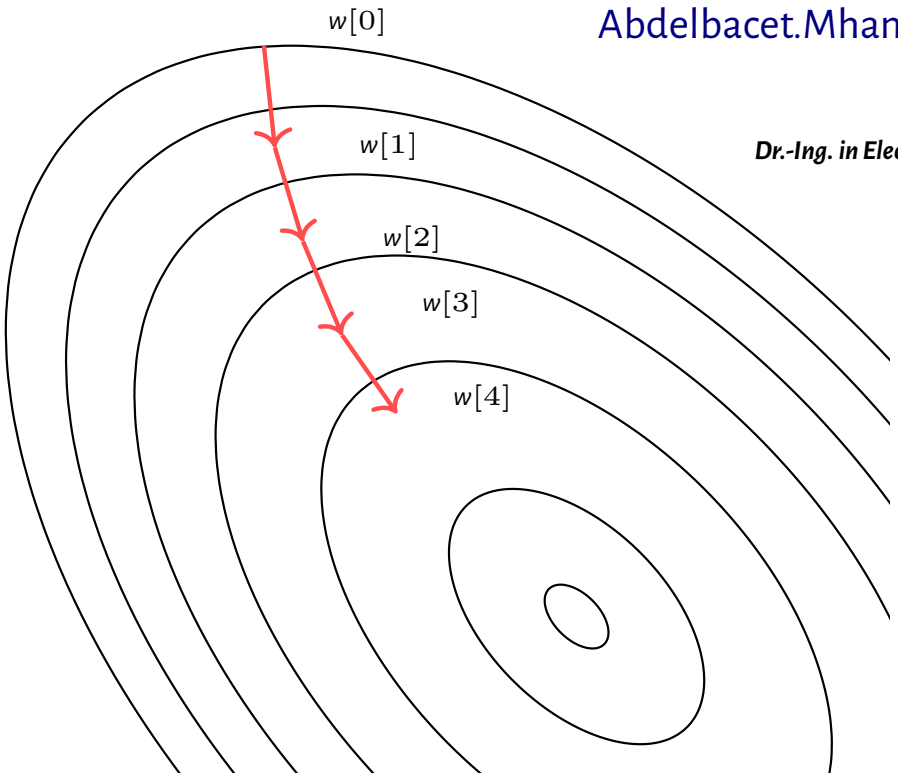
An Introduction to Machine Learning Sorcery

(An Early Draft)[1]

Abdelbacet Mhamdi
Abdelbacet.Mhamdi@bizerte.r-iset.tn

Dr.-Ing. in Electrical Engineering



"Computers are able to see, hear and learn. Welcome to the future."

Dave Waters

"This is nothing. In a few years, that bot will move so fast you'll need a strobe light to see it. Sweet dreams..."

Elon Musk

"Machine intelligence is the last invention that humanity will ever need to make."

Nick Bostrom

[1]Available @ https://github.com/a-mhamdi/isetbz/

This document features some material gathered from multiple online sources.
Please note no copyright infringement is intended, and I do not own nor claim to own any of the original material. They are used for educational purposes only.
I have included links solely as a convenience to the reader. Some links within these slides may lead to other websites, including those operated and maintained by third parties. The presence of such a link does not imply a responsibility for the linked site or an endorsement of the linked site, its operator, or its contents.

Outline

Next...

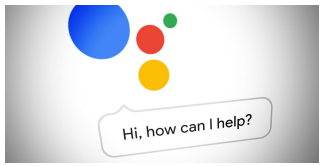
Top uses


YouTube


Gboard

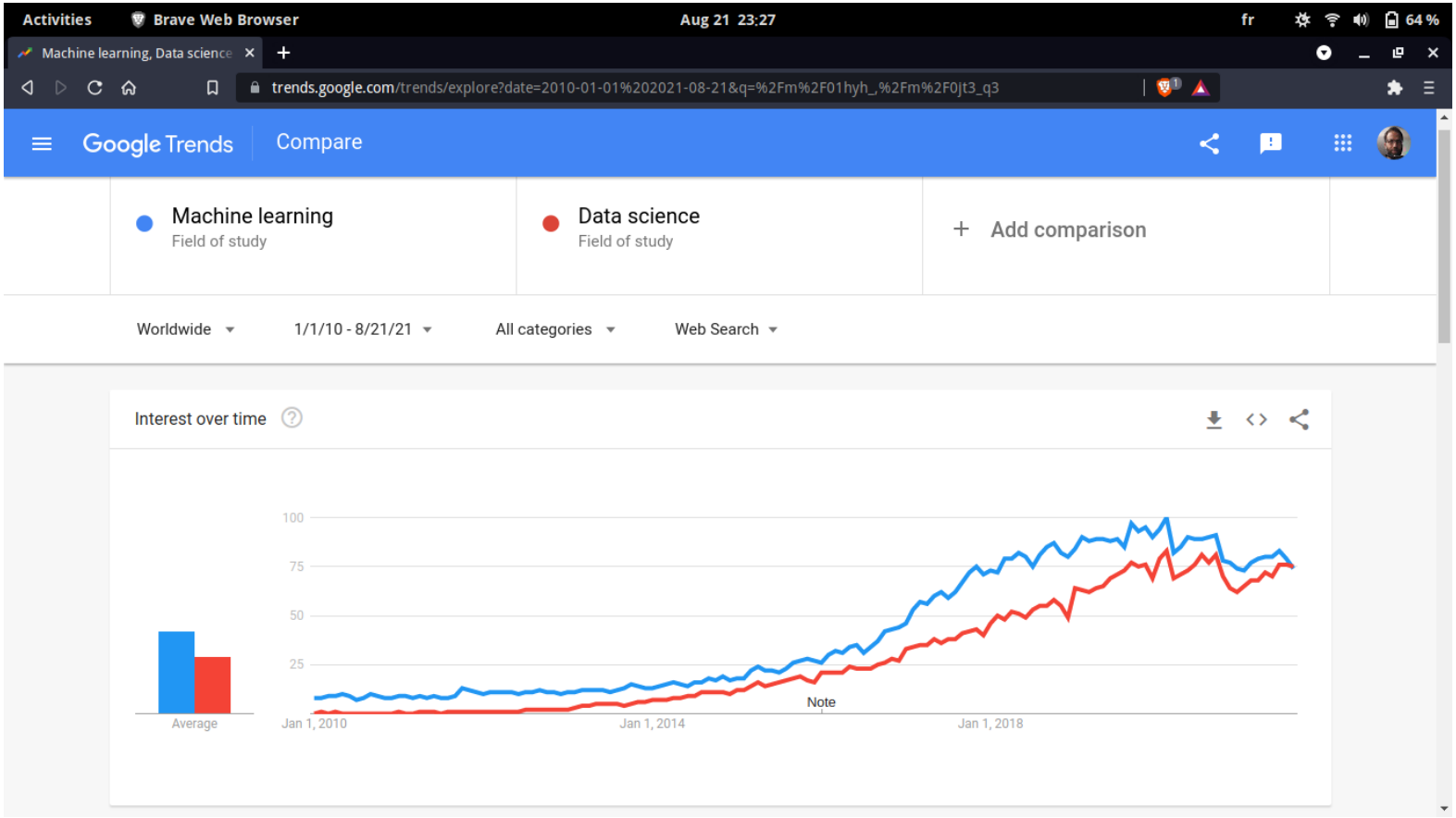

Google
Translate


amazon


Hi, how can I help?

