Advanced computational methods - Lecture 1

Hrvoje Stojic

January 8, 2016

1 Introduction

This session introduces basic parametric method for classification that relies on linear regression. Besides that, we will talk about dynamic visualization and we will see an example of it in R package Shiny.

1.1 Using .Rmd documents

In the files for this session you will find a source file of this document with .Rmd extension. To use these files simply open them as you would open any other R script. Rmd stands for rmarkdown, a combination of R code and lowly formatted text (this is why it is called mark*down*).

To be able to build PDF, HTML and other types of documents from it you need to install Rmarkdown and knitr packages. If you use RStudio there are buttons that build documents automatically, while if you are using a text editor you will need to run a specific command. For example, open a terminal in the folder containing the .Rmd file and run `R -e 'library(knitr);library(rmarkdown);rmarkdown::render("file_name.Rmd", "html_document")'`

Why knitr or Rmarkdown?

- **Reproducibility**:
  - It makes your data analysis more reproducible. The R code describes exactly the steps from the raw data to the final report. This makes it perfect for sharing reports with your colleagues.
  - It is written with almost no formatting at all (markdown), which makes it easier to convert to any other format, from nicely looking PDFs to the all-present MS docx and complete HTML documents (fancy a blog?).
- **Efficiency**:
  - Statistical output from figures to tables is automatically placed in your report. No more copy-pasting and reformatting the output from your statistical analysis program into your report.

  - You want to use a slightly different subset of the data? You want to drop that outlier observation? No problem, you can update your report with a single click instead of updating every table and figure.

  - Whoever has done some copy-pasting knows how easy is to overlook one number or one figure. This type of document also significantly reduces the chance of such errors.

- **Education & Communication**:

  - Excellent for teaching as one can check how exactly is some analysis done from the report.

  - Do not disregard this aspect, look at Github and Stackoverflow stars who get job offers on this account!

Note that I used a different font in the document than the default. To use different fonts in your documents, in the header of the .Rmd file, under pdf_document you need to set "latex_engine: xelatex", and then instead of "sansfont: Arial" use whatever font you have available to Tex system on your computer (usually whatever fonts are available on your computer).

---

2 Discriminant functions a.k.a. Linear probability models

2.1 Simple categorization problem

To illustrate discriminant functions we will use a simple artificial dataset - two dimensional data with two categories - whether a loan in the bank will be denied or approved, with two observed variables, payment-to-income ratio and solvency score.

We will assume that bot variables normally distributed variables, not necessarily independent from each other. For more general m-dimensional case we can write it as follows:

$$N(\mathbf{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} exp\{\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\}$$

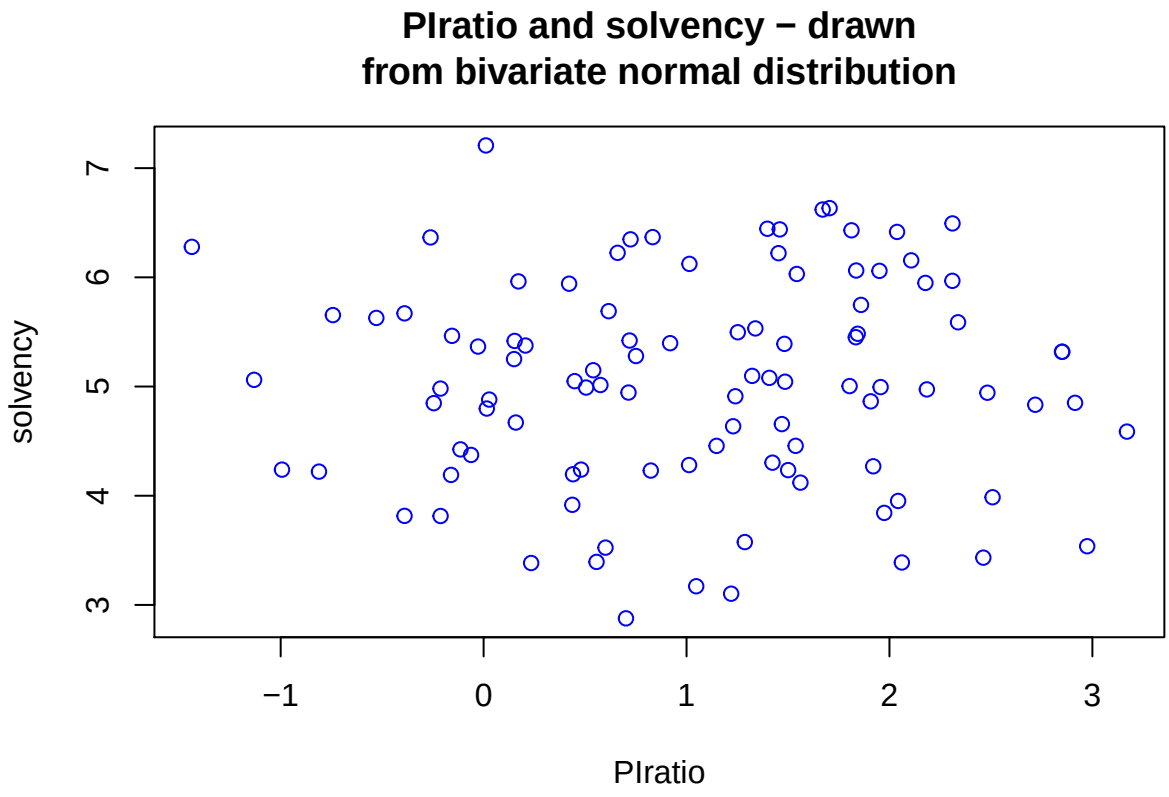
where $\mathbf{x}$ is m-dimensional vector, Σ is positive-definite covariance matrix and μ is a vector of means. Since we have a 2-dimensional case our joint distribution is governed by the following parameters:

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

where ρ is correlation between X and Y.

