Machine Learning In Finance

Introduction

AI/ML in Finance

Benefits of using AI/ML in finance

- Financial operations (e.g., trades) are based on pre-defined rules. Automation via AI/ML reduces costs and increases speed.
- Financial decisions (e.g., granting a loan, rating a bond, making investments, etc.) require quick, fact-based judgment calls. AI/ML can help automate decision-making.
- AI/ML algorithms make fact-based, objective decisions and will always comply with laws and regulation (if programmed to do so).
- AI/ML do not require economic theory, they use data to detect patterns.

A simple ML example

- **Task**: Predict the salaries of people from their age.[a]
- **Size of the data sample**: $n = 30$. Divide the data sample into a
  - Training set $n = 10$
  - Validation set $n = 10$
  - Test set $n = 10$
- **Models**: Y = salary, X = age, Parameters = a, b_i
  - Linear ($Y = a + b_1 X$)
  - Polynomial of order 2 ($Y = a + b_1 X + b_2 X^2$)
  - Polynomial of order 5 ($Y = a + b_1 X + b_2 X^2 + b_3 X^3 + b_4 X^4 + b_5 X^5$)

[a] The example and data are taken from John Hull's website.

A simple ML example

Source: Hull (2020).

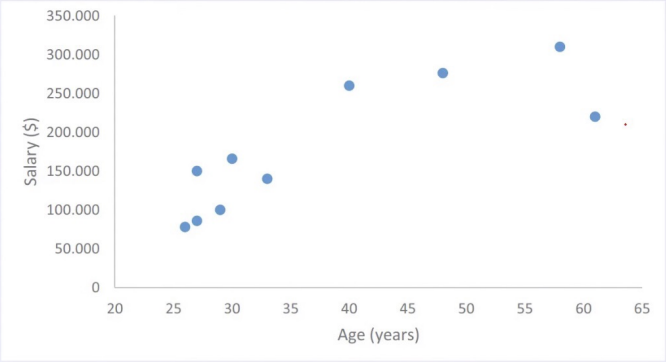
A simple ML example

Source: Hull (2020).

A simple ML example

- The polynomial model of order 5 yields the best fit. However, its ups/downs seem unrealistic thus indicating a potential overfitting.
- The linear model, in contrast, appears to be oversimplistic. It could suffer from underfitting.
- Solution: backtest the model using the second, so-called validation set (i.e., check whether the model generalizes well to a validation data set).

A simple ML example: Validation set



Source: Hull (2020).

A simple ML example

Root mean square error (rmse) of the three different model candidates:

	Linear	Polynomial (order 2)	Polynomial (order 5)	
Training set	49,731	32,932	12,902	rmse
Validation set	49,990	33,554	38,794	rmse
Difference	259	622	25,892	← Difference in rmse of Training set & Validation set.
Overfitting?	No	No	Yes	

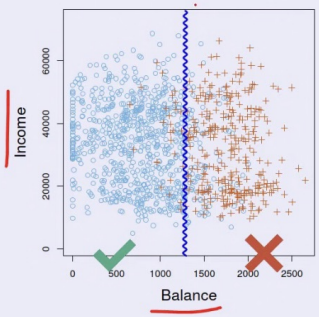
$\Rightarrow$ Polynomial model of order 2 produces the best fit w/o overfitting the data.

A simple ML example

- How accurate is the chosen model?
- $rmse = 32,932$ (training set) or $rmse = 33,554$ (validation set)?
- Answer: none of the two. Accuracy should not be measured based on data sets that were used to choose/validate a model. i.e. test set.
- $rmse = 34,273$ (test set!)

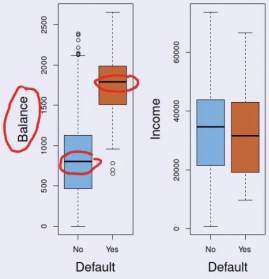
A second ML example : Classification

- **Task**: Predict the probability of credit card defaults based on annual income and monthly credit card balance.



Source: James, Witten, Hastie, and Tibshirani (2013). Defaults in orange, non-defaults in blue.

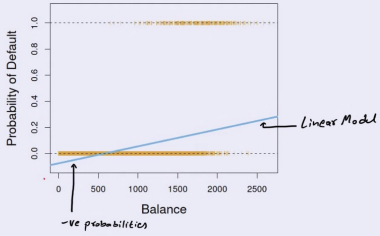
A second ML example



Source: James, Witten, Hastie, and Tibshirani (2013).

A second ML example

- One possible solution: fit a linear model $Default = a + b \times Income$ to the data. Problem: probabilities can be negative!

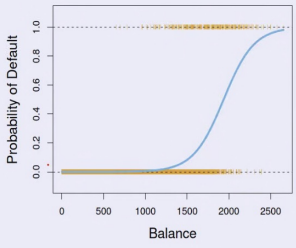


Source: James, Witten, Hastie, and Tibshirani (2013).

A second ML example

- Better solution: fit a logistic model

$$\log\left(\frac{Pr(Default=1|Income)}{1-Pr(Default=1|Income)}\right) = a + b \cdot Income$$ to the data.



Source: James, Witten, Hastie, and Tibshirani (2013).

Data Sources

Data sources/types I

- Financial data are diverse and come at different levels of complexity.
- Data can be structured, unstructured, come at low/high frequency, can be publicly available or private, etc.
- Financial data can be complemented with "alternative data", i.e., data from non-financial sources that nevertheless contain information on financial operations (e.g., satellite images).
- More data sources lead to more data; more data in turn leads to the need for AI/ML methods to process this data.
- Different data sources and types make data preprocessing necessary.

Data sources/types II

Types of financial data

Fundamental data	Market data	Analytics	Alternative data
Balance sheet items	Price/yield/implied volatility	Analyst recommendations	Images
Income statement items	Volume	Credit ratings	Google searches
Macro variables	Dividend/coupon	Earnings expectations	Twitter/chats
...	Open interest	News sentiment	Metadata
	Quotes/cancellations	...	...
	...		

Data sources/types III

In addition to paid sources like Bloomberg / Compustat / EIKON / Datastream / etc. there are also (lower quality) free of charge sources like yahoo finance. Furthermore, there are many more open data sets available on the internet that can be downloaded for free. These include

- open data set finders like **kaggle** or the **UCI Machine Learning Repository** that contain a wide variety of different data sets,
- public government data sets from **data.gov** (data from multiple US government agencies) and the **EU Open Data Portal** (data published by EU institutions),
- economics data sets from **World Bank Open Data** (economic and development indicators),
- We will learn how to import different kinds of data like csv- and pdf-files, image data, etc. in practical applications throughout the lecture.

Data Preprocessing

Data preprocessing I

- Financial (and other) data are often
  - incomplete: e.g., missing values or attributes of interest, containing only aggregated data
  - noisy: e.g., containing errors and outliers
  - inconsistent: e.g., containing discrepancies in codes or names.
- Data preprocessing is about resolving these issues and transforming raw data into an understandable format.
- Data preprocessing is key to a good model performance and most often consumes more time than actually specifying and fitting a model.
- This typically involves two steps. First understanding the data, then preparing the data before actually employing some or more ML models.

Data preprocessing II

Data understanding

- The main tasks according to Lesmeister (2015) are:
  - Collecting the data
  - Describing the data
  - Exploring the data
  - Verifying the data quality
- These tasks are performed to make sure that the data are adequate to meet the analytical goal.
- Furthermore, by exploring the data, determining its sparseness, and identifying missing values one gets a better idea of which learning methods might be appropriate.
- Verifying the data quality is critical. Understanding who collects the data and how it is collected can help to identify incomplete or erroneous data. It is also possible that rules for collecting the data change over time which might lead to structural breaks in the data.

Data preprocessing III

Data preparation

- The main tasks according to Lesmeister (2015) are:
  - Selecting the data
  - Cleaning the data
  - Constructing the data
  - Integrating the data
  - Formatting the data
- The goal of these tasks is to get the data ready to use as input in the algorithms. This includes merging data from different sources, feature engineering, and further transformations.
- Some algorithms require categorical variables like Yes/No to be in the form 1/0 or to be formatted as factors (in *R*).
- The data might be split into training, test and validation set.

Data preprocessing IV

- Data understanding and data preparation are often not performed separately but are interrelated tasks.
- As an example we will have a closer look on the German credit data set by Dua and Graff (2017) on the following slides.
- The data are readily available via the UCI Machine Learning Repository along with a **description of the data**.

Data preprocessing V

The statistical programming language *R*

- All operations (exploring and preprocessing the data, fitting and validating the model, etc.) will be performed in the statistical programming language *R*.
- *R* is available as free software under the terms of the Free Software Foundation's GNU General Public License in source code form. We recommend the usage of RStudio Desktop as IDE which can be downloaded freely (see **here**).
- Alternatively you can use one of the *RStudio* servers provided by the University Computing Center (see **here**).

Data preprocessing VI

The statistical programming language R (cont.)

- *R* is a language and environment for statistical computing and graphics and one of the major languages for data science.
- The language provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, etc.) and graphical tools.
- *R* is highly extensible via packages. There is a wide variety of *R* packages available for machine learning and many modern methods in statistical learning are implemented in *R*.
- Of course there are other alternatives, most notably the programming language *Python* with libraries like scikit-learn and keras/tensorflow.

Example: German Credit Data

Practical example I

Importing the data and first explorative analysis

```
1 #Importing the data
2 germanCredit = read.table("http://archive.ics.uci.edu/ml/
      machine?learning?databases/statlog/german/german.data")
3
4 dim(germanCredit)
5 [1] 1000   21    #1000 observations of 21 variables
6
7 #reduce number of features to make them fit on one slide,
8 #21st feature is the response/target variable (credit rating
      (good/bad))
9 germanCredit<-germanCredit[,c(1:5, 21)]
```

dim( ) -> Gives dimension of data structure.

rows Columns

<- All rows & 6 Columns 1 to 5 & 21.

Practical example II

Importing the data and first explorative analysis (cont.)

```
#Exploring the structure of the data
str(germanCredit)

'data.frame':   1000 obs. of  6 variables:
$ V1 : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4
    4 2 4 2 ...
$ V2 : int  6 48 12 42 24 36 24 36 12 30 ...
$ V3 : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3
    3 3 3 5 ...
$ V4 : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1
    8 4 2 5 1 ...
$ V5 : int  1169 5951 2096 7882 4870 9055 2835 6948 3059
    5234 ...
$ V21: int  1 2 1 1 2 1 1 1 1 2 ...
```

Practical example III

Importing the data and first explorative analysis (cont.)

The str() command gave us valuable information about the structure of the data set:

- Our data set consists of 1000 observations of 6 variables.
- The features are of type factor or of type integer.
- Factors represent categorical or ordinal variables and have the advantage that category labels are stored only once (requires less RAM and enables faster computations). The different values a factor variable can assumed are referred to as levels. For example, the first variable ("V1") has the levels "A11", "A12", "A13", and "A14" which are represented by the integers 1, 2, 3, 4.
- We can see the first observations of all variables.

Practical example IV

Importing the data and first explorative analysis (cont.)

The variable (or feature or attribute) names are not meaningful (see the next slide). Even if they were, one should always be skeptical about the provided labels.

Practical example V

A description of the variables and the values they can assume are provided **here**, e.g.:

Attribute 1 ("V1")

```
      Status of existing checking account
A11 : ... < 0 DM  <- less than 0 Deutsche mark.
A12 : 0 <= ... < 200 DM
A13 : ... >= 200 DM / salary assignments for at least 1 year
A14 : no checking account
```

Attribute 2 ("V2")

```
      Duration in month
```

Practical example VI

And most importantly:

Attribute 21 ("V21")

```
     Rating
1: Good
2: Bad
```

Practical example VII

Based on the documentation we can assign meaningful variable names. Furthermore, we transform the data.frame into a tibble which provides an enhanced print() method and has further advantages over standard data.frames (see **here**).

Adjust variable names

```
#use the documentation to assign column names
colnames(germanCredit)<-c("chkAcctStat", "duration", "
    credHist", "purpose", "amount", "rating")

#View some records
#Therefore, convert data.frame to tibble for more convenient
     printing functionality
library(tibble)
germanCredit<-as_tibble(germanCredit)
```

Practical example VIII

Show first records

```
print(germanCredit)

#A tibble: 1,000 x 6
chkAcctStat duration credHist purpose amount rating
<fct>          <int> <fct>    <fct>    <int>  <int>
1 A11            6    A34      A43     1169      1
2 A12           48    A32      A43     5951      2
3 A14           12    A34      A46     2096      1
4 A11           42    A32      A42     7882      1
5 A11           24    A33      A40     4870      2
6 A14           36    A32      A46     9055      1
7 A14           24    A32      A42     2835      1
8 A12           36    A32      A41     6948      1
9 A14           12    A32      A43     3059      1
10 A12           30    A34      A40     5234      2
#... with 990 more rows
```

Practical example IX

As a first analysis, we have a look at some summary statistics via the summary() command. The output of the function depends on the variable type.

Summary statistics

```
summary(germanCredit)
```

```
chkAcctStat    duration      credHist     purpose        amount          rating
A11:274     Min.   : 4.0   A30: 40   A43    :280   Min.   :  250   Min.   :1.0
A12:269     1st Qu.:12.0   A31: 49   A40    :234   1st Qu.: 1366   1st Qu.:1.0
A13: 63     Median :18.0   A32:530   A42    :181   Median : 2320   Median :1.0
A14:394     Mean   :20.9   A33: 88   A41    :103   Mean   : 3271   Mean   :1.3
            3rd Qu.:24.0   A34:293   A49    : 97   3rd Qu.: 3972   3rd Qu.:2.0
            Max.   :72.0             A46    : 50   Max.   :18424   Max.   :2.0
                                     (Other): 55
```

Practical example X

We now transform the integer variables into numeric (float) ones to facilitate later processing of the data. For this and further operations we rely on the dplyr package. According to the official **website**, *"dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:*

- *mutate() adds new variables that are functions of existing variables.*
- *select() picks variables based on their names.*
- *filter() picks cases based on their values.*
- *summarise() reduces multiple values down to a single summary.*
- *arrange() changes the ordering of the rows."*

We will use these operations in the practical applications throughout the lecture.

dplyr is part of **tidyverse**, a collection of *R* packages specifically designed for data science where all packages share an underlying design philosophy, grammar, and data structure (tibble).

Practical example XI

Adjust variable types

```
#transform integer variables to numeric ones
#therefore, use dplyr-style notation with the pipe operator
    %>%
germanCredit<- germanCredit %>%
                mutate_if(is.integer, as.numeric)

#transform rating into a factor
germanCredit<- germanCredit %>%
                mutate(rating=factor(rating, labels =c("Good
                    ", "Bad")))

#Show percentage of good/bad rated credit applications
table(germanCredit$rating)/nrow(germanCredit)
Good  Bad
0.7  0.3
```

Practical example XII

- On the following slides we will shortly analyze some of the variables.
- In particular, we want to gain an understanding of the predictive power of the covariates/on the dependent variable credit rating.
- This can be done via the ratio weight of evidence (WOE).

Weight of evidence

WOE encodes the relationship of a (categorical) predictor variable with a binary target variable. It originated in the finance industry to help separate "good" from "bad" risks (loan default). By now, it has been in use for more than forty years and is still most widely known in the finance and insurance industry.

Practical example XIII

Weight of evidence (cont.)

- The ratio is defined as:

$$WOE := \log\left(\frac{numberOfNonEvents(Good)}{numberOfEvents(Bad)}\right),$$

where event means a default.

- Ratios of non events to events close to 1 indicate that the corresponding category has no predictive power on the target variable. This corresponds to a value near zero after applying the logarithm.
- One should be careful in deriving conclusions from the ratio as giving a loan to a defaulting customer is worse than not giving a loan to a non-defaulting (potential) customer.

Practical example XIV

Weight of evidence (cont.)

- WOE is calculated in different groups that are formed based on the covariate of interest.
- For categorical variables these might be the categories or a pooling of multiple (smaller) subcategories.
- For continuous variables one has to create bins based on thresholds.
- Each category/bin should contain at least 5 % of all observations to avoid results to be driven by noise.

Practical example XV

Checking account status : Calculating WOE

```
#Attribute 1: (qualitative)
#Status of existing checking account
#A11 : ... < 0 DM
#A12 : 0 <= ... < 200 DM
#A13 : ... >= 200 DM / salary assignments for at least 1
    year
#A14 : no checking account

#Calculate WOE and some further simple ratios
tmpStats<-germanCredit %>%
            select(chkAcctStat, rating) %>%
            group_by(chkAcctStat) %>%
            summarize (pctOfTotalObs = length(rating)/nrow(
                germanCredit), goodRate=mean(rating=="Good")
                , WOE=log(sum(rating=="Good")/sum(rating=="
                Bad")))
```

Practical example XVI

Checking account status (cont.)

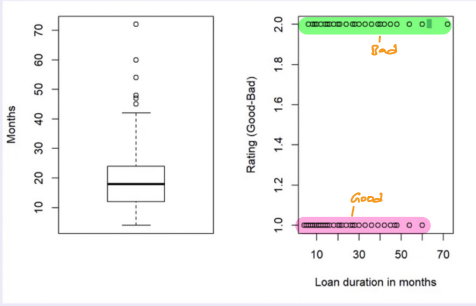
```
print(tmpStats)

#A tibble: 4 x 4
chkAcctStat pctOfTotalObs goodRate    WOE
<fct>              <dbl>    <dbl>  <dbl>
1 A11                0.274    0.507 0.0292
2 A12                0.269    0.610 0.446
3 A13                0.063    0.778 1.25
4 A14                0.394    0.883 2.02
```

Practical example XVII

Credit rating ~ checking account status

Practical example XVIII

Loan duration



Practical example XIX

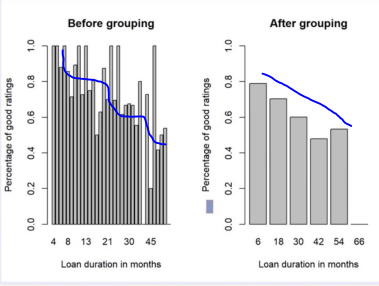
When calculating the WOE separately for each unique value of loan duration, some groups only contain very few observations. We therefore group the loan durations on a yearly basis and recalculate WOE.

WOE for loan duration

```
#Calculation for each unique value of loan duration
tmpStats<-germanCredit %>%
            select(duration, rating) %>%
            group_by(duration) %>%
            summarize(goodRate=mean(rating=="Good"))

#Form larger groups on a yearly basis
tmpStats2<-germanCredit %>%
            select(duration, rating) %>%
            mutate (interval=cut(duration, breaks=12*(0:6)))
                %>%
            group_by(interval) %>%
            summarize (goodRate=mean(rating=="Good"))
```

Practical example XX

Loan duration ~ rating



Trend line

Practical example XXI

We finally have a look at the credit history variable. For brevity, we do not consider further variables and leave this as an exercise.

Credit history

```
#A30 : no credits taken/all credits paid back duly
#A31 : all credits at this bank paid back duly
#A32 : existing credits paid back duly till now
#A33 : delay in paying off in the past
#A34 : critical account/other credits existing (not at this
    bank)

#Calculate some statistics for the five groups
tmpStats<-germanCredit %>%
            select(credHist, rating) %>%
            group_by(credHist) %>%
            summarize (pctOfTotalObs = length(rating)/nrow(
                germanCredit), goodRate=mean(rating=="Good")
                , WOE=log(sum(rating=="Good")/sum(rating=="
                Bad")))
```

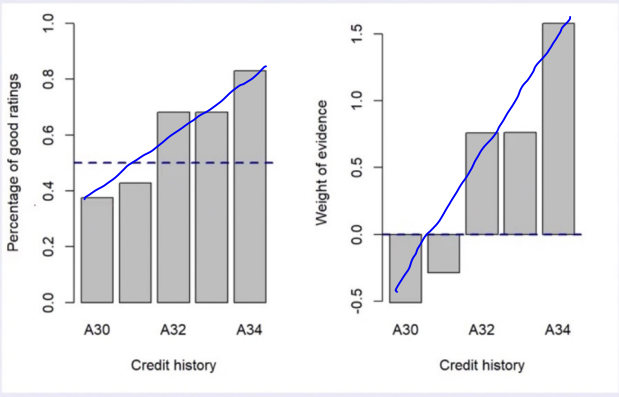
Practical example XXII

Credit history (cont.)

```
print(tmpStats)

#A tibble: 5 x 4
credHist pctOfTotalObs goodRate    WOE
<fct>            <dbl>    <dbl>  <dbl>
1 A30              0.04     0.375 -0.511
2 A31              0.049    0.429 -0.288
3 A32              0.53     0.681  0.759
4 A33              0.088    0.682  0.762
5 A34              0.293    0.829  1.58
```

Practical example XXIII

Rating ~ credit history



A34 is highly predictive & has high explanatory power for explaining bad credit rating.

Data Generation

Data generation I

- Many machine learning techniques rely on synthetic data for training and testing.
- Synthetic data is artificial data created by algorithms that mirror statistical properties of the original data while not revealing information regarding real people.
- The main purpose of using synthetic data is preserving privacy, creating training data for machine learning algorithms and testing systems.
- Many training data sets are highly imbalanced (e.g., credit card fraud data) making classification tasks difficult. In theses cases, synthetic data generation is particularly important for building accurate machine learning models.

Data generation II

Classification of synthetic data according to Aggarwal and Yu (2008)

- Fully synthetic data: The data is completely synthetic and does not contain any original data entries. Therefore, the (joint) density of the original data is estimated and random values are sampled from the estimated density function. The data has strong privacy protection but the truthfulness of the data is lost.
- Partially synthetic data: The method replaces values of selected attributes that have a high risk of disclosure with synthetic data. This technique can also be useful for imputing missing values in the data. Disclosure risk is higher than in fully synthetic data.
- Hybrid synthetic data: The data set is generated using both original and synthetic data. Each record in the original data set is replaced by the nearest record in the synthetic data. The method combines good privacy protection with high utility at the cost of more memory and processing time.

Fitting statistical distribution to original data & then sampling synthetic data from that fitted statistical distribution.

Data generation III

- There are different approaches for generating synthetic data.
- Three broad concepts are
  - (i) Generating data from a known distribution
  - (ii) Fitting a distribution to real data and generating from this distribution
  - (iii) Using deep learning

Data generation IV

(i) Generating data from a known distribution

- Applicable in cases where no real data exist but the researcher/practitioner has a comprehensive understanding of the distribution of the data of interest (e.g., normal, Student-t, exponential, chi-squared, etc.)
- One can then simulate random data from this a-priori chosen distribution.
- Utility of the data depends on the degree of knowledge about the underlying distribution of the data

Data generation V

(ii) Fitting a distribution to real data

- If there is real data, one can determine the best fit distribution chosen from a given family of distributions.
- Synthetic data can now be generated from the best fit distribution via Monte Carlo methods.
- The quality of the generated data depends on the selection of the family of distributions (e.g., only normal distributions are considered) as well as on the goodness of fit of the final distribution.
- Data can also be modeled by machine learning models such as decision trees providing an approximation to non-classical distributions (e.g., multi-modal distributions). In theses cases, overfitting might be an issue.

