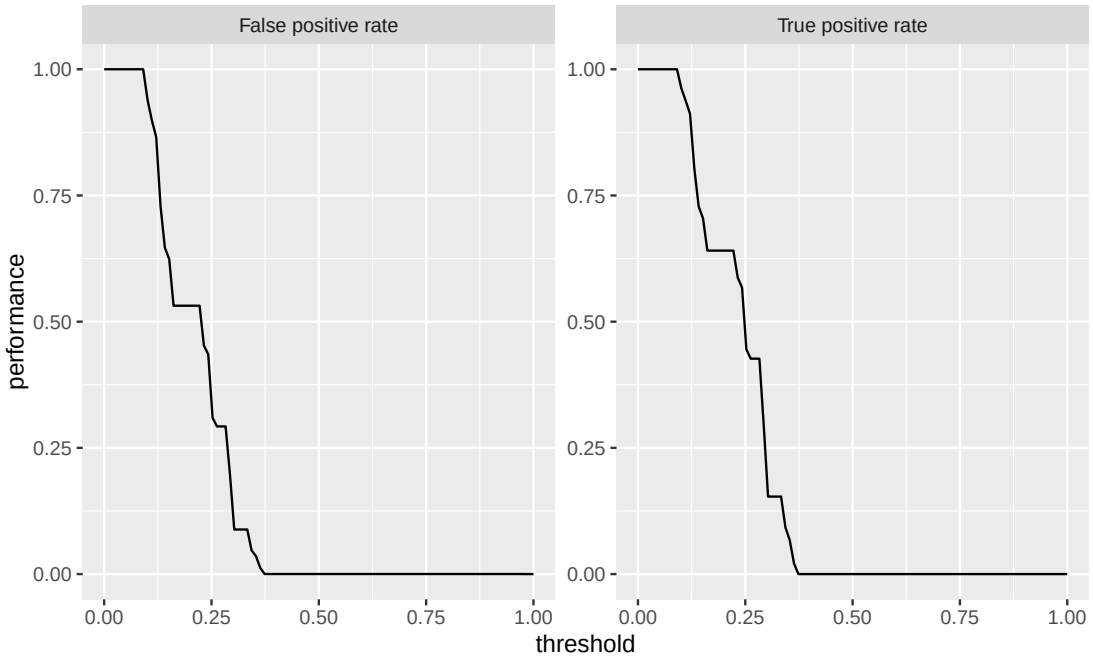
```
##             predicted
## true         Other Readmitted -SUM-
##   Other      11604        763   763
##   Readmitted  1194        147  1194
##   -SUM-       1194        763  1957
```

Given the imbalance of the dataset, CART does quite well. It is, however, quite "aggressive." Analyzing its ROC and threshold curves let us observe some more general results and patterns:

```
rpart.tvp <- generateThreshVsPerfData(rpart.res, measures=list(fpr, tpr))
plotROCCurves(rpart.tvp)
```




```
plotThreshVsPerf(rpart.tvp)
```



We now repeat these steps using the J48 algorithm.

```
# Run cross-validation using a default threshold of 50%.
j48.res <- resample(j48.lrn, classif.task, classif.rdesc,
                    measures=list(f1, auc, acc))
j48.res$aggr
```

```
##  f1.test.mean auc.test.mean acc.test.mean
##    0.01251485    0.57766670    0.90012840
```

The J48 algorithm performs similarly to CART. We can test their accuracies for statistical significance using a t-test for dependent data (as we are fitting our models on the exact same data). The t statistic is given by

$$t = \frac{\bar{X_D} - \mu_0}{\frac{s_D}{\sqrt{n}}},$$

where X_D is a random variable denoting the difference in errors.

```
measure.err <- function (lrn, task, rinst)
    function (rind) {
        mod <- train(lrn, task, subset=rinst$train.inds[[rind]])
        pred <- predict(mod, task, subset=rinst$test.inds[[rind]])
        mmce$fun(pred=pred)
    }

iters <- 15
classif.rinst <- makeResampleInstance(
    makeResampleDesc("CV", iters=iters, stratify=TRUE), classif.task)
rpart.errs <- sapply(1:iters,
```

```
                    measure.err(rpart.lrn, classif.task, classif.rinst))
j48.errs <- sapply(1:iters,
                   measure.err(j48.lrn, classif.task, classif.rinst))

t.test(rpart.errs, j48.errs, paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  rpart.errs and j48.errs
## t = -10.007, df = 14, p-value = 9.256e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.005474769 -0.003542247
## sample estimates:
## mean of the differences
##            -0.004508508
```

Despite the small p-value causing us to reject the null hypothesis, we see that the confidence interval is very close to 0. Thus, we can consider CART and J48 to be more or less equivalent in terms of accuracy.

Next, we attempt to tune J48's threshold.

```
# Fit the model on the training and testing sets.
j48.mod <- train(j48.lrn, task=classif.task, subset=classif.train)
j48.pred <- predict(j48.mod, task=classif.task, subset=classif.test)

# Tune the threshold using the previously-defined balanced measure.
j48.thresh <- tuneThreshold(j48.pred, measure=bm)
j48.pred <- setThreshold(j48.pred, j48.thresh$th)

# Check the results.
performance(j48.pred, measures=list(tpr, fpr, acc, f1, auc, bm))
```

```
##        tpr        fpr        acc         f1        auc         bm
## 0.15585384 0.09315113 0.83338197 0.15470022 0.59225217 0.38092747
```

```
j48.pred$time
```

```
## [1] 0.377
```

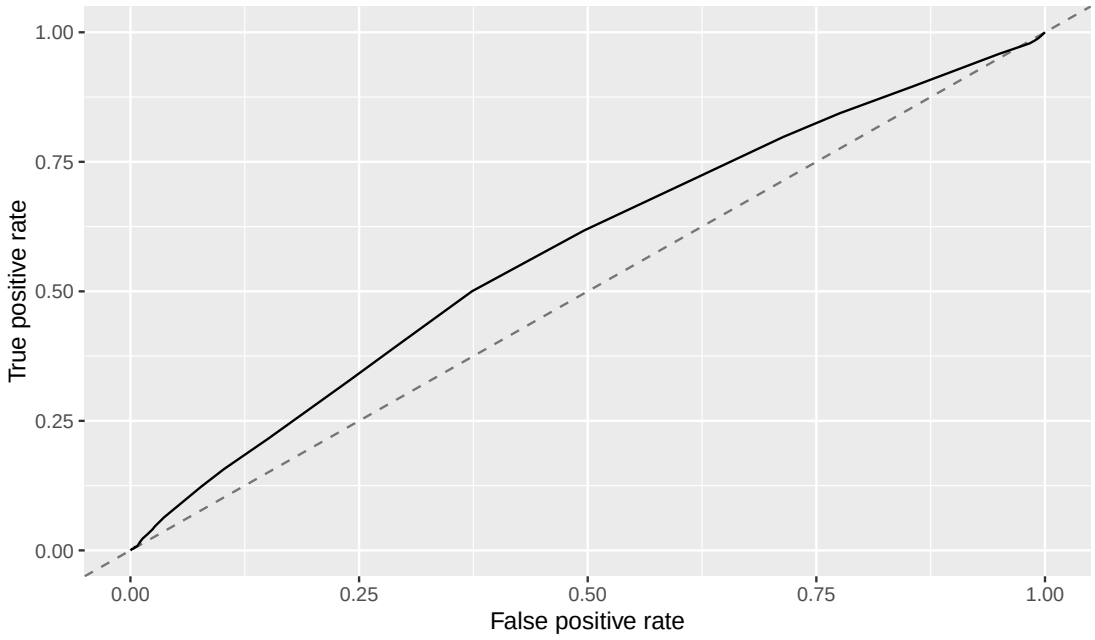
```
getConfMatrix(j48.pred)
```

```
##             predicted
## true         Other Readmitted -SUM-
##   Other      11215       1152  1152
##   Readmitted  1132        209  1132
##   -SUM-       1132       1152  2284
```