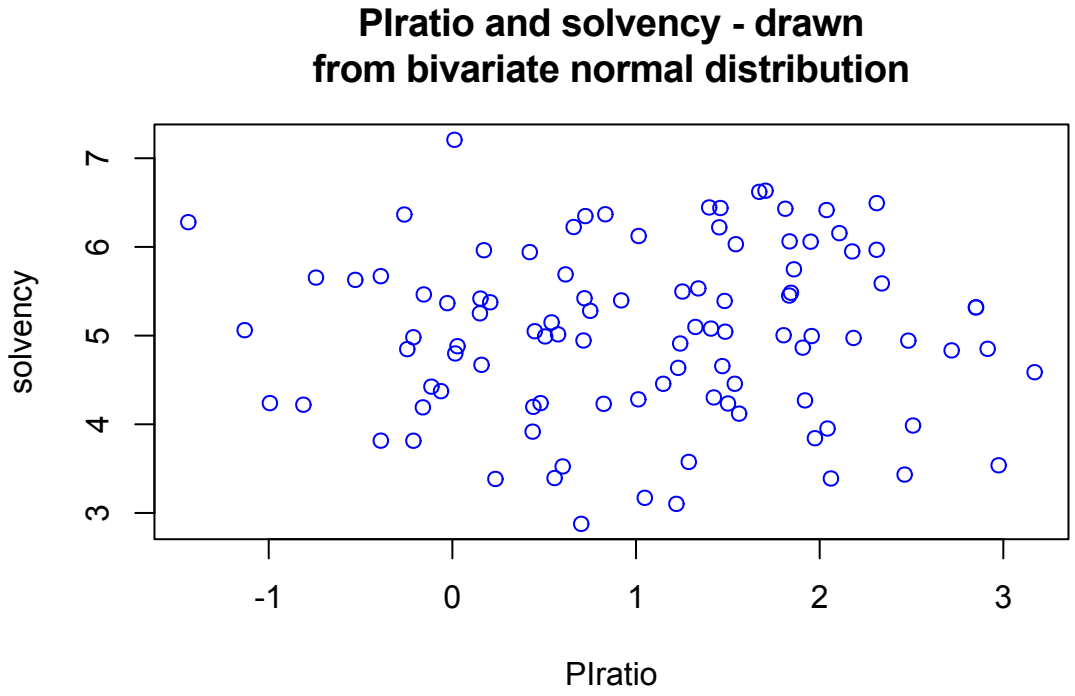
We use the R package `mvtnorm` to draw (pseudo)-random numbers from such joint normal distribution (read the docs here).

```
# simple example with uncorrelated variables
set.seed(7851)
x <- rmvnorm(n=100, mean=c(1,5), sigma=1*diag(2))
plot(x[, 1], x[, 2], col = "blue", xlab = "PIratio", ylab = "solvency")
title("PIratio and solvency - drawn\nfrom bivariate normal distribution")
```



You might have noticed that sometimes figures produced by R when saved in PDF format do not display correctly. This is because it does not embed fonts into PDF when it is generated. PDF viewers and client systems that lack the font you have used will substitute it with some other font. Actually, Helvetica, default font used in R is not present in many open source PDF viewers and operating systems. You will usually want to make sure that your users will be able to see the figure correctly. Below I show you how to embed fonts via `extrafont` package (another alternative is `showtext` package). Moreover, with this package you can expand the number of fonts you can use in your figures.

```r
# embedding fonts into our pdf figures
if (!require("extrafont")) install.packages("extrafont"); library(extrafont)

# importing fonts from your OS into R, need to be run only once
# font_import()
# show fonts available
# fonts()  # or fonttable()

# you would then embed fonts as follows
pdf("plot_embedded_fonts.pdf", family="Arial", width=4, height=4.5)
plot(x[, 1], x[, 2], col = "blue", xlab = "PIratio", ylab = "solvency")
title("PIratio and solvency - drawn\nfrom bivariate normal distribution")
dev.off()

## pdf
##   2

embed_fonts("plot_embedded_fonts.pdf")

# however, rmarkdown does that automatically, how can we instruct it
# to embed fonts?
# 1. we need to change the device to cairo_pdf(), this device can
#    automatically embed fonts into PDF plots, normal pdf() cannot
# 2. we add dev.args family argument in the chunk specs
if (output=="latex") {
    opts_chunk$set(dev = 'cairo_pdf',
    dev.args=list(family="Arial"))
}

# now plot the figure as usual
plot(x[, 1], x[, 2], col = "blue", xlab = "PIratio", ylab = "solvency")
title("PIratio and solvency - drawn\nfrom bivariate normal distribution")
```



If you need to draw a *really* high dimensional data (>1000), drawing them with the help of `mvtnorm` package might be too slow. There are `Rcpp` implementations that are faster (see for an example here)

In the figure above PIratio and solvency were uncorrelated, but this is often not realistic. Let us use `mvtnorm` to draw samples for which covariance terms in the Σ matrix are not zero.

```
# making things a bit more complicated, say we want one sample with 0.8
# correlation between solvency and PIratio, and one with -0.1

# create small wrapper functions
sigmaXY <- function(rho, sdX, sdY) {
    covTerm <- rho * sdX * sdY
    VCmatrix <- matrix(c(sdX^2, covTerm, covTerm, sdY^2),
                       2, 2, byrow = TRUE)
    return(VCmatrix)
}

genBVN <- function(n = 1, seed = NA, muXY=c(0,1), sigmaXY=diag(2)) {
    if(!is.na(seed)) set.seed(seed)
    rdraws <- rmvnorm(n, mean = muXY, sigma = sigmaXY)
    return(rdraws)
}
```

