# Data Mining and Machine Learning

February 08, 2016

# Where are we?

- Ask a question
- Get relevant data
- Prepare data for analysis
  - outliers, missing values, incorrect values
- Explore data
  - understand the world as it is (was)
- Statistical model
  - estimate/train and validate model
  - predict what will (likely) happen

  } Today
- Communicate results
  - tell a story
  - recommend

# Machine Learning vs Data Mining

- **Machine Learning** is the design of algorithms that can produce new knowledge from experience and do this automatically, without online human guidance.

- **Data mining** is carried out by a person with a particular goal in mind. It uses machine learning algorithms and may feed back to design better machine learning algorithms for the particular goal.

# General Idea of Modelling

$$Y = f(X_1, \ldots, X_p) + \epsilon$$

- $Y$ is the *dependent variable*, also known as *response* or *target*.
- $X_i$ are the *independent* variables, also known as *predictors* or *features*.

The solution of the model is $\hat{f}$, which yields

$$\hat{Y} = \hat{f}(X_1, \ldots, X_p)$$

- The error of $\hat{f}$ is *reducible*, because we can build a better approximation.
- The error $\epsilon$ is *irreducible*.

# Simple Linear Regression

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

The coefficients are calculated by minimizing the sum of squared residuals ($RSS$)

$$RSS = \sum_{i=1}^{n} |y_i - \hat{y}_i|^2 \longrightarrow \min$$

# Advertising dataset

```
> Advertising <- read.csv(
    'http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv')
>
> head(Advertising)
  X    TV Radio Newspaper Sales
1 1 230.1  37.8      69.2  22.1
2 2  44.5  39.3      45.1  10.4
3 3  17.2  45.9      69.3   9.3
4 4 151.5  41.3      58.5  18.5
5 5 180.8  10.8      58.4  12.9
6 6   8.7  48.9      75.0   7.2
> dim(Advertising)
[1] 200   5
```

# Linear Regression

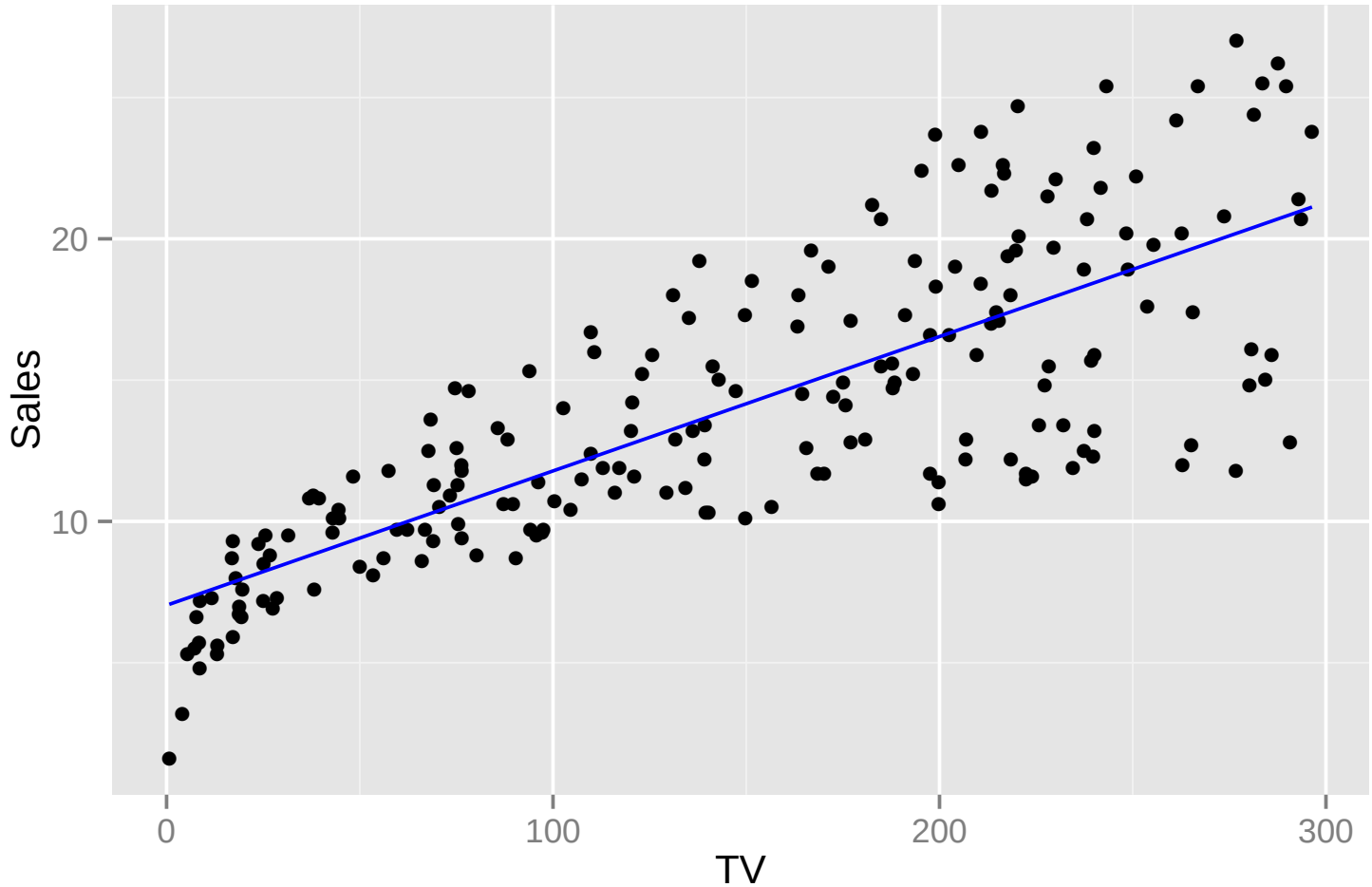
## Use `lm` to solve linear models

```
> mod <- lm(Sales ~ TV, data=Advertising)
> mod

Call:
lm(formula = Sales ~ TV, data = Advertising)

Coefficients:
(Intercept)           TV
    7.03259      0.04754
```