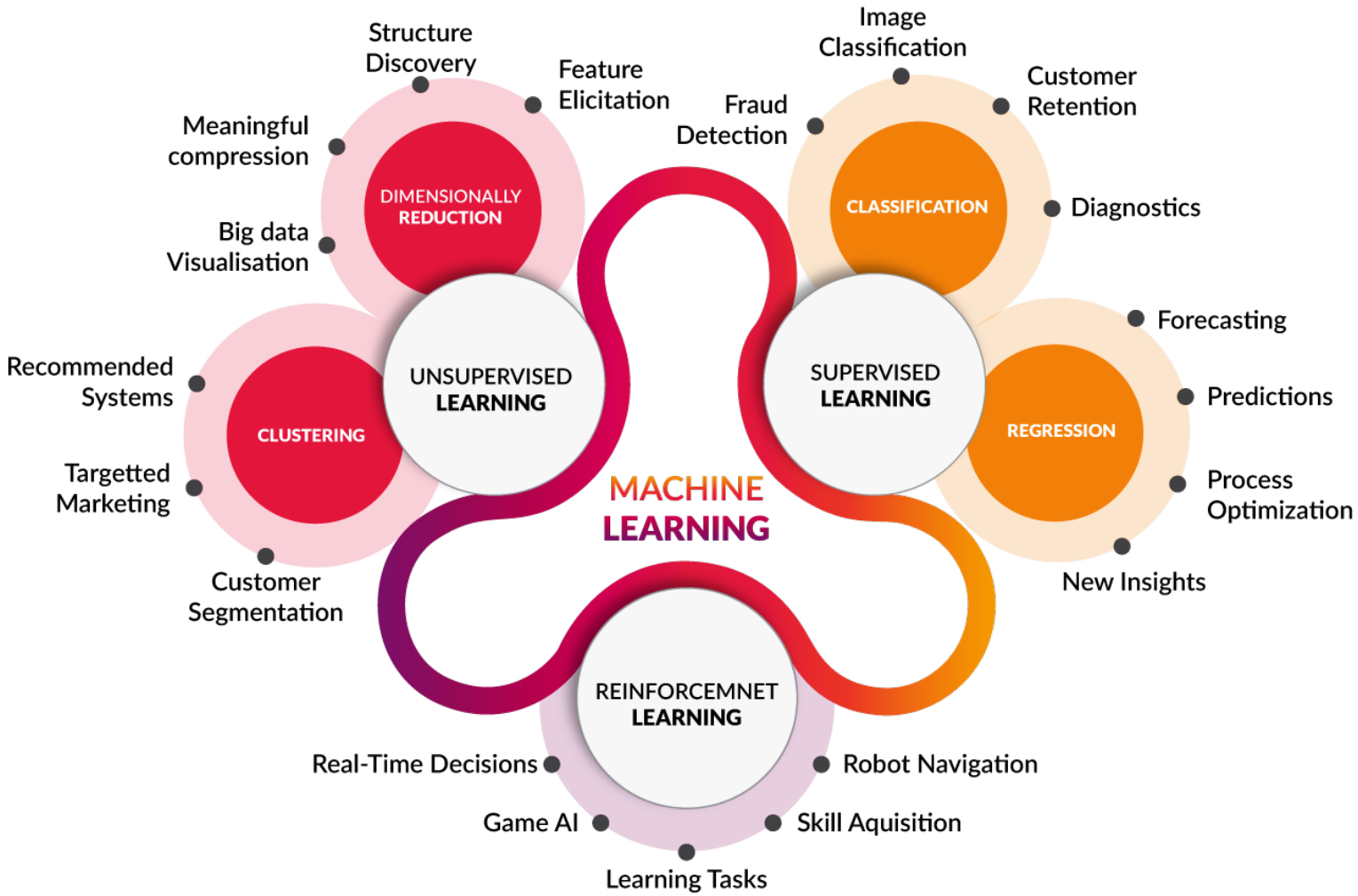

amazon
alexa


Hi, I'm Cortana.

