Exercise 5 – Evaluation I

Introduction to Machine Learning

Hint: useful `Python` *libraries*

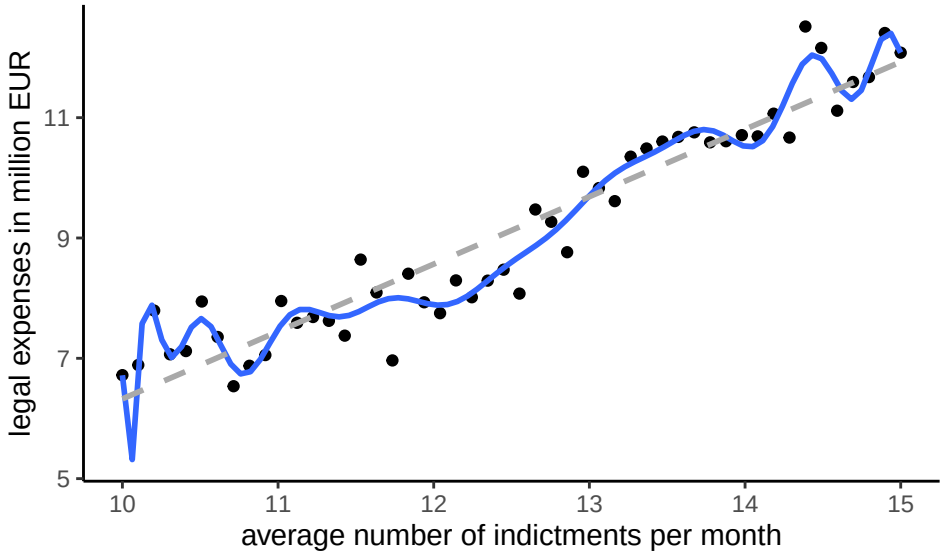
```
# Consider the following libraries for this exercise sheet:

# general
import numpy as np
import pandas as pd
import math

# plots
import matplotlib.pyplot as plt

# sklearn
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

Exercise 1: Evaluating regression learners

> Learning goals
>
> 1. Understand importance of using test error for evaluation
> 2. Discuss choice of performance metric in given situation

Imagine you work for a data science start-up and sell turn-key statistical models. Based on a set of training data, you develop a regression model to predict a customer's legal expenses from the average monthly number of indictments brought against their firm.

Due to the financial sensitivity of the situation, you opt for a very flexible learner that fits the customer's data ($n_{train} = 50$ observations) well, and end up with a degree-21 polynomial (blue, solid). Your colleague is skeptical and argues for a much simpler linear learner (gray, dashed). Which of the models will have a lower empirical risk if standard $L2$ loss is used?



Why might evaluation based on training error not be a good idea here?

Evaluate both learners using

i. mean squared error (MSE), and

ii. mean absolute error (MAE).

State your performance assessment and explain potential differences. Use the code below to create the training and test points.

R

```r
set.seed(123)
x_train <- seq(10, 15, length.out = 50)
y_train <- 10 + 3 * sin(0.15 * pi * x_train) + rnorm(length(x_train), sd = 0.5)
```

```r
data_train <- data.frame(x = x_train, y = y_train)
set.seed(321)
x_test <- seq(10, 15, length.out = 10)
y_test <- 10 + 3 * sin(0.15 * pi * x_test) + rnorm(length(x_test), sd = 0.5)
data_test <- data.frame(x = x_test, y = y_test)
```

Python

```python
np.random.seed(43)
x_train = np.linspace(10, 15, num=50)
y_train = 10 + 3 * np.sin(0.15 * math.pi * x_train)
y_train += np.random.normal(loc=0.0, scale=0.5, size=len(x_train))
data_train = pd.DataFrame({"y": y_train, "x": x_train})
np.random.seed(2238)
x_test = np.linspace(10, 15, num=50)
y_test = 10 + 3 * np.sin(0.15 * math.pi * x_test)
y_test += np.random.normal(loc=0.0, scale=0.5, size=len(x_test))
data_test = pd.DataFrame({"y": y_test, "x": x_test})
```

Exercise 2: Importance of train-test split

> Learning goals
>
> 1) Understand how strongly performance estimates can depend on data samples
> 2) Explain bias-variance trade-off of GE estimator for repeated sampling across different train/test splits

We consider the `CaliforniaHousing` data for which we would like to predict the median house value (`MedHouseVal`) from the median income in the neighborhood (`MedInc`).

R

Get data

