Lecture 5

Feature Engineering

Assoc. Prof. Dr. Burkay Genç

2024-03-18

Packages used in these slides

```
library(knitr)     # used for slides rendering
library(dplyr)
library(readr)
library(stringr)
library(lubridate)
library(xts)
library(sp)
library(COREelearn)
```

Seed used in these slides

```
set.seed(1024)
```

Modifying Data

- Many data mining tools have **requirements**
  - *Tabular* data
  - *Independent* columns and rows
  - *Numeric* or *categorical* data columns
- Or, you voluntarily make modifications on the dataset
  - *Too many features*
  - *Too many observations*
  - Unnecessary *detail* levels
  - Deriving *more useful* features

What is feature engineering?

- **Feature engineering** is the process of modifying raw data in order to make it more suitable for *efficient extraction* of wisdom through machine learning approaches.
- In R programming, feature engineering can be done through a variety of functions.
  - Trivial approaches (requires a priori knowledge)
    - Select columns with `dplyr::select()`
    - Filter rows with `dplyr::filter()`
    - Group by one or more variables with `dplyr::group_by()`
    - Normalization, scaling, encoding
  - Sophisticated approaches
    - Tidyfication
    - Temporal and spatial dependency elimination
    - Dİmensionality reduction
    - Feature extraction

How to do it?

- Feature engineering is not a one time job
- You must think about it iteratively
- If you are not satisifed with the end results, revisit feature engineering
- A few early warnings
    - Do not rush to remove data
    - Do not over simplify and lose information
    - Do not overlook details
        - Be patient and accurate

Temporal and Spatial Dependencies

Handling Observation Dependencies

- Observations may not be completely independent
  - **Contextual information**
    - determines the context of observation
    - where and when the observation was made
  - **Behavioral information**
    - the actual values measured for variables

Contextual Dependencies

- Place the observation in a context
  - *Where* it happened?
  - *When* it happened?
  - *How* it happened?
- Context usually enforces *similarity* of observations

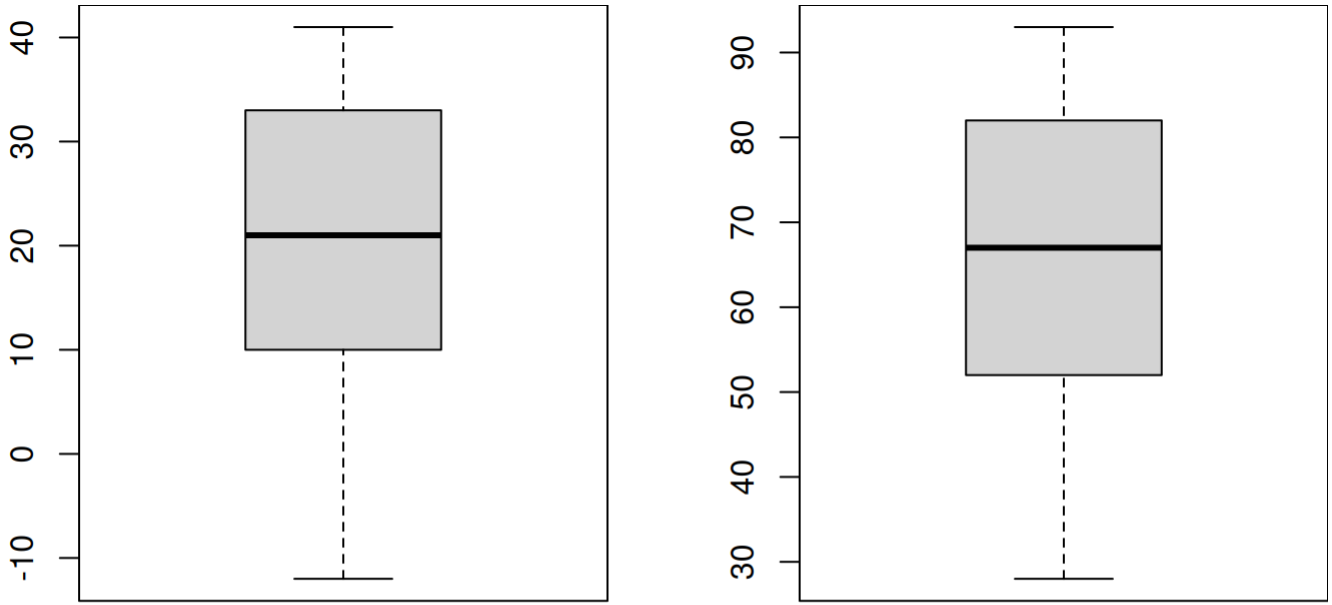
Contextual Dependency - where

City	Temperature	Humidity
Ankara	20	55
Ankara	35	28
Antalya	30	82
Antalya	22	88
Ankara	10	61
Antalya	41	93
Ankara	-12	42
Antalya	13	73
Antalya	33	78
Ankara	0	52

The data as is doesn’t tell much.