Data generation VI

(iii) Using deep learning

- Deep generative models such as Variational Autoencoder (VAE) or Generative Adversarial Network (GAN) can create synthetic data.
- VAE is an unsupervised method where the original data is compressed by an encoder into a more compact structure. The decoder then generates a representation of the original data from the compressed data. The system is trained to minimize the differences between output and original data.

Data generation VII

Using deep learning (cont.)

- In the GAN model two separate networks are trained iteratively. The first network (the generator) takes random sample data to create a synthetic data set. The second network (the discriminator) compares the synthetic data with a real data set.
- The generator network is trained to make discriminating between the generated and the real data for the discriminator network as hard as possible.
- The method is frequently used for generating image data, see **here** for a demonstration.

Simple Linear Regression

Introduction I

- We start our discussion of AI/ML methods in finance with simple models for statistical learning.
- Stat. learning refers to the set of tools used for *understanding* data.
- Broadly speaking, we can differentiate between the following types of models:
  - Supervised algorithms
  - Unsupervised algorithms

Introduction II

Supervised vs. unsupervised learning

- Supervised statistical learning: Estimate/Predict an output based on inputs.
  - Example 1: Predict Y (wage) based on X_1 (education), X_2 (gender),... (called a regression problem)
  - Example 2: Predict ups/downs of the S&P 500 ($Y = 0, 1$) based on X_1, X_2,... (called a classification problem)
- Unsupervised statistical learning: Group input observations.
  - Example 1: Group customers based on inputs X_1 (age), X_2 (income),... (called a clustering problem)

Simple linear regression I

Starting point: linear regression as the foundation for many "modern" supervised statistical learning approaches.

Simple linear regression

- Predict a quantitative response Y on the basis of a single predictor variable X:

$$Y \approx \beta_0 + \beta_1 X \tag{1}$$

- Y is "regressed" on X.
- Two unknown parameters/coefficients β_0 (intercept) and β_1 (slope) are estimated by minimizing the residual sum of squares (RSS).
- Result:

$$\hat{y} \approx \hat{\beta}_0 + \hat{\beta}_1 x \tag{2}$$

Simple linear regression II

How to estimate?

Simple linear regression

- Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ be the prediction for Y based on the ith observation of X. Then

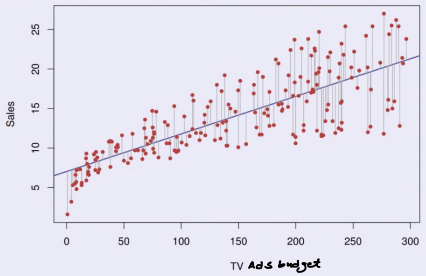
$$e_i = y_i - \hat{y}_i \tag{3}$$

  is called the ith residual (the difference between the ith observed and predicted values of the response variable).
- The residual sum of squares (RSS) is then defined as

$$RSS = \sum_{i=1}^{n} e_i^2 \tag{4}$$

Simple linear regression III

Residuals



Source: James, Witten, Hastie, and Tibshirani (2013). Data: https://www.statlearning.com/s/Advertising.csv

Simple linear regression IV

How to estimate?

Simple linear regression

Minimization of the RSS leads to the well-known OLS estimator:

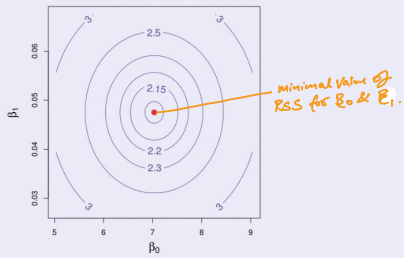
$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad (5)$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}. \quad (6)$$

with $\bar{x}$ and $\bar{y}$ being the sample means of X and Y.

Simple linear regression V

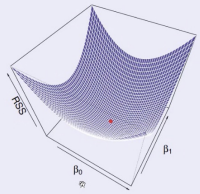
Contour plot of the RSS (response: sales; predictor: TV)



Source: James, Witten, Hastie, and Tibshirani (2013). Data: https://www.statlearning.com/s/Advertising.csv

Simple linear regression VI

Three-dimensional plot of the RSS (response: sales; predictor: TV)

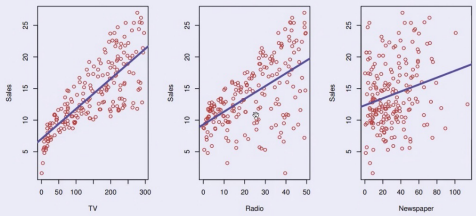


Source: James, Witten, Hastie, and Tibshirani (2013). Data: https://www.statlearning.com/s/Advertising.csv

Simple linear regression VII

Resulting OLS regression lines

- Y...sales; X_1, X_2, X_3...advertising budgets.



Source: James, Witten, Hastie, and Tibshirani (2013). Data: https://www.statlearning.com/s/Advertising.csv

TV & Sales looks to have linear regression
Not so much for Newspaper & sales.

Simple linear regression VIII

How to assess the **accuracy** of the **coefficient estimates** assuming the true relation is $Y = \beta_0 + \beta_1 X + \varepsilon$?

S.e. of the OLS coefficients

The OLS coefficients are estimated from a sample (x_i, y_i). Thus, the accuracy of the estimates can be assessed by looking at the **standard errors of the estimators**.

$$\mathrm{SE}\left(\hat{\beta}_0\right)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2}\right] \tag{7}$$

and

$$\mathrm{SE}\left(\hat{\beta}_1\right)^2 = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \tag{8}$$

with $\sigma^2 = \mathrm{Var}(\varepsilon)$.

Simple linear regression IX

How to estimate σ^2?

S.e. of the OLS coefficients

Estimate σ via the **residual standard error**:

$$\mathrm{RSE} = \sqrt{RSS/(n-2)} \tag{9}$$

Simple linear regression X

What to do with $\mathrm{SE}\left(\hat{\beta}_0\right)$ (or more precisely $\widehat{\mathrm{SE}}\left(\hat{\beta}_0\right)$)?

Confidence intervals for the OLS coefficients

The 95%-c.i. for β_0 and β_1 are approximately given by

$$\hat{\beta}_0 \pm 2 \cdot \mathrm{SE}\left(\hat{\beta}_0\right) \text{ and } \hat{\beta}_1 \pm 2 \cdot \mathrm{SE}\left(\hat{\beta}_1\right) \tag{10}$$

Simple linear regression XI

What to do with $\mathrm{SE}\left(\hat{\beta}_0\right)$ (or more precisely $\widehat{\mathrm{SE}}\left(\hat{\beta}_0\right)$)?

Hypothesis testing for the OLS coefficients

We test the hypothesis

$$H_0 : \beta_1 = 0 \text{ versus } H_1 : \beta_1 \neq 0$$

using the t-statistic

$$t = \frac{\hat{\beta}_1 - 0}{\mathrm{SE}\left(\hat{\beta}_1\right)} \tag{11}$$

which measures the no. of standard deviations that $\hat{\beta}_1$ is away from 0.

Simple linear regression XII

How to assess the **accuracy** of the **model itself** assuming that we rejected $H_0 : \beta_1 = 0$?

Residual Standard Error

Recall that associated with each individual observation (x_i, y_i) is an error term ε / e_i. Even if the coefficients were known, these error terms prevent us from perfectly predicting Y from X. Estimate the s.e. of ε via the RSE

$$\mathrm{RSE} = \sqrt{\frac{1}{n-2}\mathrm{RSS}} = \sqrt{\frac{1}{n-2}\sum_{i=1}^n (y_i - \hat{y}_i)^2} \tag{12}$$

Simple linear regression XIII

How to assess the **accuracy** of the **model itself** assuming that we rejected $H_0 : \beta_1 = 0$?
RSE provides an absolute measure of lack of fit of the model (it is given in terms of units of Y).
Problem: RSE depends on the scale of Y!

R^2

Solution: R^2, the proportion of variance explained by the model.

$$R^2 = \frac{\mathrm{TSS} - \mathrm{RSS}}{\mathrm{RSS}} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} \tag{13}$$

with $\mathrm{TSS} = \sum (y_i - \bar{y})^2$.

Note: $R^2 = r^2$ with $r = Cor(X, Y)$.

Multiple Linear Regression

Multiple linear regression I

Simple linear regression is a useful approach for predicting a response on the basis of a single predictor variable. In practice: more than one predictor (in fact, in ML often thousands!). Possible approaches:

- Estimate several simple regressions. Problem: correlations among predictors will bias the results (over-/underestimate the effect of single predictors).
- Estimate a multivariate/multiple linear regression.

Multiple linear regression II

Multiple linear regression

- Predict a quantitative response Y on the basis of several predictor variable $X_1, X_2, ...$:

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p \tag{14}$$

- Y is "regressed" on the set $X_1, \ldots, X_p$.
- $p + 1$ unknown parameters/coefficients β_0 (intercept) and β_i (slopes) are estimated by again minimizing the residual sum of squares (RSS):

Use OLS to estimate coefficients β_i.

$$\hat{\beta} = (X'X)^{-1} X'Y$$

Vector of coefficients. When X' is X transpose.

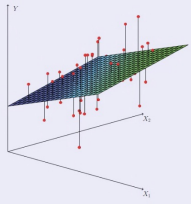
- Result:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p. \tag{15}$$

Multiple linear regression III

Resulting OLS regression plane

- Regression plane with two predictors.



Source: James, Witten, Hastie, and Tibshirani (2013).

- If we have more predictors/features, we get linear hyperplane.

Multiple linear regression IV

Multiple linear regression in *R*

Multiple regression coefficient estimates from a regression of product sales on advertising budget, community income level, average age, and education.

```
1 > library(MASS)  -> library for Regression.
2 > library(ISLR)  -> Carseats data set.
3 > lm.fit=lm(Sales~Advertising+Income+Age+Education,data=
      Carseats)  -> fitting linear model.
```

Multiple linear regression V

Multiple linear regression in *R* (cont.)

```
1 > summary(lm.fit)
2
3 Call:
4 lm(formula = Sales ~ Advertising + Income + Age + Education,
5     data = Carseats)
6
7 Residuals:
8     Min      1Q  Median      3Q     Max
9 -8.1676 -1.8869 -0.1805  1.6827  8.5494
```

Multiple linear regression VI

Multiple linear regression in *R* (cont.)

```
1  Coefficients:                    Estimate of coefficients β_i
2                Estimate Std. Error t value Pr(>|t|)   Which coefficients are significantly different from zero?
3  (Intercept)  8.487078   0.914911   9.276  < 2e-16 ***  -> 0.01% level
4  Advertising  0.110168   0.019798   5.565 4.85e-08 ***
5  Income       0.013485   0.004710   2.863  0.00442 **
6  Age         -0.040058   0.008109  -4.940 1.16e-06 ***
7  Education   -0.036805   0.050237  -0.733  0.46422     <- Not significantly different from zero, so Education can be dropped.
8  ---
9  Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
10
11 Residual standard error: 2.624 on 395 degrees of freedom
12 Multiple R-squared:  0.1453,    Adjusted R-squared:  0.1366
13 F-statistic: 16.78 on 4 and 395 DF,  p-value: 1.022e-12
```

Multiple linear regression VII

Where to go next?

- Is at least one of the predictors $X_1, X_2, \ldots, X_p$ useful in predicting the response? ($\rightarrow$ F-statistic).
- Do we need all or only a subset of the predictors? ($\rightarrow$ variable selection).
- How well does the model fit the data? ($\rightarrow$ RSE and R^2).
- Given a set of predictor values, what response value should we predict, and how accurate is our prediction? ($\rightarrow$ forecasting).

Multiple linear regression VIII

Is at least one predictor useful?

F-statistic

We test the null hypothesis,

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$$

versus the alternative

$$H_a : \text{at least one } \beta_j \text{ is non-zero.}$$

This hypothesis is tested using the following **F-statistic**:

$$F = \frac{(TSS - RSS)/p}{RSS/(n-p-1)}. \tag{16}$$

No relationship between the response and predictors: F-statistic close to 1. H_a is true: F greater than 1.

Multiple linear regression IX

Which predictors are significant?

t-statistics / p-values

- Estimate F-statistics for models from which individual predictors are omitted.
- Calculate t-statistics for each predictor (the square of each t-statistic is the corresponding F-statistic).

Caution!

If $p > n$ or even $p >> n$ then there are more coefficients β_j to estimate than observations from which to estimate them.

1. We cannot fit the regression using (multiple) OLS.
2. We cannot use the F-statistic.

→ When no. of variables is larger than no. of observations. eg: 100 variables & 20 observations so more coefficients β_i to estimate than observations from which to estimate them.

Multiple linear regression X (Variable Selection)

Which predictors are powerful?

Variable selection

Process of determining which predictors are associated with the response, and fitting a single model that involves only the significant predictors. Alternative approaches:

- Compare the fit of *all* models and calculate measures of the model's fit (e.g., Akaike's information criterion/AIC, Bayesian information criterion/BIC, adjusted R^2, etc.). Becomes infeasable as the number of different models for p variables is 2^p. (e.g., for $p = 30$ we have $1,073,741,824$ different models.
- Use an automated and efficient procedure to select a subset of models.

→ Estimate all possible model (infeasable as variables increase)

Multiple linear regression XI

Three classical approaches for Variable selection:

(i) Forward selection

- Start with the *null model* (contains an intercept only).
- Estimate p simple linear regressions and add to the null model the variable from the regression with the lowest RSS.
- Continue adding variables in this manner until some stopping rule is fulfilled.

Multiple linear regression XII

Three classical approaches for Variable selection:

(ii) Backward selection

- Start with the model that contains all p variables.
- Remove the variable with the largest p-value.
- Continue removing variables in this manner until some stopping rule is fulfilled.

Multiple linear regression XIII

Three classical approaches for Variable selection:

(iii) Mixed selection

- Start with the *null model* (contains an intercept only).
- Estimate p simple linear regressions and add to the null model the variable from the regression with the lowest RSS.
- Remove variables that turn insignificant when new variables are added to the model. (because of correlations between predictors)
- Continue adding and removing variables in this manner until some stopping rule is fulfilled.

Multiple linear regression XIV

Variable selection in *R*

```
> summary(Credit)  -> credit card data.

       ID             Income          Limit
 Min.   :  1.0   Min.   : 10.35   Min.   :  855
 1st Qu.:100.8   1st Qu.: 21.01   1st Qu.: 3088
 Median :200.5   Median : 33.12   Median : 4622
 Mean   :200.5   Mean   : 45.22   Mean   : 4736
 3rd Qu.:300.2   3rd Qu.: 57.47   3rd Qu.: 5873
 Max.   :400.0   Max.   :186.63   Max.   :13913
     Rating          Cards            Age
 Min.   : 93.0   Min.   :1.000   Min.   :23.00
 1st Qu.:247.2   1st Qu.:2.000   1st Qu.:41.75
 Median :344.0   Median :3.000   Median :56.00
 Mean   :354.9   Mean   :2.958   Mean   :55.67
 3rd Qu.:437.2   3rd Qu.:4.000   3rd Qu.:70.00
 Max.   :982.0   Max.   :9.000   Max.   :98.00
```

Multiple linear regression XV

Variable selection in *R* (cont.)

```
> summary(Credit)

     Education        Gender    Student   Married
 Min.   : 5.00    Male :193   No :360   No :155
 1st Qu.:11.00   Female:207   Yes: 40   Yes:245
 Median :14.00
 Mean   :13.45
 3rd Qu.:16.00
 Max.   :20.00
            Ethnicity      Balance  <- response variable
 African American: 99   Min.   :   0.00
 Asian           :102   1st Qu.:  68.75
 Caucasian       :199   Median : 459.50
                        Mean   : 520.01
                        3rd Qu.: 863.00
                        Max.   :1999.00
```

Multiple linear regression XVI

Variable selection in *R* (cont.)

Simple Regression (Choosing only Student Variable)

```
> summary(lm(Balance ~ Student, data=Credit))

Call:
lm(formula = Balance ~ Student, data = Credit)

Residuals:
     Min       1Q   Median       3Q      Max
-876.82 -458.82  -40.87  341.88 1518.63

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    480.37      23.43   20.50  < 2e-16 ***
StudentYes     396.46      74.10    5.35 1.49e-07 ***
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
      1
```

(high T-value; highly Significant)

Multiple linear regression XVII

Variable selection in *R* (cont.)

```
> summary(lm(Balance ~ Student, data=Credit))

[...]

Residual standard error: 444.6 on 398 degrees of freedom
Multiple R-squared:  0.06709,   Adjusted R-squared:  0.06475
F-statistic: 28.62 on 1 and 398 DF,  p-value: 1.488e-07
```

Close to 7% of total variance is explained by our Model.

Multiple linear regression XVIII

Variable selection in *R* (cont.)

Simple Regression (Choosing Credit Card Limit Variable)

```
> summary(lm(Balance ~ Limit, data=Credit))

Call:
lm(formula = Balance ~ Limit, data = Credit)

Residuals:
     Min       1Q   Median       3Q      Max
-676.95 -141.87  -11.55  134.11  776.44

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.928e+02  2.668e+01  -10.97   <2e-16 ***
Limit        1.716e-01  5.066e-03   33.88   <2e-16 ***
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
      1
```

(high T-value; highly Significant)

Multiple linear regression XIX

Variable selection in *R* (cont.)

```
> summary(lm(Balance ~ Limit, data=Credit))

Call:
lm(formula = Balance ~ Limit, data = Credit)

[...]

Residual standard error: 233.6 on 398 degrees of freedom
Multiple R-squared:  0.7425,    Adjusted R-squared:  0.7419
F-statistic:  1148 on 1 and 398 DF,  p-value: < 2.2e-16
```

Limit Variable seems to have much more explanatory power when it comes to Balance.

Multiple linear regression XX

Variable selection in *R* (cont.) (Choosing Income Variable)

Simple Regression

```
> summary(lm(Balance ~ Income, data=Credit))

Call:
lm(formula = Balance ~ Income, data = Credit)

Residuals:
    Min      1Q  Median      3Q     Max
-803.64 -348.99  -54.42  331.75 1100.25

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 246.5148    33.1993   7.425  6.9e-13 ***
Income        6.0484     0.5794  10.440  < 2e-16 ***
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
      1
```

high T-value → high significant

Multiple linear regression XXI

Variable selection in *R* (cont.)

```
> summary(lm(Balance ~ Income, data=Credit))

Call:
lm(formula = Balance ~ Income, data = Credit)

[...]

Residual standard error: 407.9 on 398 degrees of freedom
Multiple R-squared:  0.215, Adjusted R-squared:  0.213
F-statistic:   109 on 1 and 398 DF,  p-value: < 2.2e-16
```

Then, Select Variable Income for Regression.

Multiple linear regression XXII

Variable selection in *R* (cont.) Akaike Information Criterion (Lower AIC ⇒ Better Model Fit)

```
> AIC(lm(Balance ~ Student, data=Credit))
[1] 6016.933   (3rd Best)
> AIC(lm(Balance ~ Limit, data=Credit))
[1] 5501.983   (Best)
> AIC(lm(Balance ~ Income, data=Credit))
[1] 5947.894   (2nd Best)
```

Multiple linear regression XXIII

Variable selection in *R* (cont.) (Multiple Regression Using 3 Variables Income, Student & Limit)

```
> summary(lm(Balance ~ Income+Student+Limit, data=Credit))

Call:
lm(formula = Balance ~ Income + Student + Limit, data =
    Credit)
Residuals:
     Min       1Q   Median       3Q      Max
-186.60  -78.43  -17.33   62.65  314.88
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.323e+02  1.243e+01  -34.78   <2e-16 ***
Income      -7.902e+00  2.432e-01  -32.49   <2e-16 ***
StudentYes   4.270e+02  1.742e+01   24.52   <2e-16 ***
Limit        2.675e-01  3.712e-03   72.07   <2e-16 ***
```

All Variables highly significant.

Multiple linear regression XXIV

Variable selection in *R* (cont.)

```
Residual standard error: 104.4 on 396 degrees of freedom
Multiple R-squared:  0.9488,    Adjusted R-squared:  0.9484
F-statistic:  2447 on 3 and 396 DF,  p-value: < 2.2e-16

> AIC(lm(Balance ~ Income+Student+Limit, data=Credit))
[1] 4859.793
```

AIC is also much lower than before.

Almost 95%

Multiple linear regression XXV

How to assess the model fit?

Adjusted R^2 (Use this for Multiple Regression)

Adjust

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

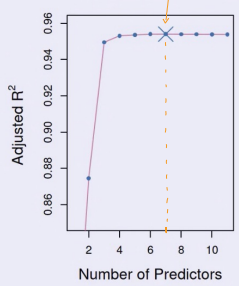
by the number of variables d used in the model:

$$\text{Adjusted } R^2 = 1 - \frac{\mathrm{RSS}/(n-d-1)}{\mathrm{TSS}/(n-1)} \tag{17}$$

Intuition: adding noise variables to a well fitted model will decrease RSS only by a little bit. Therefore, the adjusted R^2 punishes the model for every included unnecessary predictor, to avoid overfitting.

Multiple linear regression XXVI

Adjusted R^2 for the credit data



Source: James, Witten, Hastie, and Tibshirani (2013).

Multiple linear regression XXVII

How to predict from the model?

Prediction/Forecasting

Use (15) and predict values $\hat{y}$ from given values x_i and estimated coefficients $\hat{\beta}_i$.
Bias will stem from

- Estimation error / reducible error: imprecise estimates of $\hat{\beta}_i$.
- Model bias: using a linear regression function instead of $y = f(x)$.
- Irreducible error: the error term.

Multiple linear regression XXVIII

How to extend the mulitple linear regression model?

Qualitative predictors

Some variables are qualitative rather than quantitative/metric. For example: *age* and *gender* in the credit card data. How to deal with qualitative predictors?

- Predictor with only two levels: use a **dummy variable**:

$$x_i = \begin{cases} 1, & \text{if } i\text{th person is female} \\ 0, & \text{if } i\text{th person is male} \end{cases}$$

- Predictor with k levels: use $k-1$ dummy variables.
- Caution: the interpretation of the estimated coefficients changes!

Multiple linear regression XXIX

How to extend the mulitple linear regression model?

Interaction terms

Predictor X_1 increases/decreases the effect of a different predictor X_2 on Y.

- Use a so-called **interaction term**:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \varepsilon.$$

- Again caution: interpretation of β_1, β_2, and β_3 are different!
- If you include the interaction term X_1X_2, one must include X_1 and X_2 as well.
- Particularly interesting: interaction between a qualitative and a quantitative predictor.

Multiple linear regression XXX

How to extend the mulitple linear regression model?

Polynomial regression

Predictor X_1 influences Y **non-linearly**.

- One possible solution: Use **polynomial regression**:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \varepsilon.$$

- Including non-linear terms does not change the fact that this is still a linear model!
- Using higher polynomials (just as adding predictors) can lead to overfitting.

