# example_fifa

April 23, 2020

## 1 Eksempel på multivariat data: FIFA-skillz

I dette eksempelet tar jeg for meg et datasett bestående av informasjon om alle spillere på FIFA 20, hentet fra denne linken. Antagelsen min er at mange ferdigheter er høyt korrelerte (er man god på langpasninger er man gjerne god på innlegg også, for eksempel), og dermed perfekt å anvende ulike metoder var multivariat analyse på.

### 1.1 Datasett

Datasettet består av litt over 18 000 rad, der hver rad inneholder egenskapene til en fotballspiller. Hver kolonne svarer til en bestemt ferdighet. Etter å ha gjort et utvalg av hvilke ferdigheter (kolonner) som er relevante å analysere, sitter vi igjen med et datasett i form av en matrise $X$ som skriker etter å bli analysert.

### 1.2 Avhengigheter og forberedende kode

Først importerer vi alle bibliotek vi trenger, forbereder plott, og definerer datasettet vårt. Her fjerner vi alle keepere fordi disse ikke har definert alle ferdigheter vi er interesserte i. Jeg har vært lite streng, og definert alle numeriske verdier som relevante. Dette betyr blant annet at absurde forklaringsvariabler som `sofifa_id` er med, men metodene som skal brukes burde uansett være robuste nok til å ignorere disse

[62]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
```

[2]:
```
# Lag penere, mer høyoppløselige plott enn seaborn gjør med␣
↪default-innstillinger
%config InlineBackend.figure_format = 'svg'
sns.set_context("notebook")
sns.set(style="ticks", font="Latin Modern Math")
```

[7]:
```
datapath = "../datasett/fifa-20-complete-player-dataset/players_20.csv"
all_player_data = pd.read_csv(datapath)
all_player_data.sort_values('overall');
```

```
player_data_without_gk = all_player_data[all_player_data.loc[:,␣
↪'player_positions'] != 'GK']
numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
player_data = player_data_without_gk.select_dtypes(include=numeric_dtypes)
player_data = player_data.dropna(axis='columns')
```

### 1.3 Variabler

Vi er interesserte i å forklare totale ferdigheter, `overall`, som funksjon av alle andre variabler. Disse blir da hhv. $y$ og $X$. Både `overall`, og `potential` droppes fra $X$, siden disse henger så tett sammen. Jeg splitter datasettet opp i et test- og et treningssett, half ´n half.

[142]:
```
player_data_train, player_data_test = train_test_split(player_data, test_size=0.
↪2, random_state=1997)
y = player_data_train.loc[:, 'overall'].to_numpy()
y_test = player_data_test.loc[:, 'overall'].to_numpy()
X = player_data_train.drop(['overall', 'potential'], axis='columns').to_numpy()
X_test = player_data_test.drop(['overall', 'potential'], axis='columns').
↪to_numpy()
n = len(y)
n_test = len(y_test)
```

[143]:
```
theta, r, _, _ = np.linalg.lstsq(X, y)
y_hat_train = np.matmul(X, theta)
y_hat_test = np.matmul(X_test, theta)
mlr_mse_train = r/n
mlr_mse_test = sum((y_hat_test - y_test)**2)/n_test
print(f"MSE for prediksjoner på treningssett: {mlr_mse_train[0]}")
print(f"MSE for prediksjoner på testsett: {mlr_mse_test}")
```

```
MSE for prediksjoner på treningssett: 4.705397891879798
MSE for prediksjoner på testsett: 4.867091022860787
```

Litt (men ikke særlig) avhengig av hvordan man deler opp i test-og treningssett ser vi at MSE for prediksjonene til denne estimatoren ligger på ca. 4,7 for treningssett, og 4,8 for testsett. Ikke spesielt imponerende, og heller ikke spesielt lovende når man tenker på at LS eksplisitt minimerer MSE for et gitt datasett.