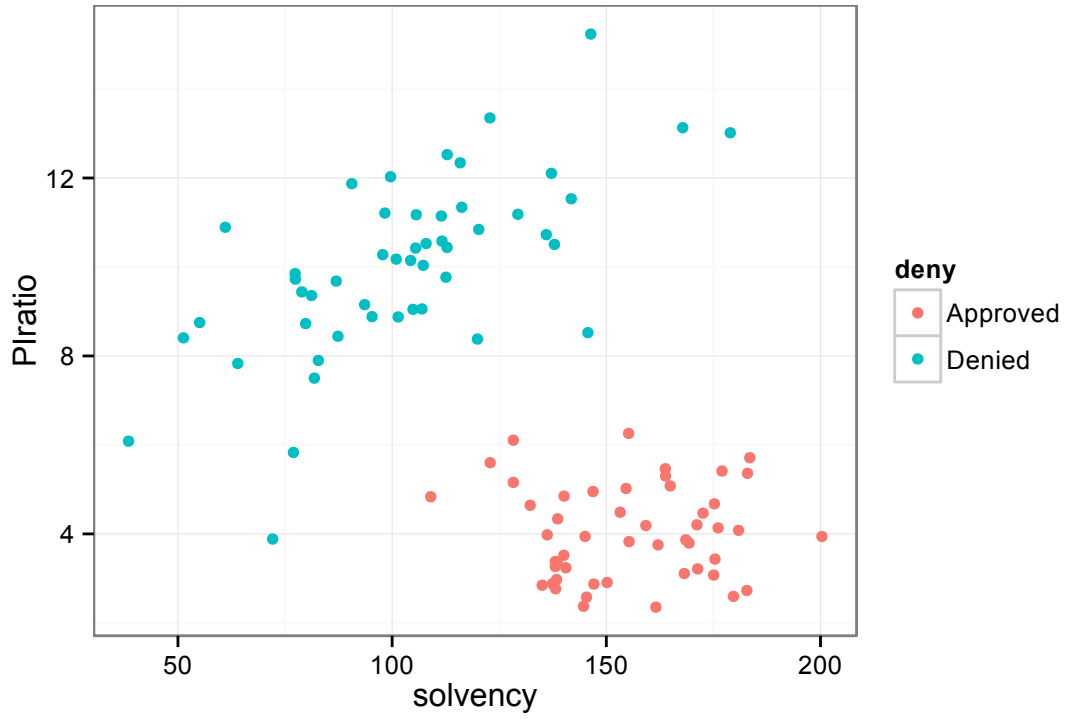
```r
# correlation is slightly negative
sigmaXY(rho=-0.1, sdX=1, sdY=20)
```

```
##      [,1] [,2]
## [1,]    1   -2
## [2,]   -2  400
```

```r
# highly positive
sigmaXY(rho=0.8, sdX=2, sdY=30)
```

```
##      [,1] [,2]
## [1,]    4   48
## [2,]   48  900
```

```r
# creating a function for all of this
loanData <- function(noApproved, noDenied, muApproved, muDenied, sdApproved,
                         sdDenied, rhoApproved, rhoDenied, seed=1111) {
    sigmaApproved <- sigmaXY(rho=rhoApproved, sdX=sdApproved[1], sdY=sdApproved[2])
    sigmaDenied <- sigmaXY(rho=rhoDenied, sdX=sdDenied[1], sdY=sdDenied[2])
    approved <- genBVN(noApproved, muApproved, sigmaApproved, seed = seed)
    denied <- genBVN(noDenied, muDenied, sigmaDenied, seed = seed+1)
    loanDf <- as.data.frame(rbind(approved,denied))
    deny <- c(rep("Approved", noApproved), rep("Denied", noDenied))
    target = c(rep(0, noApproved), rep(1, noDenied))
    loanDf <- data.frame(loanDf, deny, target)
    colnames(loanDf) <- c("PIratio", "solvency", "deny", "target")
    return(loanDf)
}

# generating some data
loanDf <- loanData(noApproved=50, noDenied=50, c(4, 150), c(10, 100),
                   c(1,20), c(2,30), -0.1, 0.6, 1221)

# illustrating the data, note that with ggplot we need to additionally
# specify font family
ggplot(data = loanDf,
       aes(x = solvency, y = PIratio, colour=deny, fill=deny)) +
geom_point() +
xlab("solvency") +
ylab("PIratio") +
theme_bw()
```



We use ggplot package from now on for producing figures - one of amazing set of R packages developed by Hadley Wickham, today a chief scientist at RStudio. (some other really useful packages he is developing are dplyr, reshape, httr, assertthat).