# Linear Regression

## Visualize the result

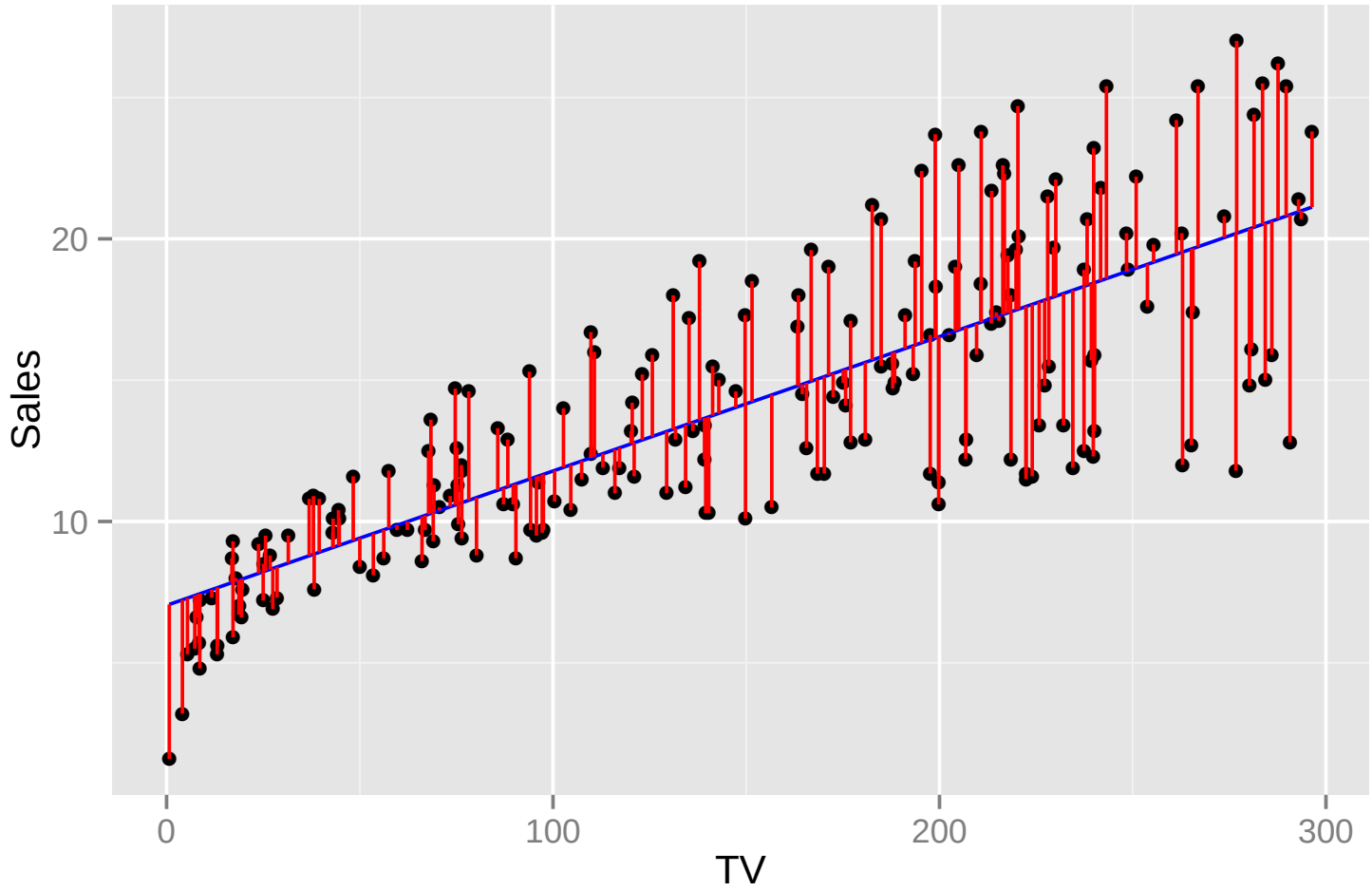
```
ggplot(mod) + geom_point(aes(x=TV, y=Sales)) +
    geom_line(aes(x=TV, y=.fitted), color="blue" )
```