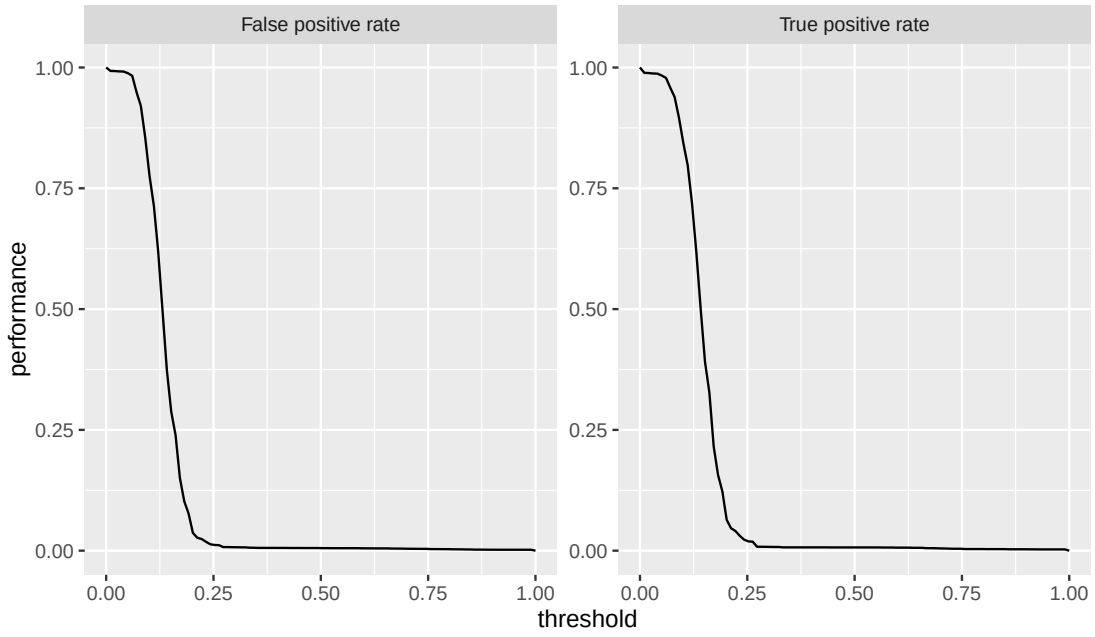
J48 appears to be more "conservative" when classifying positive samples, whereas CART more "aggressive." This results in J48 having greater overall accuracy, but smaller true positive rate. The true- and false-positive rates can be analyzed by looking at the ROC curves.

As the figures below show, the threshold curves of J48 are much smoother compared to those of CART, i.e., there are no flat regions and the spikes are not as prominent.

```
j48.tvp <- generateThreshVsPerfData(j48.res, measures=list(fpr, tpr))
plotROCCurves(j48.tvp)
```

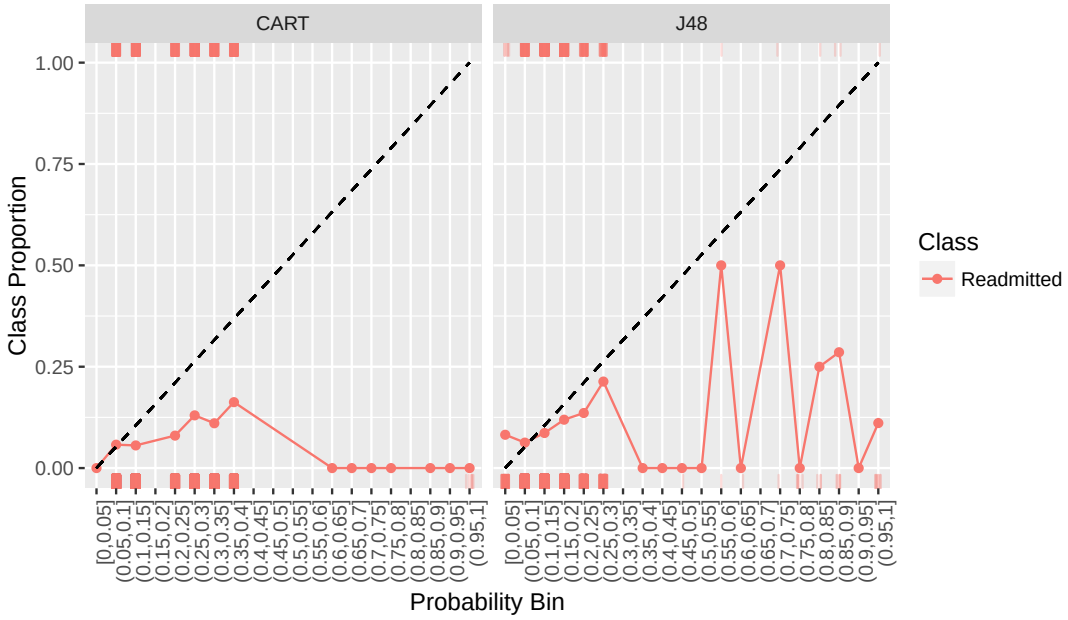



```
plotThreshVsPerf(j48.tvp)
```



At the end, I will perform calibration plots of both CART and J48. These plots visualize estimated class probabilities against the observed frequencies.

```
plotCalibration(generateCalibrationData(list(CART=rpart.pred, J48=j48.pred)))
```



Again we see that the probabilities in J48 are much more uniformly distributed, explaining the smoother threshold curves.

3.2 Naïve Bayes

I will create and evaluate the naïve Bayes classifier by repeating the procedure with the decision trees. Naïve Bayes assumes conditional independence between the attributes given the class, and makes a prediction

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\arg\max} \Pr(C_k) \prod_{i=1}^{K} \Pr(x_i \mid C_i).$$

```
# Create the naïve Bayes learner.
nb.lrn <- makeLearner("classif.naiveBayes", predict.type="prob")
nb.lrn <- wrap.learner(nb.lrn)
```

Cross-validation is used to assess preliminary performance:

```
# Run cross-validation using a default threshold of 50%.
nb.res <- resample(nb.lrn, classif.task, classif.rdesc,
                   measures=list(f1, auc, acc))
nb.res$aggr
```

```
##  f1.test.mean auc.test.mean acc.test.mean
##     0.1991073     0.6064824     0.7452507
```