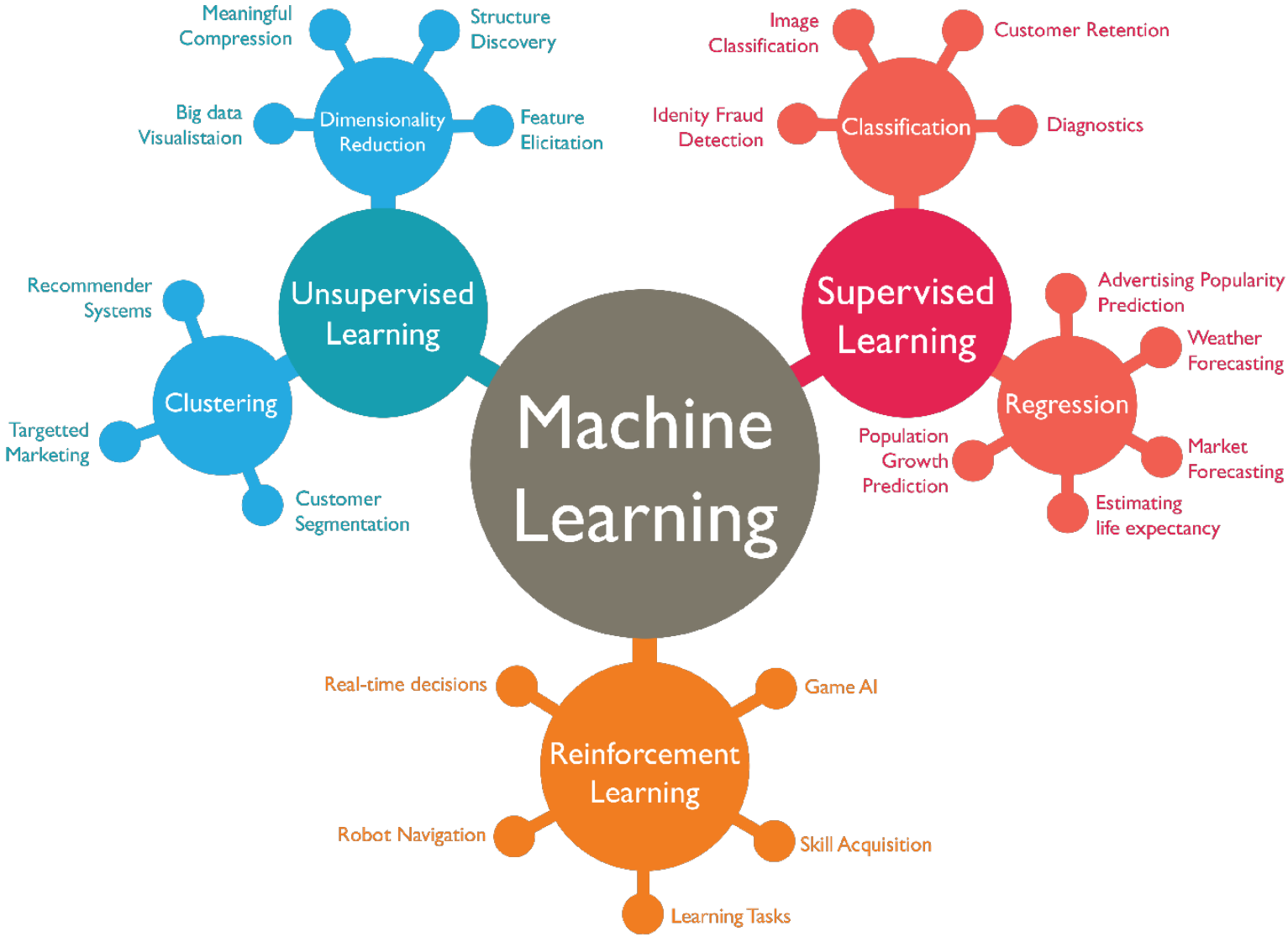


"Numbers represent search interest relative to the highest point on the chart for the given region and time.

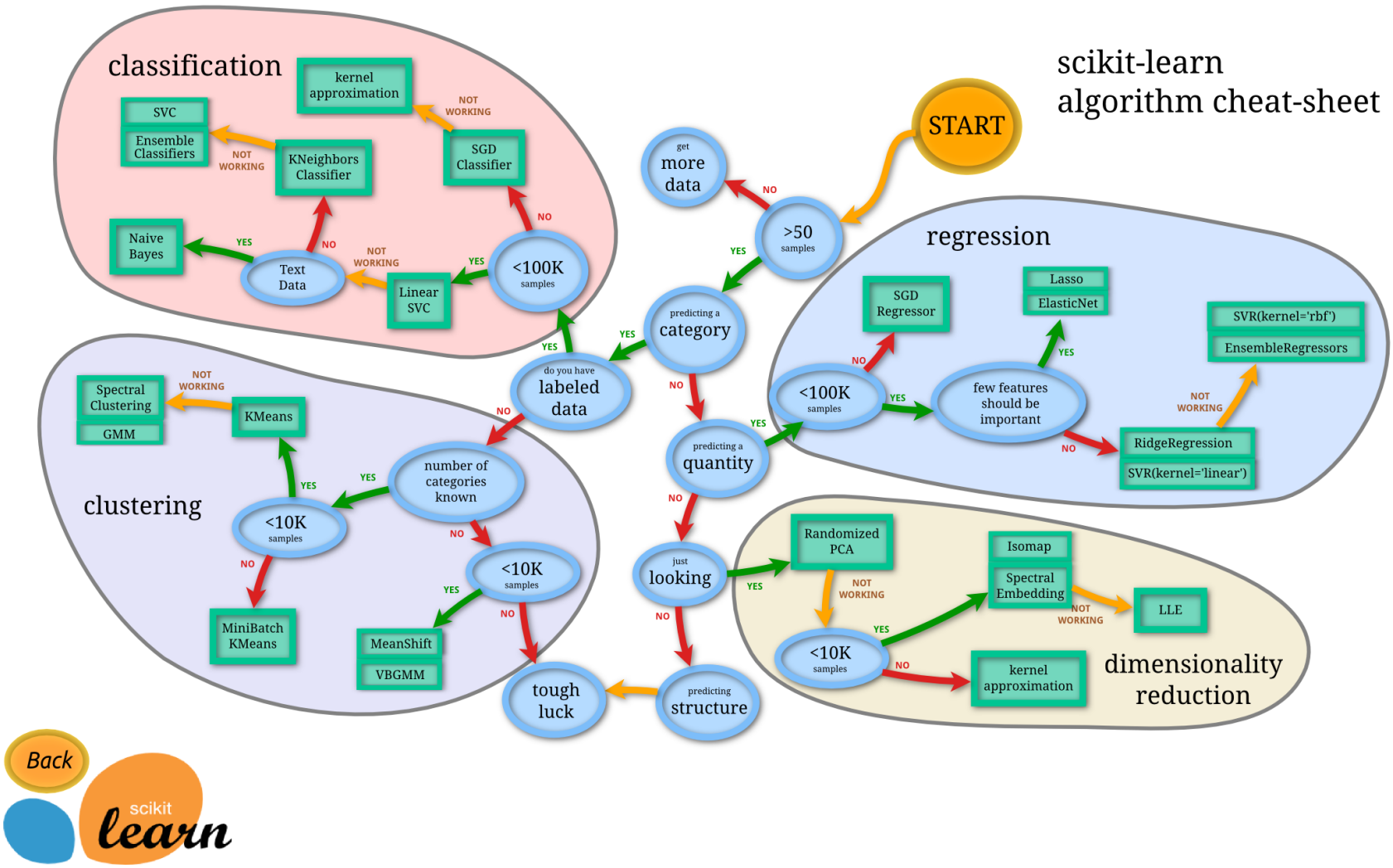
- A value of 100 is the peak popularity for the term;
- A value of 50 means that the term is half as popular;
- A score of 0 means there was not enough data for this term."