# Linear Regression

## Residuals marked in red

```
last_plot() +
    geom_linerange(aes(x=TV, ymin=.fitted, ymax=Sales),
    color="red")
```




# Linear Regression

```
> mod <- lm(Sales~TV+Radio+Newspaper, data=Advertising)
> summary(mod)

Call:
lm(formula = Sales ~ TV + Radio + Newspaper, data = Advertising)

Residuals:
    Min      1Q  Median      3Q     Max
-8.8277 -0.8908  0.2418  1.1893  2.8292

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.938889   0.311908   9.422   <2e-16 ***
TV           0.045765   0.001395  32.809   <2e-16 ***
Radio        0.188530   0.008611  21.893   <2e-16 ***
Newspaper   -0.001037   0.005871  -0.177     0.86
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.686 on 196 degrees of freedom
Multiple R-squared:  0.8972,    Adjusted R-squared:  0.8956
F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

# Model Accuracy

- Residual Standard Error ($RSE$)

$$RSE = \sqrt{\frac{RSS}{n-p-1}}$$

is an estimate of the standard deviation of $\epsilon$

- $R^2$ statistic measures the ammount of variation in $Y$ explained by the model.

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$

where $TSS$ is the *total sum of squares*.

$$TSS = \sum_{i=1}^{n} (y_i - \overline{y})^2$$

# Variable Selection

```
> summary(lm(Sales~TV, data=Advertising))
     *    *    *
Residual standard error: 3.259 on 198 degrees of freedom
Multiple R-squared:  0.6119,    Adjusted R-squared:  0.6099
F-statistic: 312.1 on 1 and 198 DF,  p-value: < 2.2e-16
>
> summary(lm(Sales~TV+Radio, data=Advertising))
     *    *    *
Residual standard error: 1.681 on 197 degrees of freedom
Multiple R-squared:  0.8972,    Adjusted R-squared:  0.8962
F-statistic: 859.6 on 2 and 197 DF,  p-value: < 2.2e-16
>
> summary(lm(Sales~TV+Radio+Newspaper, data=Advertising))
     *    *    *
Residual standard error: 1.686 on 196 degrees of freedom
Multiple R-squared:  0.8972,    Adjusted R-squared:  0.8956
F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

# Variable Selection

With $p$ predictors there are $2^{1+p}$ linear regression models!

- *Forward selection* - start with empty model and add the variable that improves the most. Iterate until improvement is not significant.
- *Backward selection* - start with a model including all variables and iteratively remove the one with the largest $p$-value.
- *Mixed selection* - like forward selection, but do a backward step if a $p$-value gets too high.

# Stepwise Regression

```
>  library(MASS)

> fit <- lm(Sales~TV+Radio+Newspaper, data=Advertising)

> step <- stepAIC(fit, direction="both")

> step$anova # display results
Stepwise Model Path
Analysis of Deviance Table

Initial Model:
Sales ~ TV + Radio + Newspaper

Final Model:
Sales ~ TV + Radio

         Step Df      Deviance Resid. Df Resid. Dev  AIC
1                                   196   556.8253  212.7868
2 - Newspaper  1 0.08871717   197   556.9140  210.8187
```

# Categorical predictors

- For binary predictor use a *dummy variable*

$$d_i = \begin{cases} 0, & x = \text{FALSE}; \\ 1, & x = \text{TRUE}. \end{cases}$$

- The coefficient $\beta_i$ is the average difference in response between the two categories.

- For predictors with $m$ levels use $m - 1$ dummy variables.

$$d_{ij} = \begin{cases} 0, & x \neq \text{category } j; \\ 1, & x = \text{category } j. \end{cases} \qquad \text{for } j = 1, \ldots, m-1$$

- Then $\beta_{ij}$ is the average difference between categories $j$ and $m$.
- Do not use integers $1, 2, \ldots, m$ to represent the $m$ categories!

# Logistic regression

- If the response, $Y$, is a binary categorical variable we may use regression to estimate the probability $p(Y)$.

$$p(Y) = \beta_0 + \beta_1 X$$

# Logistic regression

- The logistic function,

$$f(x) = \frac{e^x}{1 + e^x},$$

  is always between 0 and 1.