Contextual Dependency - where

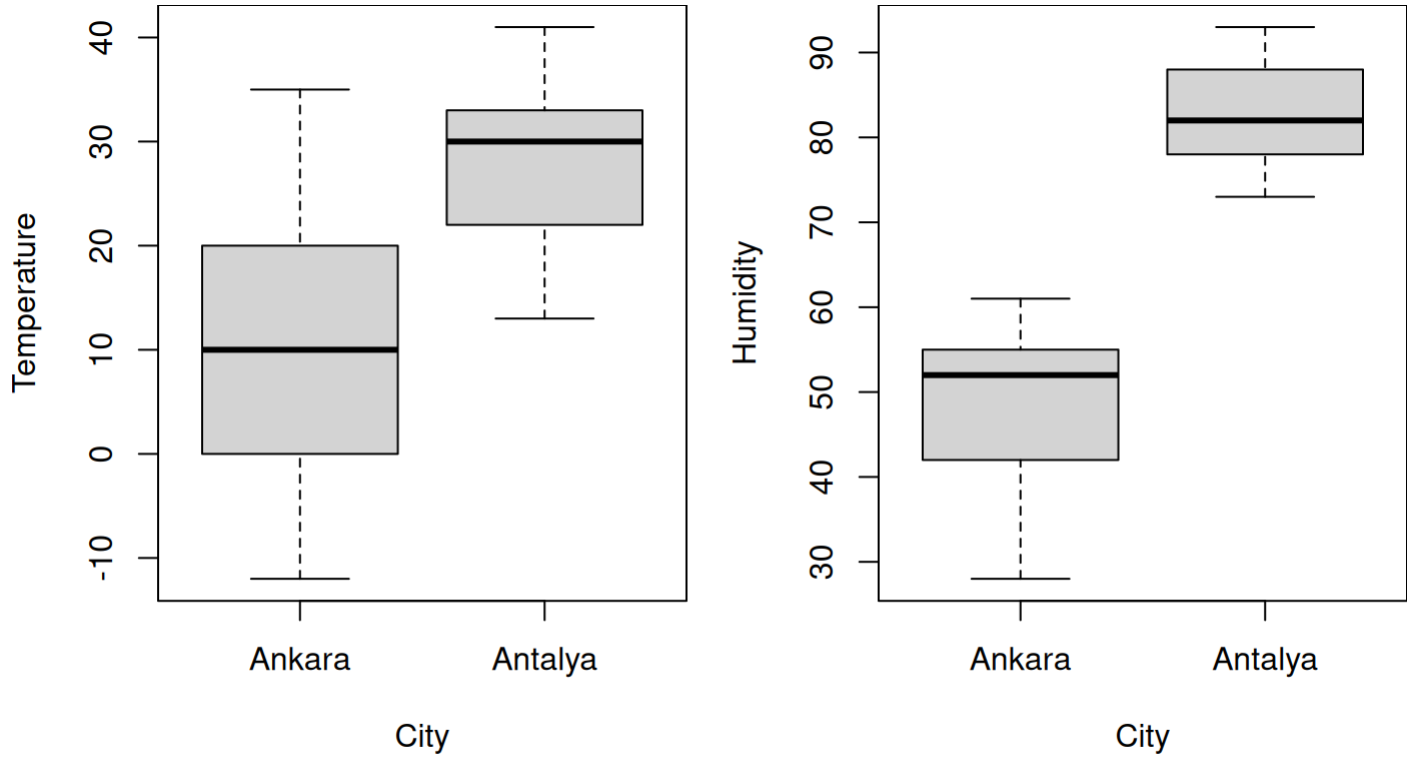
```
spatial_weather <- data.frame(
  City = c("Ankara", "Ankara", "Antalya", "Antalya", "Ankara",
           "Antalya", "Ankara", "Antalya", "Antalya", "Ankara"),
  Temperature = c(20, 35, 30, 22, 10, 41, -12, 13, 33,  0),
  Humidity =    c(55, 28, 82, 88, 61, 93,  42, 73, 78, 52)
)

attach(spatial_weather)
par(mfrow=c(1,2), mar=c(7,5,0,0))
boxplot(Temperature)
boxplot(Humidity)
```



Contextual Dependency - where

What if we add some context?

```
attach(spatial_weather)
par(mfrow=c(1,2), mar=c(7,5,0,0))
boxplot(Temperature~City)
boxplot(Humidity~City)
```