```r
# Adapt to version available on sklearn following description from
# https://gist.github.com/bgreenwell/b1330460eec5acf1c81fae71902e331c

setwd(tempdir())
```

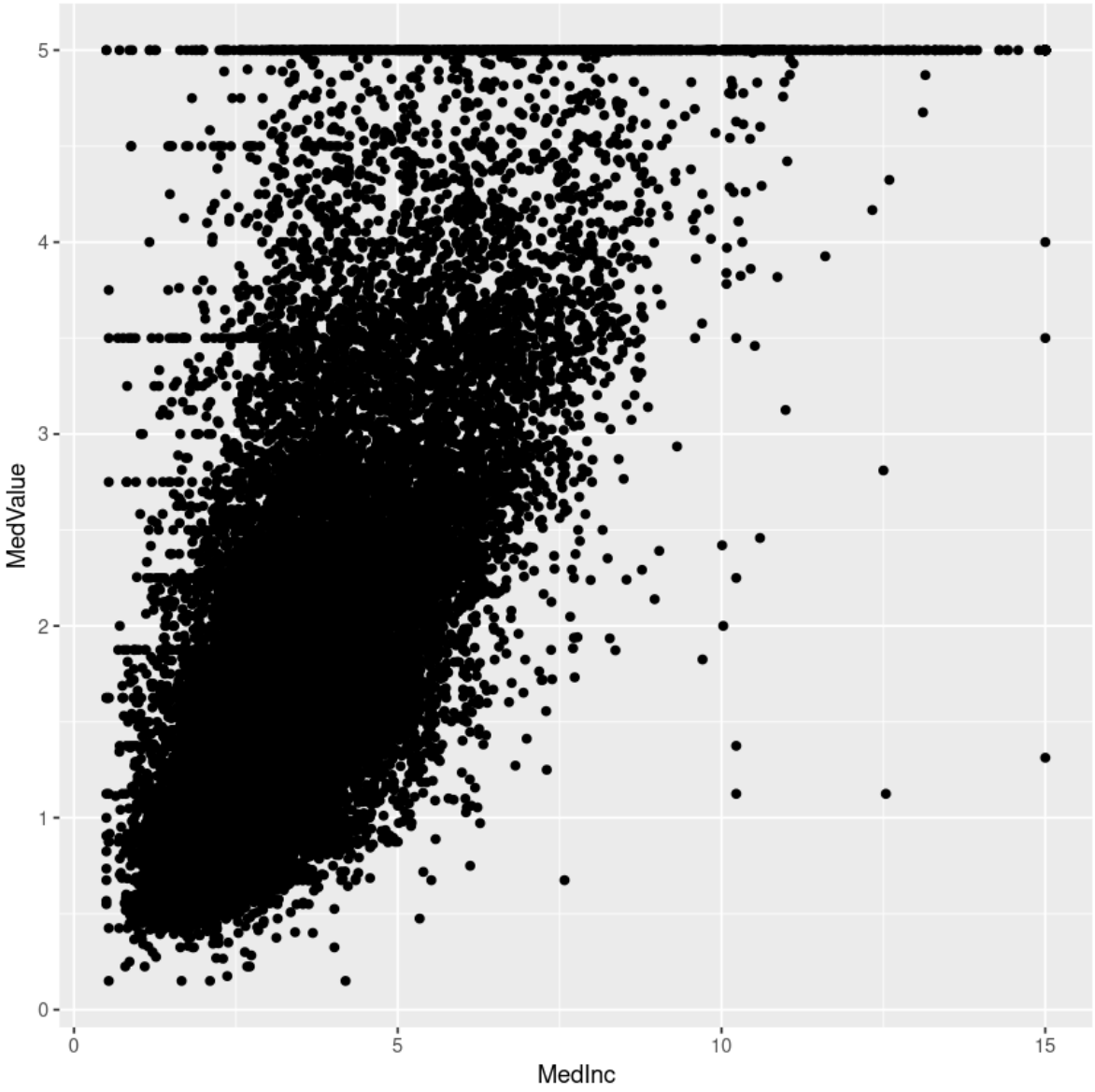
```r
url <- "https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.tgz"
download.file(url, destfile = "cal.tar.gz")
untar("cal.tar.gz")