- In logistic regression we model $p(Y)$ as

$$p(Y) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

  or, equivalently,

$$\log\left(\frac{p(Y)}{1 - p(Y)}\right) = \beta_0 + \beta_1 X$$

- The ratio $p(Y)/(1 - p(Y))$ is called *odds* of success.

# Logistic regression

- Coefficients can be interpreted as the amount of change in log-odds resulting from change in $X$ by 1 unit.
- For example, if $\beta_1 = -0.2$, then when $X$ increases from 0 to 1, the log-odds changes by $-0.2$.

  Then the odds change by a factor of $e^{-0.2} = 0.82$, i.e. the odds decline by 18%

# Logistic Regression Example

- Dataset `Default` from package `ISLR` contains simulated data of credit card defaults.
- Use `glm` with `family=binomial` to run a logistic regression.

```
> library(ISLR)
> data(Default)
> lr.fit <- glm(default ~ ., data=Default, family=binomial)
    *   *   *
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
balance      5.737e-03  2.319e-04  24.738  < 2e-16 ***
income       3.033e-06  8.203e-06   0.370  0.71152
    *   *   *
>
```

# Logistic Regression Example

```
> # Our model's response is probabilities
> lr.prob <- predict(lr.fit, type="response")
> lr.prob[1:5]
           1            2            3            4            5
0.0014287239 0.0011222039 0.0098122716 0.0004415893 0.0019355062
>
> # Convert the probabilities to predicted values
> lr.pred <- cut(lr.prob, breaks=c(0,0.5,1),
        labels=c("No", "Yes"), include.lowest=TRUE)
> lr.pred[1:5]
[1] No No No No No
Levels: No Yes
>
> # construct the confusion matrix
> lr.ct <- table(Actual=Default$default, Predicted=lr.pred)
      Predicted
Actual   No  Yes
   No  9627   40
   Yes  228  105
```

# Logistic Regression Example

- *Accuracy* of the model is the proportion of cases that were predicted correctly

```
> (lr.ct["Yes", "Yes"] + lr.ct["No", "No"]) / sum(lr.ct)
[1] 0.9732
```

```
> # Leave 20% of dataset for testing
> test <- sample(c(FALSE, TRUE), nrow(Default), replace=TRUE,
                          prob=c(0.8,0.2))
> train <- !test
> train.fit <- glm(default ~ ., data=Default, family=binomial,
                          subset=train)
> test.prob <- predict(train.fit, type="response",
                          newdata=subset(Default, test))
> test.pred <- cut(test.prob, breaks=c(0,0.5,1),
        labels=c("No", "Yes"), include.lowest=TRUE)
> (test.ct <- table(Adtual=Default$default[test], Predicted=test.pred))
      Predicted
Adtual   No  Yes
   No  1878    1
   Yes   44   25
>  sum(diag(test.ct)) / sum(test.ct) # Accuracy:
[1] 0.9768994
```

# Rare groups

- Notice that the probability of default is low, i.e. the people who default are a rare group

```
> (tbl <- table(Default$default))
  No  Yes
9667  333
> tbl["Yes"] / sum(tbl)
   Yes
0.0333
```

- If we want to analyze only the people who default, we find that our method misses more cases default than identifies! That's bad!

```
> test.ct
       Predicted
Adtual   No  Yes
   No  1878    1
   Yes   44   25
```

# Sensitivity and Specificity

- *Sensitivity* is the fraction of actual positives that were correctly classified.
- *Specificity* is the fraction of actual negatives that were correctly classified.

```
> # Sensitivity is
> test.ct["Yes", "Yes"] / sum(test.ct["Yes", ])
[1] 0.3623188
>
> # Specificity is
> test.ct["No", "No"] / sum(test.ct["No", ])
[1] 0.9994678
```

# Precision and Recall

- *Precision* is the fraction of cases classified as positives that actually are.
- *Recall* is the fraction of actual positives that were correctly classified. Same as sensitivity.

```
> # Precision is
> test.ct["Yes", "Yes"] / sum(test.ct[, "Yes"])
[1] 0.9615385
>
> # Recall is the same as sensitivity
> test.ct["Yes", "Yes"] / sum(test.ct["Yes", ])
[1] 0.3623188
```