Interestingly, naïve Bayes has a higher F_1 score "out of the box," but a lower accuracy. Perhaps it can be made even better with threshold tuning.

```
# Fit the model on the training and testing sets.
nb.mod <- train(nb.lrn, task=classif.task, subset=classif.train)
nb.pred <- predict(nb.mod, task=classif.task, subset=classif.test)

# Tune the threshold using the previously-defined balanced measure.
nb.thresh <- tuneThreshold(nb.pred, measure=bm)
nb.pred <- setThreshold(nb.pred, nb.thresh$th)

# Check the results.
performance(nb.pred, measures=list(tpr, fpr, acc, f1, auc, bm))
```

```
##       tpr       fpr       acc        f1       auc        bm
## 0.2311708 0.1283254 0.8090166 0.1914762 0.5990474 0.3973230
```

```
nb.pred$time
```

```
## [1] 30.703
```

```
getConfMatrix(nb.pred)
```

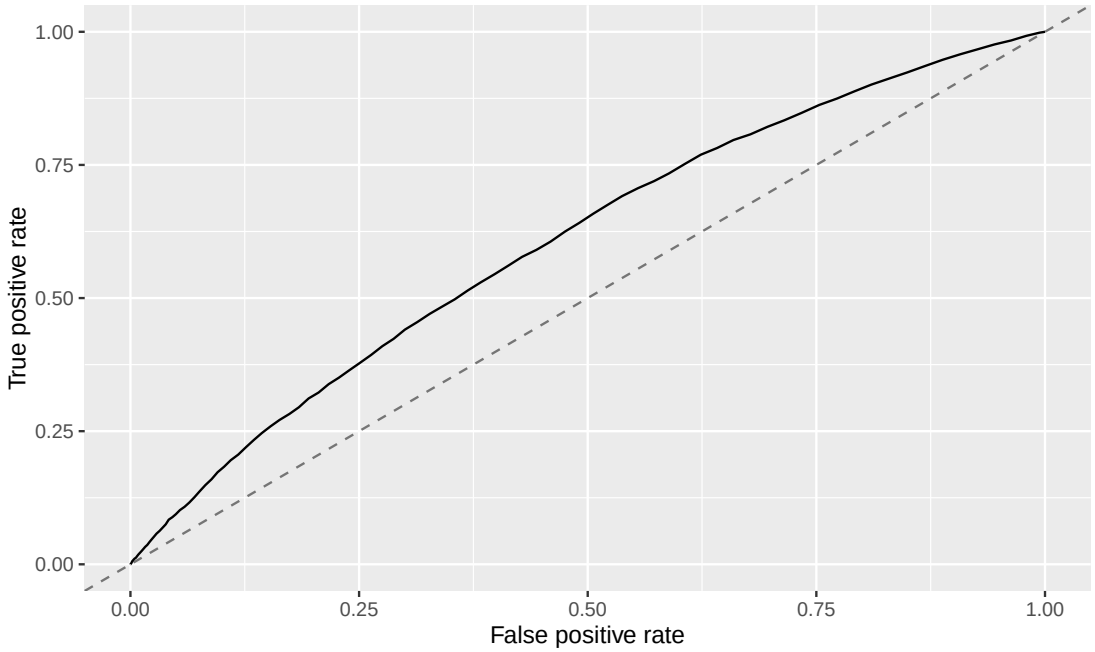
```
##             predicted
## true         Other Readmitted -SUM-
##   Other      10780       1587  1587
##   Readmitted  1031        310  1031
##   -SUM-       1031       1587  2618
```

Unfortunately, we see that optimizing the balanced measure as defined in eq. (2) lowers the F_1 score in order to compensate for accuracy. Of course, we manage to achieve a decent accuracy of 80 % (compared to the previous 75 %) while maintaining the F_1 score at around 0.2.

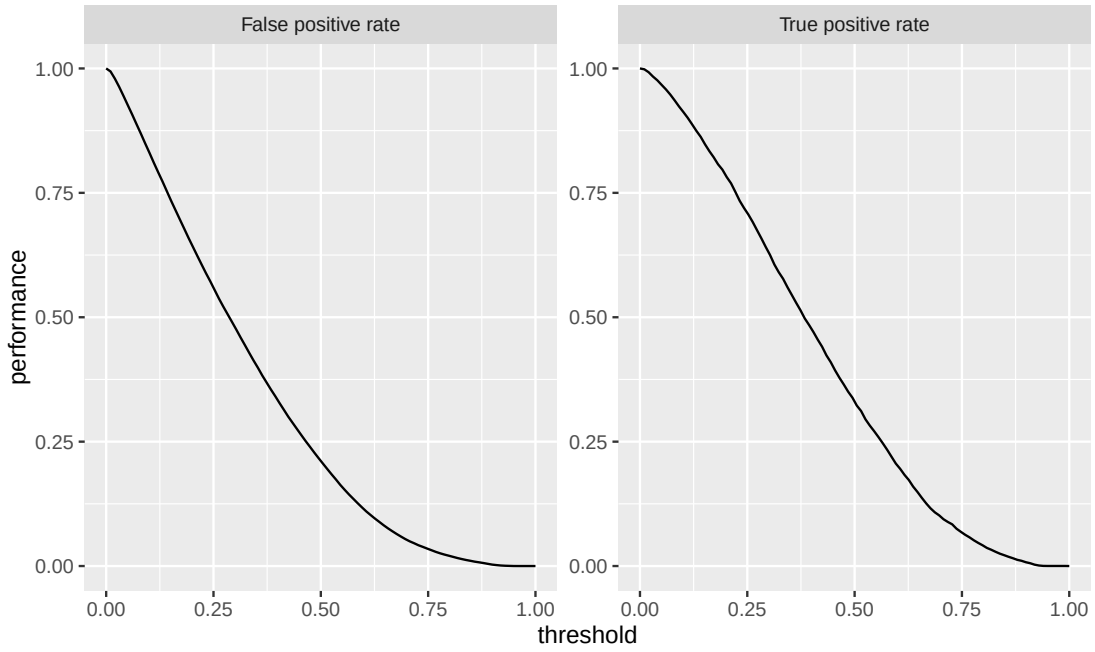
Another point is that naïve Bayes takes a much longer time to classify than the tree-based algorithms. However, this is expected.

The ROC curve and calibration plots wrap up our discussion on naïve Bayes. We observe some interesting patterns that are summarized below.

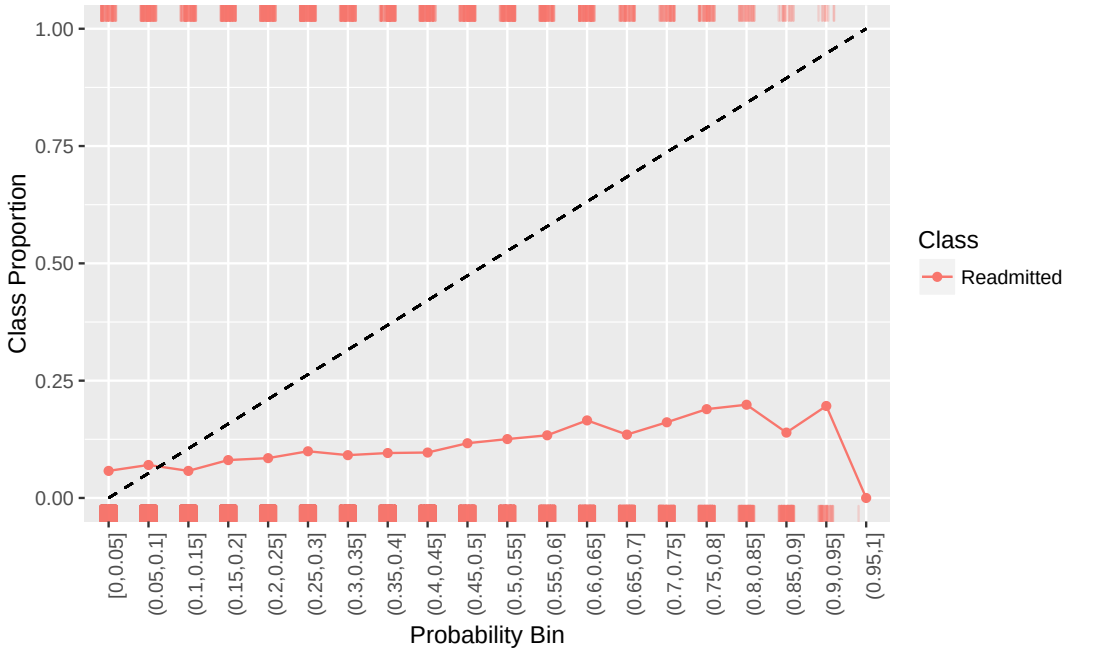
```
nb.tvp <- generateThreshVsPerfData(nb.res, measures=list(fpr, tpr))
plotROCCurves(nb.tvp)
```




```
plotThreshVsPerf(nb.tvp)
```



```
plotCalibration(generateCalibrationData(nb.pred))
```