https://www.cognub.com/index.php/cognitive-platform/

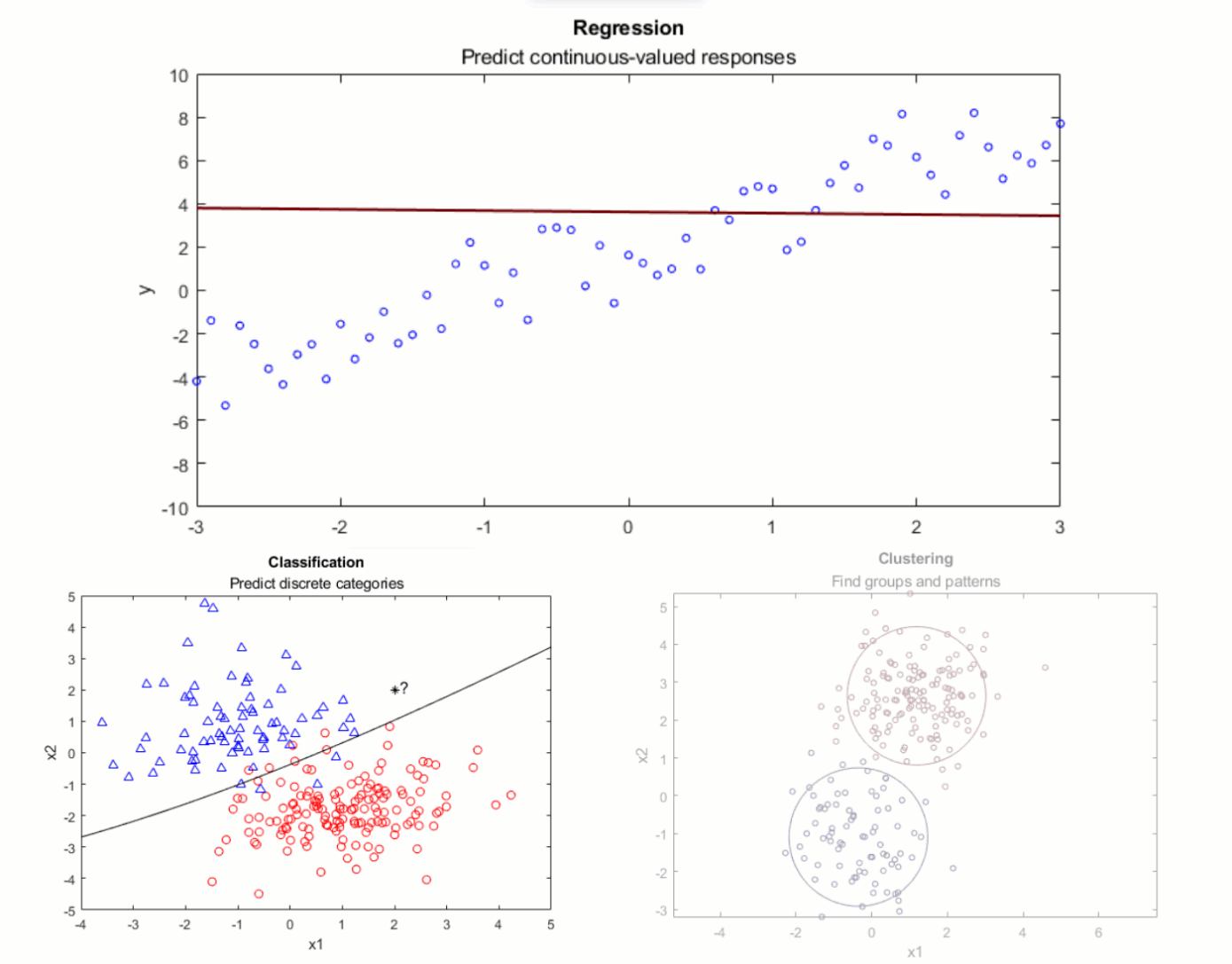


https://vitalflux.com/great-mind-maps-for-learning-machine-learning/



https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Regression | Classification | Clustering



https://github.com/MathWorks-Teaching-Resources/Machine-Learning-for-Regression

Machine Learning Definition

Arthur Samuel (1959)

Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell (1998)

Well-posed Learning Problem: A computer is said to learn from experience $\mathcal{E}$ with respect to some task $\mathcal{T}$ and some performance measure $\mathcal{P}$, if its performance on $\mathcal{T}$, as measured by $\mathcal{P}$, improves with experience $\mathcal{E}$.

Machine Learning Definition

Arthur Samuel (1959)

Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell (1998)

Well-posed Learning Problem: A computer is said to learn from experience $\mathcal{E}$ with respect to some task $\mathcal{T}$ and some performance measure $\mathcal{P}$, if its performance on $\mathcal{T}$, as measured by $\mathcal{P}$, improves with experience $\mathcal{E}$.

Task #1

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task $\mathcal{T}$ in this setting?

1. Classifying emails as spam or not spam;
2. Watching you label emails as spam or not spam;
3. The number (or fraction) of emails correctly classified as spam/not spam;
4. None of the above-this not a machine learning problem.

Machine Learning Definition

Arthur Samuel (1959)

Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell (1998)

Well-posed Learning Problem: A computer is said to learn from experience $\mathcal{E}$ with respect to some task $\mathcal{T}$ and some performance measure $\mathcal{P}$, if its performance on $\mathcal{T}$, as measured by $\mathcal{P}$, improves with experience $\mathcal{E}$.

Task #1

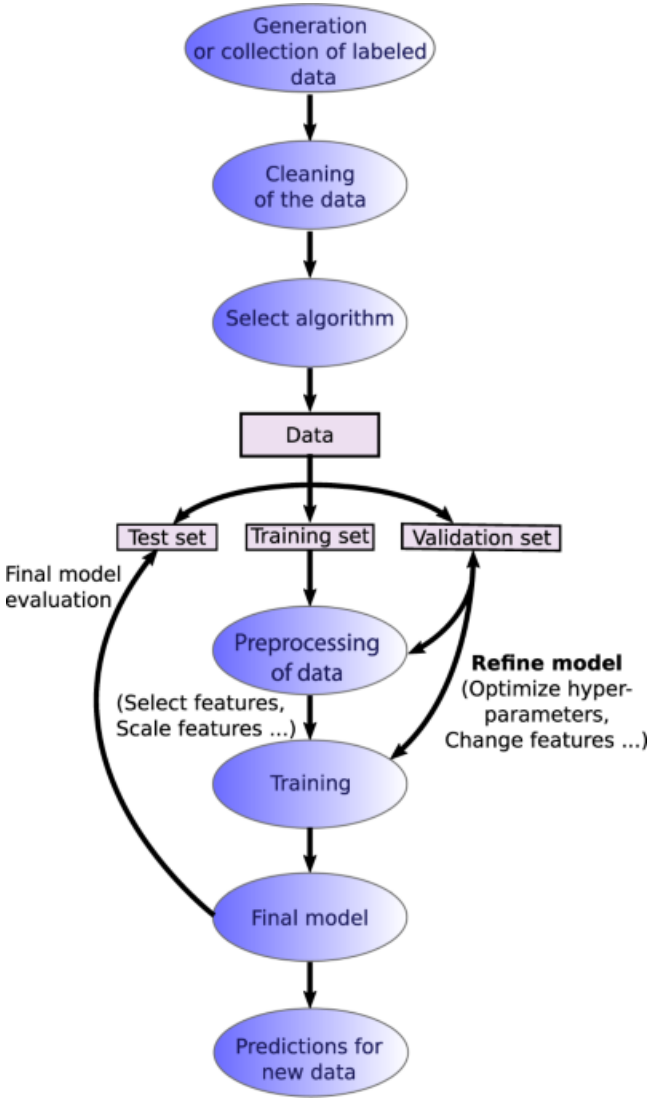
Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task $\mathcal{T}$ in this setting?

1. Classifying emails as spam or not spam;
2. Watching you label emails as spam or not spam;
3. The number (or fraction) of emails correctly classified as spam/not spam;
4. None of the above-this not a machine learning problem.

Next...