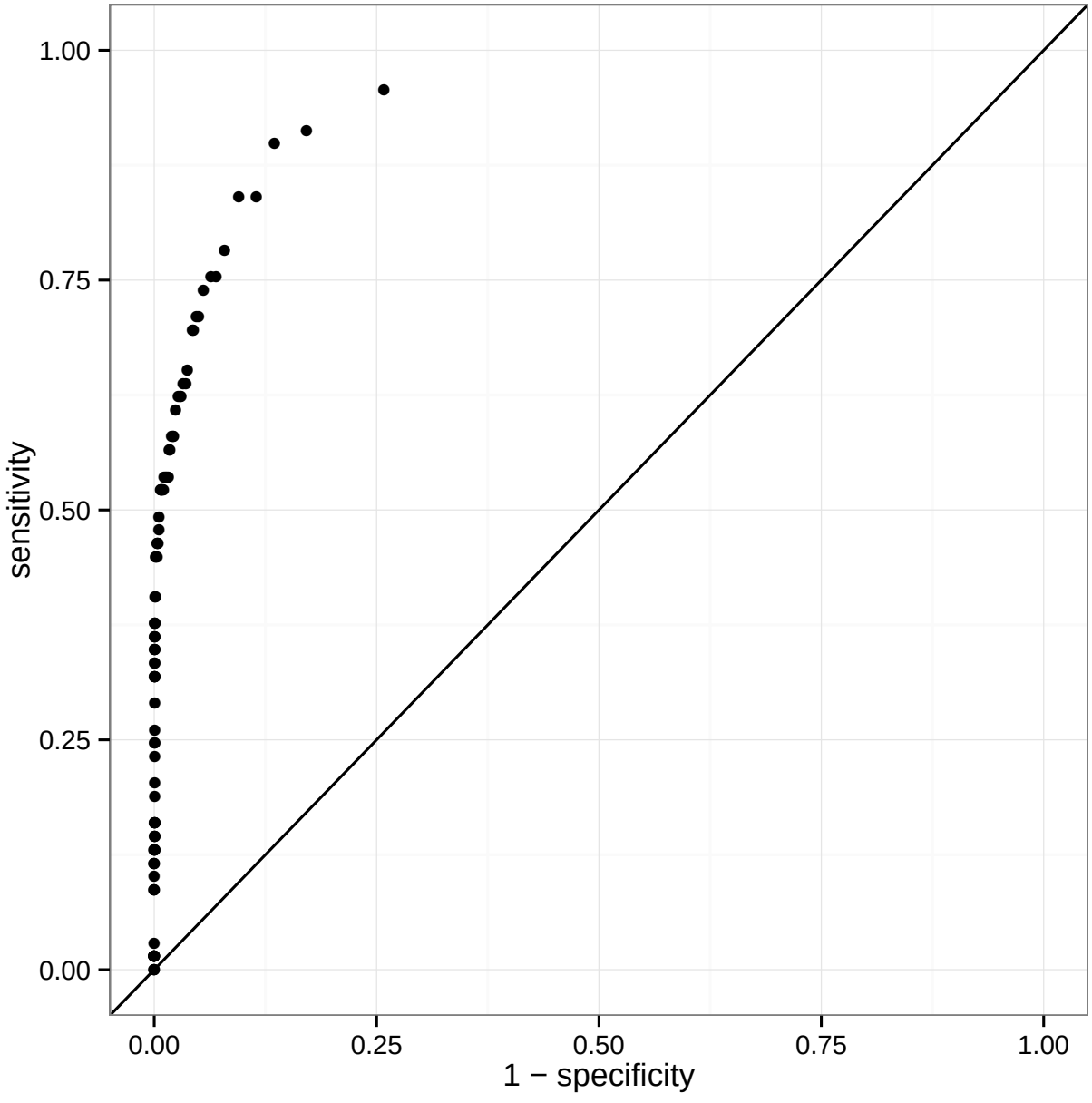
# ROC curve

- ROC curve is a plot of sensitivity vs. specificity for varying cut-off values.

```
roc.plot <- function(Actual, Prob) {
    ss <- lapply( seq(0.01, 1-0.01, by=0.01), function(p) {
        tt <- table( Actual, factor(Prob>p, levels=c(FALSE, TRUE)) )
        data.frame( sensitivity=tt[2,2]/sum(tt[2,]),
                    specificity=tt[1,1]/sum(tt[1,]) ) })
    ss <- do.call("rbind", ss)
    qplot(1-specificity, sensitivity, data=ss) +
        xlim(0,1) + ylim(0,1) +
        geom_abline(intercept=0, slope=1) + theme_bw()
}
```