2.2 Econometric perspective - Linear probability model

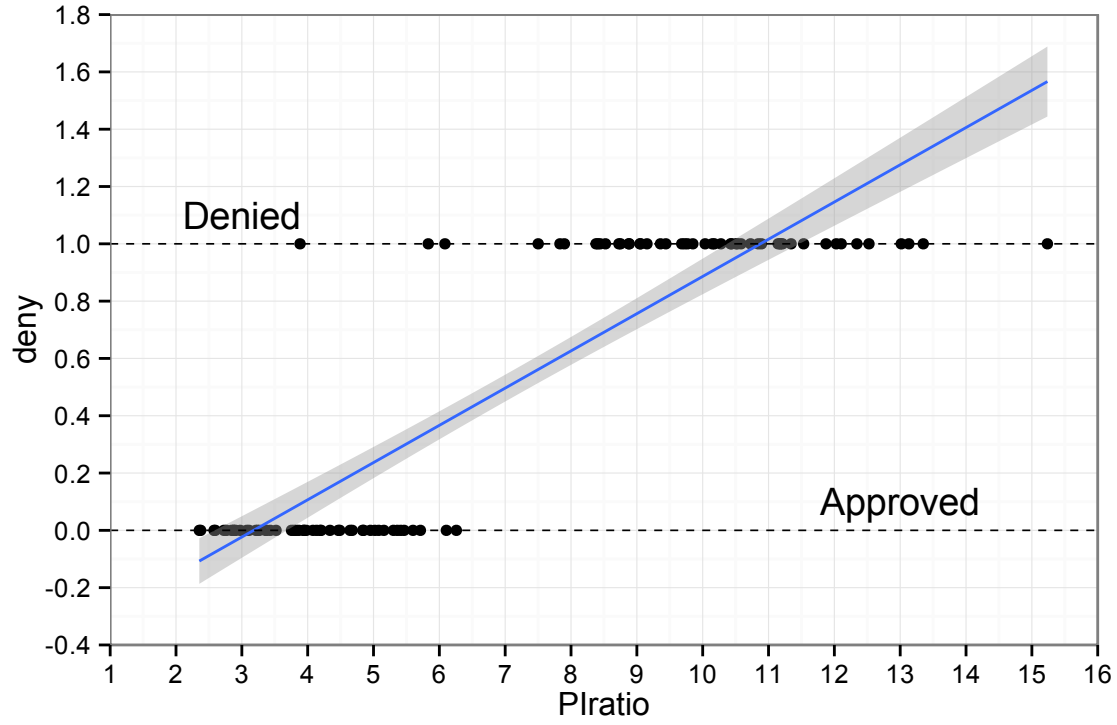
Recall the linear regression model:

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0.$$

Using Ordinary Least Squares in a problem where variable we are trying to predict takes only two values, 0 and 1, is called **Linear probability model** (LPM). Let us apply it first to our newly created dataset and later on we will worry about the consequences.

```
# illustrating the dependence of deny on PIratio variable
ggplot(data = loanDf,
       aes(x = PIratio, y = target)) +
geom_point() +
geom_smooth(method = "lm") +
scale_x_continuous("PIratio", limit = c(1, 16),
                   breaks = seq(1, 16, 1), expand = c(0, 0)) +
scale_y_continuous("deny", limit = c(-0.4, 1.8),
                   breaks = seq(-0.4, 1.8, 0.2),
```

```
                    expand = c(0, 0)) +
geom_hline(yintercept = 0, size = 0.3, linetype = 2) +
annotate("text", x = 13, y = 0.1, label = "Approved", family = "Arial") +
geom_hline(yintercept = 1, size = 0.3, linetype = 2) +
annotate("text", x = 3, y = 1.1, label = "Denied", family = "Arial") +
theme_bw() +
theme(text = element_text(family = "Arial"))
```




```
# running the linear regression on our dataset,
# but only on PIratio variable and intercept
datafit <- lm(target ~ PIratio + 1, data = loanDf)
summary(datafit)
```

```
##
## Call:
## lm(formula = target ~ PIratio + 1, data = loanDf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56630 -0.15283 -0.00954  0.10523  0.90843
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) -0.413021   0.054104  -7.634 1.52e-11 ***
## PIratio      0.129912   0.006929  18.750  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2358 on 98 degrees of freedom
## Multiple R-squared:  0.782,  Adjusted R-squared:  0.7798
## F-statistic: 351.6 on 1 and 98 DF,  p-value: < 2.2e-16
```

First question that you might ask is what does it mean to fit a line to a binary variable? How is this different than a regression with a continuous regressand?

Let us look at the figure. The scatterplot looks different than usual regression scatterplots due to our binary variable. Still, the figure seems to show there is a positive relationship between *PIratio* of a client and a loan being *denied*: very few clients with payment-to-income ratio below 6 are denied the loan, while above it most are denied. The line depicts the predicted value of *deny* as a function of PIratio. For example, when PIratio=5, the predicted value of deny is 0.2. What does that value mean?

Recall that regression is expectation, namely $E(Y|X_1, ..., X_k)$, and when expressed in terms of outcomes and probabilities we get the following expression

$$E(Y|X_1, ..., X_k) = 0 \times P(Y = 0|X_1, ..., X_k) + 1 \times P(Y = 1|X_1, ..., X_k)$$

Hence, for a binary variable $E(Y|X_1, ..., X_k) = P(Y = 1|X_1, ..., X_k)$, the predicted value from regression is the probability that Y=1, given X. Following this we can interpret the value of 0.2 as follows - when PI ratio is 5, the estimated probability of being denied a loan is about 20%, as computed with the help of coefficients: $P(Y = Denied|PIratio) = -0.41 + 5 \times 0.129$.

How do we interpret coefficients from the regression? In a similar manner, unit change in PI ratio is associated with a change of 0.12 in probability that $Y = 1$, i.e. that the loan is denied.

One of the advantages of applying OLS in this situation is that we are working with familiar framework where coefficients are easy to interpret. However, there are shortcomings. Even though in this course we will mostly be concerned with predictions, if you are interested in causal estimates you should be careful with standard errors with LPM. With binary dependent variable errors are always heteroscedastic. Standard errors computed above assume homoscedasticity by default and are incorrect, and so are p-values computed based on them. R^2 statistic is uninterpretable in this scenario - all data cannot possibly correspond to the predicted line. Finally, note that there are predicted values larger than 1 and smaller than 0 - these obviously cannot be probabilities! LPM can only provide you with an approximation of probabilities, predicted values have to be transformed in a nonlinear fashion to conform to real probabilities, and this the role of probit/logit transformation. In practice, differences between LPM and logit/probit regression might not be that big, especially if there is very few extreme values of the regressors.