Naïve Bayes appears to be a much stabler and a very "smooth" algorithm. The curves have almost no spikes and the frequencies are distributed very uniformly.

3.3 k-nearest neighbors

I will be using Weka's implementation of kNN. For this purpose, I will first write the preprocessed data in Weka's ARFF format.

```
library(RWeka)
write.arff(getTaskData(classif.task, subset=classif.train),
           "diabetic_data-train.arff")  # Training data.
write.arff(getTaskData(classif.task, subset=classif.test),
           "diabetic_data-test.arff")   # Testing data.
```

Next, I will load that data into Weka and use kNN coupled with SMOTE. In particular, I will use the same SMOTE settings as previously (oversample by a factor of 5, consider 10 nearest-neighbors).

The SMOTE command line is:

```
SMOTE -C 0 -K 10 -P 500.0 -S 13
```

Weka implements the IBk [AK91] algorithm. I will use cross-validation to determine the value for k ($k \in \{1, 2, \ldots, 13\}$) and inverse-distance weighting. As we will see, taking $k = 4$ provides the greatest accuracy.

In order to speed up the algorithm k-d trees will be used, as they allow for $\mathcal{O}(\log n)$ nearest-neighbor queries. Distance will be measured by the common Euclidean norm.

```
=== Run information ===

Scheme:       weka.classifiers.lazy.IBk -K 13 -W 0 -X -I
              -A "weka.core.neighboursearch.KDTree
              -A \"weka.core.EuclideanDistance
              -R first-last\"
              -S weka.core.neighboursearch.kdtrees.SlidingMidPointOfWidestSide
              -W 0.01 -L 40 -N" -batch-size 500
Relation:     diabetic_data-train-weka.filters.supervised.instance.SMOTE-C0-K10-P500.0-S13
Instances:    80920
Attributes:   25
              race
              gender
              age
              admission_type_id
              discharge_disposition_id
              admission_source_id
              time_in_hospital
              num_lab_procedures
              num_procedures
              num_medications
              number_inpatient
              diag_1
              diag_2
              diag_3
              number_diagnoses
              A1Cresult
              metformin
              glipizide
              glyburide
              pioglitazone
              rosiglitazone
              insulin
              change
              diabetesMed
              readmitted
Test mode:    user supplied test set:  size unknown (reading incrementally)

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 4 inverse-distance-weighted nearest neighbour(s) for classification


Time taken to build model: 0.28 seconds
```

```
=== Evaluation on test set ===

Time taken to test model on supplied test set: 38.24 seconds

=== Summary ===

Correctly Classified Instances       11006               80.2889 %
Incorrectly Classified Instances      2702               19.7111 %
Kappa statistic                          0.0195
Mean absolute error                      0.2462
Root mean squared error                  0.4081
Relative absolute error                 60.1874 %
Root relative squared error             98.4431 %
Total Number of Instances            13708

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  Class
                 0.874    0.852    0.904      0.874    0.889      Other
                 0.148    0.126    0.113      0.148    0.128      Readmitted
Weighted Avg.    0.803    0.781    0.827      0.803    0.814

                 MCC      ROC Area  PRC Area  Class
                 0.020    0.532     0.909     Other
                 0.020    0.532     0.109     Readmitted
Weighted Avg.    0.020    0.532     0.831

=== Confusion Matrix ===

     a     b   <-- classified as
 10807  1560 |     a = Other
  1142   199 |     b = Readmitted
```

Unfortunately, kNN performs quite poorly with an area under the ROC curve of only 0.532. The positive class F_1 score is poor, too; only 0.128.

4 Clustering

Since some of the algorithms we'll be exploring from here on take time—or even worse, memory—on the order of $\mathcal{O}\left(n^2\right)$ or more, I will restrict myself to a smaller subset of the data. For this I am going to use stratified sampling – I believe that associations and clusters should remain intact even within such samples.

The subset size will be around $1/3$ of the original data.

```
st <- strata(df, stratanames="readmitted",
             size=c(17500, 2500), method="srswor")
df.subset <- getdata(df, st)
# Drop unnecessary attributes.
```

```
df.subset <- subset(df.subset, select=-c(ID_unit, Prob, Stratum))
```

The data for clustering will be centered and scaled. As we will be using dummy variables, a Box–Cox transformation would not be advisable. We will also drop the `readmitted` attribute as clustering is inherently an *unsupervised* task.

```
# We won't be needing the class attribute when clustering.
cluster.df <- data.frame(model.matrix(readmitted ~ . - 1, df.subset))
cluster.pp <- preProcess(cluster.df, method=c("center", "scale"))

## Warning in preProcess.default(cluster.df, method = c("center", "scale")): These
variables have zero variances: genderUnknown.Invalid

cluster.df <- predict(cluster.pp, cluster.df)
```

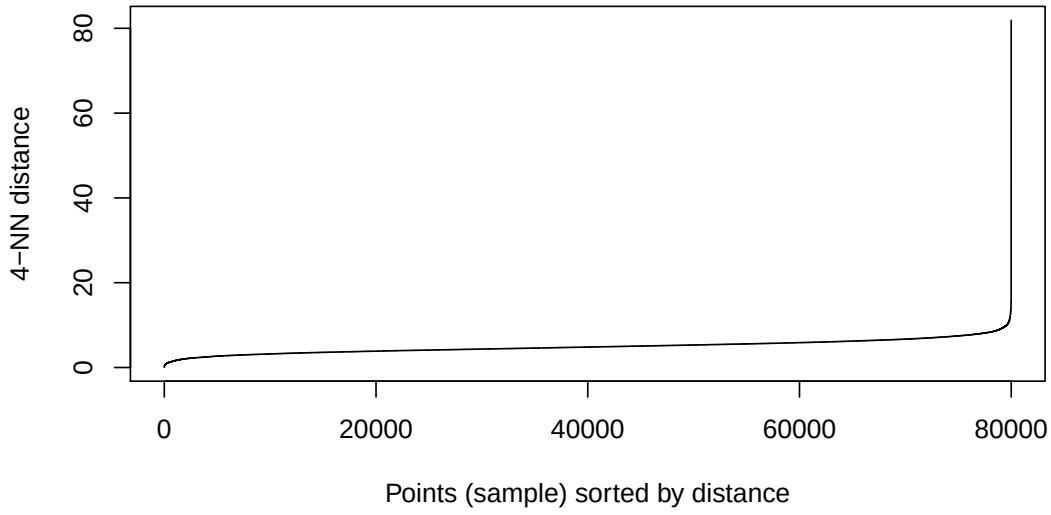
The first algorithm I will try is DBSCAN. In determining its Eps and MinPts parameters, I will use the following widely accepted method:

1. Eps is determined by looking at a knee of a kNN distance plot (I will take $k = 4$);
2. MinPts is set to $\#\{\text{dimensions}\} + 1$.

```
dim(cluster.df)

## [1] 20000    72

kNNdistplot(cluster.df, k=4)
```



Looking at the plot, one can identify a knee at around 11, so we should have Eps ≈ 11 (I will take it to be exactly 11). Since there are 72 dimensions, I will take MinPts $= 73$.

```
dbs <- dbscan(cluster.df, eps=11, minPts=73); dbs
```

```
## DBSCAN clustering for 20000 objects.
## Parameters: eps = 11, minPts = 73
## The clustering contains 4 cluster(s) and 193 noise points.
##
##     0     1     2     3     4
##   193 19359   141   157   150
##
## Available fields: cluster, eps, minPts
```

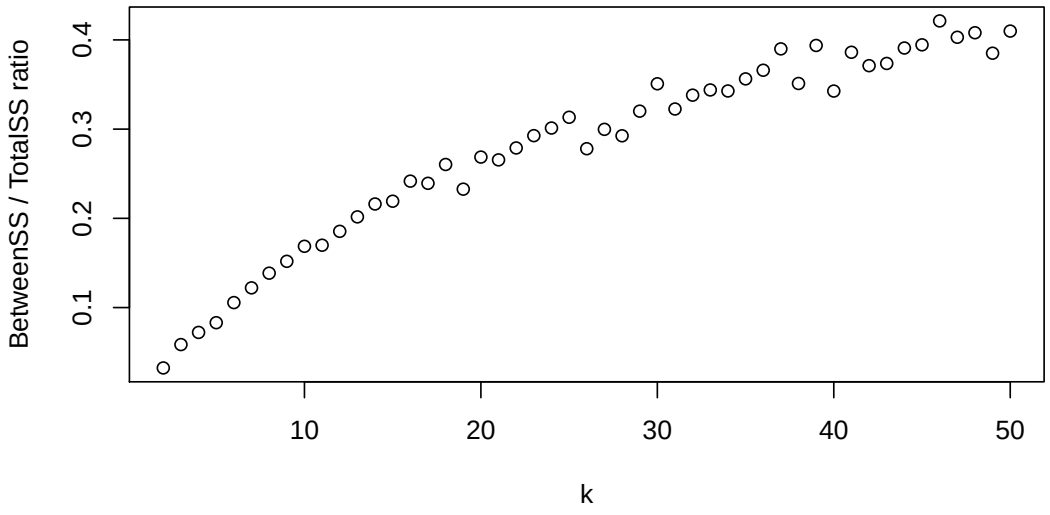
Unfortunately, it can be seen that the clusters found by DBSCAN do not correspond to our `readmitted` class.

The next algorithm I will try is k-means with $k \in \{2, \ldots, 50\}$. What I will measure is the

$$\frac{\text{BetweenSS}}{\text{TotalSS}} \tag{3}$$

ratio, which should be close to 1 if the data exhibits well-separated clusters. The rationale is that BetweenSS should be more or less equal to TotalSS = WithinSS + BetweenSS, as well-separated clusters minimize WithinSS.

```
ks <- 2:50
kmeans.errs <- sapply(ks, function (k) {
    km <- kmeans(cluster.df, k, iter.max=20)
    km$betweenss / km$totss  # Should be high (close to 1).
})
plot(ks, kmeans.errs, xlab="k", ylab="BetweenSS / TotalSS ratio")
```



Even with $k = 50$, the BetweenSS/TotalSS is very poor (around 0.4), leading me to believe that there are no well-separated clusters in terms of k-means within the data.

5 Association rule mining

I will use the apriori algorithm [AS94] for association rule mining. First, we need to discretize our numeric attributes. I will use clustering to perform the discretization.

```
assoc.df <- data.frame(lapply(df.subset, function (attr)
    if (is.numeric(attr)) discretize(attr, method="cluster") else attr))
```

Next, I will convert the data to a suitable format (sparse representation).

```
assoc.trans <- as(assoc.df, "transactions")
summary(assoc.trans)
```

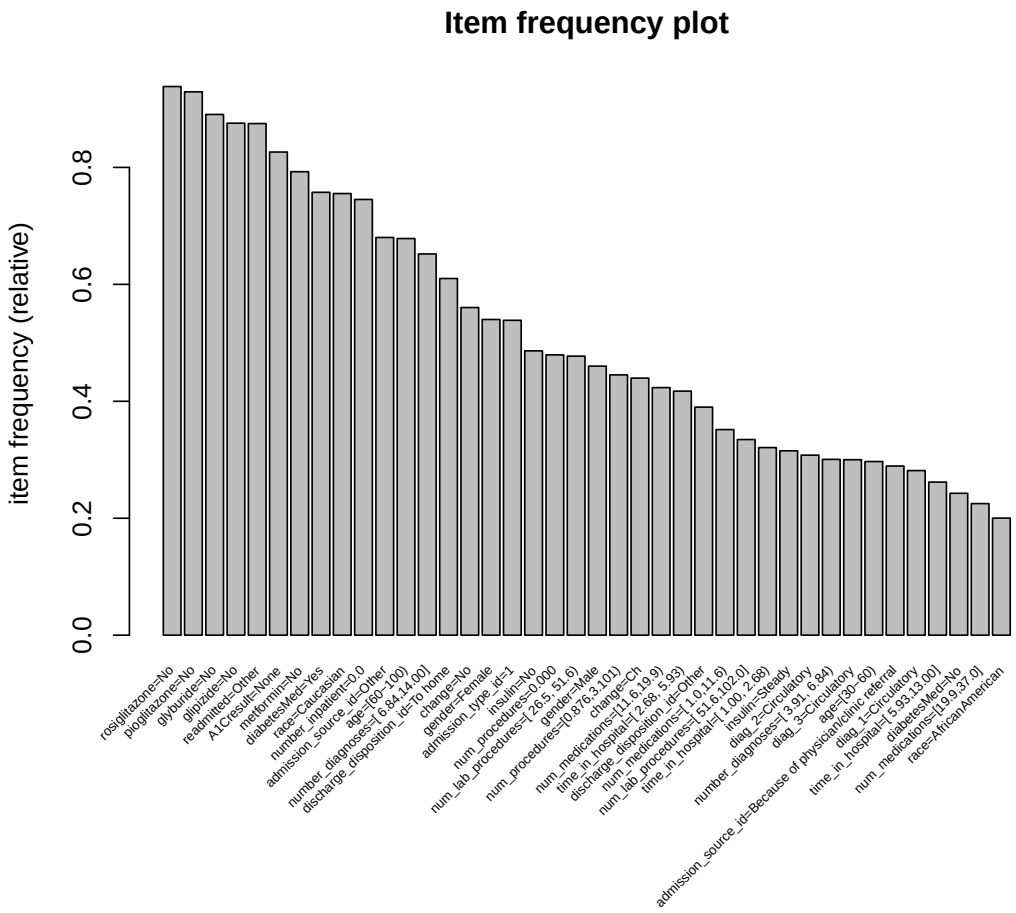
```
## transactions as itemMatrix in sparse format with
##  20000 rows (elements/itemsets/transactions) and
##  103 columns (items) and a density of 0.2427184
##
## most frequent items:
## rosiglitazone=No  pioglitazone=No     glyburide=No     glipizide=No
##            18766            18585            17811            17512
## readmitted=Other          (Other)
##            17500           409826
##
## element (itemset/transaction) length distribution:
## sizes
##    25
## 20000
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      25      25      25      25      25      25
##
## includes extended item information - examples:
##                 labels variables          levels
## 1 race=AfricanAmerican      race AfricanAmerican
## 2           race=Asian      race           Asian
## 3       race=Caucasian      race       Caucasian
##
## includes extended transaction information - examples:
##   transactionID
## 1             1
## 2             2
## 3             3
```