Multiple linear regression XXXI

Polynomial regression in *R* (last slide XXX example)

```
> library(ISLR)
> summary(lm(formula = mpg ~ horsepower + I(horsepower^2),
    data = Auto))

Call:
lm(formula = mpg ~ horsepower + I(horsepower^2), data = Auto
    )

Residuals:
     Min       1Q   Median       3Q      Max
-14.7135  -2.5943  -0.0859   2.2868  15.8961
```

need to use this in I

Multiple linear regression XXXII

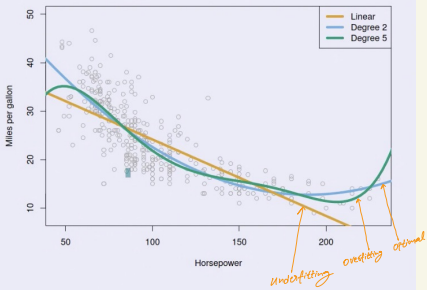
Polynomial regression in *R* (cont.)

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)    56.9000997  1.8004268   31.60   <2e-16 ***
horsepower     -0.4661896  0.0311246  -14.98   <2e-16 ***
I(horsepower^2)  0.0012305  0.0001221   10.08   <2e-16 ***

Residual standard error: 4.374 on 389 degrees of freedom
Multiple R-squared:  0.6876,    Adjusted R-squared:  0.686
F-statistic:   428 on 2 and 389 DF,  p-value: < 2.2e-16
```

highly significant

Multiple linear regression XXXIII

Polynomial regression of mpg on horsepower



Source: James, Witten, Hastie, and Tibshirani (2013).

Multiple linear regression XXXIV

What else can happen? in Multiple linear Regression.

Potential problems

(a) Non-linearity of the response-predictor relationships. ⇒ Residual plots

(b) Correlation of error terms. ⇒ Causal inference / good experimental design.

(c) Non-constant variance of error terms. ⇒ e.g., Weighted Least Squares

(d) Outliers (extremal y_i). ⇒ detection, winsorization or exclusion.

(e) High-leverage points (extremal x_i). ⇒ detection, winsorization or exclusion.

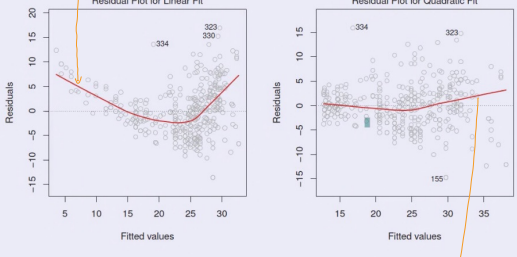
(f) Collinearity. ⇒ Compute VIF, drop/orthogonalize collinear predictors.

i.e. $X_1 \approx X_2$ Since high correlation meaning one of the variable is obsolete.

Multiple linear regression XXXV

Residual plot of polynomial regressions *(a)

There is a strong indication of non-linear relation



Source: James, Witten, Hastie, and Tibshirani (2013).

residuals
fitted values
No clear trend in residuals

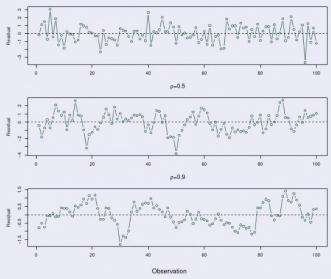
Quadratic Model is better than linear one as residual are almost constant

From these 2 plots we imply
→ Non-linear relationship
→ Quadratic Model better.

Multiple linear regression XXXVI

Residual plots, simulated data with different $\rho(\varepsilon_i, \varepsilon_{i-1})$ *(b)

No clear trend →



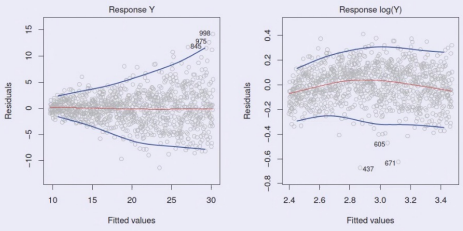
Trend here so → we have a pattern

Source: James, Witten, Hastie, and Tibshirani (2013).

Simulated data with different correlation between adjacent points & with adjacent residuals.

Multiple linear regression XXXVII

Residual plots with Y and $\log(Y)$ as responses *(c)



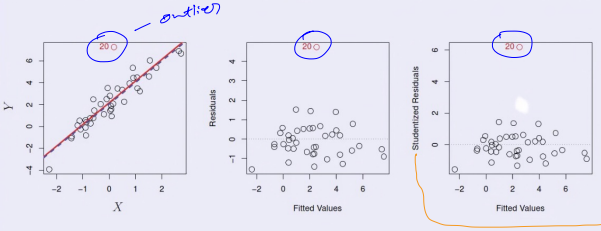
Source: James, Witten, Hastie, and Tibshirani (2013).

For the left plot even though the mean stays constant 0, the variance is increasing. By using log(Y), the right plot shows variance is constant, so sometimes using log solves this problem.

Multiple linear regression XXXVIII

Outliers are extreme values of Y.

Outliers *(d)



Source: James, Witten, Hastie, and Tibshirani (2013).

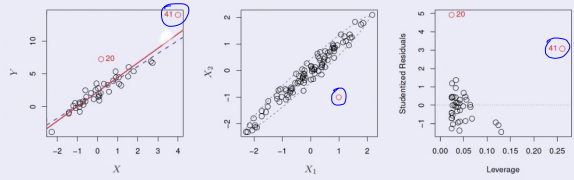
Including outliers doesn't change our Linear Model, However, it will lead to increase in RSE & Adjusted R^2 will decrease. So, the Model fit will get worse.

Means we divide by std. deviation.

Multiple linear regression XXXIX

High leverage points are extreme values for X.

High leverage points *(e)

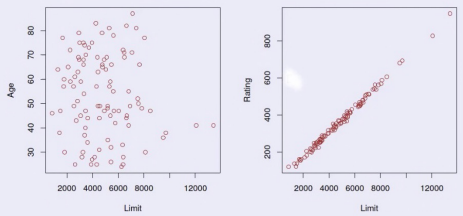


Source: James, Witten, Hastie, and Tibshirani (2013).

High leverage points changes Regression line. One single point can effect the estimation.

Multiple linear regression XXXX

Collinearity *(f)

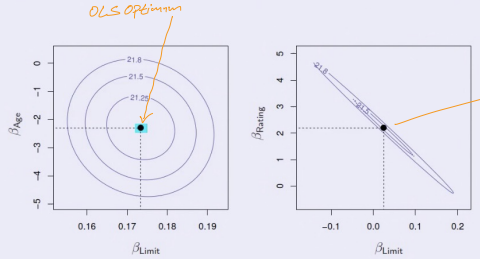


Source: James, Witten, Hastie, and Tibshirani (2013).

left plot has no correlation.
Right plot is strongly positively correlated & including limit & Rating in Regression model will lead to problems. Coefficient estimates are strongly effected due to Numerical instability.

Multiple linear regression XXXXI

Collinearity



Source: James, Witten, Hastie, and Tibshirani (2013).

Contour plot here is nasty & is much difficult to find the OLS estimate.

Penalised Regression

Penalized regression I

- Recall the problem of model/variable selection.
- One approach: subset variable selection (forward/backward stepwise selection).
- Alternative approach: **Shrinkage methods/penalized regression**

Penalized regression

In penalized regression models, *all* p predictors are used in one model and the coefficient estimates are constrained/regularized (in other words, the different algorithms *shrink* the coefficient estimates towards zero).

→ All variables are included.
Most β_i are close to zero but not equal to zero.

Penalized regression II

Ridge Regression

Instead of minimizing

$$RSS = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 \quad \text{(OLS)}$$

Response variable; Predictors; Our Estimate: $\hat{\beta}(OLS) = \hat{\beta}_j$

In OLS we minimize RSS.

one minimizes

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = RSS + \lambda\sum_{j=1}^{p}\beta_j^2 \quad \text{(Ridge Regression)} \tag{18}$$

with $\lambda \geq 0$ being a tuning parameter.

In Ridge Regression we still minimize RSS but we include the additional component $\lambda\sum_{j=1}^{p}\beta_j^2$. Note how j starts at 1 so we do not include the intercept term β_0.
As we minimise this, we are penalising coefficients (β_j) that are not equal to 0.
So, at some point coefficients tends to 0 & is governed by tuning parameter λ. For $\lambda = 0$, we get OLS.

Ridge Regression I

- As with OLS, Ridge Regression seeks to find a model that fits the data well (i.e., minimizes the RSS).
- The so-called shrinkage penalty $\lambda\sum_{j=1}^{p}\beta_j^2$ is small when the parameters β_j ($j = 1, \ldots, p$) are close to zero.
- The parameter λ controls this process of *shrinking* the parameters.

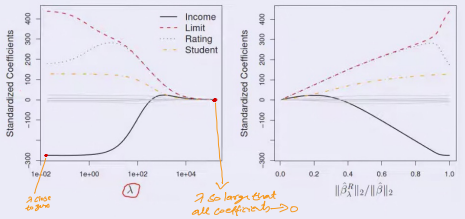
Ridge Regression - coefficients

For each value of the tuning parameter λ, Ridge regression will produce a set of coefficients $\hat{\beta}_\lambda^R$.
$\lambda = 0$: $\hat{\beta}_\lambda^R = \hat{\beta}^{OLS}$.
$\lambda \to \infty$: $\hat{\beta}_\lambda^R = 0$.

$\hat{\beta}_\lambda^R$ = Ridge Regression coefficient
In Ridge Regression, we repeat it for different values of λ. We get different coefficients for different values of λ.

Ridge Regression II

Ridge Regression - Credit Data



Source: James, Witten, Hastie, and Tibshirani (2013).

Ridge Regression III

Ridge Regression - Standardized predictors

- OLS: coefficient estimates are are scale equivariant.
- Ridge regression: coefficients can change when predictors are multiplied by a constant.

⇒ Use standardized predictors!

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2}}$$

- Standardized predictors have a s.d. of one.

Ridge Regression IV

Ridge Regression - Advantages over OLS

- Ridge regression's advantage over OLS: *bias-variance trade-off*.
- Recall the *(training) mean squared error*

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2$$

based on training data observations $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

- We are interested in fitting a model that yields a low test MSE for a previously unseen test observation (x_0, y_0).

Bias vs. Variance I

Bias-Variance trade-off

One can show that

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\varepsilon), \text{ i.e.}$$

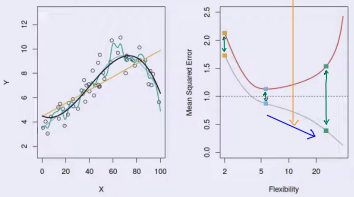
$$\text{expected test MSE} = \text{Variance of } \hat{f}(x_0) + \text{squared bias of } \hat{f}(x_0) + \text{variance of the error term.}$$

To minimize the test MSE, we need to achieve low variance AND low bias.

- **Variance** refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set.
- **Bias** refers to the error that is introduced by using a model to approximate for the unknown function f (e.g., using a *linear* model).

Bias vs. Variance II

Bias-Variance trade-off

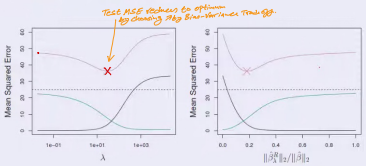


Simulated data (circles); true f: black curve; linear model in orange; smoothing splines in blue and green. Training MSE: grey curve; Test MSE: red curve. Source: James, Witten, Hastie, and Tibshirani (2013).

Ridge Regression V

Ridge Regression - Advantages over OLS

- Ridge regression: as λ increases, the flexibility of the ridge regression fit decreases and the bias increases.



Squared bias (black), variance (green), and test MSE (purple) (simulated data). Source: James, Witten, Hastie, and Tibshirani (2013).

- Thus, **we can use Ridge regression to minimize the test mean squared error.**

Ridge Regression VI

Ridge Regression - Advantages over OLS

- If $p > n$, then the OLS estimates do not have a unique solution.
- Ridge regression also has **computational advantages** over OLS combined with best subset selection (i.e., estimating 2^p models).
- **However: ridge regression will include all p predictors in the final model!** Almost always the coefficients will all be not equal to zero. Hence, we move on to LASSO.

LASSO I

- Ridge regression will always generate a model that includes relevant AND irrelevant predictors. This can make the interpretation of the model difficult as irrelevant predictors will never be excluded. Remedy: the LASSO.

LASSO (least absolute shrinkage and selection operator)

The **LASSO coefficients** $\hat{\beta}_\lambda^L$ minimize the following quantity:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\boxed{\lambda\sum_{j=1}^{p}|\beta_j|}=RSS+\lambda\sum_{j=1}^{p}|\beta_j| \quad \text{(LASSO)} \quad (19)$$

with $\lambda \geq 0$ being a tuning parameter.

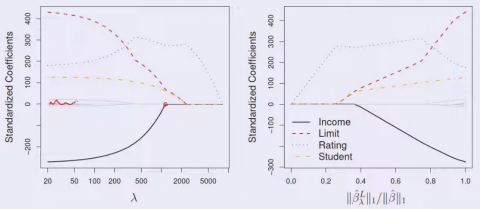
LASSO II

- In short: LASSO uses the ℓ_1 norm whereas Ridge regression uses the ℓ_2 norm to penalize too high coefficients β_j.
- Advantage of LASSO: using the ℓ_1 norm forces some coefficients to be exactly equal to zero when λ increases.
- Hence, LASSO performs *variable selection* and fitted models are sparse and thus much easier to interpret.

↓ Do not include all predictors in the model.

LASSO III

LASSO coefficients - credit data



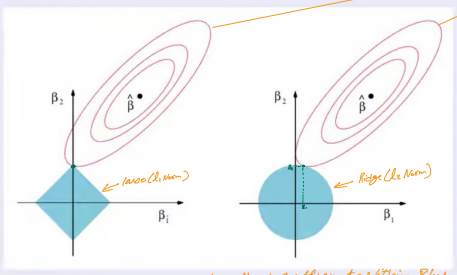
Source: James, Witten, Hastie, and Tibshirani (2013).

In Ridge Regression as λ increases the coefficients slowly tends to 0 but here it goes to 0 very fast.

LASSO IV

Why does LASSO force some coefficients to be zero?

Error/MSE and constraint functions



Only allow coefficients within Blue Region for both lasso & Ridge (constraints)

Source: James, Witten, Hastie, and Tibshirani (2013).

Contour plots of OLS/RSS.

- $\hat{\beta}$ → OLS coefficients (minimal pt. if we don't have any constraints)
- The intersection of contours with constraint f'n is the optimal point. So for Ridge we will always get two β_1 & β_2 which are non-zero. In lasso we can set $\beta_1 = 0$ & β_2 non-zero. This is why lasso forces some of the coefficients to be zero because of shape of blue region.

LASSO V

Which performs best, Ridge or LASSO?

When to use LASSO

- Neither ridge regression nor the LASSO will universally dominate the other.
- LASSO: a (rel.) small number of predictors have non-zero coefficients.
- Ridge: response is a function of many predictors, all with coefficients of roughly equal size.

→ Use LASSO when we expect small no. of predictors to have non-zero coefficients, for eg with 1000 predictors & you expect only 5 or 10 predictors to have non-zero coefficients.

When to use?

Elastic Net & Cross-Validation

Elastic Net

Can one combine LASSO and Ridge Regression? Yes:

Elastic net regularization

The estimates for the **Elastic net regularization** $\hat{\beta}_\lambda^{EN}$ are given by

$$\hat{\beta}_\lambda^{EN} = \arg\min_\beta RSS + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \text{ (Elastic net)} \qquad (20)$$

with $\lambda_1, \lambda_2 \geq 0$ being tuning parameters.

← Combining Lasso & Ridge gives Elastic Net.

How to choose the tuning parameters? for Lasso, Ridge & Elastic Net.

How should we choose the tuning parameters?

- Recall the distinction between the test error rate and the training error rate.
- If a sufficiently large test data set is available: training and test error rate are easy to calculate.
- Often not the case (i.e., no or only few test observations are available)!
- Remedy:

Validation Set Approach

Randomly divide the available set of observations into two parts: a training set and a validation/hold-out set. The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set.

Validation Set Approach

Validation Set Approach

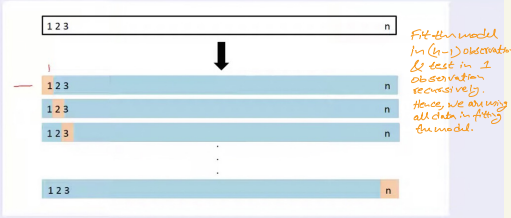
Training set (shown in blue), validation set (shown in beige). Source: James, Witten, Hastie, and Tibshirani (2013).

Drawbacks:

- The validation estimate of the test error rate can be highly variable (which observations are randomly chosen into the training/validation sets?).
- Only a subset of the available data are used for fitting the model. (We are leaving out most data & not using it for training) (Better to do Cross-validation)

Cross-Validation I To Calculate Test Error Rate

Leave-One-Out Cross-Validation

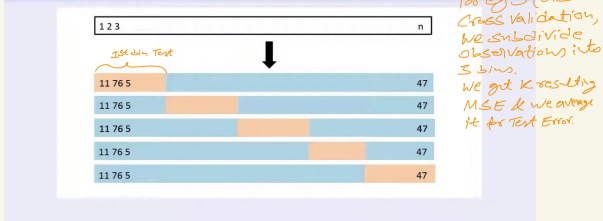


Training set (shown in blue), validation set (shown in beige). The test error is estimated by averaging the n resulting MSEs.

Source: James, Witten, Hastie, and Tibshirani (2013).

Cross-Validation II

k-fold Cross-Validation



Training set (shown in blue), validation set (shown in beige). The test error is estimated by averaging the k resulting MSEs.

Source: James, Witten, Hastie, and Tibshirani (2013).

Application

- On the following slides we will apply penalized linear regression models to the task of forecasting equity returns (one day ahead Apple returns).
- Therefore, we rely on the *RiskAnalytics* R package to download some of the data.
- Information on the package and the data can be found **here**.
- Additionally, we rely on a list of tickers of Nasdaq listed companies which can be downloaded as a csv file from **here**.

Application Of PLS

Import and preprocess data I

Import data

```
#RiskAnalytics cannot be installed directly from CRAN for
    newer versions of R. Therefore, use the following two
    lines of code:
    #library(devtools)
    #install_github("lborke/RiskAnalytics")

library(RiskAnalytics)
library(snow)
library(quantmod)

#file path to the downloaded csv file with nasdaq tickers:
pathNasdaq<-"data/nasdaqTicker.csv"
companyList<-read.csv(pathNasdaq)
#order firms in descending order by market capitalization
companyList<-companyList[order(companyList$Market.Cap,
    decreasing=TRUE),]
```

Import and preprocess data II

Import data (cont.)

```
#download return data for the 10 firms with the highest
    market capitalization within companyList
stockData <- as.data.frame(get.yahoo.data(companyList, max_
    comp_num = 10, from_date = "2010-01-01", to_date="
    2020-12-31")$returns_final)

macroData <- as.data.frame(get.macro.data(from_date = "
    2010-01-01", to_date = "2020-12-31", plot_macros=FALSE))

#manually add the corresponding date to each row
#for more information on this proceeding see the source code
    at https://github.com/lborke/RiskAnalytics/tree/master/
    R
dates<-rownames(as.matrix(getSymbols(Symbols = "^VIX", src =
    "yahoo", env = NULL, from = "2010-01-01", to = "
    2020-12-31", warnings = FALSE)[, 6]))
dates<-dates[-c(1, length(dates))]
rownames(macroData)<-dates #assign dates
```

Import and preprocess data III

Variables contained in macroData

```
#^VIX: implied volatility index
#1GSPC: S&P500 returns
#IYR: iShares Dow Jones US Real Estate returns
#3MTCM: Changes in the three-month Treasury bill rate
#Yield: Changes in the slope of the yield curve
    corresponding to the yield spread between the ten-year
    Treasury rate and the three month bill rate
#Credit: Changes in the credit spread between BAA-rated
    bonds and the Treasury rate
#see https://www.wiwi.hu-berlin.de/de/forschung/irtg/results
    /discussion-papers/discussion-papers-2017-1/sfb649dp2017
    -003-1.pdf for more details

colnames(macroData)<-c("VIX", "SP500", "realEstate", "TR3M",
    "yield", "credit") #assign new variable names
```

Import and preprocess data IV

Preprocess data

```
macroData$date<-as.Date(rownames(macroData)) #add date
    column
stockData$date<-as.Date(rownames(stockData))

#join return and macro data at the respective date via an
    inner join
completeData<-merge(stockData, macroData, by="date")
```

Import and preprocess data V

Preprocess data (cont.)

```
library(dplyr)
completeData<-as_tibble(completeData)

print(completeData,n=5)
# A tibble: 2,766 x 17
date          AAPL      MSFT      AMZN      GOOG      GOOGL
<date>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 2010-01-05  0.00173  3.23e-4  0.00588 -0.00441 -0.00441
2 2010-01-06 -0.0160  -6.16e-3 -0.0183  -0.0255  -0.0255
3 2010-01-07 -0.00185 -1.05e-2 -0.0172  -0.0236  -0.0236
4 2010-01-08  0.00663  6.87e-3  0.0267   0.0132   0.0132
5 2010-01-11 -0.00886 -1.28e-2 -0.0243  -0.00151 -0.00151
# ... with 2,761 more rows, and 11 more variables:
#   TSM <dbl>, V <dbl>, JPM <dbl>, JNJ <dbl>, WMT <dbl>,
#   VIX <dbl>, SP500 <dbl>, realEstate <dbl>, TR3M <dbl>,
#   yield <dbl>, credit <dbl>
```

Import and preprocess data VI

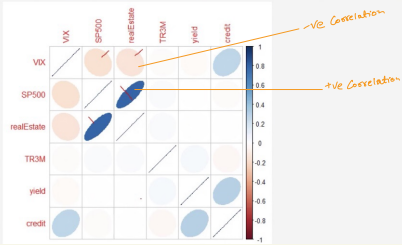
Preprocess data (cont.)