Overall Methodology

[Sch+19]



Data Preprocessing Template (1/7)

Importing the libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing the dataset

[2]:
```
df = pd.read_csv('datasets/Data.csv')
df.head()
```

[2]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes

Extracting independant and dependant variables

[3]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

Data Preprocessing Template (2/7)

[4]:
```
print(X) # Features
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

[5]:
```
print(y) # Target
```

```
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

Data Preprocessing Template (3/7)

Imputation transformer for completing missing values

https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

[6]:
```
from sklearn.impute import SimpleImputer
```

[7]:
```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

[8]:
```
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

Data Preprocessing Template (4/7)

How to encode categorical data?

Case of Independent Variable

[9]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

[10]:
```
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),␣
↪[0])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

[11]:
```
print(X)
```

```
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

Data Preprocessing Template (5/7)

Case of Dependent Variable

[12]:
```
from sklearn.preprocessing import LabelEncoder
```

[13]:
```
le = LabelEncoder()
y = le.fit_transform(y)
```

[14]:
```
print(y)
```

```
[0 1 0 0 1 1 0 1 0 1]
```

Splitting the dataset into Training set and Test set

[15]:
```
from sklearn.model_selection import train_test_split
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪2, random_state=1)
```

[17]:
```
print(X_train)
```

Data Preprocessing Template (6/7)

```
[[0.0 0.0 1.0 38.77777777777778 52000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 35.0 58000.0]]
```

[18]:
```
print(X_test)
```

```
[[0.0 1.0 0.0 30.0 54000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

[19]:
```
print(y_train)
```

```
[0 1 0 0 1 1 0 1]
```

[20]:
```
print(y_test)
```

```
[0 1]
```

Data Preprocessing Template (7/7)

Scaling of features

[21]:
```
from sklearn.preprocessing import StandardScaler
```

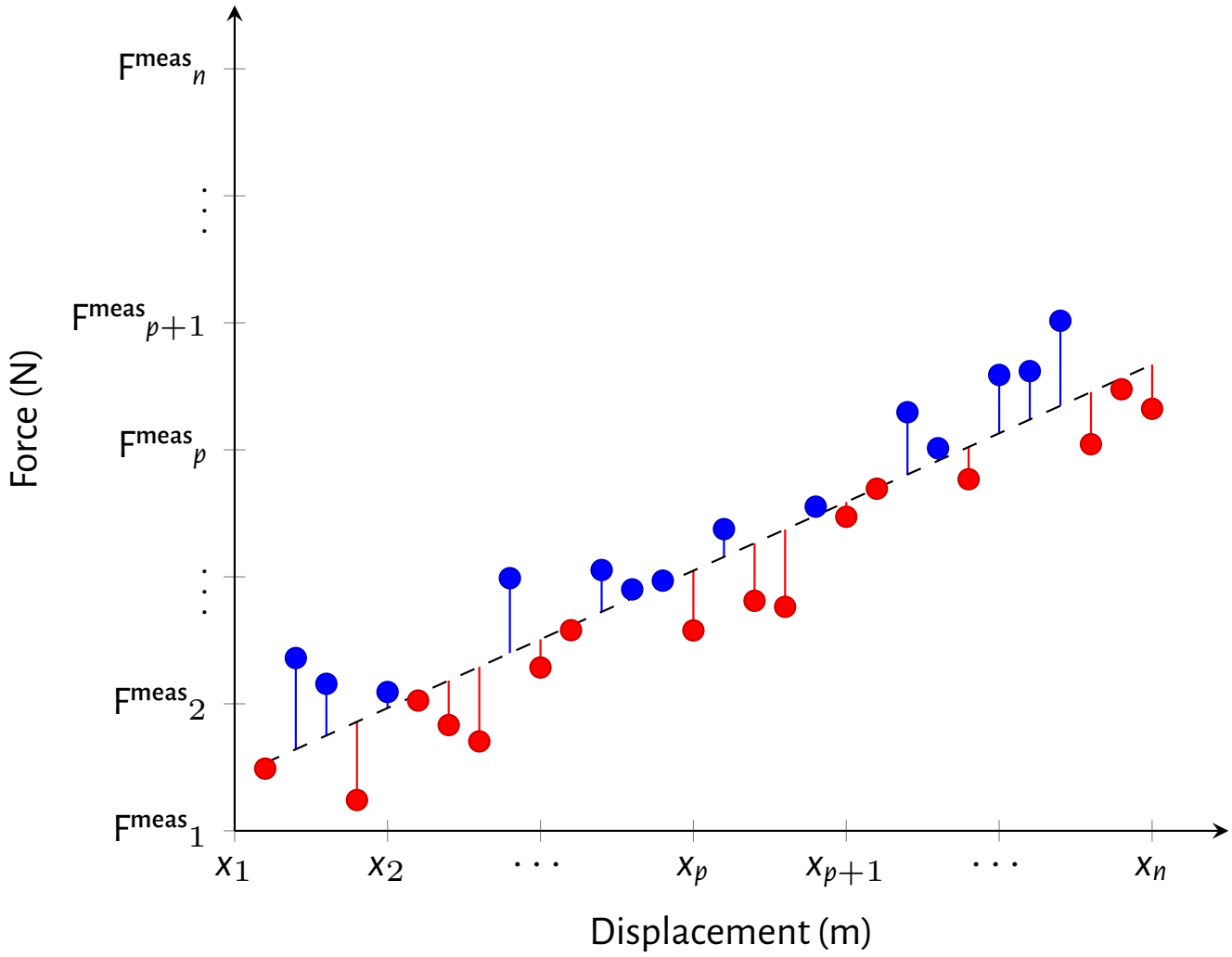
[22]:
```
sc = StandardScaler()
X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
X_test[:, 3:] = sc.transform(X_test[:, 3:])
```

[23]:
```
print(X_train)
```

```
[[0.0 0.0 1.0 -0.19159184384578545 -1.0781259408412425]
 [0.0 1.0 0.0 -0.014117293757057777 -0.07013167641635372]
 [1.0 0.0 0.0 0.566708506533324 0.633562432710455]
 [0.0 0.0 1.0 -0.30453019390224867 -0.3078661727429786 7]
 [0.0 0.0 1.0 -1.9018011447007988 -1.420463615551582]
 [1.0 0.0 0.0 1.1475343068237058 1.232653363453549]
 [0.0 1.0 0.0 1.4379472069688968 1.5749910381638885]
 [1.0 0.0 0.0 -0.7401495441200351 -0.5646194287757332]]
```

[24]:
```
print(X_test)
```