2.3 Machine learning perspective - Discriminant functions

We distinguish between three type of linear models for classification:

1. Discriminant Functions
2. Probabilistic Discriminative Models
3. Probabilistic Generative Models

Check section 1.5.4 of Bishop for more details about probabilistic vs. deterministic classification.

Strictly speaking *discriminant function* is the same as LPM, but in machine learning they think about it in a somewhat different manner - they focus much more on decision boundaries. This is useful way to think about categorization and we will use discriminant functions to introduce them.

We convert fitted values $\hat{y}(\mathbf{x})$ to fitted classes variable $\hat{C}$ by the following rule: "Approved" if $\hat{y}(\mathbf{x}) < 0.5$, "Denied" if $\hat{y}(\mathbf{x}) > 0.5$.

Why 0.5? It is a simple rationale - as soon as predicted probability for one class is greater than 0.5 we should always choose that class. This holds when costs of correct and incorrect classifications are the same (that we have been assuming here implicitly). You will learn more about this in Machine learning course in due time.

```
# simply running the linear regression on our animals dataset
datafit <- lm(target ~ solvency + PIratio + 1, data=loanDf)
summary(datafit)
```

```
##
## Call:
## lm(formula = target ~ solvency + PIratio + 1, data = loanDf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.41120 -0.08292 -0.00066  0.09504  0.50467
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.498735   0.094557   5.274 8.09e-07 ***
## solvency    -0.005506   0.000525 -10.489  < 2e-16 ***
## PIratio      0.101352   0.005490  18.462  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1622 on 97 degrees of freedom
## Multiple R-squared:  0.8979, Adjusted R-squared:  0.8958
## F-statistic: 426.4 on 2 and 97 DF,  p-value: < 2.2e-16
```

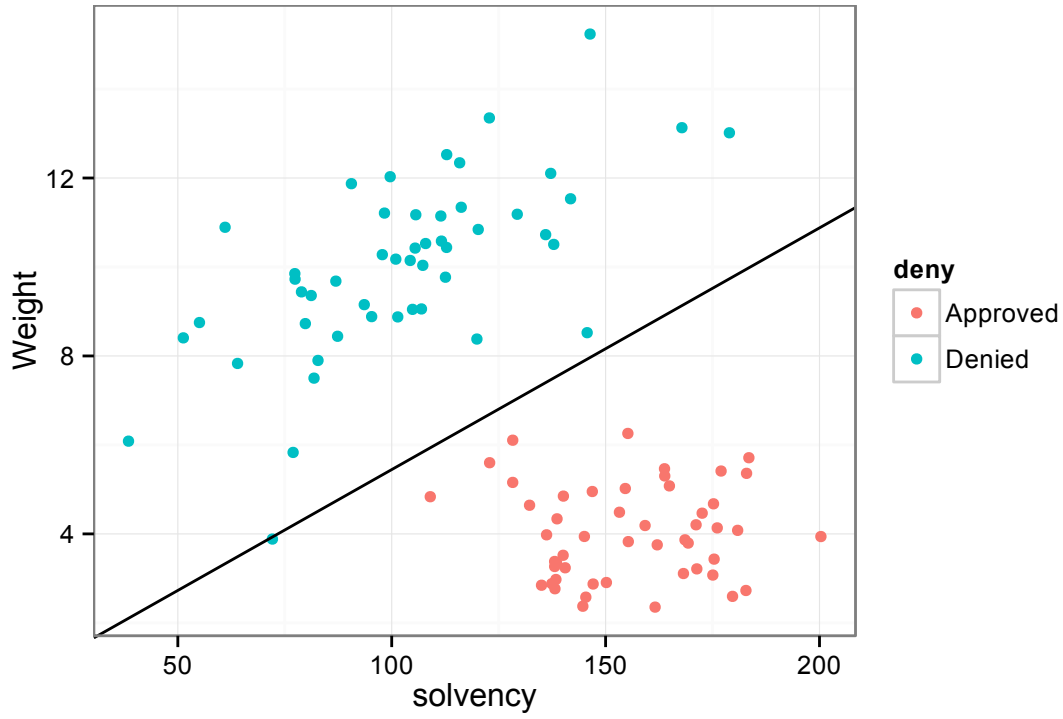
```r
# grabbing the coefficients
weights <- coef(datafit)[c("solvency", "PIratio")]
bias <- coef(datafit)[1]

# assigning labels
head(predict(datafit))
```

```
##           1           2           3           4           5           6
##  0.07643017 -0.03750601  0.27838656  0.05303728 -0.20499490  0.17504423
```