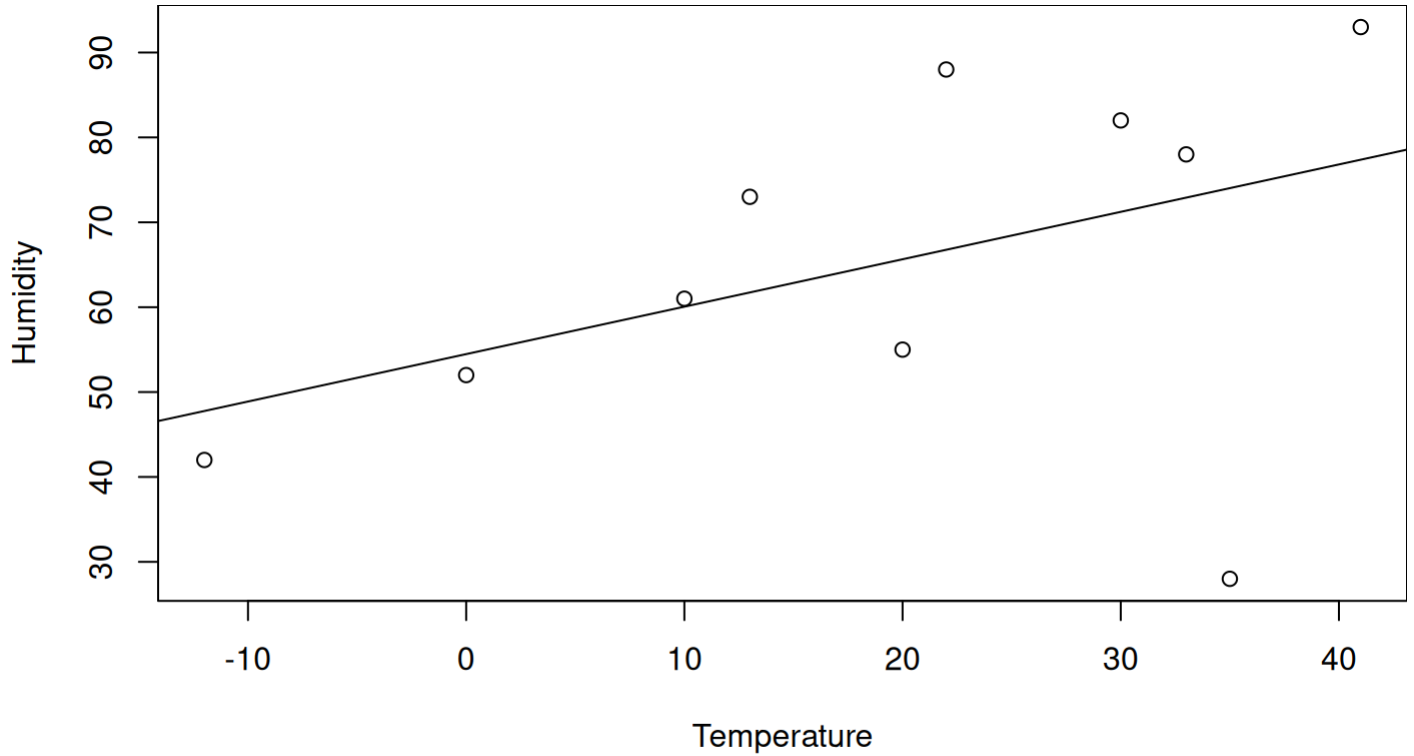



```
detach(spatial_weather)
```

Contextual Dependency - where

What about relations between variables?

```
attach(spatial_weather)
par(mar=c(7,5,0,0))
plot(Humidity~Temperature)
abline(lm(Humidity~Temperature, spatial_weather))
```