```
#Macro variables are extracted from completeData because the
     dates in the original macroData object do not coincide
    with the dates in stockData
macroData<-completeData %>% select("VIX":"credit")  #same
    dates as stockData

print(round(cor(macroData),3)) #correlation matrix
             VIX  SP500 realEstate   TR3M  yield credit
VIX        1.000 -0.163     -0.148 -0.013 -0.026  0.276
SP500     -0.163  1.000      0.790  0.026  0.006 -0.015
realEstate -0.148  0.790      1.000  0.028  0.006 -0.001
TR3M       -0.013  0.026      0.028  1.000  0.049 -0.030
yield      -0.026  0.006      0.006  0.049  1.000  0.308
credit      0.276 -0.015     -0.001 -0.030  0.308  1.000
```

Import and preprocess data VII

Visualization of pairwise correlations

```
library(corrplot)
corrplot(cor(macroData), method="ellipse")
```



Import and preprocess data VIII

Create training and test set

```
library(lubridate)
#Choose observations before 2019 as training data and
    observations afterwards as test data
trainingData<-completeData %>%
                filter(year(date)<=2018) %>%
                select(AAPL:GOOGL, JPM, JNJ, VIX:credit)
#add one day ahead apple returns to be forecasted later on
trainingData$forecast<-c(trainingData$AAPL[-1],0)
trainingData<-trainingData[-nrow(trainingData),]

testData<-completeData %>%
            filter(year(date)>2018) %>%
            select(AAPL:GOOGL, JPM, JNJ, VIX:credit)
#add one day ahead apple returns in the test set, too
testData$forecast=c(testData$AAPL[-1],0)
testData<-testData[-nrow(testData),]
```

Import and preprocess data IX

Create training and test set

```
#Some of the models need dependent and explaining variables
    separately in matrix form

xTrain<-as.matrix(select(trainingData, -forecast)) #remove
    forecast from trainingDAta
yTrain<-trainingData$forecast

xTest<-as.matrix(select(testData, -forecast))
yTest<-testData$forecast
```

Import and preprocess data X

Auxiliary functions for calculation of R^2 and rMSE

- We will evaluate the predictive performance of the various models based on the out-of-sample R^2 as well as the rMSE.
- Therefore, we introduce the following auxiliary functions:

```
rSquared<-function(pred, actual){
    rss<-sum((pred-actual)^2)
    tss<-sum((actual-mean(actual))^2)
    rsq<-1-rss/tss
    return(rsq)
}

rmse<-function(pred, actual){
    return(sqrt(mean((pred-actual)^2)))
}
```

Simple linear model I (OLS)

Simple linear model

```
# forecast future Apple returns with past returns of the
    stock itself and various other variables
linModel<-lm(forecast~ ., data=trainingData)
                                  .dot means use all the variables for the model.
summary(linModel)
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.018068   0.013017  -1.388   0.1653
AAPL         0.036055   0.026788   1.346   0.1785
MSFT        -0.052560   0.033369  -1.575   0.1154
AMZN        -0.020489   0.022059  -0.929   0.3531
GOOG        -0.100961   0.228608  -0.442   0.6588
GOOGL        0.118844   0.228915   0.519   0.6037
JPM         -0.031266   0.034880  -0.896   0.3702
JNJ         -0.085395   0.048611  -1.757   0.0791 .
VIX          0.006921   0.005037   1.374   0.1696
SP500        0.020498   0.024484   0.837   0.4026
...
```

JNJ stock return has slightly more statistical significance in explaining future Apple stock return

Simple linear model II

Simple linear model (cont.)

```
...
realEstate   0.012901   0.013165   0.980   0.3272
TR3M        -0.005056   0.008193  -0.617   0.5372
yield        0.002831   0.002025   1.398   0.1623
credit      -0.002966   0.002524  -1.175   0.2400
---
Signif. codes:  0 '***' '0.001' '**' 0.01 '*' 0.05 '.' 0.1

Residual standard error: 0.01619 on 2248 degrees of freedom
Multiple R-squared:  0.007153,  Adjusted R-squared:
    0.001412
F-statistic: 1.246 on 13 and 2248 DF,  p-value: 0.2397
```

Both Multiple & Adjusted R^2 is very small. Linear Model is NOT so Good.

Simple linear model III

Ouf-of-sample performance

```
pred<-predict(linModel, newdata=testData)

rSquared(pred=pred, actual=yTest)
    -0.01376846

rmse(pred=pred, actual=yTest)
    0.02405451
```

Model doesn't perform better in Test data & R^2 has become -Ve.

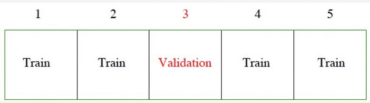
Choosing a value for the tuning parameters I

- The penalized linear regression models rely on one (ridge, LASSO) or two (elastic net) tuning parameters that control the amount of regularization.
- We want to choose the tuning parameter such that it generalizes well to new data.
- This will be done via cross-validation.

Choosing a value for the tuning parameters II

Cross-validation

- To perform k-fold cross-validation, we split the training data into k parts or "folds".
- One usually chooses $k = 5$ or $k = 10$.
- k-fold cross-validation then considers training on all but the kth part and then validating the model performance on that kth part, iterating over k.
- Cross-validation cannot only be used for selecting hyperparameters, but also for assessing the performance of a model.



Ridge regression I

Determining the regularization parameter

```
1  library(glmnet)
2  set.seed(2021) # for reproducibility
3
4  #Find optimal lambda parameter via cross-validation:
5  cvRidge<-cv.glmnet(x=xTrain, yTrain, nfolds=10, nlambda=100,
       alpha=0, lambda.min.ratio=0.0001)
6  #nfolds: number of folds in cross validation
7  #nlambda: number of regularization parameters to be tested
8  #alpha: determines the weighting to be used, 0 in case of
       ridge regression, 1 gives LASSO.
9  #lambda.min.ratio: ratio of smallest lambda to largest
       lambda considered
10 #for further details see https://glmnet.stanford.edu/
       articles/glmnet.html for further details
11
12 optimalLambda<-cvRidge$lambda.min
13 optimalLambda
14     0.455634    <- optimal λ
```

Ridge regression II

Fitting the model

```
1  #fit ridge regression model with the lambda determined on
       the previous slide
2  ridgeReg<-glmnet(x=xTrain, yTrain, alpha=0, lambda=
       optimalLambda)
3
4  coef(ridgeReg) #coefficients of the penalized linear model
5  (Intercept)  6.308607e-04
6  AAPL         9.831866e-04
7  MSFT        -8.763122e-04
8  AMZN        -3.372965e-04
9  GOOG         2.089550e-04
10 GOOGL        2.573408e-04
11 ...
12 TR3M        -1.985538e-04
13 yield        9.563131e-05
14 credit      -7.623069e-06
```

Ridge regression III

Effect of high/low lambda

Asymptotically we have:

- $\lambda \to 0 : \beta_{\text{ridge}} \to \beta_{\text{OLS}}$
- $\lambda \to \infty : \beta_{\text{ridge}} \to 0$

```
1  highLambda<-10
2  lowLambda<-0.00001
3  coef(glmnet(x=xTrain, yTrain, alpha=0, lambda=highLambda))
4  (Intercept)  7.729305e-04
5  AAPL         4.554773e-05
6  MSFT        -4.106883e-05
7  AMZN        -1.584365e-05
8  GOOG         9.446636e-06
9  GOOGL        1.164191e-05
10 ...
11 TR3M        -9.432911e-06
12 yield        4.532762e-06
13 credit      -2.308404e-07
```

Using high λ=10

Ridge regression IV

Using low λ we get same as OLS coefficients

Effect of high/low lambda (cont.)

```
1  coef(glmnet(x=xTrain, yTrain, alpha=0, lambda=lowLambda))
2
3  (Intercept) -0.018050381
4  AAPL         0.036021284
5  MSFT        -0.052962852
6  AMZN        -0.020486622
7  GOOG        -0.057921967
8  GOOGL        0.076092514
9  ...
10 TR3M        -0.005064605
11 yield        0.002834675
12 credit      -0.002962296
```

Ridge regression V

In and ouf-of-sample performance

```
1  #in-sample R^2
2  predTrain<-predict(ridgeReg, newx=xTrain)
3  rSquared(pred=predTrain, actual=yTrain)
4      0.0003141296   -> Much lower than OLS model, so better
5  #reminder: in-sample R^2 simple linear model = 0.007153
6
7  #out-of-sample performance
8  predTest<-predict(ridgeReg, newx=xTest)
9  rSquared(pred=predTest, actual=yTest)
10     -0.005109216   -> slightly better than OLS.
11 #reminder: out-of-sample R^2 simple linear model = -0.0138
12 rmse(pred=predTest, actual=yTest)
13     0.02395156
14 #reminder: out-of-sample rMSE simple linear model = 0.0241
```

Ridge regression VI

Since the parameter λ governs the training process it is called hyperparameter instead of parameter.

Summary

- We determined the hyperparameter λ via 10-fold cross-validation.
- Using this λ we obtained a linear model with coefficients that are smaller (in terms of their absolute value) than the corresponding coefficients in the simple OLS model. The degree of penalization of large coefficients is governed by λ.
- The OLS model achieves by construction the highest in-sample R^2. However, the ridge regression model might generalize better to previously unseen data (out-of-sample). This is true in our application where the ridge regression model achieved a (slightly) higher out-of-sample R^2 and a lower MSE compared to the OLS model.

LASSO regression I

Determining the regularization parameter

```
1  library(glmnet)
2  set.seed(2021) # for reproducibility
3
4  cvLASSO<-cv.glmnet(x=xTrain, yTrain, nlambda=100, alpha=1,
       lambda.min.ratio=0.0001)
5  #parameters are the same as for ridge regression except for
       alpha
6  #alpha: determines the weighting to be used, 0 for ridge and
        1 for LASSO regression
7
8  optimalLambda<-cvLASSO$lambda.min
9  optimalLambda
10      0.0005000578
```

LASSO regression II

Fitting the model

```
1  #fit LASSO regression model with the lambda determined on
       the previous slide to the training data
2  LASSOReg<-glmnet(x=xTrain, yTrain, alpha=1, lambda=
       optimalLambda)
3
4  coef(LASSOReg)
5  (Intercept) 0.0007795497
6  AAPL        0.0000000000
7  MSFT        .
8  AMZN        .
9  GOOG        .
10 GOOGL       .
11 ...
12 TR3M        .
13 yield       .
14 credit      .
```

LASSO regression III

In and ouf-of-sample performance

```
1  #in-sample R^2
2  predTrain<-predict(LASSOReg, newx=xTrain)
3  rSquared(pred=predTrain, actual=yTrain)
4      0
5
6  #out-of-sample performance
7  predTest<-predict(LASSOReg, newx=xTest)
8  rSquared(pred=predTest, actual=yTest)
9      -0.005150357
10 #reminder: out-of-sample R^2 simple linear model = -0.0138
11 rmse(pred=predTest, actual=yTest)
12     0.02395205
13 #reminder: out-of-sample rMSE simple linear model = 0.0241
```

Elastic net I

Determining the regularization parameters

```
1  library(caret)
2  set.seed(2021) #for reproducibility
3  #specify trainingControl object that specifies how repeated
       cross validation will take place
4  trainingControl<-trainControl(method="repeatedcv", number=5,
        repeats=10, search="random")
5
6  #builds the model in which a range of possible alpha and
       lambda values are tested and the optimum values are
       selected.
7  #tuneLength specifies the number of different alpha and
       lambda combinations to be tested
8  elasticNet<-train(forecast~., data=trainingData, method="
       glmnet", tuneLength=10, trControl=trainingControl)
9
10 elasticNet$bestTune #Optimal parameters
11          alpha    lambda
12       0.8154215 5.488304
```

Elastic net II

In and ouf-of-sample performance

```
1  #in sample predictions based on the optimal parameters from
       the previous slide:
2  predTrain<-predict(elasticNet, xTrain)
3  rSquared(pred=predTrain, actual=yTrain) #in-sample R^2
4      0
5
6  #out-of-sample predictions based on the optimal parameters
       from the previous slide:
7  predTest<-predict(elasticNet, xTest)
8  rSquared(pred=predTest, actual=yTest)
9      -0.005150357
10 #reminder: out-of-sample R^2 simple linear model = -0.0138
11 rmse(pred=predTest, actual=yTest)
12     0.02395205
13 #reminder: out-of-sample rMSE simple linear model = 0.0241
```

Bayes & KNN Classifier

Model accuracy - classification I

Recall the classification problem. How can we assess the model accuracy of classifiers (in contrast to regression methods)?

Training error rate (qualitative response)

The **training error rate**, i.e. the proportion of mistakes that are made if we apply our estimate $\hat{f}$ to the training observations, is defined as

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i) \qquad (21)$$

Indicator fn. Equals 1 when $y_i \neq \hat{y}_i$

with $\hat{y}_i$ being the predicted class label for the ith observation.

Model accuracy - classification II

How can we assess the model accuracy of classifiers when applied to new data?

Test error rate (qualitative response)

The **test error rate**, i.e. the proportion of mistakes that are made if we apply our estimate $\hat{f}$ to the test observations (x_0, y_0), is defined as

$$\text{Ave}\left(I(y_0 \neq \hat{y}_0)\right), \qquad (22)$$

i.e., the percentage of incorrectly classified test observations.

Bayes classifier (Most Basic Classifier)

Bayes classifier

The **Bayes classifier** is the classifier that assigns each observation to the most likely class, given its predictor values: Assign a test observation with predictor vector x_0 to the class j for which the cond. probability

$$Pr(Y = j | X = x_0)$$

is largest.

- The test error rate given is minimized, on average, by the Bayes classifier.
- **Problem: Distribution of Y given X is not known!** So only good in Theory but not in practice as Conditional prob. Not known.

k-nearest neighbors I (Use this Since Bayes can't be used in practice)

k-nearest neighbors (kNN) classifier

Starting point: a positive integer k and a test observation x_0

Procedure:

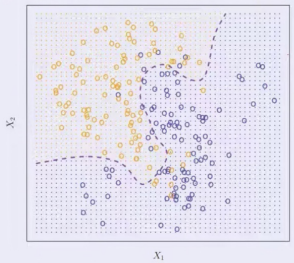
1. Identify the k points in the training data that are closest to x_0, represented by $\mathcal{N}_0$.
2. Estimate the conditional probability for class j as the fraction of points in $\mathcal{N}_0$ whose response values equal j:

$$Pr(Y = j | X = x_0) = \frac{1}{k}\sum_{i \in \mathcal{N}_0} I(y_i = j). \qquad (23)$$

3. Apply Bayes rule and classify the test observation x_0 to the class with the largest probability.

k-nearest neighbors II

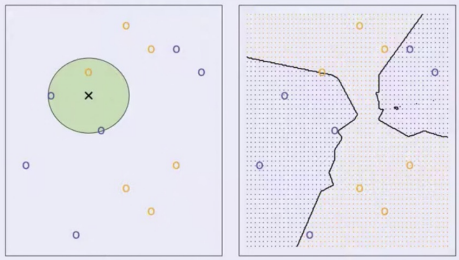
Classification - Bayes decision boundary



Simulated data with two classes. Bayes decision boundary in purple. Source: James, Witten, Hastie, and Tibshirani (2013).

k-nearest neighbors III

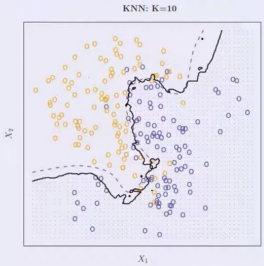
kNN - Simulated data $k = 3$



Source: James, Witten, Hastie, and Tibshirani (2013).

k-nearest neighbors IV

kNN vs. Bayes decision boundaries

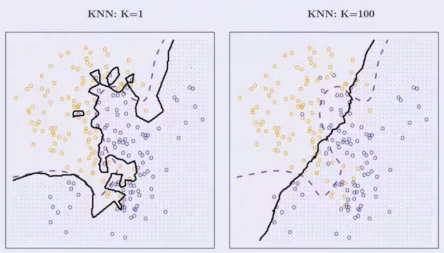


Source: James, Witten, Hastie, and Tibshirani (2013).

In Finance, Classification is normally used in Credit Risk Management. For eg. Classify Good & Bad loans, defaulting Customers Vs Non-defaulters, Customers prone to switch insurance.

k-nearest neighbors V

kNN decision boundaries for $k = 1$ and $k = 100$



Source: James, Witten, Hastie, and Tibshirani (2013).

K is a hyperparameter. Needs to be chosen before applying algorithm via Cross validation.
K=1 ⇒ More wiggly
K=100 ⇒ very coarse even linear decision boundary.
Bias–Variance tradeoff in play.

Maximal Margin Classifier

Support Vector Machines I

- **SVMs**: usually the summary of three distinct methods: the maximal margin classifier, the support vector classifier, and the support vector machine.
- Like kNN, it is used for classification.

Hyperplane

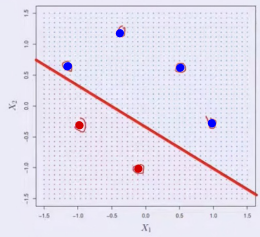
in p dimension

A **hyperplane** is a flat affine (linear) subspace of dimension $p - 1$. $p = 2$: a line; $p = 3$: a plane. A hyperplane divides the p-dimensional space into two halves and is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$ ← Hyperplane Eqn

Support Vector Machines II

Hyperplane



Hyperplane defined by $1 + 2X_1 + 3X_2 = 0$. Source: James, Witten, Hastie, and Tibshirani (2013).

Support Vector Machines III

observations
observations in

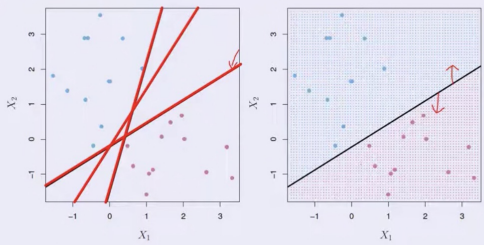
- **Input**: $n \times p$ data matrix X of n p-dimensional training observations

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

  with each observation belonging to one class, i.e., $y_1, \ldots, y_n \in \{-1; 1\}$ where -1 represents one class and 1 the other class, and a test observation with a p-vector of observed features $x^* = (x_1^*, \ldots, x_p^*)^T$
- **Output**: classification of x^* using a **separating hyperplane**.

Support Vector Machines IV

Classifier based on a separating hyperplane



Source: James, Witten, Hastie, and Tibshirani (2013).

Support Vector Machines V

Problem: if *one* separating hyperplane exists, then an *infinite* number of such hyperplanes exists. Which one should we use?

Maximal Margin Classifier

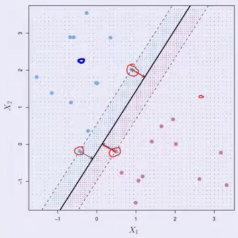
Natural choice: the maximal margin hyperplane, which is the separating hyperplane that is farthest from the training observations.

- Compute the (perpendicular) distance from each training observation to a given separating hyperplane.
- The smallest such distance is known as the margin.
- The maximal margin hyperplane is the separating hyperplane for which the margin is largest. S.t. distance from blue & Red pts are equal.

Maximum Margin classifier only depends on points called Support vectors so adding points that are not support vectors doesn't change Maximal margin hyperplane.

Support Vector Machines VI

Maximal margin hyperplane/classifier



Red Circled points are Support Vectors

The observations on the dashed lines that are equidistant from the separating hyperplane are the **support vectors**. Source: James, Witten, Hastie, and Tibshirani (2013).

Support Vector Machines VII

Construction of the Maximal Margin Classifier

The **Maximal Margin hyperplane** is the solution to the optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{\arg\max} \; M \tag{24}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \tag{25}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M, \forall i = 1, \ldots, n. \tag{26}$$

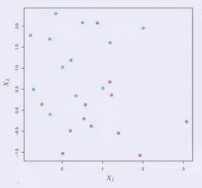
Constraint (26): ensures that each observation will be on the correct side of the hyperplane (with some buffer/margin M).
Constraint (25) ensures that each observation is at least a distance M from the hyperplane.

$y_i = 1$ or -1

Support Vector Machines VIII

If a separating hyperplane exists, use the Maximal Margin Classifier. But is this always the case? No.

Non-separable case



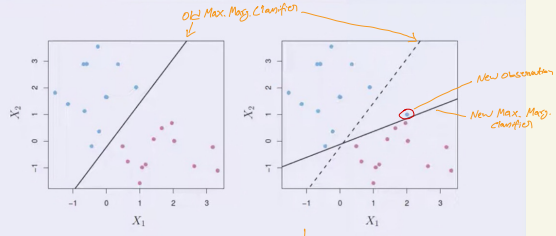
Can't Use Maximal Margin Classifier here.

Source: James, Witten, Hastie, and Tibshirani (2013).

Support Vector Machines IX

If a separating hyperplane exists, should we always use a classifier based on it? No. Since it's very sensitive to New Observations

Sensitivity of the Maximal Margin Classifier



Source: James, Witten, Hastie, and Tibshirani (2013).

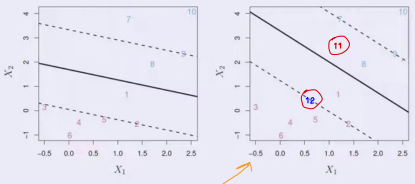
Additional point moves hyperplane by quite a lot instead of little movement. Hence, the Bias will be low but variance will be quite high as it depends all like of Red points which is opposite of Max. Marg. Classifier.

Support Vector Machines

Support Vector Machines X

Solution: Using a classifier that allows for some observations to be incorrectly classified.

Support Vector Classifier



Source: James, Witten, Hastie, and Tibshirani (2013).

With addition of two points 11 (Red) & 12 (Blue), the decision boundary doesn't change that drastically by using Support Vector Classifier 11 is incorrectly classified.

Support Vector Machines XI

Construction of the Support Vector Classifier

The **Support Vector hyperplane** is the solution to the problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M}{\arg\max} \; M \tag{27}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \tag{28}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \tag{29}$$

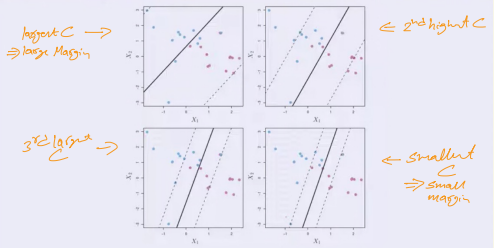
$$\epsilon_i \geq 0, \; \sum_{i=1}^{n} \epsilon_i \leq C, \tag{30}$$

with C being a nonnegative tuning parameter chosen via cross-validation. M is the width of the margin that we want to maximize. ϵ_i are so-called *slack variables* that allow observations to be on the wrong side of the margin or hyperplane.

When C is larger, there is a high tolerance for observations to be incorrectly classified. Hence, Margin will be large.

Support Vector Machines XII

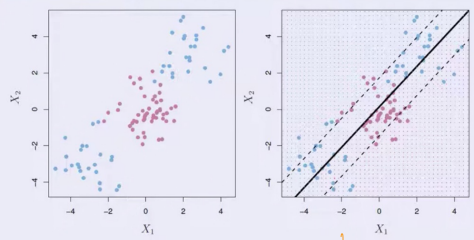
Support Vector Classifier - Influence of the tuning parameter C



Source: James, Witten, Hastie, and Tibshirani (2013).

Support Vector Machines XIII

Is a linear classifier always warranted?



Source: James, Witten, Hastie, and Tibshirani (2013).

Requires Non-linear classification here.

Support Vector Machines XIV

The Support Vector Classifier is a *linear* classifier. How can we (automatically) convert it to a classifier with *non*-linear decision bounds?

Classifier with non-linear decision bounds

$$\underset{\beta_0,\beta_{11},\beta_{12}\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n,M}{\arg\max} \quad M \tag{31}$$

$$\text{subject to } y_i\left(\beta_0+\sum_{j=1}^{p}\beta_{j1}x_{ij}+\sum_{j=1}^{p}\beta_{j2}x_{ij}^2\right)\geq M(1-\epsilon_i), \tag{32}$$

$$\epsilon_i\geq 0,\ \sum_{i=1}^{n}\epsilon_i\leq C,\ \sum_{j=1}^{p}\sum_{k=1}^{2}\beta_{jk}^2=1. \tag{33}$$

Previously we had a linear hyperplane, we now allow a polynomial f'n. This makes decision boundary to be polynomial.
This is one way of doing it, however it might not be enough & we need more Non-linearity.

Support Vector Machines XV

Recall the standard (or euclidean) inner product:

$$\langle x_i, x_{i'}\rangle=\sum_{j=1}^{p}x_{ij}x_{i'j}.$$

Linear Support Vector Classifier - Alternate representation

The linear Support Vector Classifier can be represented as

hyperplane ⟶ $$f(x)=\beta_0+\sum_{i=1}^{n}\alpha_i\langle x, x_i\rangle \tag{34}$$

← different representation of SV classifier.

with n parameters α_i. To estimate these and β_0, we only need the $n(n-1)/2$ inner products $\langle x, x_i\rangle$ between all pairs of training observations. Moreover, α_i is nonzero only for the support vectors in the solution (i.e., usually a much smaller number of parameters will suffice).

Support Vector Machines XVI

Why do we need a different representation of SV classifier?
⇒ Because we can extend it with Kernel.

How can we extend the linear SV classifier? Replace the inner product with a more general **Kernel**.

Kernel

A **Kernel** K (in the context of machine learning) is a function that quantifies the similarity of two observations.

If we have two observations x_1 & x_2, a kernel measures the similarity between x_1 & x_2, could be distance between two points. In case of inner product it measures distance.

After extending SV classifier with Kernel & substituting linear kernel with Non-linear kernel we get Support Vector Machine.

Support Vector Machines XVII

Kernel - examples

two vectors x_i & $x_{i'}$

Linear Kernel:

$$K(x_i,x_{i'})=\sum_{j=1}^{p}x_{ij}x_{i'j}. \tag{35}$$

Polynomial Kernel of degree d:

$$K(x_i,x_{i'})=\left(1+\sum_{j=1}^{p}x_{ij}x_{i'j}\right)^d \tag{36}$$

Radial Kernel:

$$K(x_i,x_{i'})=\exp\left(-\gamma\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2\right),\ (\gamma>0). \tag{37}$$

Support Vector Machines XVIII

Support Vector Machine

When using a non-linear kernel in combination with the Support Vector Classifier, the resulting classifier is known as a **Support Vector Machine**

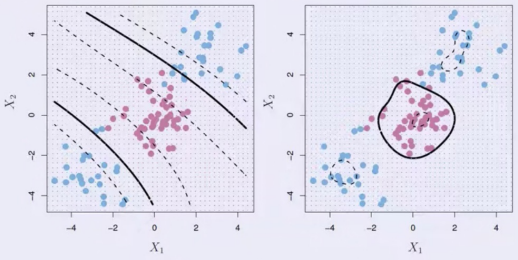
$$f(x)=\beta_0+\sum_{i\in\mathcal{S}}\alpha_i K(x,x_i) \tag{38}$$

← we don't need all n parameters as α_i will be non-zero only for support vectors.

with $\mathcal{S}$ being the set of support vectors (i.e., the training observations for which α_i is nonzero).

Support Vector Machines XIX

Support Vector Machines - Polynomial and Radial Kernels



Source: James, Witten, Hastie, and Tibshirani (2013).

← Much finer way of classification with SVM.

Application Of SVMs

Application I

Which customers are prone to terminate their contract.

- On the following slides we will apply the kNN model and SVMs to a dataset of credit card customers to predict customer churn.
- The data are available at **kaggle**. There you can find various (Python) notebooks featuring a wide variety of different methods for accomplishing this task.
- However, in this application we only use the kNN model and SVMs (linear and with a radial basis function (RBF) kernel).

Application II

Motivation

- The manager responsible for the credit card customers is worried about an increasing fraction of these customers quitting their contracts.
- To slow down customer churn, she wants to contact these customers proactively in order to change their decision.
- Therefore, she needs a prediction of who is going to terminate the contract.

Application III

The dataset

- The dataset contains information on more than 10,000 customers including age, salary, credit card limit, and other features.
- The data are unbalanced with only about 16 % of the customers having churned. This complicates training and interpretation of the predictive performance of the models.
- It might be more favorable to contact a customer who is not about to churn than not contacting a customer who is.
- In this lecture, our primary focus is on the models. We will therefore not specifically address the class imbalance problem.

Class imbalance → Many more instances of some classes than others.

Application IV

For fitting the kNN model and the SVMs (as well as the elastic net model from the previous section) we rely on the *caret* R-package by Kuhn (2020). Before we work with the data we will therefore shortly introduce the package.

caret

- Caret is short for classification and regression training.
- It summarizes activities related to model development in a streamlined process. It allows you to test different models with very little changes to the code and offers nearly automatic cross validation and parameter tuning.
- The package provides a consistent modeling syntax. For example, by simply changing the method argument one can easily change the underlying model. In total, the package provides access to more than 200 different models.

Application V

caret (cont.)

- Note that behind the scenes the package is not performing the modeling itself. Instead, it relies on various other specialized R-packages.
- For example, when using the method *lm* caret uses the lm() function from the *stats* package to estimate a linear regression model.
- When performing kNN classification, caret relies on the *class* package by Venables and Ripley (2002).
- For a complete list of models caret provides access to see **here**.

Import and preprocess data I

Import data