# ROC curve

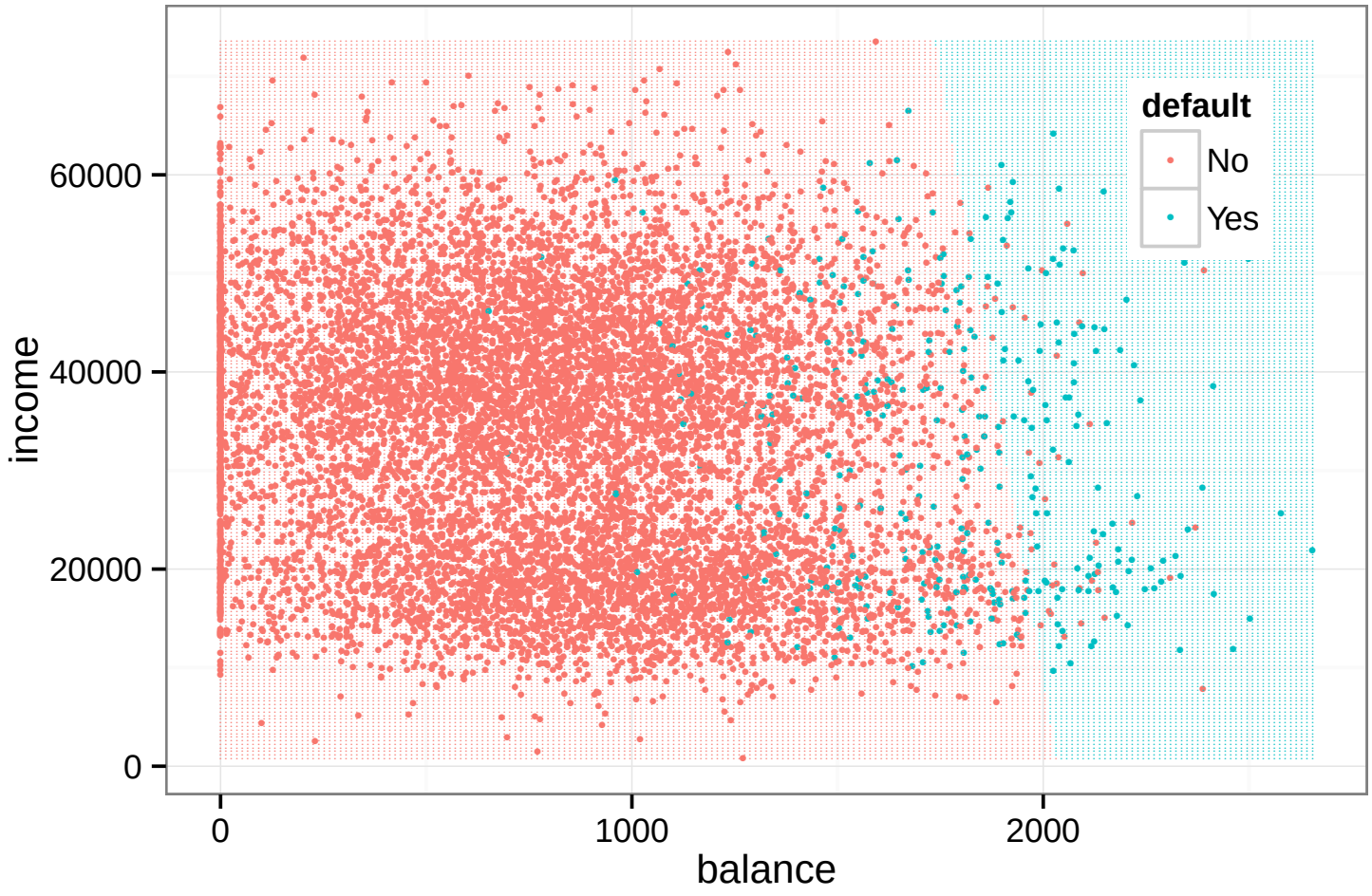
```
roc.plot(Default$default[test], test.prob)
```




# Visualization

```
train.fit <- glm(default ~ balance + income, data=Default,
             family=binomial, subset=train)
grid <- with(Default, expand.grid(
  balance=seq(floor(min(balance)), ceiling(max(balance)), length.out=201),
  income=seq(floor(min(income)), ceiling(max(income)), length.out=201)))
grid <- mutate(grid, prob=predict(train.fit, newdata=grid,
type="response"), pred=factor(prob>0.5, labels=levels(Default$default)))
qplot(balance, income, data=Default, colour=default, size=I(0.9)) +
     geom_point(aes(color=pred), data=grid, size=I(0.2))
```