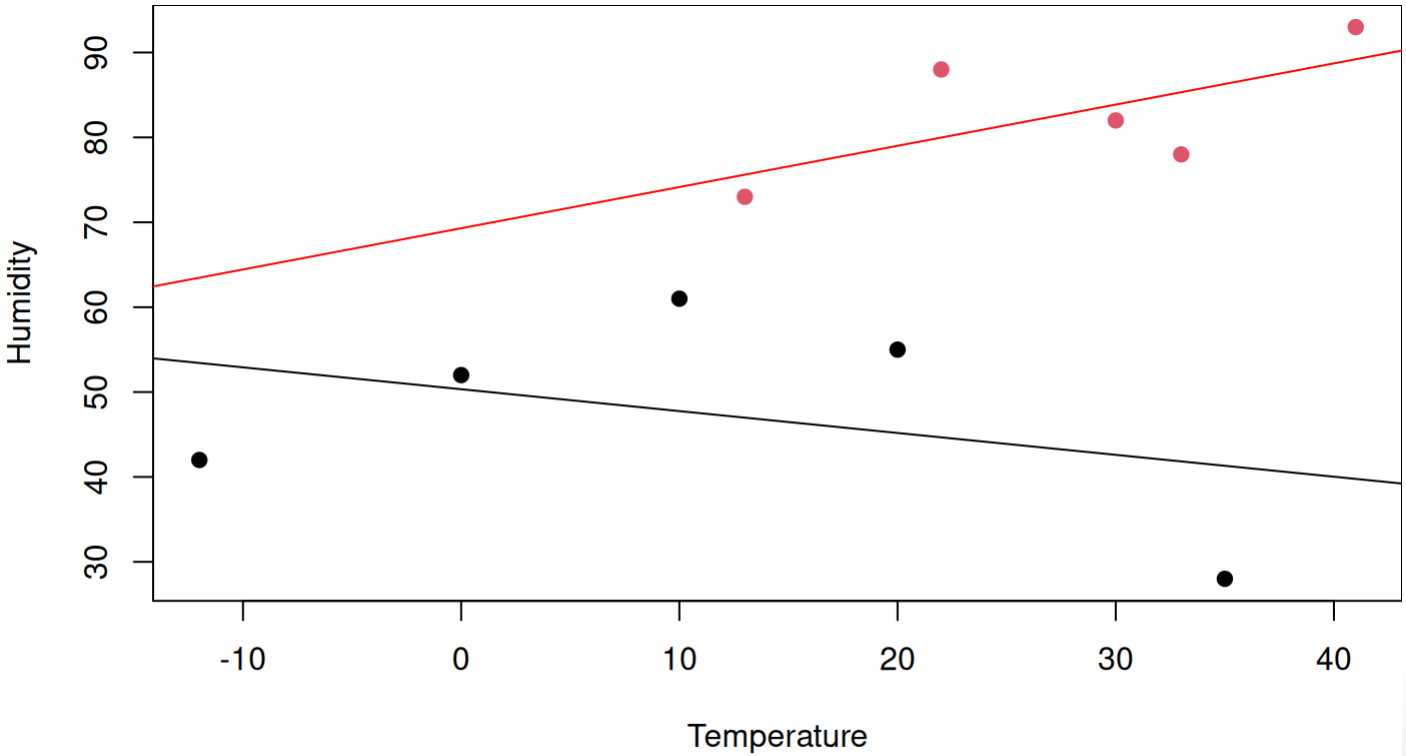



```
detach(spatial_weather)
```

Contextual Dependency - where

With context:

```
attach(spatial_weather)
par(mar=c(7,5,0,0))
plot(Humidity~Temperature, col=factor(City), pch=19)
abline(lm(Humidity~Temperature,
          spatial_weather[spatial_weather$City=="Ankara",]),
       col="black")
abline(lm(Humidity~Temperature,
          spatial_weather[spatial_weather$City=="Antalya",]),
       col="red")
```



Handling Observation Dependencies

- Contextual Dependencies
  - Time dependency
  - Space dependency
    - Lattitude, longitude, etc.
- What to do?
  - Use **related** suitable tools
    - such as time series specific algorithms
      - Autoregressive, ARIMA, SARIMA, Exponential Smoothing…
  - Modify data to remove dependencies or **make them useful**

Time Dependency

- Temperature data
  - You expect a correlation between time t and $t + 1$
  - This is called a *times series* data
  - And it has its own field of research and tools
    - TimeSeries Task View
- Doing data mining on time series data is tricky
  - The time dependency may easily dominate other patterns
  - House pricing, car pricing, etc.

`xts` package

- The `xts` package is used for working with time series data

```
sp500 <- xts(c(1102.94,
               1104.49,
               1115.71,
               1118.31),
             ymd(c("2010-02-25",
                   "2010-02-26",
                   "2010-03-01",
                   "2010-03-02")))
sp500
```

```
##                 [,1]
## 2010-02-25 1102.94
## 2010-02-26 1104.49
## 2010-03-01 1115.71
## 2010-03-02 1118.31
```

- Note that the time stamps are given as row names

```
sp500["2010-03-02"]
```

```
##                 [,1]
## 2010-03-02 1118.31
```

```
sp500["2010-03"]
```

```
##                 [,1]
## 2010-03-01 1115.71
## 2010-03-02 1118.31
```

```
sp500["2010-03-01/"]
```

```
##                 [,1]
## 2010-03-01 1115.71
## 2010-03-02 1118.31
```