The apriori algorithm is based on two crucial measures, *support* and *confidence*, defined as

$$\text{Support}(X \to Y) = \frac{\sigma(X \cup Y)}{N} = \Pr(X \cup Y), \quad \text{Confidence}(X \to Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} = \Pr(Y \mid X).$$

Given this, the first step is to generate *frequent itemsets* – itemsets with support greater than some minimum threshold. In order to get a feel of the data, I will plot the frequent items. These items make for good left-hand side candidates, as long as they do not lead to trivial rules.

```
itemFrequencyPlot(assoc.trans, topN=40,
                  cex.names=0.5, main="Item frequency plot")
```



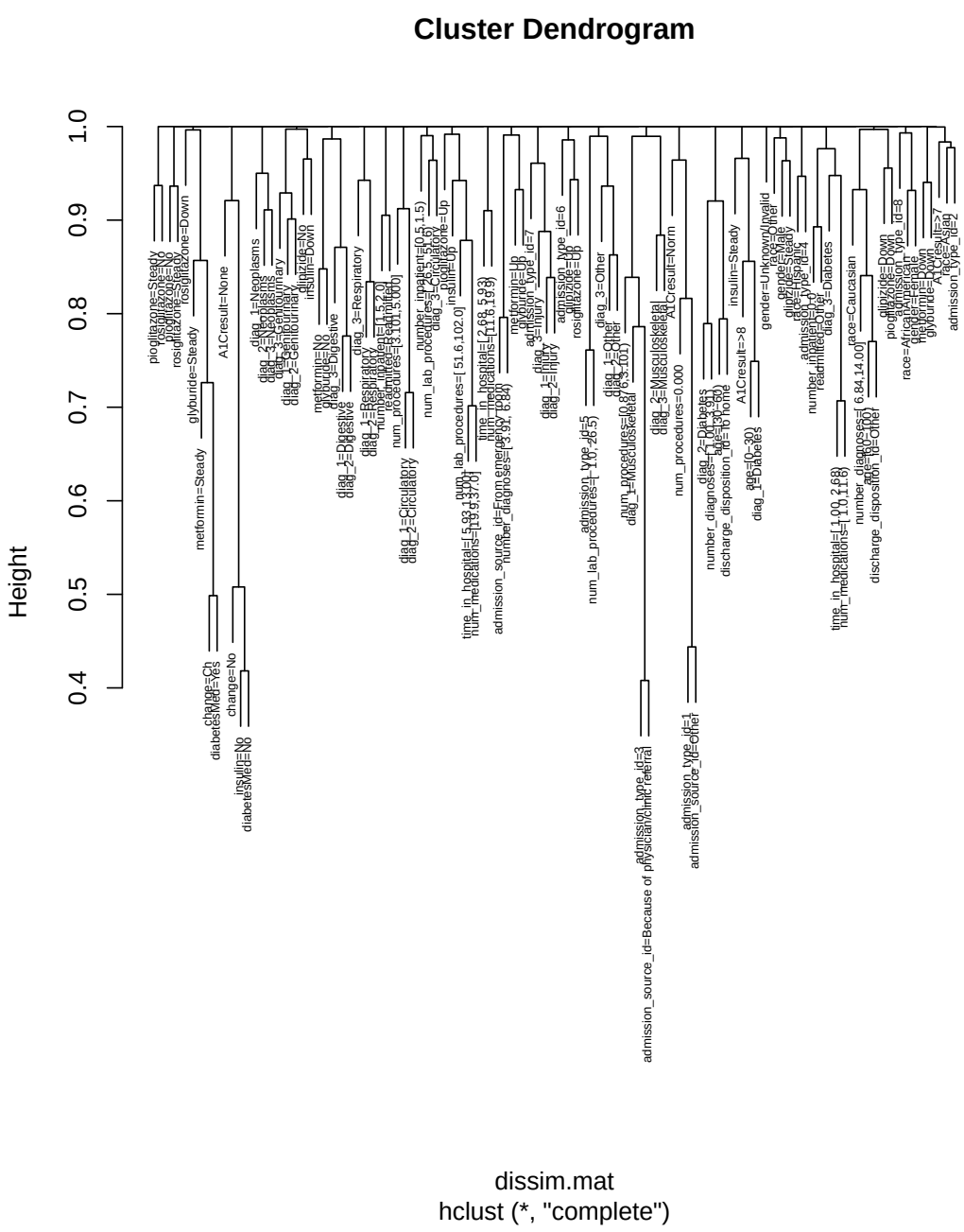
Several "items" appear to be present in relatively high frequency, which is hopefully a good sign – again, as long as they do not lead to trivial rules. Now we can check for similarities between items: whether there is any redundancy in the transactions. This will be done with hierarchical clustering.

Observing the dendrogram lets us look at how up in the tree items are merged (using complete linkage). The higher up they are the more dissimilar they are.

```
# Generate the dissimilarity matrix.
dissim.mat <- dissimilarity(assoc.trans, method="phi", which="items")

# Plot the results of the hierarchical clustering.
```

```
# (Uses complete linkage by default)
plot(hclust(dissim.mat), cex=0.5)
```



As it can be seen, the vast majority of the items are dissimilar from one another and there is no need to further preprocess the data. When generating the rules using the priory algorithm, I will take an interesting support to mean having at least 5000 observations:

```r
5000 / nrow(assoc.trans)  # Find a minimum support size.
```

```
## [1] 0.25
```

In other words, the minimum support will be set to 0.25. The minimum confidence will be set to 0.75, and rules will be restricted to include at most 15 items.

```r
# Generate the rules.
assoc.rules <- apriori(assoc.trans,
                       parameter=list(target="rules",
                                      supp=0.25, conf=0.75,
                                      minlen=2, maxlen=15))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##        0.75    0.1    1 none FALSE            TRUE    0.25      2     15
##  target   ext
##   rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 5000
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[102 item(s), 20000 transaction(s)] done [0.06s].
## sorting and recoding items ... [37 item(s)] done [0.01s].
## creating transaction tree ... done [0.04s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 done [1.59s].
## writing ... [25170 rule(s)] done [0.01s].
## creating S4 object  ... done [0.03s].
```