```
#download bankChurners.csv from here: https://www.kaggle.com
    /sakshigoyal7/credit-card-customers/download
#then import csv-file
library(readr)
bankChurners <- read_csv("data/BankChurners.csv")
#drop two last columns (according to dataset description at
    https://www.kaggle.com/sakshigoyal7/credit-card-
    customers)
bankChurners<-bankChurners[,1:(ncol(bankChurners)-2)]
```

Import and preprocess data II

Explorative analysis

```
print(bankChurners,n=7)

# A tibble: 10,127 x 21
   CLIENTNUM Attrition_Flag    Customer_Age Gender
       <dbl> <chr>                    <dbl> <chr>
1 768805383 Existing Customer           45 M
2 818770008 Existing Customer           49 F
3 713982108 Existing Customer           51 M
4 769911858 Existing Customer           40 F
5 709106358 Existing Customer           40 M
6 713061558 Existing Customer           44 M
7 810347208 Existing Customer           51 M
# ... with 10,120 more rows, and 17 more variables:
```

Import and preprocess data III

Variable description

- CLIENTNUM: unique identifier for the customer holding the account
- Attrition_Flag: if the account is closed then 1 else 0 (1 => Customer cancelled contract)
- Dependent count: number of dependents (Gender, age etc.)
- ...
- A description of all variables can be found **here**.

Import and preprocess data IV

Explorative analysis (cont.)

```
any(is.na(bankChurners))      => Check for missing values
    FALSE # no missing values

any(duplicated(bankChurners)) => Check for duplicates
    FALSE # no duplicates

# class imbalance => Checking class imbalance but we are not solving this in this example.
round(table(bankChurners$Attrition_Flag)/nrow(bankChurners)
    ,3)

Attrited Customer Existing Customer
            0.161             0.839
```

16.1% Customer Churning — 83.9% remaining Customer

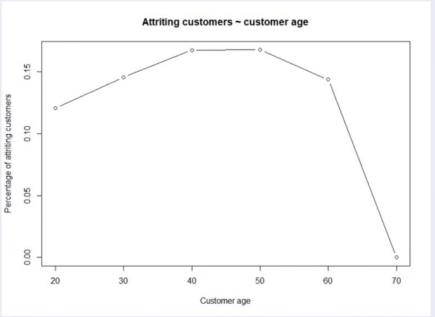
Import and preprocess data V

Explorative analysis (cont.)

```
# Analyze churning customers by age
library(dplyr)
percAgeGrouped<-bankChurners %>%
    mutate (interval=as.numeric(cut(Customer_Age, breaks=10*
        (2:8)))) %>%
    group_by(interval) %>%
    summarize(percentageAttrited=mean(Attrition_Flag=="
        Attrited Customer"))

plot(x=10*(2:7), y=percAgeGrouped$percentageAttrited, xlab="
    Customer age", ylab="Percentage of attriting customers",
     type="b")
```

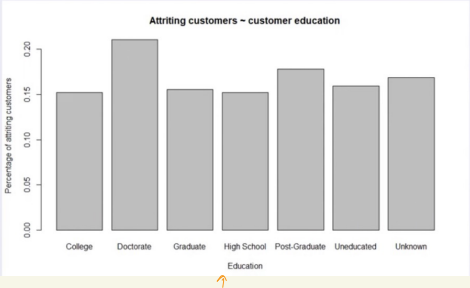
Import and preprocess data VI

Explorative analysis (cont.)



Import and preprocess data VII

Explorative analysis (cont.)



In this case since doctorate influences attriting customers, we could use one dummy variable & encode 1 for doctorate & zero for rest.

Import and preprocess data VIII

Categorical features

- Customer education is a categorical feature and cannot simply be encoded in a numeric variable as is sometimes possible for ordinal variables.
- Customer education might be encoded via 6 binary dummy variables. However, this raises the dimensionality of the data significantly. This is an especially serious problem for non-parametric methods like kNN classification because of the curse of dimensionality.
- For simplicity, we therefore drop all non-numeric variables. In practice, one should carefully decide for each (categorical) variable if it should be included or not.

Import and preprocess data IX

Create training and test set

- We remove all non-numeric features and the column CLIENTNUM (no predictive value).
- Furthermore, we create the training and test set by randomly including 80 % of observations in the training and 20 % of observations in the test set.

```
set.seed(2021) #for reproducibility
index<-sample.int(n=2, size=nrow(bankChurners), prob=c
    (0.8,0.2),replace=TRUE)
table(index)
      1    2
   8061 2066

xTrain<-bankChurners %>%
    select (-c("CLIENTNUM", "Attrition_Flag")) %>%
    select(where(is.numeric)) %>% filter(index==1)
```

Import and preprocess data X

Create training and test set (cont.)

```
xTest<-bankChurners %>%
    select (-c("CLIENTNUM", "Attrition_Flag")) %>%
    select(where(is.numeric)) %>%
    filter(index==2)

yTrain<-as.factor(bankChurners$Attrition_Flag[index==1])
yTest<-as.factor(bankChurners$Attrition_Flag[index==2])
```

bankChurners & Attrition_Flag is our response variable.

Import and preprocess data XI

Create training and test set (cont.)

By default, kNN is based on the Euclidean distance. To ensure that all features contribute equally to the measured distances we scale the data based on the minimum and maximum values of the training data.

```
minVec<-apply(xTrain,2,min) # determine per-column minimum
maxVec<-apply(xTrain,2,max) # determine per-column maximum

# scale variables
xTrainScaled<-as.data.frame(t(apply(xTrain,1,function(row){(
    row-minVec)/(maxVec-minVec)})))

xTestScaled<-as.data.frame(t(apply(xTest,1,function(row){(
    row-minVec)/(maxVec-minVec)})))
```

Import and preprocess data XII

Create training and test set (cont.)

All variables in the training set now have minimum 0 and maximum 1. As scaling is based on data from the training set, the same is not necessarily true for the test set:

```
# calculate maximum per column --> apply()
# then provide summary statistics and round results to four
    decimal places
round(summary(apply(X=xTrainScaled, MARGIN=2, FUN=max)),4)
       Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
          1       1       1       1       1       1

round(summary(apply(X=xTestScaled, MARGIN=2, FUN=max)),4)
       Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     0.7741  0.9614  1.0000  0.9633  1.0000  1.0000
```

kNN classification I

Introduction

- For our kNN classification model we rely on the Euclidean distance. Many other distance metrics are possible (e.g., Manhattan distance, cosine distance, etc.).
- The number of neighbors is a hyperparameter. We determine an appropriate value via 10-fold cross validation.
- Computations can be performed in parallel to speed up the estimation process.
- We determine the optimal parameter and evaluate a models' predictive performance based on the metrics accuracy and Cohen's kappa (more details later).

kNN classification II

Hyperparameter tuning

cross validation

```
1  library(caret)
2  trControl <- trainControl(method  = "cv", number  = 10)
3  set.seed(2021) #for reproducibility
4
5  # perform cross validation in parallel
6  # if you encounter any problems with the PSOCK-cluster, the
       corresponding lines can be commented out.
7  library(doParallel)
8  cl <- makePSOCKcluster(6) #number of cores/threads to use
9  registerDoParallel(cl)
10
11 kNNModel<-train(x=xTrainScaled, y=yTrain, method="kNN",
       tuneGrid=expand.grid(k=1:10), trControl=trControl,
       metric="Accuracy") # consider 1, 2, ..., 10 neighbors
12
13 stopCluster(cl)
```

training parameter

kNN classification III

Model summary

```
1  print(kNNModel) #optimal number of neighbors is the same for
       both accuracy and Cohen's kappa. In general, this is
       not necessarily the case.
2
3  Resampling results across tuning parameters:
4     k   Accuracy   Kappa
5     1  0.8988959  0.6070618
6    ...
7     5  0.9131598  0.6398903
8     6  0.9160140  0.6506041
9     7  0.9150200  0.6423632
10   ...
11   10  0.9125397  0.6288506
12
13 Accuracy was used to select the optimal model using the
       largest value.
14 The final value used for the model was k = 6.
```

kNN classification IV

Predictive performance

```
1  # predict class labels for the test set
2  yPredKnn<-predict(kNNModel,newdata=xTestScaled)
3
4  # print confusion matrix (see the next slide)
5  print(confusionMatrix(data=yPredKnn, reference=yTest))
```

confusion Matrix

kNN classification V

Bad for prediction Accuracy
Good for prediction Accuracy

Predictive performance (cont.)

Confusion Matrix and Statistics

To compare prediction vs Truth

Prediction	Reference: Attrited Customer	Reference: Existing Customer
Attrited Customer	189	53
Existing Customer	152	1672

```
               Accuracy : 0.9008
                 95% CI : (0.8871, 0.9133)
    No Information Rate : 0.8349
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5925

 Mcnemar's Test P-Value : 7.668e-12

            Sensitivity : 0.55425
            Specificity : 0.96928
         Pos Pred Value : 0.78099
         Neg Pred Value : 0.91667
             Prevalence : 0.16505
         Detection Rate : 0.09148
   Detection Prevalence : 0.11713
      Balanced Accuracy : 0.76176

       'Positive' Class : Attrited Customer
```

kNN classification VI

Predictive performance (cont.)

- Accuracy is defined as the percentage of correctly classified observations. In our example, we obtain an accuracy of

$$90.08\ \% = \frac{189 + 1672}{189 + 1672 + 53 + 152} = \frac{1861}{2066}.$$

- No Information Rate (NIR) is defined as the largest proportion of the observed classes. In our example this corresponds to the proportion of existing customers

$$83.49\ \% = \frac{53 + 1672}{2066}.$$

NIR is the highest accuracy which can be achieved by a constant prediction.

kNN classification VII

Predictive performance (cont.)

- Cohen's kappa is a measure of a classifier's performance relative to how well the model would have performed simply by chance. Therefore, one compares the accuracy of the model to the hypothetical probability of an agreement by chance.
- In our example, 83.49 % (NIR) of observations belong to existing customers while 16.51 % belong to attrited customers. The kNN model classifies $\frac{152+1672}{2066} = 88.29\ \%$ as existing and $\frac{189+53}{2066} = 11.71\ \%$ as attrited customers. The probability for an agreement by chance p_e can thus be calculated as
$p_e = 83.49\ \% \cdot 88.29\ \% + 16.51\ \% \cdot 11.71\ \% = 75.65\ \%$. Cohen's kappa κ is now defined as

$$\kappa := \frac{\text{accuracy} - p_e}{1 - p_e} = \frac{90.08\ \% - 75.65\ \%}{1 - 75.65\ \%} = 0.5926.$$

kNN classification VIII

Predictive performance (cont.)

- When comparing two models, a higher kappa signals a better predictive performance. The maximum value is 1.
- There is, however, no standardized way of interpreting its values.
- A negative value of kappa signals that the model's predictions are worse than predicting by chance.

kNN classification IX

Predictive performance (cont.)

- Error costs of positives (in our example attrited customers) and negatives (existing customers) are usually different (think, e.g., of credit default or Covid-19 tests). In this context, sensitivity and specificity are more informative than accuracy.
- Sensitivity (recall or true positive rate) is defined as the number of correct positive predictions divided by the total number of positives (55.43 % in our example).
- Specificity (true negative rate) is defined as the number of correct negative predictions divided by the total number of negatives (96.93 %).
- Our kNN classifier is good in predicting customers that do not terminate their credit card contract but bad in predicting customers that do.
  Possible reasons, possible problems? due to unbalanced data

SVM I

We now use SVMs for the same classification task.

Introduction

- We will fit a linear SVM as well as a SVM with a RBF kernel to the training data to predict labels for the test data.
- For the linear SVM (without kernel) we determine the most appropriate value for the hyperparameter C, also known as *Cost*, out of the set (grid) $\{2^{-5}, 2^{-4}, \ldots, 2^4, 2^5\}$. Again, this can be done in parallel.
- The SVM with RBF kernel has two hyperparameters (sigma and C). We do not provide a set of possible values but let caret try 10 reasonable parameter combinations that is chooses itself (*tuneLength* parameter).

SVM II

Hyperparameter tuning (linear SVM)

```
set.seed(2021)
trControl <- trainControl(method  = "cv", number  = 5) #
    only use 5-fold cross validation to speed up estimation
grid<-expand.grid(C=2^(-5:5))

cl <- makePSOCKcluster(6) #number of cores/threads
registerDoParallel(cl)

svmLinear <- train(x=xTrainScaled, y=yTrain, method = "
    svmLinear", trControl=trControl, tuneGrid = grid)

stopCluster(cl)
```

SVM III

Model summary (linear SVM)

```
print(svmLinear)
Resampling results across tuning parameters:
      C          Accuracy   Kappa
    0.03125  0.9017494  0.5792921
    0.06250  0.9010053  0.5806835
    0.12500  0.9011295  0.5831063
    0.25000  0.9006332  0.5821980
   ...
   32.00000  0.9003851  0.5823553

Accuracy was used to select the optimal model using the
    largest value.
The final value used for the model was C = 0.03125.
```

In this case, choosing C with regard to accuracy leads to a different value than choosing it with regard to Cohen's kappa.

SVM IV

Hyperparameter tuning (SVM with RBF kernel)

```
set.seed(2021)
cl <- makePSOCKcluster(6) #number of cores/threads
registerDoParallel(cl)

svmRadial <- train(x=xTrainScaled, y=yTrain, method = "
    svmRadial", trControl=trControl, tuneLength = 10)

stopCluster(cl)
```

SVM V

Model summary (SVM with RBF kernel)

```
print(svmRadial)

Resampling results across tuning parameters:
      C        Accuracy   Kappa
      0.25  0.9175042  0.6405062
   ...
      4.00  0.9405779  0.7624900
      8.00  0.9420669  0.7711532
     16.00  0.9410748  0.7703549
   ...
   128.00  0.9304059  0.7378950

Tuning parameter 'sigma' was held constant at a value of
     0.05209594. Accuracy was used to select the optimal
     model using the largest value. The final values used for
      the model were sigma = 0.05209594 and C = 8.
```

SVM VI

Predictive performance

```
yPredSvmLinear<-predict(svmLinear,newdata=xTestScaled)
yPredSvmRadial<-predict(svmRadial,newdata=xTestScaled)

# output on the next slides
print(confusionMatrix(data=yPredSvmLinear, reference=yTest))
print(confusionMatrix(data=yPredSvmRadial, reference=yTest))
```

SVM VII

Predictive performance (linear SVM)

```
Confusion Matrix and Statistics

                    Reference
Prediction          Attrited Customer Existing Customer
  Attrited Customer               186                46
  Existing Customer               155              1679

               Accuracy : 0.9027
                 95% CI : (0.8891, 0.9152)
    No Information Rate : 0.8349
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5951

 Mcnemar's Test P-Value : 2.582e-14

            Sensitivity : 0.54545
            Specificity : 0.97333
         Pos Pred Value : 0.80172
         Neg Pred Value : 0.91549
             Prevalence : 0.16505
         Detection Rate : 0.09003
   Detection Prevalence : 0.11229
      Balanced Accuracy : 0.75939

       'Positive' Class : Attrited Customer
```

SVM VIII

Better than linear SVM & KNN in this Example.

Predictive performance (SVM with RBF kernel)

```
Confusion Matrix and Statistics

                    Reference
Prediction          Attrited Customer Existing Customer
  Attrited Customer               258                45
  Existing Customer                83              1680

               Accuracy : 0.938
                 95% CI : (0.9268, 0.9481)
    No Information Rate : 0.8349
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7647

 Mcnemar's Test P-Value : 0.001074

            Sensitivity : 0.7566
            Specificity : 0.9739
         Pos Pred Value : 0.8515
         Neg Pred Value : 0.9529
             Prevalence : 0.1651
         Detection Rate : 0.1249
   Detection Prevalence : 0.1467
      Balanced Accuracy : 0.8653