```
[[0.0 1.0 0.0 -1.4661817944830124 -0.9069571034860727]
 [1.0 0.0 0.0 -0.44973664397484414 0.2056403393225306]]
```


Force (N)
F^{meas}_n
F^{meas}_{p+1}
F^{meas}_p
F^{meas}_2
F^{meas}_1
x_1
x_2
x_p
x_{p+1}
x_n
Displacement (m)

Consider the example of a spring. Our main goal is to determine the stiffness k of this spring, given some experimental data. The mathematical model *(Hooke's law)*:

$$F = kx \tag{1}$$

Restoring force is proportional to displacement.

Table: Measurements of couple $(x_i,\ \mathrm{F}^{\mathrm{meas}}{}_i)$

x_i	x_1	$\cdots$	x_p	$\cdots$	x_n
$\mathrm{F}^{\mathrm{meas}}{}_i$	$\mathrm{F}^{\mathrm{meas}}{}_1$	$\cdots$	$\mathrm{F}^{\mathrm{meas}}{}_p$	$\cdots$	$\mathrm{F}^{\mathrm{meas}}{}_n$

$$\begin{aligned} \mathrm{F}^{\mathrm{meas}}{}_i &= \mathrm{F}_i + \varepsilon_i \\ &= kx_i + \varepsilon_i, \end{aligned} \tag{2}$$

where F_i denotes the unknown real value of the force applied to the spring. In order to estimate the stiffness value k, we can consider the quadratic criterion:

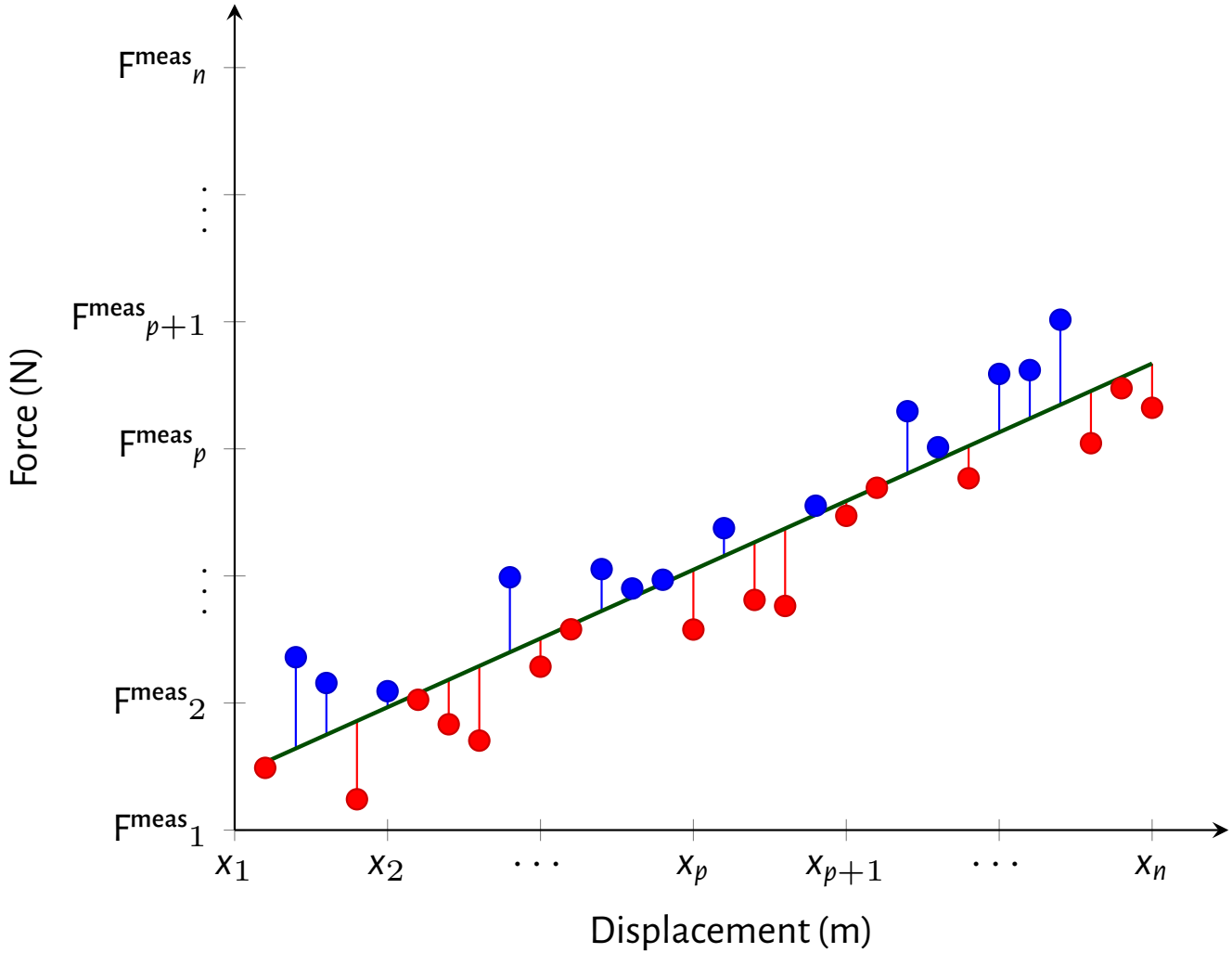
$$\begin{aligned} \mathcal{J} &= \sum_{i=1}^{n} \varepsilon_i^2 \\ &= \sum_{i=1}^{n} \left(\mathrm{F}^{\mathrm{meas}}{}_i - kx_i\right)^2 \end{aligned}$$

$$\frac{\partial \mathcal{J}}{\partial k} = 0 \tag{3}$$

$$2\sum_{i=1}^{n}\left(\mathrm{F}^{\mathrm{meas}}{}_i - kx_i\right)\sum_{i=1}^{n}\frac{\partial\left(\mathrm{F}^{\mathrm{meas}}{}_i - kx_i\right)}{\partial k} = 0$$

$$\sum_{i=1}^{n}\left(\mathrm{F}^{\mathrm{meas}}{}_i - kx_i\right)\sum_{i=1}^{n}x_i = 0$$

$$\sum_{i=1}^{n}\mathrm{F}^{\mathrm{meas}}{}_i\, x_i = k\sum_{i=1}^{n}x_i^2 \iff \hat{k} = \frac{\sum_{i=1}^{n}\mathrm{F}^{\mathrm{meas}}{}_i\, x_i}{\sum_{i=1}^{n}x_i^2}$$


F^{meas}_n
F^{meas}_{p+1}
F^{meas}_p
F^{meas}_2
F^{meas}_1
Force (N)
x_1
x_2
x_p
x_{p+1}
x_n
Displacement (m)

Simple Linear Regression (1/6)

Importing the libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing the dataset

[2]:
```
df = pd.read_csv('datasets/Salary_Data.csv')
df.head()
```

[2]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

[3]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

Splitting the dataset into training set and test set

Simple Linear Regression (2/6)

[4]:
```
from sklearn.model_selection import train_test_split
```

[5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/
↪3, random_state=0)
```