# Read the data into R and provide column names
df_california <- read.csv("CaliforniaHousing//cal_housing.data", header = FALSE)
columns_index <- c(9, 8, 3, 4, 5, 6, 7, 2, 1)
df_california <- df_california[, columns_index]
names(df_california) <- c(
    "MedValue",
    "MedInc",
    "HouseAge",
    "AveRooms",
    "AveBedrms",
    "Population",
    "AveOccup",
    "Latitude",
    "Longitude"
)
df_california$MedValue <- df_california$MedValue / 100000
df_california <- df_california[, c("MedInc", "MedValue")]
head(df_california)
```

A data.frame: 6 × 2

	MedInc <dbl>	MedValue <dbl>
1	8.3252	4.526
2	8.3014	3.585
3	7.2574	3.521
4	5.6431	3.413
5	3.8462	3.422
6	4.0368	2.697

Inspect

```r
ggplot(dataset_california, aes(x = MedInc, y = MedValue)) +
    geom_point()
```



Python

Get data

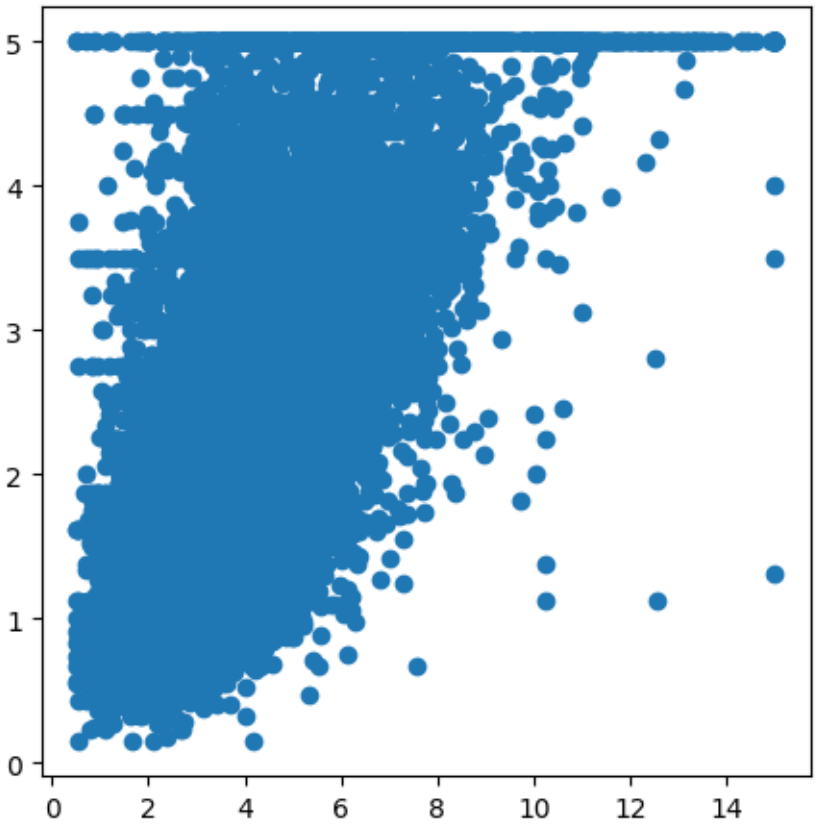
```
from sklearn.datasets import fetch_california_housing
dataset_california = fetch_california_housing(as_frame=True)
df_california = dataset_california.frame.loc[:, ['MedInc', 'MedHouseVal']]
```

```
df_california.head()
```

	MedInc	MedHouseVal
0	8.3252	4.526
1	8.3014	3.585
2	7.2574	3.521
3	5.6431	3.413
4	3.8462	3.422

Inspect

```
# instantiate plot
fig = plt.figure(figsize=(4, 4))
# Creating axes instance
ax = fig.add_axes([0, 0, 1, 1])
# Creating plot
sp = ax.scatter(
    x=df_california.loc[:, ['MedInc']],
    y=df_california.loc[:, ['MedHouseVal']]
)
plt.show()
```



Use the first 100 observations as training data to compute a linear model and evaluate the performance of your learner on the remaining data using MSE.

What might be disadvantageous about the previous train-test split?

Now, sample your training observations from the data set at random. Use a share of 0.1 through 0.9, in 0.1 steps, of observations for training and repeat this procedure ten times. Afterwards, plot the resulting test errors (in terms of MSE) in a suitable manner.

Hint

R

`rsmp` is a convenient function for splitting data – you will want to choose the "holdout" strategy. Afterwards, `resample` can be used to repeatedly fit the learner.

Python

`from sklearn.model_selection import train_test_split` is a convenient function for splitting data. It has an optional parameter `random_state`, which can be used to split the data randomly in each iteration.

Interpret the findings from the previous question.