Eksperimantiranje

Prvo ćemo importati potrebne pakete i fiksirati *random seed* kako bi kod bio reproducibilan.

In [ ]:
```
from enum import Enum
from time import sleep

import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm

random_state = 42
rng = np.random.RandomState(random_state)
```

Sljedeća funkcija služi generiranju uzorka jediničnih normalnih veličina x i y td.

$$y = \beta x + \varepsilon, \;\; \varepsilon \sim N(0, \sigma^2)$$
$$\mathrm{Cov}(x_i, x_j) = \sigma^2 > 0, \;\; i \neq j.$$

In [ ]:
```
def get_sample(beta: np.array, rho: float, sample_size: int, error_var: floa

    """Generates sample of X and y such that y = X*beta + eps, eps ~ N(0, sq

    Args:
        beta (np.array): Linear transformation vector.
        rho (float): Covariance between covariates.

    Returns:
        sample: generated sample.
    """

    N = beta.shape[0]
    cov = rho * np.ones([N, N])
    cov = cov + np.diag(1-rho * np.ones(N))

    N = beta.shape[0]
    X = rng.multivariate_normal(mean = np.zeros(N), cov=cov, size=sample_siz

    error = rng.normal(loc=0, scale=np.sqrt(error_var), size=X.shape[0])
    y = np.matmul(X, beta) + error

    return X, y
```

Brzinski *sanity check*: ako je $\rho = 0$, tada je

$$\mathrm{Var}(y) = \sum_{k=0}^{N} \beta_k^2 \mathrm{Var}(X_k) + \mathrm{Var}(\varepsilon),$$

što bi u slucaju $N = 3, \beta = (1, \dots, 1)$ moralo biti 4. Zaista, za dovoljno velik uzorak imamo da je uzoračka varijanca ≈ 4:

In [ ]:
```
N = 3
beta = np.ones(3)
rho = 0
sample_size = 1000000

X, y = get_sample(beta=beta, rho=rho, sample_size=sample_size)
print(f'>>> Var(y) = {y.var(ddof=1)}')
```

```
>>> Var(y) = 3.9966141119858167
```

Nastavljamo definiranjem funkcija za treniranje LS, PCR i PLS modela, pri čemu potonja dva kao parametar primaju i broj glavnih komponenti koje koriste.

In [ ]:
```
class Model(str, Enum):
    linreg = "linreg"
    pcr = "pcr"
    pls = "pls"

    @staticmethod
    def train(model_name: str, X: np.array, y: np.array, n_components: int |

        if model_name not in list(Model):
            raise ValueError(f'No such model. Available models are {list(Mod

        if model_name == Model.linreg:
            model = LinearRegression()

        elif model_name == Model.pcr:
            model = make_pipeline(PCA(n_components=n_components, random_stat

        elif model_name == Model.pls:
            model = PLSRegression(n_components=n_components)

        model.fit(X, y)

        return model
```

Jedan razuman način validacije naših modela bio bi da koristeći distribucije iz kojih smo generirali podatke izračunamo populacijski β pa za grešku modela uzmemo koliko se njegov koeficijent razlikuje od populacijskog, tj. ako je nas model dan s $y = \hat{\beta}x$, njegovu grešku možemo računati kao

$$\text{Err}(\text{Model}) = \|\beta - \hat{\beta}\|.$$

Međutim, kako je prilikom visoke korelacije kovarijata taj β "nestabilan", mi ćemo umjesto toga testirati naše modele na velikom testnom uzorku. Preciznije, prvo ćemo izgenerirati jako velik uzorak, zatim trenirati model na njegovom malom dijelu, a na ostatku izračunati $R2$, što ce nam biti primarna metrika za validaciju modela. Takav pristup ima dvije prednosti:

1. Metrike poput kvaratne greške i $R2$ su interpretabilnije od udaljenosti do stvarnog β.
2. Tako se stvari rade u praksi (jer ne znamo stvarne distribucije pa ni vrijednost populacijskog koeficijenta); istrenira se model na uzorku koji nam je dan, a zatim validira na testnom skupu pa ide u produkciju.

In [ ]:
```
SAMPLE_SIZE = 50_000
```

In [ ]:
```
def train_and_evaluate_all_models(X_train: np.array, X_test: np.array, y_tra

    """Trains LS, PLS and PCR models on X_train and y_test and evaluates the

    Args:
        n_components (int): Number of relevant components to use for predici

    Returns:
        dict: Contains model names with their respective R2 values.
    """

    score_dict = {}

    for model_name in list(Model):
        model = Model.train(model_name=model_name, X=X_train, y=y_train, n_c
        score_dict[model_name.value] = model.score(X_test, y_test)

    return score_dict
```

Donji primjer pokazuje kako kod "skoro" nezavisnih kovarijata PLS bolje predviđa nego PCR. To je i očekivano, pogotovo ako je broj komponenti puno manji od N jer tada PCA nužno gubi bitne informacije za predviđanje. S druge strane, PLS, rastavljajuci zavisnu varijablu skupa s nezavisnima, ne izgubi gotovo ništa te predviđa jednako dobro kao i linearna regresija, ali u puno manjoj dimenziji pa je stoga interpretabilniji od nje.

In [ ]:
```
N = 300
train_sample_size = 1000
n_components = 5
rho= 0.1

beta = rng.normal(loc=0, scale=25, size=N)
X, y = get_sample(beta=beta, rho=rho, sample_size=SAMPLE_SIZE)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=randc
```

In [ ]:
```
score_dict = train_and_evaluate_all_models(X_train=X_train, X_test=X_test, y
score_dict
```

Out[ ]:
```
{'linreg': 0.9999894545075303,
 'pcr': 0.005928099252595187,
 'pls': 0.9946321320636914}
```

Sad ćemo za razne parametre provesti eksperiment i podatke o R2 spremiti u datoteku *scores.csv*.

In [ ]:
```
SAMPLE_SIZE = 100_000

index_columns = ['train_sample_size', 'N', 'n_components', 'rho']
score_df = pd.DataFrame(None, columns=index_columns+[x.value for x in Model]

Ns = [500, 100, 50, 10, 5]
loader = tqdm(Ns)

N_train_sample_size_ratios = [10, 5, 3, 2, 1]
n_components_N_ratios = [0.01, 0.1, 0.25, 0.5]
rhos = [0.01, 0.1, 0.2, 0.5, 0.9, 0.99]

for N in loader:
    for rho in rhos:
```

```
        beta = rng.normal(loc=0, scale=25, size=N)
        X, y = get_sample(beta=beta, rho=rho, sample_size=SAMPLE_SIZE)

        for train_sample_size in [N*x for x in N_train_sample_size_ratios]:

            X_train, X_test, y_train, y_test = train_test_split(X, y, random

            for n_components in [int(N*x) for x in n_components_N_ratios]:

                if n_components == 0:
                    continue

                score_dict = train_and_evaluate_all_models(X_train=X_train,
                hparams_dict = dict(train_sample_size=train_sample_size, N=N
                new_row = hparams_dict | score_dict

                score_df.loc[len(score_df), :] = new_row

                loader.set_postfix(**new_row)

        score_df.to_csv('scores.csv', index=False)

score_df.to_csv('scores.csv', index=False)
```

```
100%|██████████| 5/5 [14:30<00:00, 174.03s/it, N=5, linreg=0.999, n_compone
nts=2, pcr=0.996, pls=0.997, rho=0.99, train_sample_size=5]
```