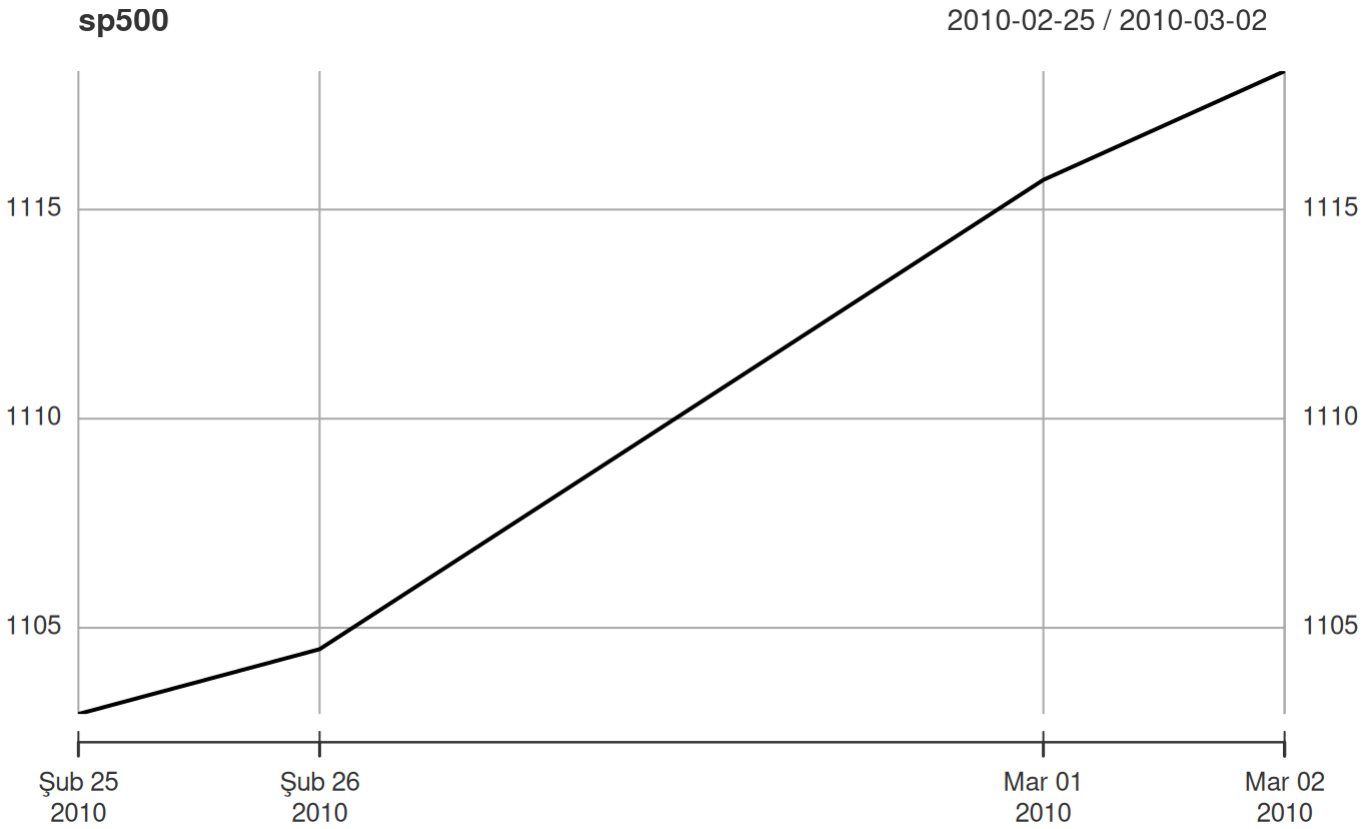
```
sp500["2010-02-26/2010-03-01"]
```

```
##                 [,1]
## 2010-02-26 1104.49
## 2010-03-01 1115.71
```

`xts` package

- `xts` also allows nice plots of time series data

```
plot(sp500)
```



Time Dependency

- If there is a **trend in target value** w.r.t. time, generic methods fail to make future predictions
  - because the past and the future are inherently different
- We can transform target values to minimize effects of time
  - **Relative vs absolute values**: the following formula converts absolute values to relative values

$$Y_t = \frac{X_t - X_{t-1}}{X_{t-1}}$$

  - avoid using ($X_t - X_{t-1}$) for tranformation as it doesn't fully hide the effect of time
    - the difference may also grow in time

Example

```
data(AirPassengers)
AirPassengers
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

```
ap <- as.xts(AirPassengers)
head(ap)
```

```
##          [,1]
## Oca 1949  112
## Şub 1949  118
## Mar 1949  132
## Nis 1949  129
## May 1949  121
## Haz 1949  135
```

Example

```
head(diff(ap))
```

```
##          [,1]
## Oca 1949   NA
## Şub 1949    6
## Mar 1949   14
## Nis 1949   -3
## May 1949   -8
## Haz 1949   14
```

```
tail(diff(ap))
```

```
##          [,1]
## Tem 1960   87
## Ağu 1960  -16
## Eyl 1960  -98
## Eki 1960  -47
## Kas 1960  -71
## Ara 1960   42
```