Training the Simple Linear Regression model on the Training set

[6]:
```
from sklearn.linear_model import LinearRegression
```

[7]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

[7]:
```
LinearRegression()
```

Predicting the Test set results

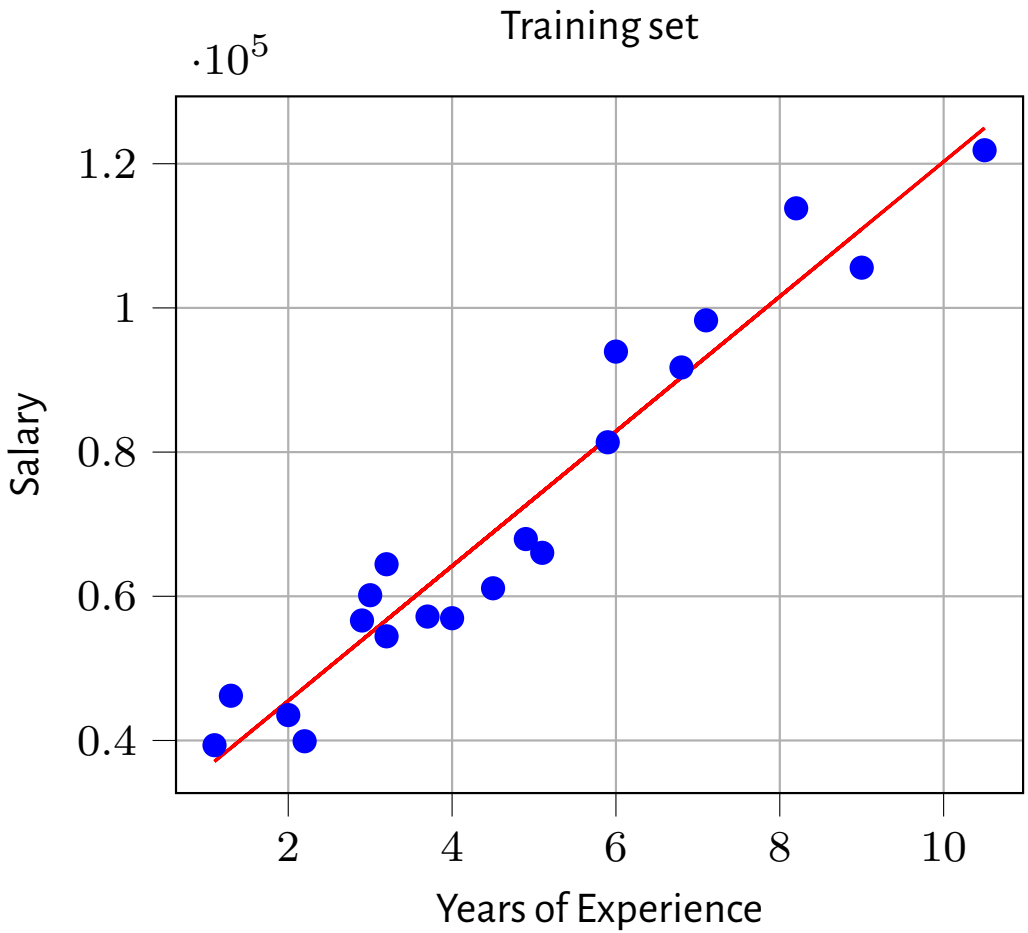
[8]:
```
y_pred = regressor.predict(X_test)
```

Visualising the Training set results

Simple Linear Regression (3/6)

[9]:
```
plt.scatter(X_train, y_train, color='blue')
plt.plot(X_train, regressor.predict(X_train), color='red')
plt.title('Training set')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.grid()
```

Simple Linear Regression (4/6)

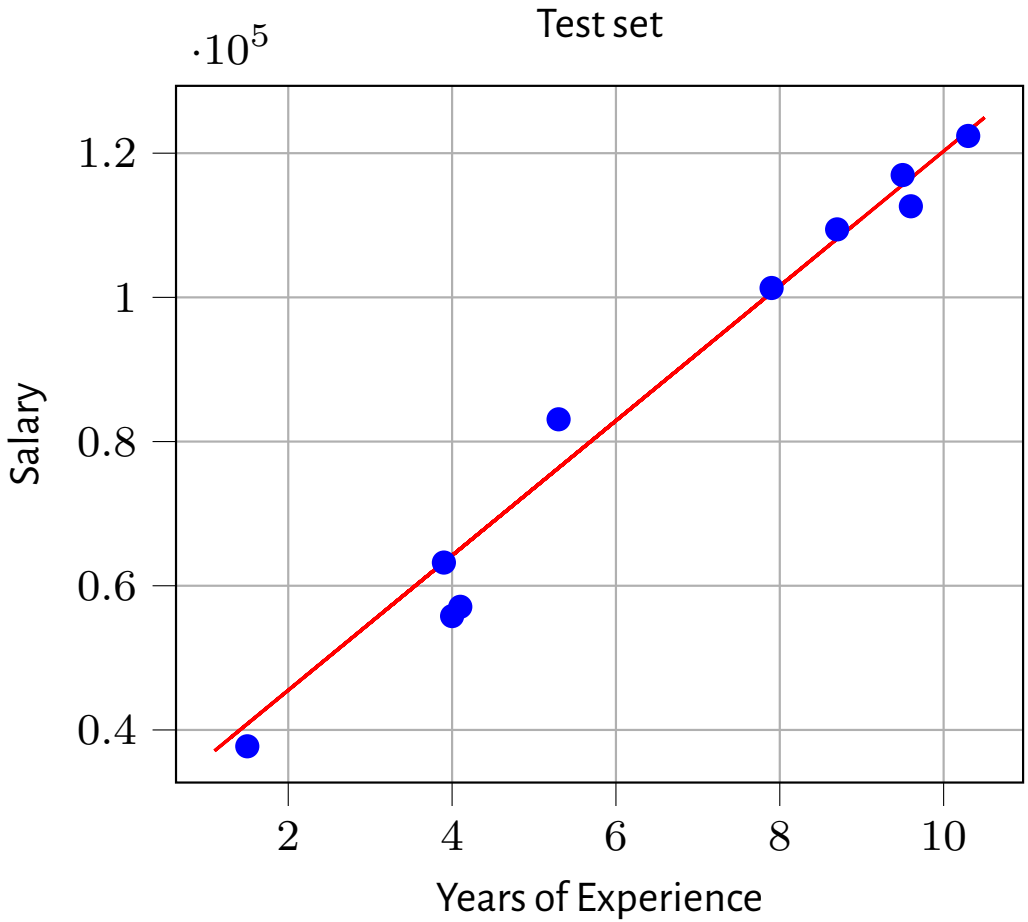


Simple Linear Regression (5/6)

Visualising the Test set results

[10]:
```
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_train, regressor.predict(X_train), color='red')
plt.title('Test set')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.grid()
```

Simple Linear Regression (6/6)