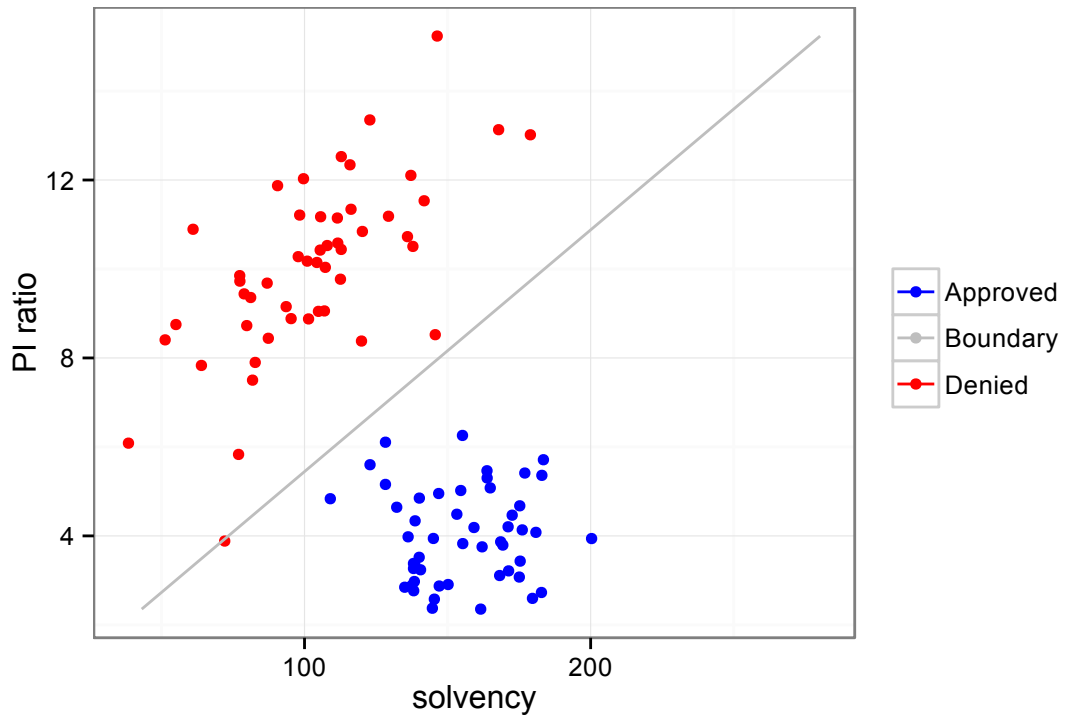
```r
sum(predict(datafit)>0.5)
```

```
## [1] 49
```

```r
# Computing the boundary: since it is a 2-dimensional example the boundary
# is a line.
intercept <- (-bias + 0.5)/weights["PIratio"]
slope <- -(weights["solvency"]/weights["PIratio"])

# illustrating the data, now with the boundary, we use geom_abline(),
# this will work only for lines
ggplot(data = loanDf, aes(x = solvency, y = PIratio,
       colour=deny, fill=deny)) +
geom_point() +
xlab("solvency") +
ylab("Weight") +
theme_bw() +
theme(text=element_text(family="Arial")) +
geom_abline(intercept = intercept, slope = slope)
```




```
# when plotting, a more general solution is to use geom_line()
x <- seq(min(loanDf["PIratio"]), max(loanDf["PIratio"]),
         length.out = nrow(loanDf))
y <- -(weights["PIratio"]/weights["solvency"])*x +
    (0.5-bias)/weights["solvency"]

# careful, colnames have to match!
boundaryDf <- data.frame(PIratio=x, solvency=y,
                         deny=rep("Boundary", length(x)))

# now plotting again, but with geom_line(), and we create a plot function,
plotDiscFnc <- function() {
    ggplot(data = loanDf,
           aes(x = solvency, y = PIratio, colour=deny)) +
    geom_point() +
    xlab("solvency") +
    ylab("PI ratio") +
    theme_bw() +
    geom_line(data=boundaryDf) +
    scale_color_manual("",
                       values = c("Boundary" = "grey",
                        "Approved" = "blue", "Denied" = "red"))
}
plotDiscFnc()
```



Estimated coefficients give us a solution to the following equation:

$$w_{PIratio}x_{PIratio} + w_{solvency}x_{solvency} + w_0 = 0.5$$

All inputs that satisfy this equation form the boundary. To visualize the boundary in PIratio/solvency coordinates we have to transform the solution

$$x_{PIratio} = -\frac{w_{solvency}}{w_{PIratio}}x_{solvency} + \frac{0.5 - w_0}{w_{PIratio}}$$

We use this equation to create points for plotting the boundary. Points on this line are neither in one nor the other category.

2.3.1 Confusion matrix

Let us create a slightly harder categorization problem where distributions overlap to some extent, to illustrate the usage of confusion matrices. Confusion matrices are often used to illustrate the performance of the classification algorithm. They give more detailed insight in what way the algorithm errs. This might be particularly relevant if the costs of errors are asymmetric. For example, in health diagnostics false negatives are much more costly than false positives. Hence, you would tolerate high misclassification rate on false positives, but not false negatives.

```
noApproved <- 50; noDenied <- 50
loanDf <- loanData(noApproved, noDenied, c(8, 120), c(10, 100),
                   c(1,20), c(2,30), -0.2, 0.6)

# optimizing
datafit <- lm(target ~ solvency + PIratio + 1, data=loanDf)
summary(datafit)
```

```
##
## Call:
## lm(formula = target ~ solvency + PIratio + 1, data = loanDf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.86978 -0.30739  0.02863  0.30608  0.93511
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.056326   0.247279   0.228     0.82
## solvency    -0.007267   0.001423  -5.107 1.64e-06 ***
## PIratio      0.138986   0.020943   6.636 1.85e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.391 on 97 degrees of freedom
## Multiple R-squared:  0.4069, Adjusted R-squared:  0.3947
## F-statistic: 33.28 on 2 and 97 DF,  p-value: 9.894e-12
```

```
# compute misclassification
predictedLabels <- ifelse(predict(datafit) < 0.5, "Approved", "Denied")

# confusion matrices
confMatrixFreq <- table(loanDf$deny, predictedLabels)
confMatrixFreq
```

```
##           predictedLabels
##            Approved Denied
##   Approved       42      8
##   Denied         11     39
```

```
confMatrixProp <- prop.table(confMatrixFreq, 1)
confMatrixProp
```

```
##           predictedLabels
```

```
##            Approved Denied
##   Approved     0.84   0.16
##   Denied       0.22   0.78
```

2.3.2 Sensitivity to outliers

Once you are thinking in terms of decision boundary another shortcoming becomes apparent. Regression lines are very sensitive to points very far from it, in other words, they are very sensitive to outliers. Why is this property very bad for classification purposes? Intuitively, points that are very far from the boundary (on the correct side) should not matter much as they should be categorized with confidence. In fact, points that should influence the boundary the most are the points closest to it.