```
apRel <- diff(ap) / ap
head(apRel)
```

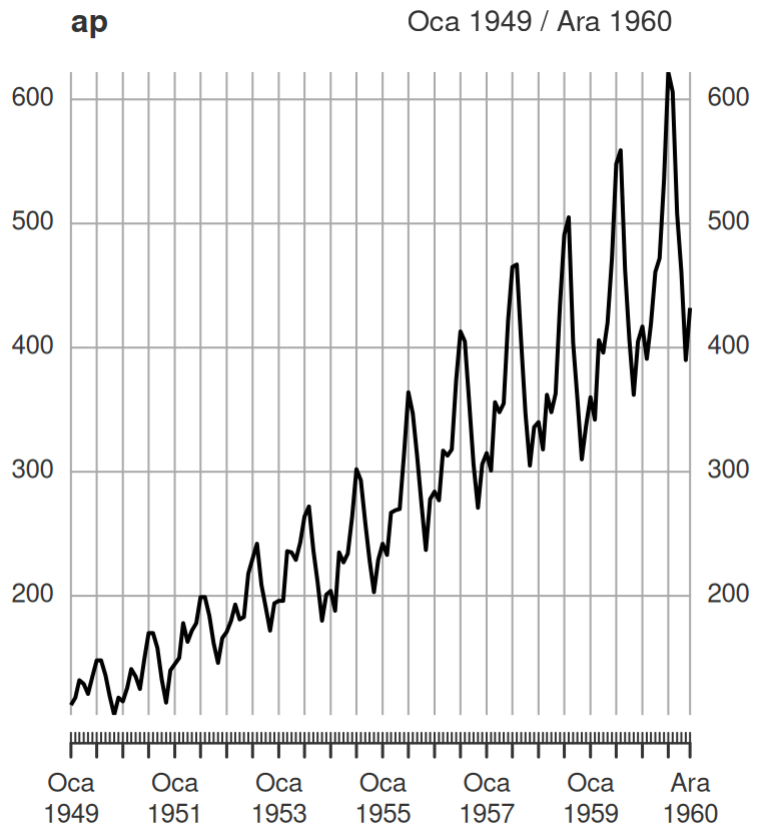
```
##                  [,1]
## Oca 1949           NA
## Şub 1949  0.05084746
## Mar 1949  0.10606061
## Nis 1949 -0.02325581
## May 1949 -0.06611570
## Haz 1949  0.10370370
```

```
tail(apRel)
```

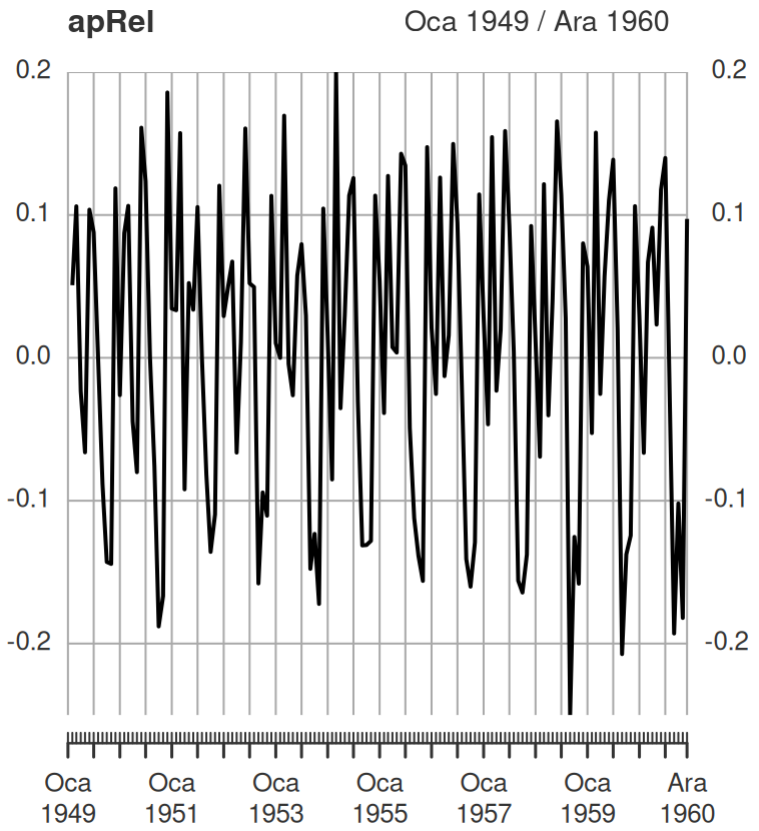
```
##                  [,1]
## Tem 1960  0.13987138
## Ağu 1960 -0.02640264
## Eyl 1960 -0.19291339
## Eki 1960 -0.10195228
## Kas 1960 -0.18205128
## Ara 1960  0.09722222
```

Example

```
plot(ap)
```




```
plot(apRel)
```



embed Function

```
head(apRel, 10)
```

```
##                 [,1]
## Oca 1949          NA
## Şub 1949  0.05084746
## Mar 1949  0.10606061
## Nis 1949 -0.02325581
## May 1949 -0.06611570
## Haz 1949  0.10370370
## Tem 1949  0.08783784
## Ağu 1949  0.00000000
## Eyl 1949 -0.08823529
## Eki 1949 -0.14285714
```