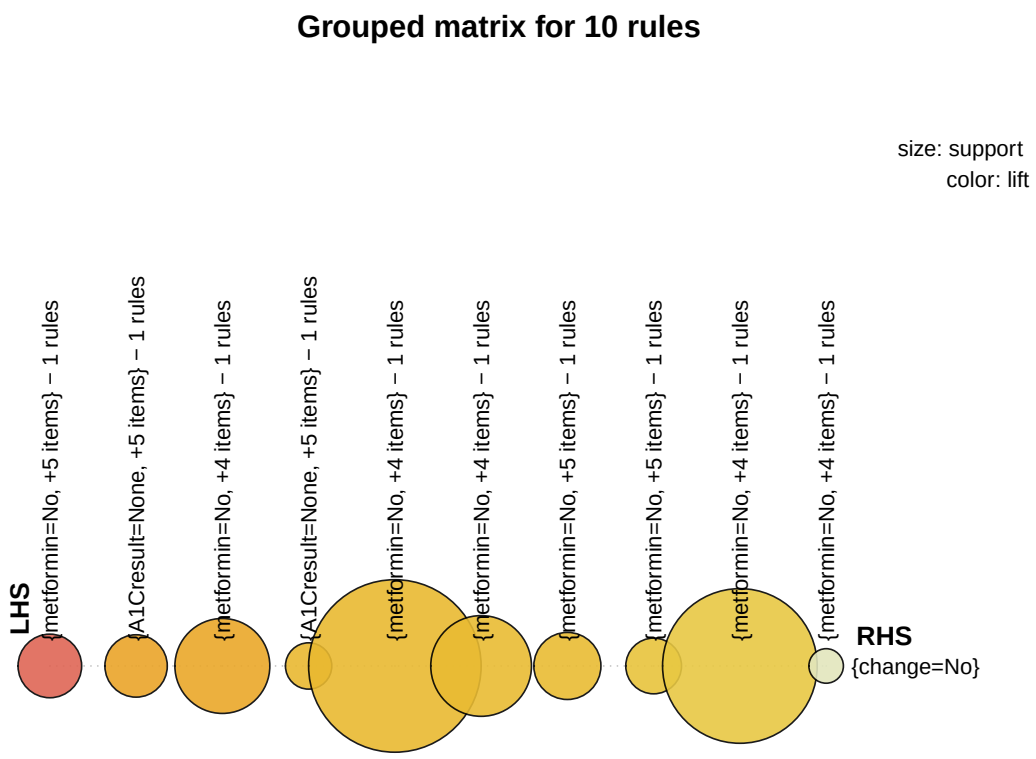
```r
# Sort by lift.
rules.by.lift <- head(sort(assoc.rules, by="lift"), 10)
inspect(rules.by.lift)
```

```
##    lhs                   rhs          support confidence     lift
## 1  {metformin=No,
##     glipizide=No,
##     glyburide=No,
##     pioglitazone=No,
##     rosiglitazone=No,
##     insulin=No}       => {change=No} 0.26130  0.9927812 1.771716
## 2  {A1Cresult=None,
##     metformin=No,
##     glyburide=No,
##     pioglitazone=No,
##     rosiglitazone=No,
```

```
##     insulin=No}       => {change=No} 0.26090  0.9845283 1.756988
## 3  {metformin=No,
##     glipizide=No,
##     glyburide=No,
##     pioglitazone=No,
##     insulin=No}       => {change=No} 0.27165  0.9838827 1.755836
## 4  {A1Cresult=None,
##     metformin=No,
##     glipizide=No,
##     pioglitazone=No,
##     rosiglitazone=No,
##     insulin=No}       => {change=No} 0.25560  0.9815668 1.751703
## 5  {metformin=No,
##     glyburide=No,
##     pioglitazone=No,
##     rosiglitazone=No,
##     insulin=No}       => {change=No} 0.29725  0.9813470 1.751311
## 6  {metformin=No,
##     glipizide=No,
##     glyburide=No,
##     rosiglitazone=No,
##     insulin=No}       => {change=No} 0.27355  0.9811693 1.750994
## 7  {metformin=No,
##     glyburide=No,
##     pioglitazone=No,
##     rosiglitazone=No,
##     insulin=No,
##     readmitted=Other} => {change=No} 0.26245  0.9811215 1.750908
## 8  {metformin=No,
##     glipizide=No,
##     pioglitazone=No,
##     rosiglitazone=No,
##     insulin=No,
##     readmitted=Other} => {change=No} 0.25870  0.9799242 1.748772
## 9  {metformin=No,
##     glipizide=No,
##     pioglitazone=No,
##     rosiglitazone=No,
##     insulin=No}       => {change=No} 0.29135  0.9793277 1.747707
## 10 {metformin=No,
##     glipizide=No,
##     glyburide=No,
##     insulin=No,
##     readmitted=Other} => {change=No} 0.25170  0.9740712 1.738326
```

We see that our 10 best rules (in terms of lift) can be used to help us predict whether change in diabetes medication should be administered to the patient. We can visualize these rules:

```
plot(rules.by.lift, method="grouped", measure="support", shading="lift")
```



On the other hand, ranking the rules by confidence gives rules which tell whether medication is to be administered:

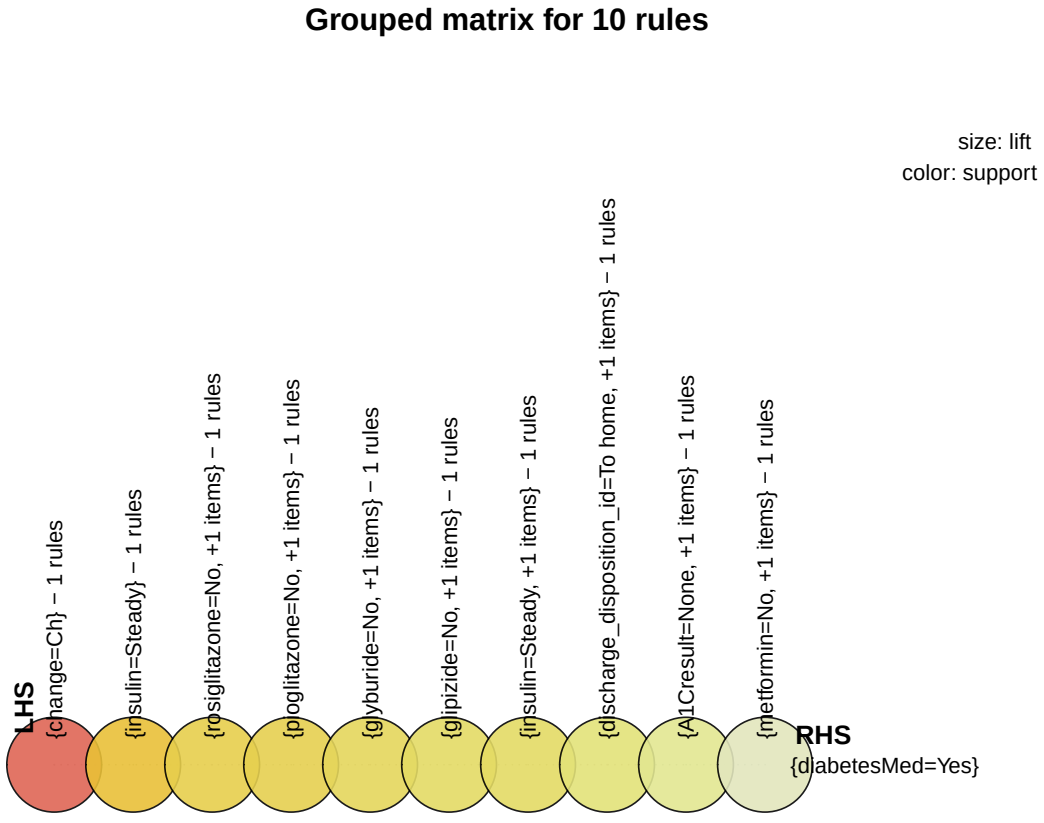
```
# Sort by confidence.
rules.by.conf <- head(sort(assoc.rules, by="confidence"), 10)
inspect(rules.by.conf)
```

```
##    lhs                                  rhs                 support confidence     lift
## 1  {insulin=Steady}                  => {diabetesMed=Yes} 0.31530          1 1.320393
## 2  {change=Ch}                       => {diabetesMed=Yes} 0.43965          1 1.320393
## 3  {metformin=No,
##     insulin=Steady}                  => {diabetesMed=Yes} 0.25100          1 1.320393
## 4  {A1Cresult=None,
##     insulin=Steady}                  => {diabetesMed=Yes} 0.25455          1 1.320393
## 5  {insulin=Steady,
##     readmitted=Other}                => {diabetesMed=Yes} 0.27485          1 1.320393
```

```
## 6  {glipizide=No,
##     insulin=Steady}                    => {diabetesMed=Yes} 0.27545          1 1.320393
## 7  {glyburide=No,
##     insulin=Steady}                    => {diabetesMed=Yes} 0.28320          1 1.320393
## 8  {pioglitazone=No,
##     insulin=Steady}                    => {diabetesMed=Yes} 0.29255          1 1.320393
## 9  {rosiglitazone=No,
##     insulin=Steady}                    => {diabetesMed=Yes} 0.29440          1 1.320393
## 10 {discharge_disposition_id=To home,
##     change=Ch}                         => {diabetesMed=Yes} 0.26365          1 1.320393
```