This issue is due to the objective function that OLS uses to estimate the paramters - sum of squared errors. This is desirable property if we want to estimate the precise function that generates the data, but for categorization we can do better by relaxing that requirement - we have to know the function in as much as it leads to correct classification.

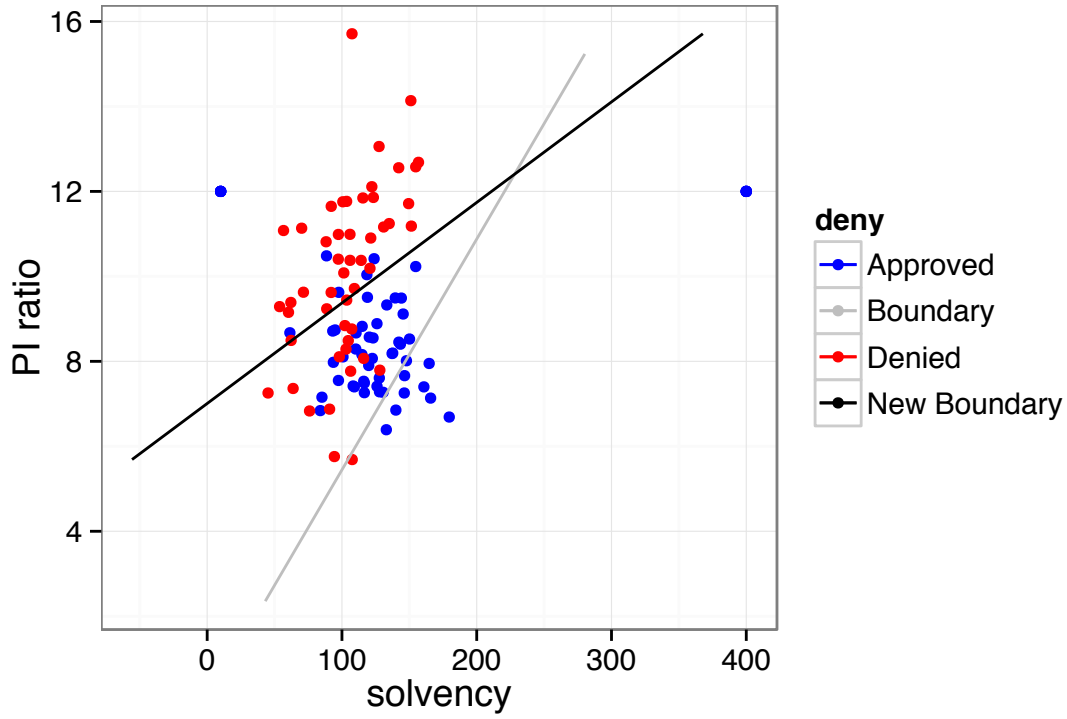
We will come back to the notion of distance from the boundary later on when we will talk about support vector machines (SVM). When optimization problem is formulated differently, distance from the boundary can give a powerful boost in classification performance.

```
# adding some outliers
outlier <- data.frame(rep(12,10), c(400, 10), rep("Approved",10), rep(0,10))
colnames(outlier) <- colnames(loanDf)
loanDf <- rbind(loanDf, outlier)

# optimizing
datafit <- lm(target ~ solvency + PIratio + 1, data=loanDf)
summary(datafit)
```

```
##
## Call:
## lm(formula = target ~ solvency + PIratio + 1, data = loanDf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.94258 -0.33425 -0.08381  0.40579  0.85971
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.1520725  0.2162548  -0.703 0.483453
## solvency    -0.0022020  0.0006336  -3.475 0.000739 ***
## PIratio      0.0930560  0.0224494   4.145 6.82e-05 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4546 on 107 degrees of freedom
## Multiple R-squared:  0.1891, Adjusted R-squared:  0.174
## F-statistic: 12.48 on 2 and 107 DF,  p-value: 1.344e-05
```

```
# grabbing the coefficients
weights <- coef(datafit)[c("solvency", "PIratio")]
bias <- coef(datafit)[1]

# the boundary
x <- seq(min(loanDf["PIratio"]), max(loanDf["PIratio"]),
         length.out = noApproved+noDenied)
y <- -(weights[2]/weights[1])*x + (0.5-bias)/weights[1]
boundaryDf2 <- data.frame(PIratio=x, solvency=y,
                          deny=rep("New Boundary", length(x)))

# plotting
ggplot(data = loanDf, aes(x = solvency, y = PIratio,
       colour=deny, fill=deny)) +
geom_point() +
xlab("solvency") +
ylab("PI ratio") +
theme_bw(base_size = 14, base_family = "Helvetica") +
geom_line(data=boundaryDf) +
geom_line(data=boundaryDf2) +
scale_color_manual("deny",
   values = c("New Boundary" = "black", "Boundary" = "grey",
              "Approved" = "blue", "Denied" = "red"))
```



2.3.3 More general implementation

How would you apply discriminant function to a problem with more than two categories?

A more general formulation is needed for K categories problems. Each category is modeled with a separate linear function:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \theta(\mathbf{x}) + w_{k0},$$

and input x is assigned to category C_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$.