```
head(embed(apRel[-1], 5))
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]
## [1,]  0.10370370 -0.06611570 -0.02325581  0.10606061  0.05084746
## [2,]  0.08783784  0.10370370 -0.06611570 -0.02325581  0.10606061
## [3,]  0.00000000  0.08783784  0.10370370 -0.06611570 -0.02325581
## [4,] -0.08823529  0.00000000  0.08783784  0.10370370 -0.06611570
## [5,] -0.14285714 -0.08823529  0.00000000  0.08783784  0.10370370
## [6,] -0.14423077 -0.14285714 -0.08823529  0.00000000  0.08783784
```

Time Dependency - Other Methods

- Discrete Wavelet Transform (DWT)
- Discrete Fourier Transform (DFT)
- Their details are beyond our scope

Spatial Dependency

> “Everything is related with everything else, but near things are more related than distant things.”
>
> First law of geography, (Tobler, 1970)

- Usually comes in lat, long value pairs
- Also, city names, municipalities, town names, street names etc.
- Package `sp` is useful in spatial analysis

Spatial Dependency - sp

```
# https://web.cs.dal.ca/~ltorgo/AuxFiles/forestFires.txt
ff <- read_csv("forestFires.txt")
print(ff, width=70)
```

```
## # A tibble: 25,000 × 14
##    FID_    CID ano1991 ano1992 ano1993 ano1994 ano1995 ano1996 ano1997
##    <lgl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 NA        1       0       0       0       0       0       0       0
##  2 NA        2       0       0       0       0       0       0       0
##  3 NA        3       0       0       0       0       0       0       0
##  4 NA        4       0       0       0       0       0       0       0
##  5 NA        5       0       0       0       0       0       0       0
##  6 NA        6       0       0       0       0       0       0       0
##  7 NA        7       0       0       0       0       0       0       0
##  8 NA        8       0       0       0       0       0       0       0
##  9 NA        9       0       0       0       0       0       0       0
## 10 NA       10       0       0       0       0       0       0       0
## # i 24,990 more rows
## # i 5 more variables: ano1998 <dbl>, ano1999 <dbl>, ano2000 <dbl>,
## #   x <dbl>, y <dbl>
```

- Each row is a location
- First two columns are irrelevant
- `anoX` : Fire happened in year X
- Last two columns give location coordinates

Spatial Dependency - sp

```
spatialCoord <- select(ff, long = x, lat = y)
spatialData <- select(ff, Year2000 = ano2000)
coordRefSys <- CRS("+proj=longlat +ellps=WGS84")
fires <- SpatialPointsDataFrame(spatialCoord,
                                spatialData,
                                proj4string = coordRefSys)
head(fires)
```

```
##           coordinates Year2000
## 1 (-7.31924, 38.5406)        0
## 2 (-7.63557, 40.5022)        0
## 3 (-7.90273, 40.3418)        0
## 4 (-7.25657, 39.2572)        0
## 5 (-8.50379, 37.3445)        0
## 6  (-8.05975, 41.562)        0
```

Spatial Dependency - sp

```
bbox(fires)
```

```
##           min      max
## long -9.49174 -6.20743
## lat  36.98050 42.14360
```

```
head(coordinates(fires))
```

```
##          long     lat
## [1,] -7.31924 38.5406
## [2,] -7.63557 40.5022
## [3,] -7.90273 40.3418
## [4,] -7.25657 39.2572
## [5,] -8.50379 37.3445
## [6,] -8.05975 41.5620
```

Text Processing

Handling Text Datasets

- Text mining has its own literature
  - **bag of words**
    - binary
    - counts
  - **TF-IDF**
  - **n-grams**
- Package `tm` provides useful functions

Dimensionality Reduction

Dataset Size

Columns

- Having many columns is good
- Having **too** many columns is bad
- Having **many more columns than rows** is very bad

Rows

- Having many rows is good
- Having **more rows than your RAM can handle** is bad

Sampling Rows

- A good way of cutting down on rows is **random sampling** them
  - use `sample()`

```
data(iris)
sampleRate <- 0.7
sampledRows <- sample(1:nrow(iris), nrow(iris) * sampleRate)
iris.sample <- iris[sampledRows,]
```

- or `sample.int()`