       'Positive' Class : Attrited Customer
```

SVM IX

Predictive performance (summary)

- The predictive performance of the linear SVM is very similar to that of the kNN classifier in terms of accuracy, Cohen's kappa, sensitivity, and specificity.
- The SVM with RBF kernel can improve on these results with an accuracy of 93.80 % and a kappa of 0.7647.
- This substantial increase is mainly due to an improvement in predicting attriting customers (sensitivity 75.66 %). This compares to a sensitivity of 54.55 % for the linear SVM.

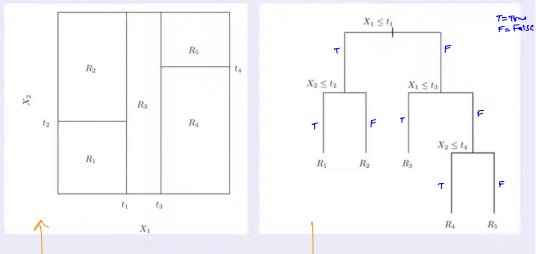
Classification And Regression Trees

Introduction I

- Trees can be applied to both classification and regression problems.
- Trees involve stratifying or segmenting the predictor space into a number of simple box-shaped regions.
- This segmenting of the predictor space is performed based on a set of splitting rules. These rules can be summarized in a tree.

Introduction II

Exemplary partition of a two-dimensional feature space and the corresponding tree



Source: James, Witten, Hastie, and Tibshirani (2013).

Segmenting the predictor Space. Predictors X_1 & X_2.

Tree

Making predictions

- Predictions for a specific observation are typically made by a majority vote (classification) or by using the mean or mode of the training observations (regression) in the region corresponding to the given observation.
- For a given region, the prediction for every observation that falls into this region is the same!

Building a tree I

- In theory, for making predictions as on the previous slide the regions could have any shape.
- However, for simplicity and ease of interpretation the predictor space is divided into high-dimensional rectangles or boxes.
- These boxes are chosen such that the classification error (classification) or the squared errors of the residuals (regression) are minimized.

Building a tree II

(Possible to partition the predictor space into as no. of boxes)

- It is computationally infeasible to consider any possible partition of the feature space into a finite number of boxes.
- Therefore, one typically relies on a *top-down, greedy* approach that is commonly referred to as *recursive binary splitting*.
  - top-down: The approach begins at the top of the tree where all observations belong to a single region and then successively splits the feature space.
  - greedy: At each step the best split at that particular step is made. One does not look ahead and pick a split that will lead to a better tree at some future step.

Building a tree III

- For performing recursive binary splitting, one first selects the predictor X_j and the cutpoint s such that the squared errors (regression) or the classification error rate (classification) are minimized over the resulting regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$. In classification tasks, sometimes also the Gini index or entropy[1] is used as a criterion for making tree splits.
- In this process, all predictors $X_1, \ldots, X_p$ and all possible values for the cutpoint s are considered.
- This process is then repeated in each of the resulting regions. Therefore, one again looks for the best predictor and best cutpoint to minimize the squared regression errors or the classification error rate further.

[1] The Gini index is defined as $G := \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$, where $\hat{p}_{mk}$ represents the proportion of training observations in the m'th region that are from the k'th class. Entropy is defined as $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$.

Building a tree IV

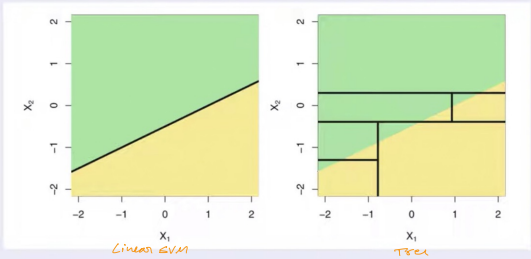
- This process continues until some stopping criterion is met.
- By adding further splits to the tree and growing the tree deeper and deeper, the squared regression errors or the classification error rate can only decrease (or stay the same).
- Consequently, the stopping criterion is crucial to avoid overfitting due to a tree that is too complex! Indeed, smaller trees with fewer splits might lead to lower variance and better interpretation at the cost of some additional bias.

Building a tree V

- A possible remedy to overfitting trees is to grow the tree only as long as the reduction in classification or regression error exceeds some threshold, yielding smaller trees.
- This minimum required reduction can be described by a complexity parameter balancing reductions in classification or regression error against the complexity of the model.
- In the following application, we will see that a high value of the complexity parameter leads to more shallow trees while a low value yields deeper trees.
- The optimal value of the complexity parameter can be determined via cross-validation.

Trees vs. linear models I

Linear boundary



Source: James, Witten, Hastie, and Tibshirani (2013).

For linear boundary using a Tree leads to large bias & huge classification error.

Trees vs. linear models II

Non-linear boundary

Huge classification Error

Source: James, Witten, Hastie, and Tibshirani (2013).

For Non-linear boundary using Tree here we get a perfect classification.

Advantages and disadvantages I

Advantages

Trees

- are easy to explain.
- closely mirror human decision-making.
- can be displayed graphically.
- can handle qualitative predictors.

Advantages and disadvantages II

Disadvantages

- (Single) trees often exhibit an inferior predictive accuracy as compared to many other approaches.
- Trees can be very non-robust. A small change in the data might lead to a completely different tree.

By aggregating many trees via methods like bagging, random forests, or boosting, their predictive performance can be improved substantially. An example is included in the application.

Classification Trees: Applications

Applications

- On the following slides we will employ tree models for both classification and regression tasks.
- We will again use the credit card customer dataset from the previous lecture to predict customer churn (classification).
  To provide a more complete picture we will also employ boosted decision trees (a special kind of random forests).
- In the regression task, we will forecast health insurance premia. The data are available at **kaggle**.

Classification I

We start with the classification task. Here is a short reminder of the credit card customer dataset:

Credit card customer churn prediction - reminder

- The manager responsible for the credit card customers is worried about an increasing fraction of the customers quitting their credit card services.
- To slow down customer churn she wants to proactively contact customers who are about to leave their credit card services to change their decision.
- Therefore, she needs predictions on who is going to quit the services.

Classification II

Credit card customer churn prediction - reminder (cont.)

- The dataset contains information on more than 10,000 customers including age, salary, credit car limit, and other features.
- The data is unbalanced with only about 16 % of the customers having churned. This complicates training and interpretation of the accuracy of the models.
- Again, we focus on the application of the models and will not specifically address the class imbalance problem.

Import and preprocess data I

We proceed similar to slides 220 ff.

Import data

```
#download bankChurners.csv from here: https://www.kaggle.com
    /sakshigoyal7/credit-card-customers/download
#then import csv-file

library(readr)
bankChurners <- read_csv("data/bankChurners.csv")

#drop two last columns (according to dataset description at
    https://www.kaggle.com/sakshigoyal7/credit-card-
    customers)
bankChurners<-bankChurners[,1:(ncol(bankChurners)-2)]
```

Import and preprocess data II

As opposed to the kNN/SVM example, we do not exclude categorical features.

Create training and test set

```
# We randomly select 80 % of observations for the training
    and the remaining 20 % of observations for the test set.
# We exclude CLIENTNUM which is a number for identifying
    individual customers and has no predictive value.
library(dplyr)
set.seed(2021) #for reproducibility
index<-sample.int(n=2, size=nrow(bankChurners), prob=c
    (0.8,0.2),replace=TRUE)

xTrain<-bankChurners %>% select (-c("CLIENTNUM", "Attrition_
    Flag")) %>%
  filter(index==1)
```

Import and preprocess data III

Create training and test set (cont.)

```
xTest<-bankChurners %>% select (-c("CLIENTNUM", "Attrition_
    Flag")) %>%
  filter(index==2)

yTrain<-as.factor(bankChurners$Attrition_Flag[index==1])
yTest<-as.factor(bankChurners$Attrition_Flag[index==2])
```

Decision trees do not require feature scaling because they are not sensitive to the variance in the data.

Fitting the classification tree I

Hyperparameter tuning

```
library(caret)
library(doParallel)
trControl <- trainControl(method  = "cv", number  = 10)
set.seed(2021) -> hyperparamter tuning / selecting

# if you encounter any problems with the PSOCK-cluster, the
    according lines can be commented out.
cl <- makePSOCKcluster(6) #number of cores/threads
registerDoParallel(cl)
treeModel<-train(x=xTrain, y=yTrain, method="rpart",
                 tuneLength=20,  # training trees is
                     relatively fast, so we can consider
                     more possible values
                 trControl=trControl, metric="Accuracy")
stopCluster(cl)
```

partitioning

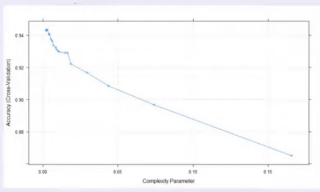
With increasing complexity, the accuracy from Cross Validation decreases.

Fitting the classification tree II

Hyperparameter tuning (cont.)

```
print(treeModel$bestTune)
          cp
0.002851218
plot(treeModel)
```

-> complexity parameter



Fitting the classification tree III

Hyperparameter tuning - the complexity parameter

- The complexity parameter *cp* in the *rpart* package (implicitly used by caret) is the minimum improvement required at each node to make a further split.
- The parameter balances possible reductions in the classification error via further splits against the complexity of the model (number of splits, depth).
- A high value of cp leads to more shallow trees while a low value yields deeper trees (that might overfit). We will exemplify this on the following slides.

Fitting the classification tree IV

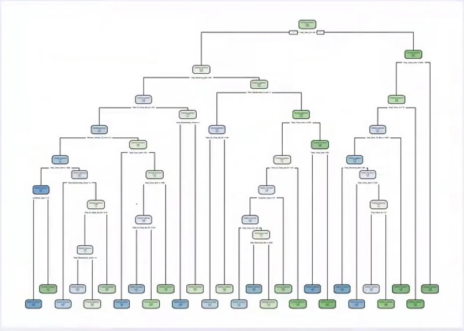
The *rpart.plot* R-package provides convenient visualizations of tree models. However, it requires a tree model that was fitted by rpart directly (and not via caret). We will therefore again fit the respective model with the optimal parameters determined in the hyperparameter tuning step.

Visualization of the tree

```
library(rpart.plot)
bestParam<-treeModel$bestTune # best complexity parameter
    from cross validation

# fit tree on training set (index==1) and remove first
    column (CLIENTNUM)
treeModel1<-rpart(Attrition_Flag ~.,
          data=bankChurners[index==1,-1],
          cp=bestParam)

rpart.plot(treeModel1) # next slide
```

Fitting the classification tree V

Visualization of the tree (cont.)



Fitting the classification tree VI

We will now raise the complexity parameter to obtain a more shallow tree.

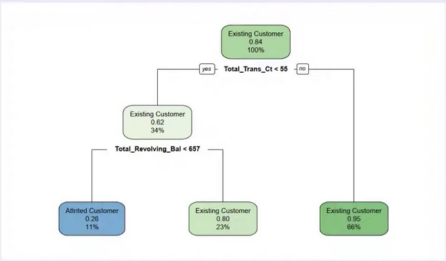
Shallow tree

```
# multiply the complexity parameter from the current tree
# (bestParam) with 50
treeModel2<-rpart(Attrition_Flag ~.,
          data=bankChurners[index==1,-1],
          cp=bestParam*50)

rpart.plot(treeModel2) # see next slide
```

Fitting the classification tree VII

Shallow tree (cont.)

Total_Trans_Ct denotes the total transaction count in the last 12 months. *Total_Revolving_Bal* denotes the total revolving balance on the credit card.

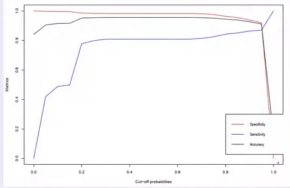


Fitting the classification tree VIII

> Box → Class 1
< Box → Class 2

Cut-off level

Classification in each node is performed by majority vote. However, it is possible and sometimes useful to choose a cut-off level different from 0.5. By varying the cut-off level, we can influence sensitivity and specificity. Accuracy closely follows specificity due to the class imbalance in the data. The figure is based on the training data and the deeper tree.



Predictive performance of the classification tree I

Predictive performance

For prediction in the test set we rely on the model determined via cross validation (the deeper tree).

```
yPredTree<-predict(treeModel,newdata=xTest)

print(confusionMatrix(data=yPredTree, reference=yTest)) #
    see the next slide
```

Predictive performance of the classification tree II

Predictive performance (cont.)

```
Confusion Matrix and Statistics

                Reference
Prediction       Attrited Customer Existing Customer
  Attrited Customer               252               48
  Existing Customer                89             1677

               Accuracy : 0.9337
                 95% CI : (0.9221, 0.944)
    No Information Rate : 0.8349
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7472

 Mcnemar's Test P-Value : 0.0006322

            Sensitivity : 0.7390
            Specificity : 0.9722
         Pos Pred Value : 0.8400
         Neg Pred Value : 0.9496
             Prevalence : 0.1651
         Detection Rate : 0.1220
   Detection Prevalence : 0.1452
      Balanced Accuracy : 0.8556

       'Positive' Class : Attrited Customer
```

The predictive performance is comparable to that of the SVM with a RBF kernel, see slide 248.

Boosted Classification Trees

Improving the predictive accuracy & predictive performance of decision & classification trees.

Excursion I

- Deep trees are prone to overfitting while shallow trees might underfit the data.
- A possible solution to this problem is to construct multiple (different) trees at training time and produce forecasts based on the mode of the classes (classification) or mean/median of the response variable (regression) of the individual trees.
- This ensemble technique is called random forest.

Excursion II

- Random forests can be constructed by introducing randomness into the construction of each tree, e.g., by choosing random subsamples, variables, and splits, etc. This yields trees that are built independently of each other. This concept is often referred to as bagging (bootstrap aggregating).
- Another approach is to use multiple trees in a boosting framework where weak learners (shallow trees) are combined to yield a stronger estimator such that at each iteration step classification or regression error is reduced. This yields boosted classification/regression trees where the individual trees are no longer built independently of each other. For an illustration see, e.g., **here** or **there**.
- On the next slides we will apply boosted decision trees to our classification problem. Therefore, we rely on the XGBoost algorithm.

Excursion III

Fit boosted decision trees

```
library(caret)
library(doParallel)
trControl <- trainControl(method  = "cv", number  = 5,
    verboseIter = FALSE) # fitting boosted decision trees is
     computationally very intense. We therefore only use 5-
    fold cross validation.
set.seed(2021) # for reproducibility

# XGBoost can only handle numeric data. Therefore, we
    exclude all non-numeric features.
xTrain<-xTrain %>% select(where(is.numeric))
xTest<-xTest %>% select(where(is.numeric))
```

Excursion IV

Fit boosted decision trees (cont.)

```
# the underlying XGBoost algorithm already uses parallel
    processing (implicitly). We therefore only employ two
    parallel processes in caret (wrapped around the implicit
     parallelism).
cl <- makePSOCKcluster(2)
registerDoParallel(cl)

# we set tuneLength=2 as caret selects 5 different
    parameters leading to 2^5 different parameter
    combinations that are considered in cross validation
xgBoostModel<-train(x=xTrain, y=yTrain, method="xgbTree",
    tuneLength=2, trControl=trControl, metric="Accuracy")

stopCluster(cl)
```

XGBoost

Excursion V

Fit boosted decision trees (cont.)

```
print(t(xgBoostModel$bestTune))

     nrounds          100.0
     max_depth          2.0
     eta                0.4
     gamma              0.0
     colsample_bytree   0.8
     min_child_weight   1.0
     subsample          1.0
```

Excursion VI

Forecasting accuracy of boosted decision trees

```
yPredXgBoost<-predict(xgBoostModel,newdata=xTest)

print(confusionMatrix(data=yPredXgBoost, reference=yTest))
```

Excursion VII

Forecasting accuracy of boosted decision trees (cont.)

```
Confusion Matrix and Statistics

                Reference
Prediction          Attrited Customer Existing Customer
  Attrited Customer               298               26
  Existing Customer                43             1699

               Accuracy : 0.9666
                 95% CI : (0.9579, 0.9739)
    No Information Rate : 0.8349
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8764

 Mcnemar's Test P-Value : 0.05408

            Sensitivity : 0.8739
            Specificity : 0.9849
         Pos Pred Value : 0.9198
         Neg Pred Value : 0.9753
             Prevalence : 0.1651
         Detection Rate : 0.1442
   Detection Prevalence : 0.1568
      Balanced Accuracy : 0.9294

       'Positive' Class : Attrited Customer
```

Much better than what we have had before like KNN, SVM or decision tree.

Excursion VIII

Forecasting accuracy of boosted decision trees (cont.)

- The predictive performance of the boosted decision tree model exceeds that of all previous models (decision tree, SVMs, kNN classifier).
- In particular, the sensitivity is substantially improved from 73.9 % for the single tree to 87.39 % for the boosted tree ensemble.
- By relying on the predictions from the boosted classification tree ensemble, for 298 out of 341 customers we can (rightly) predict that they are about to quit their credit card services. By, e.g., offering special price conditions to these customers, the bank manager might prevent them from quitting their contracts.
- On the other hand, we would falsely approach only 26 out of 1725 existing customers.

Regression Trees (Predicting Insurance Premium)

Regression trees I

We now continue with regression trees.

The dataset

- For our application we rely on the insurance dataset available at **kaggle**.
- The dataset contains information on more than 1,300 health insurance policies including four numeric features (age, bmi, number of children, expenses) and three nominal features (sex, smoker, region).
- We are interested in predicting the insurance premia (expenses).

Regression trees II

Import data

```
1  library(readr)
2  # download data (see link on the previous slide) and save
       the file as insurance.csv
3  insurance <- read_csv(``data/insurance.csv'')
4
5  sum(duplicated(insurance))
6      1 # there seems to be a duplicate entry
7  insurance<-insurance[!duplicated(insurance),] # remove
       duplicate entry
8
9  sum(is.na(insurance))
10     0 # no missing values
11
12 summary(insurance$expenses) -> insurance Premium Summary
13    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
14    1122    4740    9382   13270   16640   63770
```

Regression trees III

View data

```
1  library(tibble)
2  insurance<-as_tibble(insurance)
3
4  print(insurance, n=8)
5       age sex      bmi children smoker region    expenses
6     <dbl> <chr>  <dbl>    <dbl> <chr>  <chr>        <dbl>
7  1     19 female  27.9        0 yes    southwest   16885.
8  2     18 male    33.8        1 no     southeast    1726.
9  3     28 male    33          3 no     southeast    4449.
10 4     33 male    22.7        0 no     northwest   21984.
11 5     32 male    28.9        0 no     northwest    3867.
12 6     31 female  25.7        0 no     southeast    3757.
13 7     46 female  33.4        1 no     southeast    8241.
14 8     37 female  27.7        3 no     northwest    7282.
15 # ... with 1,327 more rows
```

Regression trees IV

Create training and test data

```
1  index<-sample(c(1,2), size=nrow(insurance), prob=c(0.8,0.2),
       replace=TRUE) # sample with replacement from the set
       {1,2} with probabilities 80 % and 20 %.
2
3  xTrain<-insurance[index==1, 1:6] # exlude last column (
       expenses)
4  yTrain<-insurance$expenses[index==1]
5
6  xTest<-insurance[index==2, 1:6]
7  yTest<-insurance$expenses[index==2]
```

Regression trees V

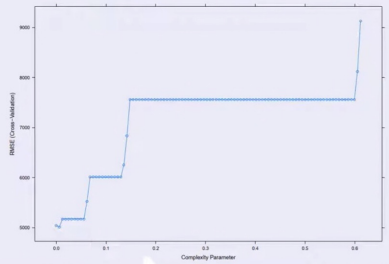
Fitting the regression tree - hyperparameter tuning

```
1  library(caret)
2  trControl <- trainControl(method  = "cv", number  = 10,
       verboseIter = FALSE)
3  set.seed(2021) # for reproducibility
4
5  # Single trees are fitted very fast. Therefore, we do not
       need parallel computing and can consider various
       hyperparameters (tuneLength=100)
6  regressionTree<-train(x=xTrain, y=yTrain, method="rpart",
7                        tuneLength=100, trControl=trControl, metric
                            ="RMSE")
8
9  print(regressionTree$bestTune) # complexity parameter
10           cp
11     0.00617757
```

Can't use "Accuracy" Metric which is for classification. So we will use RMSE for Regression task.

Regression trees VI

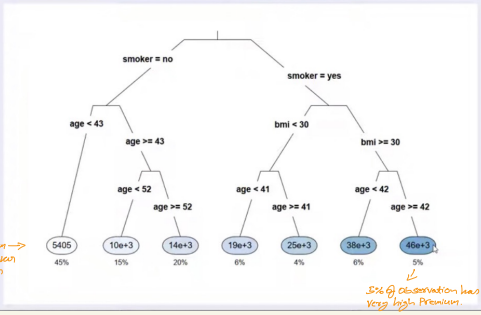
Fitting the regression tree - hyperparameter tuning (cont.)



As complexity parameter increases RMSE increases.

Regression trees VII

Visualization of the optimal tree (optimal cp)



Insurance Premium -> low for Non-Smoker, high for Smoker

5% of observation has very high Premium.

Regression trees VIII

Predictive accuracy