Decision boundary between categories C_k and C_j is then determined by $y_k(\mathbf{x}) = y_j(\mathbf{x})$.

```
noApproved <- 50; noDenied <- 50
loanDf <- loanData(noApproved, noDenied, c(7, 150), c(10, 100),
                   c(2,20), c(2,30), -0.5, 0.3)

# add target variable, coded in a particular way
loanDf <- cbind(loanDf,
                target1 = c(rep(0, noApproved), rep(1, noDenied)),
                target2 = c(rep(1, noApproved), rep(0, noDenied))
                )

# analytical solution
```

```
X <- as.matrix(cbind(ind=rep(1, nrow(loanDf)),
                loanDf[,c("PIratio", "solvency")]))
Y <- cbind(target1 = c(rep(0, noApproved), rep(1, noDenied)),
           target2 = c(rep(1, noApproved), rep(0, noDenied))
           )
weightsOptim <- solve(t(X)%*%X) %*% t(X) %*% Y

# compute predictions
predictions <- X %*% weightsOptim
head(predictions)
```

```
##          target1   target2
## [1,] -0.05615153 1.0561515
## [2,]  0.05612504 0.9438750
## [3,]  0.87400775 0.1259922
## [4,]  0.07249453 0.9275055
## [5,]  0.11559319 0.8844068
## [6,] -0.08431786 1.0843179
```

```
# classify according to the argmax criterion
denied <- (predictions==apply(predictions, 1, max))[,1]
predictedLabels <- ifelse(denied, "Denied", "Approved")

# classification algorithm performance
confMatrixFreq <- table(loanDf$deny, predictedLabels)
confMatrixFreq
```

```
##           predictedLabels
##            Approved Denied
##   Approved       45      5
##   Denied          6     44
```

```
confMatrixProp <- prop.table(confMatrixFreq, 1)
confMatrixProp
```

```
##           predictedLabels
##            Approved Denied
##   Approved     0.90   0.10
##   Denied       0.12   0.88
```

What is the rationale behind this approach? Recall that linear regression assumes that class conditional distributions are Gaussians. Accidentally, the data is generated from (bivariate) Gaussian distribution and so our solution is actually close to optimal. It is not optimal due

to unequal covariances. If covariances were equal then our solution would have been Bayes optimal.

What if observations were generated in quite different way? Say mixture of Gaussians? Linearity assumption will fail in such a case - optimal boundary is nonlinear. Discriminant function relies heavily on the assumption that a linear decision boundary is appropriate. In other words, it has low variance and potentially high bias. In the next seminar class we will see that k-nearest-neighbor categorization method does not rely on such strict assumptions about the underlying data and will fare better on data where linearity assumption does not hold.

---

3 Visualization v2.0

Figures are nice, but we can go further, we can illustrate our results in interactive web applications!

Why?

- Achieving prediction accuracy with a fancy method is great, but without successful communication it is all for naught.
- Geometrical and visual representation enhances the understanding.
- It will improve our own understanding of the problem and the methods.
- It's fun and eye catching!

3.1 Developing a simple webapp with R package Shiny

There is a nice gallery of examples on the shiny website and they are a good starting point. Shiny is still in development so it is sometimes behaving inconsistently. At the moment they are hosting shiny applications for free on Shinyapps server, but that is not a long term solution if you would use it a lot - you will need to learn a bit about servers etc.

Web applications in shiny consist of two files: UI.R and Server.R. Best way to learn it is to take a look at an example. These are the UI.R and server.R from the "K means" example.

UI.R

```
shinyUI(pageWithSidebar(
  headerPanel('Iris k-means clustering'),
  sidebarPanel(
```

```
    selectInput('xcol', 'X Variable', names(iris)),
    selectInput('ycol', 'Y Variable', names(iris),
                selected=names(iris)[[2]]),
    numericInput('clusters', 'Cluster count', 3,
                 min = 1, max = 9)
  ),
  mainPanel(
    plotOutput('plot1')
  )
))
```

server.R

```
palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
  "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

shinyServer(function(input, output, session) {

    # Combine the selected variables into a new data frame
        selectedData <- reactive({
            iris[, c(input$xcol, input$ycol)]
    })

    clusters <- reactive({
        kmeans(selectedData(), input$clusters)
    })

    output$plot1 <- renderPlot({
        par(mar = c(5.1, 4.1, 0, 1))
        plot(selectedData(),
            col = clusters()$cluster,
            pch = 20, cex = 3)
            points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
    })

})
```

We can launch it locally by navigating first to the directory where our application is located and then running the following command: `shiny::runApp()`. This will open the application in a browser or simply give you an URL. In RStudio you have a nice button for it and it does everything automatically.

To deploy it publicly, you will need to set up an account on Shinyapps or install Shiny Server on your own server. In former case, after signing up and setting an account,

you would deploy your app by navigating to your application directory and running `shinyapps::deployApp(appName="someName")`.

There are similar commands for terminating (`shinyapps::terminateApp(appName="someName")`) and modifying the configuration of your app, e.g. using an extra large multiple core instance (`shinyapps:configureApp("someName", size="xxlarge")`).

3.2 "Alternatives" to Shiny

1. plotly - Language independent data visualization, together with hosting and data. Aimed at achieving the ideal of reproducibility.

2. D3 - Powerful Javascript library for interactive graphics in HTML. It is quite versatile, but there is a rather steep learning curve to get to the basic level. There are some packages that create basic type of interactive graphics in D3, take a look at the R2D3.

3. Google Charts - famous Hans Rosling TED talk few years ago featured moving charts that nicely illustrated evolution of some indicators over time (e.g. infant mortality and GDP in the world over time). Finally, Google bought the visualization libraries and improved them. There is an R package that facilitated creating such charts with R - googleVis, and it can be used within Shiny as well (albeit not without issues).

---

4 References

- For more details on discriminant functions, see Bishop sections 4.1.1 to 4.1.4. and Hastie, Tibshirani, Friedman, sections 2.3, 2.4, 4.1 and 4.2.
- For linear probability models, see for example Stock and Watson (2011): "Introduction to Econometrics"