```
data(iris)
sampleRate <- 0.7
sampledRows <- sample.int(nrow(iris), nrow(iris) * sampleRate)
iris.sample <- iris[sampledRows,]
```

- However, we still have to load the dataset into RAM
  - What if we can't?

Sampling Very Large Datasets

A pseudocode for very large dataset sampling

- Read the large file **line by line**
- For each line **create a random number between 0 and 1**
  - If the random number **is less than a threshold**
    - Write the line to a **temp file**
- Read the sample data from the temp file

Sampling Very Large Datasets

Potential problems:

- We may get **more rows** than we want
  - **Trim** the excess records
- We may get **less rows** than we want
  - **Be content** with what you get :)
  - or, try to **get more rows** than you need
    - you will later trim as above

Variable Selection

- Remove irrelevant variables
  - If a variable has nothing to say we don't need it
- Remove highly correlated variables
  - If two variables say the same thing, we need only one of them
- Reduce columns when `ncol() > nrow()`

Variable Selection

- Mainly two groups of approaches
  - **filter methods**
    - select features individually based on a metric
  - **wrapper methods**
    - consider the effects of choosing a variable to the efficiency of a model
    - *iteratively* construct a feature set

Variable Selection

- Alternative groups of approaches
  - **supervised**
    - select variables based on their *relations* with the “target” variable
  - **unsupervised**
    - select variables individually based on their **value distribution**
    - **same** value on all observations -> **remove**
    - **unique** values on all observations -> **remove**

Supervised Methods

- Correlation - simple
- Information theoretic metrics
  - Entropy

$$H(Y) = -\sum_{c_i \in \mathcal{Y}} P(Y = c_i) \times \log P(Y = c_i)$$

  - Conditioned class entropy

$$H(Y|X) = -\sum_{v_i \in \mathcal{X}, c_i \in \mathcal{Y}} P(X = v_i, Y = c_i) \log \frac{P(X=v_i, Y=c_i)}{p(X=v_i)}$$

  - Information Gain

$$IG(X) = H(Y) - H(Y|X)$$

  - Gain Ratio

$$GR(X) = \frac{IG(X)}{H(X)}$$

Supervised Methods

- These methods are implemented in `FSelector` and `CORElearn`
  - `attrEval` is from `CORElearn`

```
data(iris)
attrEval(Species ~ ., iris, estimator = "GainRatio")
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.5919339    0.3512938    1.0000000    1.0000000
```

```
attrEval(Species ~ ., iris, estimator = "InfGain")
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.5572327    0.2831260    0.9182958    0.9182958
```

```
attrEval(Species ~ ., iris, estimator = "Gini")
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.2277603    0.1269234    0.3333333    0.3333333
```

```
attrEval(Species ~ ., iris, estimator = "Relief")
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.1974074    0.1874074    0.7267797    0.7088889
```

Supervised Methods

- All supervised methods for classification tasks

```
infoCore(what = "attrEval")
```

```
##  [1] "ReliefFequalK"      "ReliefFexpRank"     "ReliefFbestK"
##  [4] "Relief"             "InfGain"            "GainRatio"
##  [7] "MDL"                "Gini"               "MyopicReliefF"
## [10] "Accuracy"           "ReliefFmerit"       "ReliefFdistance"
## [13] "ReliefFsqrDistance" "DKM"                "ReliefFexpC"
## [16] "ReliefFavgC"        "ReliefFpe"          "ReliefFpa"
## [19] "ReliefFsmp"         "GainRatioCost"      "DKMcost"
## [22] "ReliefKukar"        "MDLsmp"             "ImpurityEuclid"
## [25] "ImpurityHellinger"  "UniformDKM"         "UniformGini"
## [28] "UniformInf"         "UniformAccuracy"    "EqualDKM"
## [31] "EqualGini"          "EqualInf"           "EqualHellinger"
## [34] "DistHellinger"      "DistAUC"            "DistAngle"
## [37] "DistEuclid"
```

- All supervised methods for regression tasks

```
infoCore(what = "attrEvalReg")
```

```
## [1] "RReliefFequalK"      "RReliefFexpRank"     "RReliefFbestK"
## [4] "RReliefFwithMSE"     "MSEofMean"           "MSEofModel"
## [7] "MAEofModel"          "RReliefFdistance"    "RReliefFsqrDistance"
```

Supervised Methods

- Note that some methods can do feature selection as part of the modeling process
  - Tree based methods

PCA

- Instead of filtering variables
  - Create new ones with equivalent power
  - …but much less in count

```
data(iris)
pca <- princomp(iris[,1:4])
loadings(pca)
```

```
##
## Loadings:
##              Comp.1 Comp.2 Comp.3 Comp.4
## Sepal.Length  0.361  0.657  0.582  0.315
## Sepal.Width          0.730 -0.598 -0.320
## Petal.Length  0.857 -0.173        -0.480
## Petal.Width   0.358        -0.546  0.754
##
##                Comp.1 Comp.2 Comp.3 Comp.4
## SS loadings      1.00   1.00   1.00   1.00
## Proportion Var   0.25   0.25   0.25   0.25
## Cumulative Var   0.25   0.50   0.75   1.00
```

PCA

```
new.iris <- data.frame(pca$scores[, 1:2], Species = iris$Species)
head(new.iris)   %>% kable()
```

Comp.1	Comp.2	Species
-2.684126	0.3193972	setosa
-2.714142	-0.1770012	setosa
-2.888991	-0.1449494	setosa
-2.745343	-0.3182990	setosa
-2.728716	0.3267545	setosa
-2.280860	0.7413304	setosa

- There are other methods as well
  - Independent Component Analysis
  - Singular Value Decomposition
- They all have some drawbacks
  - Numeric input is one major problem
  - Less readable by humans