Note that the lift of these rules is comparably smaller because the event of medication being administered appears often in our dataset. These rules also look similar to the ones ranked by lift, as they also pertain to a single right-hand size.

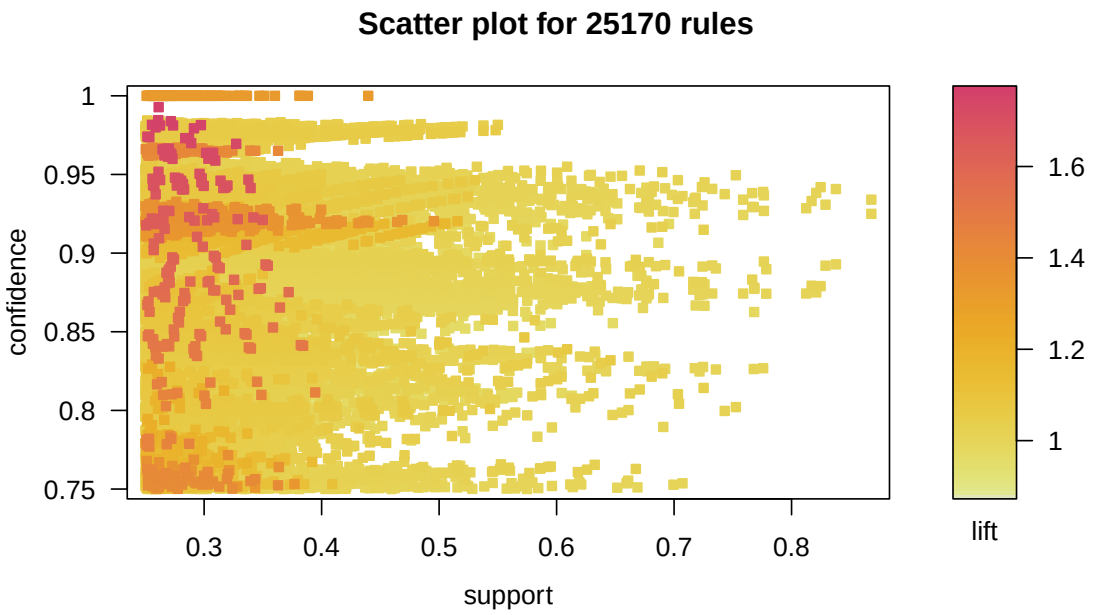
```
plot(rules.by.conf, method="grouped", measure="lift", shading="support")
```



We finish off with a visualization of all rules, which shows that there are rules which have both high

confidence and lift (meaning they are accurate while also not being "common knowledge").

```
plot(assoc.rules)
```



6 Conclusion

The dataset that I have mined shows heavy imbalance in class distribution. This leads to problems in classification. We observe that despite having solid accuracy, the classification algorithms struggle with classifying positive observations even if sophisticated upsampling techniques such as SMOTE are used. However, after tuning the algorithms' classification threshold, satisfying results are obtained.

Further directions in classification worth exploring include:

- SMOTE likely leads to overfitting. Thus, it might be worthwhile to look at ensemble-based methods such as random forests and AdaBoost, as they are known to be less susceptible to overfitting. Additionally one might combine several completely different learners, i.e., naïve Bayes and kNN.
- Aggressive cross-validation for parameter tuning, e.g., tune even SMOTE's parameters using separate data.
- Look at pairwise interactions for adding nonlinearities.

The data does not exhibit well-separated clusters in terms of density (DBSCAN) or centroids (k-means). It is unlikely that any clustering technique will yield meaningful results, partly due to the nature of the data, partly due to the underlying imbalance in class distributions.

Nevertheless, there are meaningful association rules that can be extracted from the data. These rules allow us to draw conclusion regarding medication changes or administrations.

Last, but not least, it is safe to say that the data is quite ungainly. In order to tackle the issue, perhaps additional attributes should be measured regarding patients. Unfortunately, there is also the possibility of diabetes being too unpredictable to lack any meaningful causal structure.

References

[AK91] David W. Aha and Dennis Kibler. "Instance-Based Learning Algorithms". In: *Machine Learning* 6 (1991), pp. 37–66 (cit. on p. 25).

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". In: *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 487–499. ISBN: 1-55860-153-8. URL: http://dl.acm.org/citation.cfm?id=645920.672836 (cit. on p. 30).

[BC64] George E. P. Box and David R. Cox. "An Analysis of Transformations". In: *Journal of the Royal Statistical Society. Series B (Methodological* (1964), pp. 211–252. URL: http://fisher.osu.edu/~schroeder.9/AMIS900/Box1964.pdf (cit. on p. 13).

[Bre+84] Leo Breiman et al. *Classification and Regression Trees*. Chapman and Hall, 1984. ISBN: 0412048418 (cit. on p. 17).

[Cha+02] Nitesh V. Chawla et al. "SMOTE: Synthetic Minority Over-Sampling Technique". In: *J. Artif. Int. Res.* 16.1 (June 2002), pp. 321–357. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622407.1622416 (cit. on p. 13).

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-238-0 (cit. on p. 17).

[Str+14] Beata Strack et al. "Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records". In: *BioMed Research International* (2014). DOI: 10.1155/2014/781670 (cit. on pp. 3, 6).