# Visualization

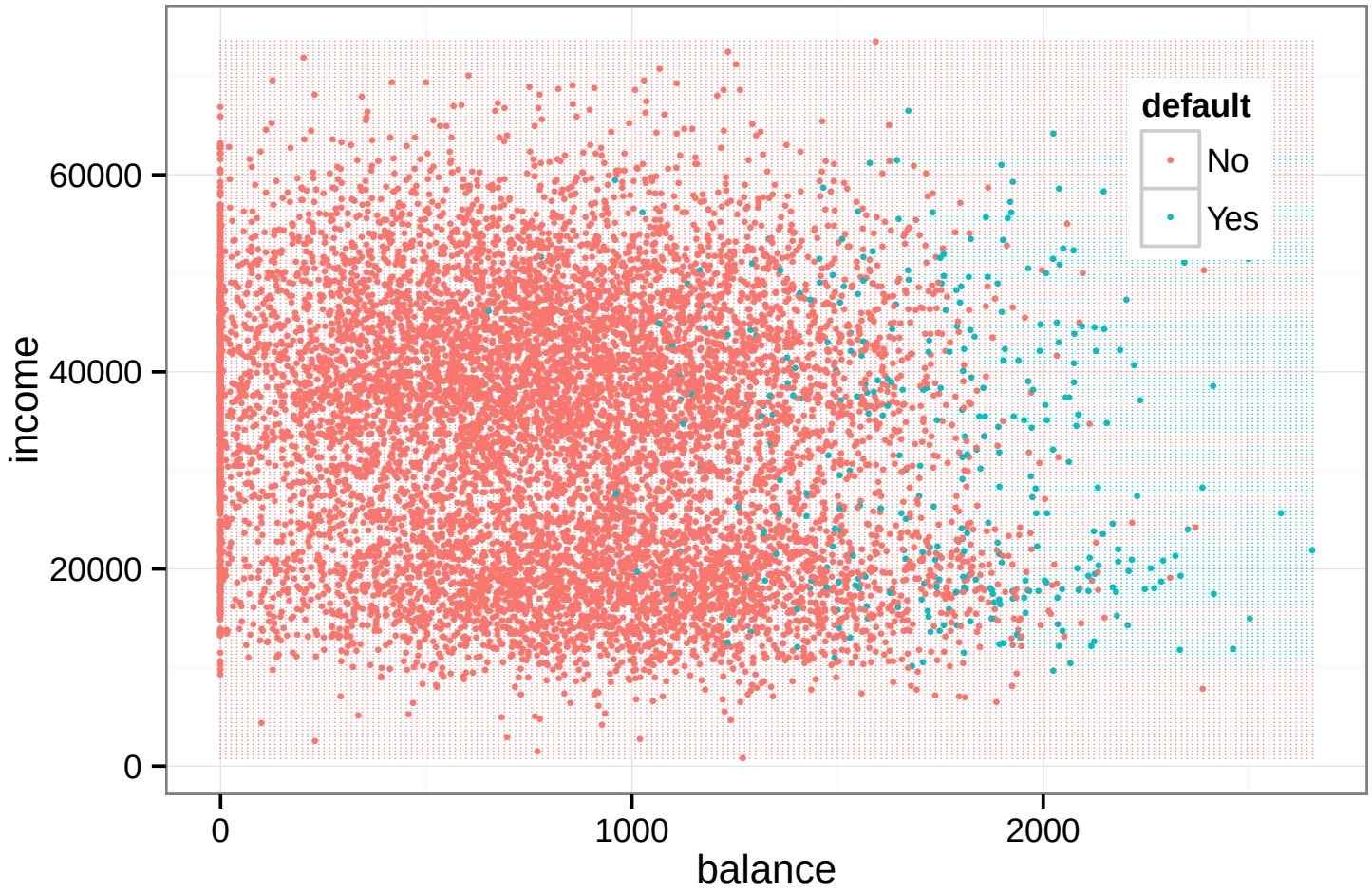



# *k*-Nearest Neighbour

- For each point where prediction is needed, find the *k* nearest points in the training set and use majority vote. Ties are broken at random.

```
library(class)
train.X <- model.matrix(~0+balance+income, data=subset(Default,train))
test.X <- model.matrix(~0+balance+income, data=subset(Default,test))
train.Y <- Default$default[train]
test.Yknn <- knn(train.X, test.X, train.Y, k=5)
table(Actual=Default$default[test], Predicted=test.Yknn)
      Predicted
Actual   No  Yes
   No  1912   11
   Yes   54    4

grid.knn <- knn(train.X, grid[,c("balance", "income")], train.Y, k=5)
p <- qplot(balance, income, data=Default, colour=default, size=I(0.9))
p <- p + geom_point(aes(color=grid.knn), data=grid, size=I(0.2))
p + theme_bw() + theme(legend.justification=c(1,1),
                       legend.position=c(0.95,0.95))
```

# Visualization of $k$-NN




# Trees

- Split the $p$-dimensional space of $X_1, \ldots, X_p$ into $J$ disjoint regions, $R_1, \ldots, R_J$.
- For every point in $R_j$ set the predicted value of $\hat{Y}$ to the average observed $Y$ in $R_j$.

How to form the regions $R_j$, so that:

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} \left( y_i - \overline{y}_{R_j} \right)^2 \longrightarrow \min$$

# Trees

- *Recursive binary splitting* algorithm is a greedy, top-down solution.
- We select a predictor $X_j$ and a cut point $c_j$ and form two regions $R_1 = \{X|X_j < c_j\}$ and $R_2 = \{X|c_j \leq X_j\}$ so that the resulting *RSS* is minimal.
- At the next step we do the same, but this time we split one of the regions we already have in a way that makes the greatest improvement of *RSS*.