```
1  yPredRegressionTree<-predict(regressionTree,newdata=xTest)
2  metrics<-postResample(pred=yPredRegressionTree, obs=yTest)
3
4  print(metrics)
5          RMSE  Rsquared       MAE
6       4238.624 0.8779503 2909.371
7
8  # RMSE: root mean squared error
9  # Rsquared: R^2 -> 87.79%. In this model close to 88% of variance can be explained by this Regression Tree.
10 # MAE: mean absolute error
```

Regression trees IX

Boosted regression trees

- We could also use boosted regression trees. However, XGBoost only works with numeric features. Excluding categorical variables like *smoker* leads to an inferior model performance (RMSE = 11452, R^2 = 0.09, MAE = 8993).
- Instead relying on bagging via the *method="treebag"* (and using all features) does not lead to an improved performance compared to the single tree model (RMSE = 4888, R^2 = 0.81, MAE = 3045)

This alternate Models doesn't perform well for this example.

Neural Networks: Structure

Introduction I

- The term neural network encompasses a large class of models. The main focus in this lecture is on "vanilla" neural networks which are often referred to as single-layer perceptron or single hidden layer back-propagation networks.
- We also shortly cover deeper networks (multilayer perceptron) as well as convolutional neural networks.
- This section's contents are based on Hastie, Tibshirani, and Friedman (2017) and Goodfellow, Bengio, and Courville (2016).

Introduction II

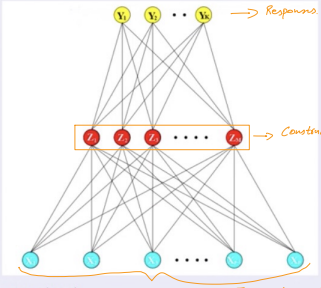
Historical trends

- Neural networks are not new. In fact, neural networks and deep learning date back to the 1940s and have since gone by many names.
- In the 1940s-1960s, deep learning was known as cybernetics, in the 1980s-1990s it was known as connectionism, and only since around 2006 it is commonly referred to as deep learning.
- With an increasing amount of training data, neural networks have become more useful.
- Over time, models have grown in size as both hardware and software have improved.
- Neural networks have helped to solve increasingly complicated applications with increasing accuracy.

Structure of a neural network I

The following figure shows the network diagram of a single-layer perceptron:

Single hidden layer, feed-forward neural network



Responses For insurance premium example we will have only one response Y_1. But this model can predict multiple responses/classes

Constructing variables Z_i in this hidden layer by combining features linearly

Features $(X_1, \ldots, X_p)$

Source: Hastie, Tibshirani, and Friedman (2017).

Instead of using features to directly predict Y using regression analysis/Trn, we would first compute the hidden variable Z_i then again recombine these variables to predict Y.

Structure of a neural network II

- The network structure from the previous slide is suitable for both regression and classification tasks.
- This single-layer perceptron can be seen as a two-stage regression or classification model, where in the intermediate layer so-called hidden features $(Z_1, \ldots, Z_M)$ are derived as a non-linear function of the inputs. These features are then used to model the targets $(Y_1, \ldots, Y_K)$.
- For regression, one typically sets $K = 1$ and employs only one output node Y_1 at the top. Nevertheless, as neural networks can handle multiple quantitative outputs at the same time, we deal with the more general case of K output nodes.
- For classification with K classes, the K output units model the probability of each class. In the training data, the $Y_k, 1 \leq k \leq K$ correspond to hot coded 0-1 variables for the k'th class.

Structure of a neural network III

The single-layer perceptron can be defined as follows:

Formal description

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, \ldots, M,$$
$$T_k = \beta_{0k} + \beta_k^T Z, k = 1, \ldots, K,$$
$$f_k(X) = g_k(T), k = 1, \ldots, K,$$

where $Z = (Z_1, \ldots, Z_M)$, and $T = (T_1, \ldots, T_K)$. The so-called activation function σ is often chosen to be the sigmoid

$$\sigma_{\text{sigmoid}}(\nu) := \frac{1}{1 + e^{-\nu}}$$

or the rectified linear unit (ReLU) function

$$\sigma_{\text{ReLU}}(\nu) := \max(0, \nu).$$

Structure of a neural network IV

Formal description (cont.)

- $f_k(X)$ is the model's prediction for the k'th output feature Y_k ($1 \leq k \leq K$) given the input vector X.
- These predictions are constructed as a linear combination of the hidden features $Z_1, \ldots, Z_M$ to which a final transformation g_k is applied.
- For regression, often no final transformation is performed, i.e., the functions g_k are chosen to be the identity function.
- For classification, usually the softmax function

$$g_k(T) := \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

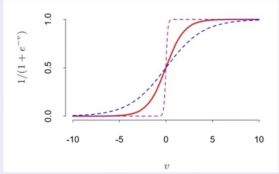
is employed. This is the same function as in the multilogit model.

Structure of a neural network V

- The units in the middle of the network are called hidden units, and the layer itself hidden layer, because the values $Z_1, \ldots, Z_M$ cannot be observed directly.
- Instead, these features are learned and modeled from the (observable) input features $X_1, \ldots, X_P$.
- The hidden features are then used to produce predictions on the (observable) output features $Y_1, \ldots, Y_K$.
- Note that when σ is the identity function, the entire model collapses to a linear model in the inputs. Consequently, a neural network can be thought of as a nonlinear generalization of the linear model.
- Deeper network architectures (multilayer perceptron) can be obtained by stacking multiple hidden layers on each other. Examples can be found in the application.
- An excellent graphical illustration of neural networks is provided in this **video**.

Structure of a neural network VI

Sigmoid function



Plot of the sigmoid function $\sigma(\nu) = \frac{1}{1+e^{-s\nu}}$. s is a scaling parameter that controls the activation rate, where the "standard" sigmoid function is obtained for $s = 1$ (red curve) and the blue and the purple curve are obtained for $s = 0.5$ and $s = 10$, respectively.

Source: Hastie, Tibshirani, and Friedman (2017).

Fitting Neural Networks

Fitting neural networks I

- The neural network model has a lot of unknown parameters that have to be fitted to the data.
- These parameters are often referred to as weights (the vectors $\alpha_m, m = 1, \ldots, M$ and $\beta_k, k = 1, \ldots, K$ on Slide 301) and biases (the parameters $\alpha_{0m}, m = 1, \ldots, M$ and $\beta_{0k}, k = 1, \ldots, K$ on the same slide). Sometimes, the weights and biases are simply referred to as weights.
- The parameters are chosen such that the model predictions fit the training data well.

intercepts

Fitting neural networks II

- For regression, one usually relies on the sum of squared errors as a measure of fit:

cost f'n → $$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2,$$ ← response, ← prediction

  where θ is the vector of *all* trainable parameters and N denotes the number of training examples.
- For classification, one often uses the cross-entropy (deviance):

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i).$$

  The corresponding classifier is $G(x) := \arg\max_k f_k(x)$. That is, the class to which the highest probability is assigned, is chosen as the prediction.

Fitting neural networks III

- The generic approach towards minimizing $R(\theta)$ is via gradient descent.
- In the setting of neural networks, gradient descent is often referred to as back-propagation as the gradient can easily be derived by the chain rule of differentiation. This can be done in a forward and backward sweep over the network.
- For details on the back-propagation equations, we refer to Hastie, Tibshirani, and Friedman (2017, p. 396).
- An excellent graphical illustration and explanation on the training error in neural networks and the back-propagation algorithm can be found in **this** and **that** video.

Fitting neural networks IV

- Calculating gradients based on the whole data sample as on Slide 306 and back-propagating them through the network is called batch learning. This can be costly in computational terms as datasets can be quite large.
- Therefore, one usually relies on the *stochastic* gradient descent (SGD) algorithm which is also referred to as mini-batch learning. The SGD algorithm only uses small random subsamples to update the network weights. As a consequence, the computational burden for each iteration step does not increase with the (total) number of training examples.
- The number of training examples in each mini-batch is referred to as batch size while a complete sweep over the entire dataset is referred to as one epoch. A neural network is typically trained over multiple epochs.

Select small Random Sub-samples to update the Network Weights.

Fitting neural networks V

Overfitting

- Neural networks involve numerous parameters and are therefore prone to overfitting at the global minimum of the cost function R (Slide 306).
- While there are many means to mitigate overfitting, e.g., by using smaller batch sizes that have a regularizing effect, there are some methods that explicitly address the problem of overfitting. In the application, we discuss and apply *dropout* and *early stopping*.
  - Dropout is a frequently used approach where non-output units are randomly removed from the network during training.
  - Early stopping addresses the question of how long to train a neural network by stopping training when the model performance starts to deteriorate on a validation set.

Already include the Validation Set in our Training.

Before we see this methods in work in the application, we quickly introduce convolutional neural networks.

Convolutional Neural Networks

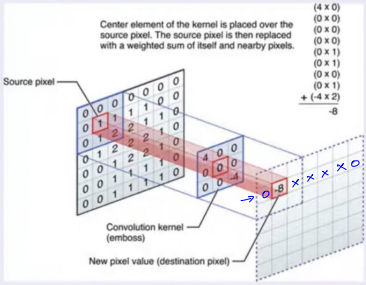
Convolutional neural networks I

Motivation

- Convolutional neural networks (CNNs) are a specialized kind of neural network for processing data with a grid-like topology (e.g., images)
- CNNs have been tremendously successful in practical applications.
- The term "convolutional" indicates that CNNs use a mathematical operation called convolution, which is a specialized kind of linear operation. CNNs are thus neural networks that use convolution instead of general matrix multiplication (like in the single-layer or multilayer perceptron) in at least one of their layers.

Convolutional neural networks II

The convolution operation



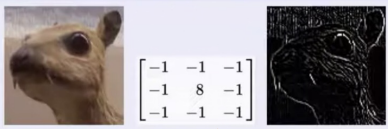
We are loosing two pixels as we go along with computation filling all the boxes.

Source: Graphical representation of 2D Convolution by Jon Hoffman licensed under CC BY 4.0.

Convolutional neural networks III

Convolution kernels are also often referred to as *filters* where each filter is meant to extract a specific feature.

Feature extraction (edge detection)



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

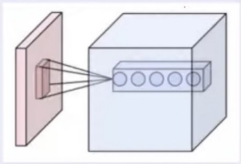
Convolutional Kernel

Source: Work by Michael Plotke licensed under CC BY-SA 3.0.

Convolutional neural networks IV

- Typically, many different filters are used in each layer of a convolutional neural network in parallel. This is done to extract various different features at the same time.
- As a result, the output of a convolutional layer has the structure of a cuboid.

Output structure of a convolutional layer with multiple filters



Source: Input volume connected to a convolutional layer by Aphex34 licensed under CC BY-SA 4.0.

Convolutional neural networks V

- Some further architectural components and considerations like, e.g., pooling layers are covered in the application.
- An extensive treatment of the CNN model can be found in Chapter 9 of the book by Goodfellow, Bengio, and Courville (2016).
- In the same book, you can also find information on other network architectures, e.g., recurrent neural networks, that are beyond the scope of this lecture.

We now turn to an application of both multilayer perceptron models and CNNs.

Neural Networks Application

Application I

- In most companies, a variety of documents only exists in paper form (e.g., invoices, bank statements, printouts of static-data, business cards, etc.)
- Of course, a scanned image can not be searched in its native state.
- It is therefore common to digitize printed texts to enable electronic editing, searching, compact storage, and online display. The retrieved "hidden text" behind an image can also be fed directly into further machine processes, e.g., for automatically managing invoices.
- The technique behind this is called optical character recognition (OCR).

Application II

- On the following slides we employ neural networks for the task of handwritten digit recognition (a classification problem with 10 classes).
- We start with a multilayer perceptron (MLP) model and discuss some regularization techniques.
- Finally, we also fit a CNN to the data.
- For the application, we rely on the MNIST database that is provided within the *keras* R-package. The database contains 60,000 training and 10,000 test examples of handwritten digits. More details on the dataset can be found **here**.

Application III

Keras

- We employ the keras R-package for fitting the neural networks.
- The package provides an interface for the open-source Python library Keras which in turn acts as an interface for the TensorFlow library. The syntax in the R-package is very similar to the Python syntax in the original library.
- Keras is one of the leading high-level neural network APIs.
- It has a focus on enabling fast experimentation. It is very user-friendly making it easy to quickly prototype deep learning models.
- Keras allows the training of neural networks on both CPUs and GPUs without changing the code.
- It supports arbitrary network architectures and is therefore appropriate for building essentially any deep learning model.

Application IV

- Before we can start to work with the data we have to install the (Python) Keras and TensorFlow backend first.
- This can mainly be done from within R. During the process, you might be asked which TensorFlow version you want to have installed (specifying "default" yields the CPU version which we will use here). Please note that installing Python Keras via the keras R-package seems not to work on the RStudio-Servers ... so please use your own computer.
- Before executing the following lines of code, please install anaconda with the default settings from **here**.

Installing Keras and TensorFlow backend

```
1 library(keras)
2 ?install_keras
3 install_keras() # this will install miniconda and several
      python-packages (keras, numpy, etc.)
```

Import and preprocess data I

digitized image data

Import data

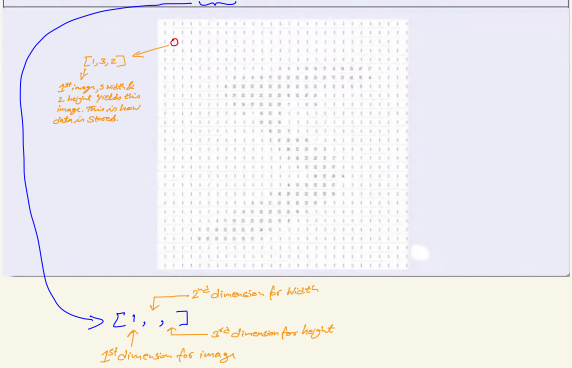
```
1  library(keras)
2
3  mnist<-dataset_mnist() # downloads the MNIST database
4  object.size(mnist)/1000000
5      219.8 bytes  #size in MB
6
7  str(mnist)
8      List of 2
9       $ train:List of 2
10       ..$ x: int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0  ...
11       ..$ y: int [1:60000(1d)] 5 0 4 1 9 2 1 3 ...
12      $ test :List of 2
13       ..$ x: int [1:10000, 1:28, 1:28] 0 0 0 0 0 0 0 ...
14       ..$ y: int [1:10000(1d)] 7 2 1 0 4 1 4 9 ...
```

Import and preprocess data II

Visualize data

first observation in dataset. A matrix of numbers

```
1 View(mnist$train$x[1,,]) # 28x28 grayscale image
```



Import and preprocess data III

Visualize the data (cont.)

```
1 mnist$train$y[1] #check image label
2     5   <- response/output of 1st observation.
3 image((mnist$train$x[1,,]),useRaster = TRUE, axes=FALSE, col
      =gray.colors(10, rev=TRUE))
```

Import and preprocess data IV

Create training and test set

The data are stored in a three dimensional array (image x width x height), see Slide **319**. To be fed into a MLP the matrix has to be flattened, that is, transformed into a vector. Additionally, we transform the gray-scale values (currently between 0 and 255) to the range $[0, 1]$.

```
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

# reshaping the data
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
# rescaling the data
x_train <- x_train / 255
x_test <- x_test / 255
```

Import and preprocess data V

Create training and test set (cont.)

The y data (y_train, y_test) are integer vectors with values ranging from 0 to 9. For training, we one-hot encode the two vectors into binary class matrices. This is done via the Keras *to_categorical()* function.

```
str(y_train) #structure before transformation
    int [1:60000(1d)] 5 0 4 1 9 2 1 3 1 4 ...

y_train <- to_categorical(y_train, 10)
str(y_train) #now a matrix
    num [1:60000, 1:10] 0 1 0 0 0 0 0 0 0 0 ...

y_test <- to_categorical(y_test, 10)
```

5 0 4 1 9 2 ...
No Yes No No No No : Are they zero
0 1 0 0 0 0 : Encoding in binary

Multilayer Perceptron - Application

Categorising hand written digits 0–9 using MLP (Neural Network)

MLP I

Building a first model

- The core data structure in Keras is a model. The simplest type of model is a sequential one where (potentially different kinds of) layers are stacked sequentially on each other.
- We start with defining a sequential model via *keras_model_sequential()* and subsequently add layers to it.
- Instead of the object-oriented syntax in the Python Keras library (*model.add()*), the R-package uses the pipe operator (%>%) that we are already familiar with from the *dplyr* R-package.
- Even shallow models can exhibit hundreds of thousands of parameters. Therefore, be careful of overfitting!

MLP II

Building a first model (cont.)

```
model1 <- keras_model_sequential() #define a sequential
    model

# add one hidden layer with 256 neurons and the relu
    activation function
# The input_shape corresponds to the length of the flattened
    images (28*28=784)
model1 %>%
    layer_dense(units=256, activation="relu", input_shape=c
        (784)) %>%
    layer_dense(units=10, activation=''softmax'') #output
        layer with one neuron for each class reflecting the
        probability for each digit
```

Hidden layer
output layer

MLP III

Building a first model (cont.)

```
# printing a model summary
summary(model1)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param
=================================================================
dense (Dense)                (None, 256)               200960
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
_________________________________________________________________
```

lots of parameters, hence bound to suffer from Overfitting

MLP IV

Building a first model (cont.)

Next, we compile the model from the previous slides with an appropriate loss function, optimizer, and error metric:

```
model1 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
```

MLP V

Building a first model (cont.)

Now, we can train the model. The results from the training steps are saved in the history object:

```
history <- model1 %>% fit(
  x_train, y_train,
  epochs = 50,      #epoch = one sweep over the whole dataset
  batch_size = 64, #number of images processed after which
      the parameters are updated
  validation_split = 0.2 #to provide an indication of the
      generalization performance of the model
)
```

20% Validation
80% train

MLP VI

Evaluating the model

```
# Evaluate model1 on the test set:
res1 <- model1 %>% evaluate(x_test, y_test)
print(res1) # prints categorical_crossentropy loss and
    classification accuracy (the selected loss function and
    error metric specified in the compile-stage)
            loss  accuracy
       0.1864205 0.9816000

# The model overfits in the training set:
print(model1 %>% evaluate(x_train, y_train))
             loss   accuracy
       0.04544459 0.99624997
```

Very Accurate but this could be due to overfitting in training set. We need to see if it generalise well for new data.

MLP VII

Making predictions from the model

```
predictions<-model1%>%predict_classes(x_test)
predictions[1] #prediction for first image in the test set
     7
y_test[1,] #true value
     0 0 0 0 0 0 0 1 0 0
```

Binary value for 7.

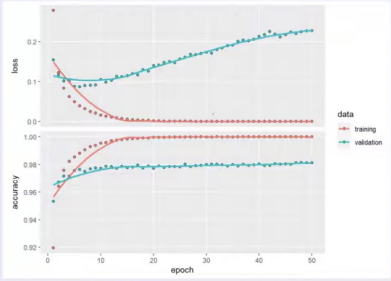
Prediction matches with True Value.

MLP VIII

Visualization of the training progress

```
plot(history) #overfitting!
```

in Training set due to more parameters than observations.



Loss increases with every epoch for Validation set. So, the Model doesn't generalise well for New data.

Multilayer Perceptron : Overfitting

MLP IX

Visualization of the training progress (cont.)

- It is obvious from the previous slide (and by comparing the accuracy (and loss) in the training and in the test set) that our first model overfits the data. The model ends up memorizing the training sample (with its more than 200,000 parameters) and does not generalize well to previously unseen data.
- While the loss continuously decreases on the training set, it starts increasing from epoch 10 on the validation set (20 % of the training sample randomly selected via the validation_split parameter).
- As a consequence, the classification accuracy on the validation set does also not further increase (as opposed to the accuracy on the training set).
- A possible solution to overfitting in neural networks are regularization techniques such as dropout and early stopping.

MLP X

Dropout

- proposed by Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov (2014)
- powerful regularization method applicable to a broad family of models
- computationally inexpensive
- frequently used in the current literature

MLP XI

Dropout (cont.)

- Dropout trains an ensemble of subnetworks of a given neural network.
- Therefore, non-output units are randomly removed from the network. This is typically achieved by multiplying the outputs of the respective neurons with zero. For each mini-batch, a different subsample of hidden units is used.
- Calculating gradients and backpropagating these through the network continues as usual.

MLP with dropout I

We now continue by adding a dropout layer to model1. The dropout rate specifying the percentage of neurons excluded per mini-batch serves as a hyperparameter to the training process. Here, we choose a dropout rate of 50 %.

Specifying a regularized model

```
model2 <- keras_model_sequential()
model2 %>%
  layer_dense(units = 256, activation = 'relu', input_shape
    = c(784)) %>%
  layer_dropout(rate = 0.5)  %>%
  layer_dense(units = 10, activation = 'softmax')
```

One extra layer added this time.

MLP with dropout II

Model summary of the regularized model

```
summary(model2)
    Model: "sequential_1"
    ________________________________________________
    Layer (type)          Output Shape      Param
    ================================================
    dense_2 (Dense)       (None, 256)       200960
    ________________________________________________
    dropout (Dropout)     (None, 256)       0
    ________________________________________________
    dense_3 (Dense)       (None, 10)        2570
    ================================================
    Total params: 203,530
    Trainable params: 203,530
    Non-trainable params: 0
    ________________________________________________
```

Note that the model relies on the same number of parameters as before.

MLP with dropout III

We continue as before ...

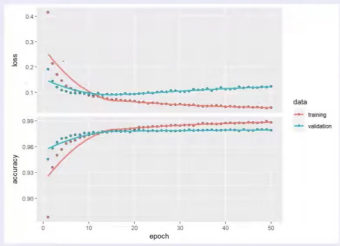
Compiling and fitting the regularized model

```
model2 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

history <- model2 %>% fit(
  x_train, y_train,
  epochs = 50,
  batch_size = 64,
  validation_split = 0.2
)
```

MLP with dropout IV

Visualizing the training progress of the regularized model

```
plot(history)
```



loss in Validation set increases but Not as bad as last time (model 1)

MLP with dropout V

- Accuracy in the validation set (slightly) increases with the number of epochs while the loss over the validation set only slightly increases → overfitting is not a (major) issue any more.
- While the accuracy of the regularized model (about 99 %) is lower in the training set (compared to the original model), it is higher (98.2 %) in the test set → generalization performance has improved.
- This is also reflected in a lower loss over the test set (0.111 for the regularized vs. 0.186 for the original model).

Hence, it a Good Way to regularized a Neural Network & to prevent from Overfitting.

MLP XII

Early stopping

- addresses the question of how long to train a neural network
- Two little training might lead to underfitting while too much training might cause overfitting (bad generalization to the test set).
- Early stopping proposes a compromise to this problem by stopping training at the point when performance on a validation set starts to degrade.
- simple, effective, and widely used

Multilayer Perceptron : Deep Model

Deep model I

- Now that overfitting is no longer a (major) issue, we can fit a deeper model to the data.
- Therefore, we add two additional hidden layers to the previous model where we reduce the number of neurons by a factor of two in each consecutive layer.
- Again, we apply dropout (with a dropout rate of 50 %) to each hidden layer.

Deep model II

Specifying a deep model

```
model3 <- keras_model_sequential()
model3 %>%
  layer_dense(units = 256, activation = 'relu', input_shape
      = c(784)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 10, activation = 'softmax')
```

Deep model III

Model summary

```
summary(model3)
    Layer (type)          Output Shape      Param
    ================================================
    dense_4 (Dense)       (None, 256)       200960
    dropout_1 (Dropout)   (None, 256)       0
    ------------------------------------------------
    dense_5 (Dense)       (None, 128)       32896
    dropout_2 (Dropout)   (None, 128)       0
    ------------------------------------------------
    dense_6 (Dense)       (None, 64)        8256
    dropout_3 (Dropout)   (None, 64)        0
    ------------------------------------------------
    dense_7 (Dense)       (None, 10)        650
    ================================================
    Total params: 242,762
    Trainable params: 242,762
    Non-trainable params: 0
```

Deep model IV

We continue as before ...

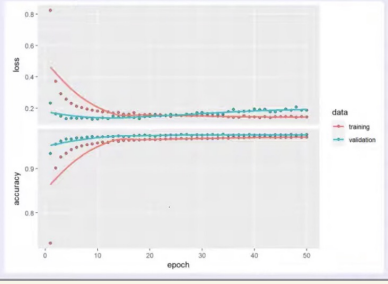
Compiling and fitting the model

```
model3 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

history <- model3 %>% fit(
  x_train, y_train,
  epochs = 50,
  batch_size = 64,
  validation_split = 0.2
)
```

Deep model V

Visualizing the training progress

```
plot(history)
```



Deep model VI

Evaluating the model

```
# Evaluate the classification accuracy on the test set
res3 <- model3 %>% evaluate(x_test, y_test)
print(res3)
      loss  accuracy
 0.1864311 0.9768000
```

Multilayer Perceptron : Early Stopping

Early stopping I

- Finally, we fit a model with early stopping, see Slide **335**.
- Therefore, we rely on the previous model with three hidden layers.
- For illustrative purposes we reduce regularization by setting the dropout rate to 30 %. Apart from this, the model specification is the same as on Slide **343**. (model3)

Early stopping II

Compiling and fitting the model

```
# Compiling is the same as for the previous model:
model4 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy'))

# Training with early stopping is performed via introducing
    a so-called callback monitoring the accuracy on the
    validation set:
history <- model4 %>% fit(
  x_train, y_train,   epochs = 50, batch_size = 64,
      validation_split = 0.2,
  callbacks = list(callback_early_stopping(
    monitor = "val_accuracy",
    patience = 5,
    restore_best_weights = TRUE)))
```

Early stopping III

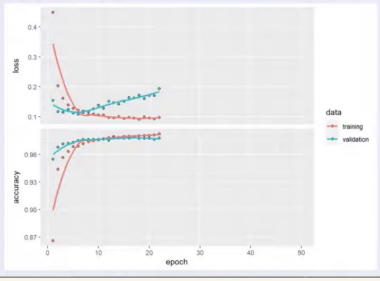
The parameters to the *callback_early_stopping()* function are explained **here**.

- monitor: quantity to be monitored
- patience: number of epochs with no improvement after which training is stopped
- restore_best_weights: whether to restore model weights from the epoch with the best value of the monitored quantity
- ....

Early stopping IV

Visualizing the training progress

```
plot(history)
```



Early stopping V

Evaluating the model

```
res4 <- model4 %>% evaluate(x_test, y_test)

print(res4)
      loss  accuracy
   0.1458071 0.9781000
```

The classification accuracy has (slightly) improved compared to the previous (deep) model. However, to assess whether the improvement is due to early stopping, the lower dropout-rate, or just happened by chance one would have to perform further analyses.

Early stopping VI

model3 & model4

Evaluating the model (cont.)

- Both deep models do not achieve a better performance than the shallow model with dropout. Deeper models are not better per se.
- Neural networks need to be carefully specified and trained. Deeper models might perform better with more training examples, with a different dropout rate, different numbers of neurons, layers, etc.
- On the following slides, we consider a different network architecture, namely CNNs. This type of network is especially suited for data with a grid-like structure such as images.

Convolutional Neural Networks Application

CNN I

- For the MLP models we flattened the input data, that is, we transformed the 2D images into vectors of gray-scale values.
- The CNN, however, exploits the grid-like structure of the images. Therefore, we need the training and test set in a different structure.

Preprocessing the data

```
x_train <- mnist$train$x/255 # grayscale values in [0,1]
x_test  <- mnist$test$x/255
dim(x_train)
     60000    28    28      -> 60000 images, 28 Pixels in height & width
dim(x_train)<-c(dim(x_train),1) #the CNN takes images in
    three dimensions, where the last dimension typically has
     three values (the RGB channels). As we have gray-scale
    images, we only need one channel. Adding one more dimension.
dim(x_train)
     60000    28    28     1    -> 60000 images, 28 Pixels, 28 Pixels, 1 RGB Channel
dim(x_test)<-c(dim(x_test),1) #same transformation for test
    set
```

CNN II

- CNNs can automatically learn a large number of filters (32 in our case in the first convolutional layer) in parallel.
- Each filter provides highly specific features that can be detected anywhere on the input images.
- In this example, we apply filters of size 3x3.
- The following lines of code specify the whole model as sequential and add the first convolutional layer.

Adding the first convolutional layer

