Stochastic Machine Learning
01 - Intro Teil 3

Thorsten Schmidt

Abteilung für Mathematische Stochastik

www.stochastik.uni-freiburg.de
thorsten.schmidt@stochastik.uni-freiburg.de

WS 2020/21

Definition

A computer program learns from experience E with respect to tasks T, if its performance P improves with experience E.

This quite vague definition allows us to develop some intuition about the situation.

- **Experience** is given by an increasing sequence of observations, for example $X_1, X_2, \ldots, X_t$ could represent the information at time t. This is typically decoded in a **filtration**: a filtration is an increasing sequence of sub-σ-fields $(\mathscr{F}_t)_{t \in \mathcal{T}}$.
- The performance is often measured in terms of an **utility function**. For example the utility at time t could be given by $U(X_t)$ with an function U. U could of course depend on more variables. One could also look for the accumulated utility

$$\sum_{t=1}^{T} U(X_t).$$

One very simple learning algorithm is linear regression, a classical statistical concept. Here it arises as an example of **supervised learning**.

Example (Linear Regression)

Suppose we oberseve pairs $(x_i, y_i)_{i=1,\ldots,n}$ and want to predict y on basis of x. **Linear** regression requires

$$\hat{y}(x) = \beta x$$

with some weight $\beta \in \mathbb{R}$. We specify a loss function[1]

$$\mathsf{RSS}(\beta) := \sum_{i=1}^{n} \left(y_i - \hat{y}(x_i)\right)^2$$

and minimize over β.

One could choose $-$MSE as utility function. So how does the system **learn**?

[1] Given by the Residual Sum of Squares here.

The system learns by maximizing the utility, i.e. minimizing the MSE for each n. And additional data will lead to a better prediciton. We will later see that this is in a certain sense indeed optimal.

We use the **first-oder condition** to derive the solution letting $\boldsymbol{x} = (x_1, \ldots, x_n)$ and similar for $\boldsymbol{y}$,

$$\begin{aligned} & 0 = \partial_\beta (\boldsymbol{y} - \beta \boldsymbol{x})^2 = \partial_\beta (\boldsymbol{y}^2 - 2\boldsymbol{y}^\top \beta \boldsymbol{x} + \beta^2 \boldsymbol{x}^\top \boldsymbol{x}) \\ \Leftrightarrow \quad & 0 = -2\boldsymbol{x}^\top \boldsymbol{y} + 2\beta \boldsymbol{x}^\top \boldsymbol{x} \end{aligned}$$

such that we obtain

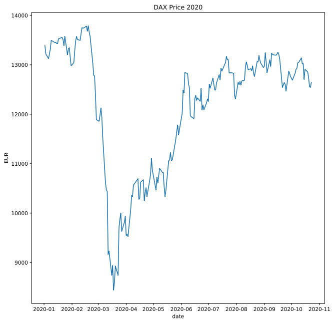
$$\hat{\beta} = (\boldsymbol{x}^\top \boldsymbol{x})^{-1} \boldsymbol{x}^\top \boldsymbol{y}.$$

Note that typically one considers affine functions of x without mentioning, i.e. one looks at functions $y = \alpha + \beta x$. This can simply be achieved with the linear approach by augmenting $\boldsymbol{x}$ by an additional entry 1.

- Of course many generalizations are possible:
- To higher dimensions: consider data vectors $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $i = 1, \dots, n$,
- To nonlinear functions: include $x_i^1, \dots, x_i^p$ into the covariates
- and many more.

Let us consider a linear regression in python.

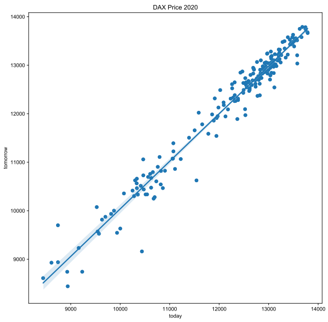
```
import yfinance as yf
import matplotlib.pyplot as plt

DAX = yf.Ticker('%5Egdaxi')
DAX_History = DAX.history(start="2020-01-01", end="2020-10-26")

plt.figure(figsize=(10,10))
plt.plot(DAX_History.index, DAX_History['Close'])

# Linear Regression example: regress tomorrow on today
x = DAX_History['Close'][:-1]    # without the last value
y = DAX_History['Close'][1:]     # without the first value

import numpy as np
from numpy import array
from sklearn.linear_model import Linea

model = LinearRegression()
x = array(x).reshape(-1,1)   # The lin
y = array(y).reshape(-1,1)
model.fit(x, y)  # values in model.int

# Give a very sophisticated plot
import seaborn as sns; sns.set_theme(c
ax = sns.regplot(x=x, y=y)
plt.show()
```



Could we improve this ? Suggestions ?

Let us consider a linear regression in python.

```
import yfinance as yf
import matplotlib.pyplot as plt

DAX = yf.Ticker('%5Egdaxi')
DAX_History = DAX.history(start="2020-01-01", end="2020-10-26")

plt.figure(figsize=(10,10))
plt.plot(DAX_History.index, DAX_History['Close'])

# Linear Regression example: regress tomorrow on today
x = DAX_History['Close'][:-1]    # without the last value
y = DAX_History['Close'][1:]     # without the first value

import numpy as np
from numpy import array
from sklearn.linear_model import Linea

model = LinearRegression()
x = array(x).reshape(-1,1)   # The lin
y = array(y).reshape(-1,1)
model.fit(x, y)  # values in model.int

# Give a very sophisticated plot
import seaborn as sns; sns.set_theme(c
ax = sns.regplot(x=x, y=y)
plt.show()
```



Could we improve this ? Suggestions ?

What is the difference to Statistics ?

In a statistical approach we start with a **parametric model**:

$$Y_i = \alpha + \beta x_i + \epsilon_i, \quad i = 1, \ldots, n$$

and assume that $\epsilon_1, \ldots, \epsilon_n$ have a certain structure (for example, i.i.d. and $\mathcal{N}(0, \sigma^2)$). The one can derive (see, e.g. Czado & Schmidt (2011)) **optimal estimators** for α and β. One can also relax the assumptions and gets weaker results.

So what ? What are the advantages of the statistical approach ?

One particular outcome is that we are able to provide **confidence intervals**, **predictive intervals** and **test** hypothesises.

Questions

- What is the definition of Machine Learning?
- Give examples
- Give surprising examples
- Derive the main equation of linear regression
- (do it in 1 dimension first - this goes back to Gauss)
- Write your own python code, providing a linear regression on your favourite stock
- Do this with your least favourite stock
- Can you regress two stocks on each other ?
- Can you predict better the value of the stock tomorrow ? (You can also research on this ...)