# Trees

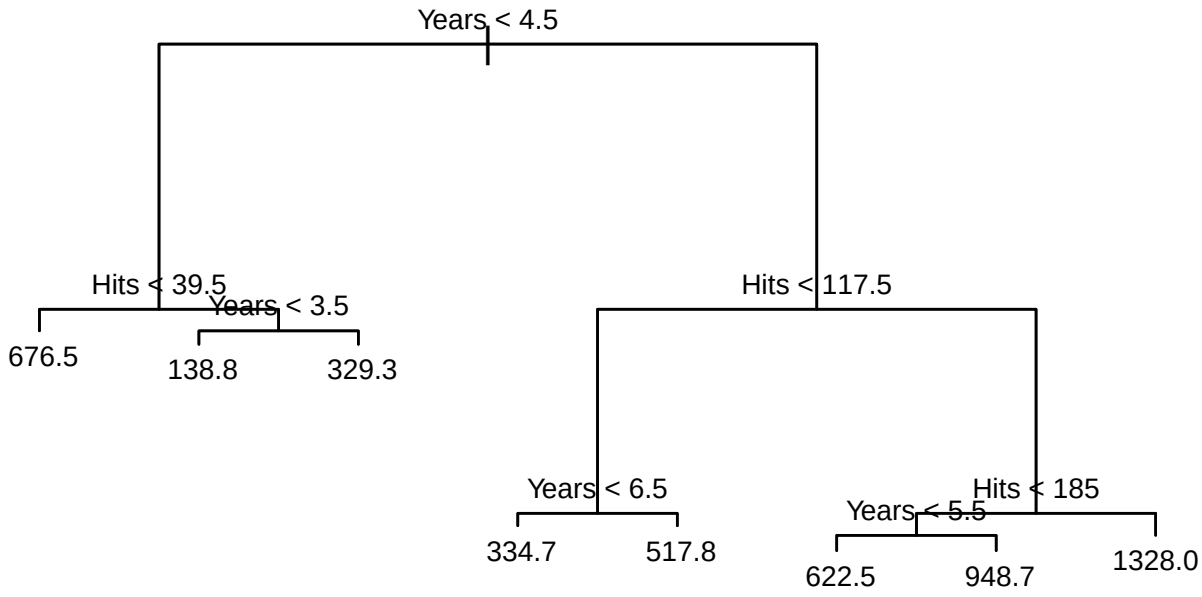
- Trees are intuitive: easy to explain and easy to understand by non-experts.
- Easily handle numerical and categorical variables with any number of levels in both predictors and response without need for dummy variables.
- Generally the predictive power is lower than other methods.

# Example of Regression Tree

```
data(Hitters)
H <- Hitters[, c("Hits", "Years", "Salary")]
H <- subset(H, !is.na(Salary))
library(tree)
tree.H <- tree(Salary~Hits+Years, data=H)

node), split, n, deviance, yval
      * denotes terminal node

 1) root 263 53320000  535.9
   2) Years < 4.5 90  6769000  225.8
     4) Hits < 39.5 5  3131000  676.5 *
     5) Hits > 39.5 85  2564000  199.3
      10) Years < 3.5 58   309400  138.8 *
      11) Years > 3.5 27  1586000  329.3 *
   3) Years > 4.5 173 33390000  697.2
     6) Hits < 117.5 90  5312000  464.9
      12) Years < 6.5 26   644100  334.7 *
      13) Years > 6.5 64  4048000  517.8 *
     7) Hits > 117.5 83 17960000  949.2
      14) Hits < 185 76 13290000  914.3
        28) Years < 5.5 8    82790  622.5 *
        29) Years > 5.5 68 12450000  948.7 *
      15) Hits > 185 7  3571000 1328.0 *
```

# Example of Regression Tree

```
plot(tree.H)
text(tree.H, cex=0.6)
```




# Example of Classification Tree

```
tree.D <- tree(default~., data=Default, subset=train)
D.pred <- predict(tree.D, newdata=subset(Default, test), type="class")
table(Default$default[test], D.pred)
     D.pred
        No  Yes
  No  1906   17
  Yes   42   16
tree.D
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 8019 2395.00 No ( 0.965706 0.034294 )
   2) balance < 1472.99 7204  566.70 No ( 0.993476 0.006524 )
     4) balance < 1099.01 5669   80.33 No ( 0.999118 0.000882 ) *
     5) balance > 1099.01 1535  385.10 No ( 0.972638 0.027362 ) *
   3) balance > 1472.99 815  966.10 No ( 0.720245 0.279755 )
     6) balance < 1891.6 669  638.50 No ( 0.816143 0.183857 )
      12) balance < 1706.96 465  353.80 No ( 0.873118 0.126882 ) *
      13) balance > 1706.96 204  253.80 No ( 0.686275 0.313725 ) *
     7) balance > 1891.6 146  173.40 Yes ( 0.280822 0.719178 ) *
```

# Example of Regression Tree

```
plot(tree.D)
text(tree.D, cex=0.7)
```