```
modelCNN <- keras_model_sequential() %>%
  layer_conv_2d(
    filters=32,
    kernel_size=c(3,3),
    activation="relu", input_shape=c(28,28,1))
```

CNN III

- The output feature map (obtained from the various filters) is sensitive to the location of the features in the input.
- Pooling layers are an approach to down sample feature maps. They summarize the presence of features in patches of the feature map.
- Common pooling methods are average and maximum pooling. Average pooling summarizes the average presence of a feature while maximum pooling captures the most activated presence of a feature ("cat present in the respective part of the image or not").
- In our example, we use maximum pooling over patches of size 2x2. This reduces each 26x26 feature map obtained by the convolutional layer to a 13x13 map. -> because in each dimension we are loosing 2 pixels.

Adding a pooling layer

```
modelCNN <- modelCNN %>%
    layer_max_pooling_2d(pool_size=c(2,2))
```

CNN IV

We now add another convolutional layer, this time with 64 filters for detecting more detailed features in the image, followed by another maximum pooling layer.

Adding another convolution and pooling layer

```
modelCNN <- modelCNN %>%
    layer_conv_2d(filters=64, kernel_size=c(3,3), activation
        ="relu") %>%
    layer_max_pooling_2d(pool_size=c(2,2))
```

CNN V

- To complete the model, we feed the outputs from the last convolutional layer (with attached pooling layer) into a dense layer to perform classification.
- The dense layer has the same structure as a layer in the MLP.
- Before the outputs from the convolutional layer can be fed into the dense layer, the 3D output has to be flattened to 1D.

Adding a dense layer

```
modelCNN <- modelCNN %>%
        layer_flatten() %>%
        layer_dense(units=10, activation="softmax")
```

CNN VI

The whole model architecture looks like the following:

Model summary

```
summary(modelCNN)
  Layer (type)                 Output Shape              Param
  ============================================================
  conv2d (Conv2D)              (None, 26, 26, 32)        320
  max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
  ------------------------------------------------------------
  conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
  max_pooling2d_1 (MaxPooling2D) (None, 5, 5, 64)        0
  ------------------------------------------------------------
  flatten (Flatten)            (None, 1600)              0
  dense_2 (Dense)              (None, 10)                16010
  ============================================================
  Total params: 34,826
  Trainable params: 34,826
  Non-trainable params: 0
```

CNN VII

Dimensionality of the layers

- The input data consist of 28x28 gray-scale images.
- Applying filters of size 3x3 to them ("convolution") leads to the loss of two pixels in each dimension (if we do not apply some kind of padding). This is because a filter typically starts at the upper left corner of the image with the left-hand side of the filter sitting on the far left pixels of the image. The filter then moves across the image until the far right pixels of the image are reached. This yields 26x26 feature maps.
- As in the first convolutional layer we apply 32 different filters, the output dimensionality is 26x26x32.
- By maximum pooling with a patch size of 2x2 the dimensionality is reduced to 13x13x32.

CNN VIII

Dimensionality of the layers (cont.)

- The next convolutional layer applies 64 different filters of size 3x3 to the output (tensor) of the previous layer.
- Again, the input feature map (13x13x32) loses two "pixels" in the first two dimensions, yielding an output dimensionality of 11x11x64.
- By maximum pooling with patches of size 2x2, the number of "pixels" in the first two dimensions is essentially divided by 2 yielding the output dimension of 5x5x64.
- Flattening this output tensor yields a vector of length $5 \cdot 5 \cdot 64 = 1600$.

CNN IX

Number of parameters in each layer

- A very important feature of CNNs is parameter sharing. This is achieved by moving each filter over the picture thereby employing the same parameters at each location. As a consequence, the first convolutional layer of our network only employs 320 different parameters. This compares to 200,960 parameters in the first hidden layer of the previous MLP models!
- Each 3x3 filter involves $3 \cdot 3$ parameters $+ 1$ bias parameter. As we employ 32 filters, this yields $(3 \cdot 3 + 1) \cdot 32 = 320$ parameters.
- Pooling does not involve any trainable parameters. (The size of the patches over which pooling is performed is a hyperparameter.)

CNN X

Number of parameters in each layer (cont.)

- Each of the 64 filters of size 3x3 from the second convolutional layer is applied to all 32 feature maps from the previous layer at once.
- This yields $(3 \cdot 3 \cdot 32 + 1) \cdot 64 = 18,496$ parameters.
- Pooling and flattening do not involve (trainable) parameters.
- The last dense layer requires 10 weights for each of the 1600 neurons (for mapping to the 10 output neurons). Additionally, each output neuron requires a bias parameter. This yields $(1600 + 1) \cdot 10 = 16,010$ parameters.
- In total, we obtain 34,826 parameters which are still substantially less parameters than are employed in the one hidden layer MLP model (203,530 parameters).

CNN XI

- Compiling and training the model is performed analogously to the MLP models.
- As fitting the convolutional network is computationally very expensive, we only perform 10 epochs.
- Our University Computing Center provides a GPU cluster which is suitable for fitting large (or many) neural networks. The GPU cluster features 6 nodes with 4 Tesla V100 GPUs and 512GB RAM as well as 23 nodes with 8 RXT2080TI and 512GB RAM.

CNN XII

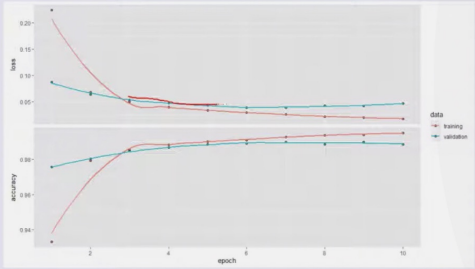
Compiling and training the CNN

```
modelCNN %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = c("accuracy")
)

history <- modelCNN %>% fit(
  x_train, y_train,
  epochs = 10, #train the model only for 10 epochs
  batch_size = 64,
  validation_split = 0.2 #to visualize the generalization
      performance during training
)
```

CNN XIII

Visualization of the training progress

```
plot(history)
```



CNN XIV

Evaluating the classification performance

```
resCNN <- modelCNN %>% evaluate(x_test, y_test)
print(resCNN)
           loss  accuracy
      0.0330491 0.9899000
```

- The CNN provides a higher forecasting accuracy on the test set than any of the MLP models.
- Classification accuracy could probably be improved by further tuning the model specification and by increasing the number of epochs. (e.g. 50 epochs)
- This is, however, beyond the scope of this lecture.

Regulation : Regtech

Regtech I

To comply with law & Regulations

Definition

The use of AI & ML **in the management of regulatory processes** for the financial industry. Main functions are **regulatory monitoring, reporting, compliance**.

Examples:

- Anti-fraud and risk management for digital transactions (**IdentityMind Global**)
- Management of consent for customer service data (**Trunomi**)

Regtech in Numbers [2] III

- Dominated by start-ups; 70% of firms are younger than 5 years
- 44,000 people employed globally
- About $4.9bn annual revenue
- About $9.7bn raised in external funding

Market Environment

Market and regulatory environment are rated to be **generally favourable** to Regtech companys by vendors in the sector. The **pace of regulatory changes has increased** after the Financial Crisis 08/09, expensive punishment for non-compliance with regulatory rulings lead to a surge in the demand for automated and reliable methods.

[2] The Global RegTech Industry Report by Cambridge Center for Alternative Finance and supported by EY Japan

Regtech - Clients [3] V

The Cambridge Center for Alternative Finance in collaboration with EY has surveyed 658 Regtech firms (about 80% of firms in the sector) with the following results concerning clientèle of Regtech companies:

- 89-94% of Regtech companies target banks as clients
- 61% target insurers
- and 57% serve FinTechs

However,

- 58% also have clients outside of the financial service sector such as advisory, consultancy and regulators

This means, there is **substantial overlap between what we consider Regtech and Suptech** companies. Where many firms can be subsumed under both definitions depending on the client they are currently serving.

[3] The Global RegTech Industry Report by Cambridge Center for Alternative Finance and supported by EY Japan

Regtech - Regulation VII

Regulators

Regulators have to adapt to ever new technology-enabled financial services, which may present a challenge especially for emerging and developing economies.

- **Regulatory Sandboxes** - formal programs that allow certain financial services and business models which are not yet fully compliant with existing laws. The aim is to learn about the opportunities and risks that a particular innovation carries and to develop evidence-based policy.
- **Innovation Hubs / Offices** - places where innovators and regulators meet to discuss solutions to sector challenges.
  - **Example: TechSprints** from the UK Financial Conduct Authority (FCA)
    - So far 7 TechSprints since 2016 with an 8th planned in October 2021
    - 2-day events including industry and innovators
    - past focus areas were e.g. (1) regulatory reporting; (2) financial service and mental health; (3) anti-money laundering

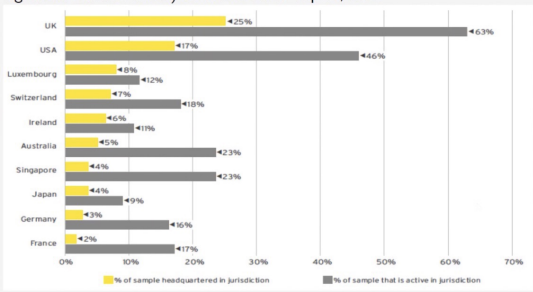
Regulator Sandbox (German Regulator) came up with this. eg: crypto.

Regtech - Market Segments II

- **Profiling and Due Diligence:** Collect or integrate data from multiple internal and external sources. Aims to profile an entity, confirm their identity, or categorize them e.g. according to regulatory requirements.
- **Reporting and Dashboards:** Collect or integrate data from multiple internal sources. Aims to build standardized reports for management or compliance purposes.
- **Risk Analytics:** Collect or integrate data from multiple internal and external sources. Aims to assess the risk of fraud, market abuse, or other misconduct at the transaction level.
- **Dynamic Compliance:** Using Machine Learning methods to facilitate and monitor regulatory changes, ensuring flexible adaptation of policies and controls.
- **Market Monitoring:** Collect or integrate data from multiple external sources. Aims to match market-level adverse outcomes to regulatory or business rules. For example, poor product performance, market manipulation, etc.

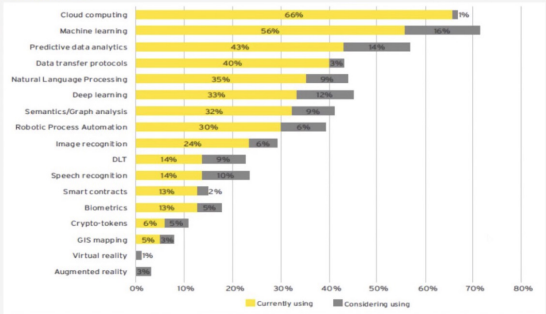
Regtech - Markets IV

Top 10 RegTech markets, by % of firms present (headquarters or significant market share) from the CCAF report, 2019



Source: CCFA Report.

Regtechs - Technologies and Tools used by Regtech companies VI



Source: CCFA Report.

Suptech And Systemic Risk (AI using by Regulators)

Suptech I

Definition

A sub-discipline of Regtech focussing on innovative technologies such as Big Data and AI **supporting financial authorities in their work**. The focus is on **misconduct analysis, reporting, data management**.

Examples:

- Collection and management of detailed data on loans in the euro area (**ECB**)
- Accelerating Suptech solutions and prototyping (R^2A)

Suptech - Survey II

Survey

The financial stability institute (FSI) surveyed 39 financial authorities from 31 jurisdictions worldwide about the implementation of suptech strategies.

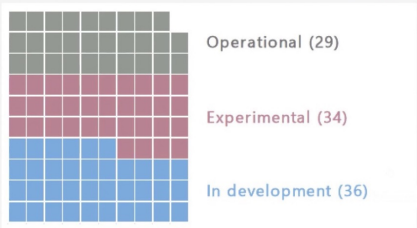
- two types of approaches were identified (which can also be combined)
  1. **Specific Suptech Roadmaps** based on particular needs of a department. The approach tends to be more experimental.
  2. **Institution-wide Digital Transformation and Data-driven Innovation programs (DT&DI)** is a broader approach encompassing large parts of the authority

Suptech - Focus areas III

Source: FSI Report.

Suptech - Conclusion IV

- Suptech is still in its infancy, but is gaining momentum.
- In 2019, most suptech solutions were still experimental or in a developmental stage.
- Authorities are willing to explore the area, yet funding and infrastructure may be problematic.



Source: FSI Report.

Systemic Risks due to AI/ML in finance V

Drivers of systemic risk, when using AI in regulatory functions:

- **AI is unable to reason about events it has not yet been exposed to**. Humans can draw on a broad range of prior experience and imagination, while AI can only extrapolate from what it has seen before.
- **We do not know how AI makes decisions**, it is too complex for us to follow. (less transparency causes uncertainty to investors & debtors)
- **AI is more likely to amplify current cycles** (pro-cyclicality) than human regulators. Automation favours homogenous methodologies and standardization
- The high predictability and transparency of **AI will enable individuals to bet against it**.

Exogenous risk VI

Exogenous Risk

Exogenous risk is caused by events from the outside. Much like an asteroid falling on London. It is easy to measure.

- AI is suited for the evaluation and management of exogenous risk
- It uses large data-sets, well-established statistical methods, and repeated events to train.
- Therefore, it is well suited for micro-regulation and internal risk management.

Tighten Regulation in crisis & loosen Regulation when market is booming leading to another crisis.

Endogenous risk VII

Endogenous Risk

Endogenous risk is caused by events from within the system. It starts when individual entities within the system stop acting independently but synchronize their behavior. It is very difficult to measure.

AIs are trained with various games for situations including interactions between entities...

- **Full information games:** In Chess, all possible moves can be known. Learning Algorithms and Deep Neural Networks succeed in these games, they assume their opponent is their clone (i.e., knows the same). They do not prepare for endogenous risk situations.
- **Incomplete information games:** In Poker, the opponents cards are unknown. AI performs worse than human players, as it cannot theorize about the opponent's intentions.
- **Cooperative Games**: Success is even more difficult for AI, when cooperation leads to multiple local optima (e.g. Diplomacy).

AI/ML & Ethics

Ethical Considerations [5] I

Ethical risks at different levels...

- data
  - training data must be free of biases and include all relevant types of stimuli (e.g., racial or sexist biases)
  - AI should only be deployed in environments it has been trained in (i.e., no cross-use for a different purpose)
  - privacy laws must be taken into account when collecting and processing data
- algorithm
  - unethical coding influenced by biases of the developers (most developers are white males)
  - AI must be controlled for emerging biases during the life cycle
- business use
  - the purpose of using AI should be ethical in the first place
  - unintended impact of AI can be unethical

[5] Study by Thomaz et al. 2021

Ethical Considerations II

Racism and Sexism in AI

Machines in themselves have no bias, but the learning data sets and algorithms are most likely as biased as our current society is. Most of the AI developers are white males, lacking the perspectives of minorities. Consequently, **existing biases will be amplified, if we are not actively working against them**.

Organisations working against biased algorithms

- American Civil Liberties Union (**ACLU**) - fights in a wide variety of issues, but has exposed *Amazon's Rekognition* as racially biased
- Algorithmic Justice League (**AJI**) - founded in 2016 with the mission to raise public awareness to biases in AI
- several independent researchers

Ethical Considerations III

Practical Recommendations

- implementation of a general statement on the firm's intention for AI ethics [a]
- should be an extension to the firms already existing mission or purpose statement
- implementation of internal application-specific design plans
- regular auditing of processes (flagging risks and concerns)
- records of decisions concerning ethical trade-offs for transparency

[a] Study by Thomaz et al. 2021

Regulation of AI/ML in finance IV

- Regulation of artificial intelligence is still in its infancy
- Artificial intelligence and machine learning not yet clearly defined
- European Commission has stated in its Digital Finance Strategy that it intends to clarify, together with the European Supervisory Authorities, by 2024 at the latest whether and how existing financial market regulation should be applied to the use of Big Data and Artificial Intelligence (BDAI)
- BaFin has proposed several principles for the general use of algorithms in decision-making processes of financial firms[6]
  - Principles represent preliminary considerations for minimum supervisory requirements for the use of artificial intelligence

[6] BaFin: Big Data und künstliche Intelligenz: Prinzipien für den Einsatz von Algorithmen in Entscheidungsprozessen

Regulation of AI/ML in finance V

Definition of AI according to BaFin

Artificial Intelligence is a combination of large amounts of data (Big Data), computing resources, and machine learning (ML).[a]

In machine learning, computers are given the ability to learn from data and experience on the basis of special algorithms. Compared to rule-based methods, learning takes place without the programmer specifying which results are to be derived from certain data constellations and how.

[a] BaFin (2018): Big Data trifft auf Künstliche Intelligenz: Herausforderungen und Implikationen für Aufsicht und Regulierung von Finanzdienstleistungen, URL: https://www.bafin.de/dok/10985478

Regulation of AI/ML in finance VI

Algorithms

Algorithms are rules of action that are usually integrated into a computer program and solve an (optimization) problem or a class of problems.

In addition to the distinction according to the type of algorithms (how is the problem technically solved), applications of ML can also be differentiated according to result types (the basic distinction between classification, regression, and clustering) and data types (e.g., for text, language, and image data).

Regulation of AI/ML in finance VII

Overriding Principles

Clear resposibility of the management

- Management is responsible for company-wide strategies and guidelines or policies for the use of algorithm-based decision-making processes.
- Potentials of such processes as well as their limits and risks should be taken into account and clearly stated.
- Company-wide strategy for the use of algorithm-based decision processes should also be reflected in the IT strategy.

Regulation of AI/ML in finance VIII

Overriding Principles

Adequate risk and outsourcing management

- Establishment of a risk management system adapted to the use of algorithm-based decision-making processes.
- If applications are sourced from a service provider, management must also have an effective outsourcing management.
- Responsibility, reporting, and control structures must be clearly defined.
- When establishing adequate risk management, one needs to consider the risks of an algorithmic decision-making process → risk mitigating measures and processes should start exactly where a risk originates according to the polluter pays principle.

Regulation of AI/ML in finance IX

Overriding Principles

Bias prevention

- Avoidance of a bias, i.e., the systematic distortion of results, in algorithm-based decision-making processes
- Bias must be avoided in order to:
  - be able to make business decisions based on results that are not systematically distorted,
  - exclude the possibility of systematic bias-based discrimination against individual customer groups and the resulting reputational risks.
- In accordance with the polluter pays principle, the risk of bias must be identified where it can arise, it must be analyzed and either eliminated or at least mitigated

Regulation of AI/ML in finance X

Overriding Principles

Exclude differentiation prohibited by law

- For some financial services, it is also stipulated by law that certain characteristics may not be used for differentiation, i.e., for calculating risk and prices.
- Danger of discrimination exists if these characteristics are replaced by an approximation.
- Would be associated with increased reputational risks and legal risks.
- Companies should establish (statistical) verification processes that exclude discrimination.

Ethics of AI in finance XI

EIOPA's Group on Digital Ethics (GDE) AI governance principles

1. Human oversight
2. Robustness and performance
3. Data governance and record keeping
4. Transparency and explainability
5. Fairness and non-discrimination
6. Principle of proportionality

EIOPA's GDE AI governance principles XII

Human oversight

Insurance firms should establish adequate levels of human oversight taking into account the impact of specific AI use cases and other governance and control measures in place.

- Selection of the level of human oversight should be proportionate to the nature, scale, and complexity of the risks inherent to the specific AI use case.
- Different roles and responsibilities for the staff involved in AI processes should be clearly defined in policy documents.

EIOPA's GDE AI governance principles XIII

Robustness and performance

Insurance firms should assess and monitor the performance of AI systems on an on-going basis and taking due consideration of their limitations and potential shortcomings.

- Performance metrics should be adapted to the objective pursued and the nature of the data used.
- Sound data management is key to ensure the performance of AI systems.
- AI systems should produce stable outcomes over time.
- Insurance firms should develop resilient IT systems and infrastructures.

EIOPA's GDE AI governance principles XIV

Data governance and record keeping

Insurance firms should adapt the data governance and record keeping measures to the impact of specific AI use cases.

- Data used in AI models should be accurate, complete, and appropriate.
- Sound data governance should be applied throughout the AI model lifecycle.
- Data used in AI models should be handled and stored in a secure manner.
- Appropriate records of the data and the modelling methods should be kept to ensure their reproducibility/traceability.

EIOPA's GDE AI governance principles XV

Transparency and explainability

Insurance firms should adapt the types of explanations to specific AI use cases and to the recipient stakeholders.

- Firms should adapt their explanations to the different types of stakeholders.
- Firms should strive to use explainable AI models, in particular in high-impact AI use cases.
- Data used need to be transparently communicated.

EIOPA's GDE AI governance principles XVI

Fairness and non-discrimination

- Sound and transparent governance processes are key to ensure fairness and non-discrimination.
- Insurance firms should conduct their business in a fair manner when using AI and make reasonable efforts to take into account the outcomes of AI systems.
- Consumers not willing to share very personal and sensitive data are not strictly necessary for risks assessments should still have access to affordable insurance coverage.
- Insurance firms should respect the principle of human autonomy by developing AI systems that support consumers in their decision-making process and avoid unfair nudging practices.

EIOPA's GDE AI governance principles XVII

Principle of proportionality

Insurance undertakings should establish the necessary governance measures that are proportionate to the nature, scale, and complexity of their operations.

- AI use case impact assessment and the governance measures should be proportionate to the potential impact of a specific AI use case on consumers and/or insurance firms.
- Insurance firms should then assess the combination of measures put in place in order to ensure an ethical and trustworthy use of AI.

Literature XVIII

Aggarwal, C. C., and P. S. Yu (2008): *Privacy-Preserving Data Mining: Models and Algorithms*, vol. 34 of *Advances in Database Systems*. Springer, New York, NY and Heidelberg.

Dua, D., and C. Graff (2017): "UCI Machine Learning Repository," .

Goodfellow, I., Y. Bengio, and A. Courville (2016): *Deep Learning*. MIT Press, Cambridge, MA, USA.

Hastie, T., R. Tibshirani, and J. H. Friedman (2017): *The elements of statistical learning: Data mining, inference, and prediction*, Springer series in statistics. Springer, New York, NY, second edition, corrected at 12th printing 2017 edn.

Hull, J. C. (2020): *Machine Learning in Business*. Second Printing, Toronto, second edn.

James, G., D. Witten, T. Hastie, and R. Tibshirani (2013): *An Introduction to Statistical Learning*. Springer, New York, first edn.

Kuhn, M. (2020): *caret: Classification and Regression Training*,R package version 6.0-86.

Lesmeister, C. (2015): *Mastering machine learning with R: Master machine learning techniques with R to deliver insights for complex projects*, Community experience distilled. Packt Publishing, Birmingham and Mumbai.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014): "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, 15(56), 1929–1958.

Venables, W. N., and B. D. Ripley (2002): *Modern Applied Statistics with S*. Springer, New York, fourth edn., ISBN 0-387-95457-0.