This example consists on determining the unknown couple $(y_0,\ v_0)$ of a mobile solid. We assume that the trajectory is linear. The mathematical model that relates the position y to time t is given by this equation:

$$y = y_0 + v_0 t \tag{4}$$

Table: Measurements of position y

t_i	t_1	$\cdots$	t_p	$\cdots$	t_n
$y^{meas}{}_i$	$y^{meas}{}_1$	$\cdots$	$y^{meas}{}_p$	$\cdots$	$y^{meas}{}_n$

$$\begin{aligned} y^{meas}{}_i &= y_i + \varepsilon_i \\ &= y_0 + v_0 t_i + \varepsilon_i, \end{aligned} \tag{5}$$

where y_i denotes the unknown real value of the position y at time point t_i.
In order to estimate the values taken by the couple $\left[\ y_0,\quad v_0\ \right]^T$, we consider the quadratic criterion again, as follows:

$$\begin{aligned} \mathcal{J} &= \sum_{i=1}^{n} \varepsilon_i^2 \\ &= \varepsilon^T \times \varepsilon \end{aligned}$$

The vector ε is set by $\varepsilon_i, \ \forall i \geq 1$:

$$\varepsilon = \begin{bmatrix} \varepsilon_1 & \cdots & \varepsilon_n \end{bmatrix}^T$$

$$\frac{\partial \mathcal{J}}{\partial \begin{bmatrix} y_0 \\ \\ v_0 \end{bmatrix}} = 0 \tag{6}$$

Multiple Linear Regression (1/4)

Importing the libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing the dataset

[2]:
```
df = pd.read_csv('datasets/50_Startups.csv')
df.head()
```

[2]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

[3]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

[4]:
```
print(X[:5])
```

Multiple Linear Regression (2/4)

```
[[165349.2 136897.8 471784.1 'New York']
 [162597.7 151377.59 443898.53 'California']
 [153441.51 101145.55 407934.54 'Florida']
 [144372.41 118671.85 383199.62 'New York']
 [142107.34 91391.77 366168.42 'Florida']]
```

[5]:
```
print(y[:5])
```

```
[192261.83 191792.06 191050.39 182901.99 166187.94]
```

Encoding categorical data

[6]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

[7]:
```
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),␣
↪[3])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

[8]:
```
print(X[:5])
```

Multiple Linear Regression (3/4)

```
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]]
```

Splitting the dataset into training set and test set

[9]:
```
from sklearn.model_selection import train_test_split
```

[10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪2, random_state=0)
```

Training the multiple linear regression model on the training set

[11]:
```
from sklearn.linear_model import LinearRegression
```

[12]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

[12]:
```
LinearRegression()
```

Making predictions using the X test set and comparison

Multiple Linear Regression (4/4)

[13]:
```
y_pred = lr.predict(X_test)
np.set_printoptions(precision=2)
print(type(y_pred), y_pred.shape)
y_pred = y_pred.reshape(-1,1)
print(type(y_pred), y_pred.shape)
y_test = y_test.reshape(-1,1)
print(np.concatenate((y_pred, y_test), axis=1))
```

```
<class 'numpy.ndarray'> (10,)
<class 'numpy.ndarray'> (10, 1)
[[103015.2  103282.38]
 [132582.28 144259.4 ]
 [132447.74 146121.95]
 [ 71976.1   77798.83]
 [178537.48 191050.39]
 [116161.24 105008.31]
 [ 67851.69  81229.06]
 [ 98791.73  97483.56]
 [113969.44 110352.25]
 [167921.07 166187.94]]
```

Consider the following multivariate equation:

$$y = \theta_1 x_{(1)} + \theta_2 x_{(2)} + \cdots + \theta_m x_{(m)} \tag{7}$$

For a particular single measurement, eq. (7) can be updated as

$$y_k = \theta_1 x_{(1,k)} + \theta_2 x_{(2,k)} + \cdots + \theta_m x_{(m,k)} + \varepsilon_k \tag{8}$$

We denote hereafter by θ the vector $\begin{bmatrix} \theta_1 \\ \theta_2 \\ . \\ . \\ . \\ \theta_n \end{bmatrix}$. The function y_k becomes:

$$y_k = \underbrace{\left[x_{(1,k)}, x_{(2,k)}, \cdots, x_{(m,k)}\right]}_{x_k^T} \theta + \varepsilon_k$$

We assume that we have n measurements for y. Then we can transform the previous equation into

$$y = H\theta + \varepsilon,$$

where $y^T = [y_1, y_2, \cdots, y_n], X = \begin{bmatrix} x_1^T \\ x_2^T \\ . \\ . \\ . \\ x_n^T \end{bmatrix}$, and $\varepsilon^T = [\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_n]$.

We can consider the mean squared error or quadratic criterion in order to compute the approximated value of θ:

$$\begin{aligned} \mathcal{J} &= \sum_{k=1}^{n} \varepsilon_k^2 \\ &= \varepsilon^T \varepsilon \end{aligned}$$

The best well estimated value of $\hat{\theta}$ corresponds to the absolute minimum of $\mathcal{J}$. This leads to calculate the gradient of $\mathcal{J}$ with respect to θ:

$$\frac{\partial \mathcal{J}}{\partial \theta} = \frac{\partial \left(\varepsilon^T \varepsilon\right)}{\partial \theta} \tag{9}$$

$$\frac{\partial \left(\varepsilon^T \varepsilon\right)}{\partial \theta} = 2 \left(\frac{\partial \varepsilon}{\partial \theta}\right)^T \varepsilon \tag{10}$$

Recall that $\varepsilon = y - X\theta$, the term $\dfrac{\partial \varepsilon}{\partial \theta}$ hence becomes:

$$\frac{\partial \varepsilon}{\partial \theta} = -X \tag{11}$$

$$\begin{aligned}\frac{\partial J}{\partial \theta} &= 2(-X)^T (y - X\theta) \\ &= 0\end{aligned}$$

The regressor is given by

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

$X^T X$ **is not invertible (singular/degenerate)**

▼ Redundant Features

Some features are linearly dependant, i.e, $\exists$ some $x_p \propto$ some x_l for instance x_p in feet and x_l in m.

▼ Too many features

Fewer observations compared to the number of features, i.e, $m \geq n$.

▲ Delete some features
▲ Add extra observations
▲ Use regularization

Gradient Descent

$$\theta_i = \theta_i - \underbrace{\alpha}_{\text{Learning Rate}} \frac{\partial \mathcal{J}}{\partial \theta_i}$$

Recall that $\mathcal{J} = {}^1/_{2n} \sum_{k=1}^{n} (y_k - h_\theta(x_k))^2 \implies \frac{\partial \mathcal{J}}{\partial \theta_i} = {}^1/_n \sum_{k=1}^{n} (y_k - h_\theta(x_k))\, x_{(i,k)}$

$$\theta_i \leftarrow \theta_i - \alpha \frac{1}{n} \sum_{k=1}^{n} (y_k - h_\theta(x_k))\, x_{(i,k)}$$

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{n} \sum_{k=1}^{n} (y_k - h_\theta(x_k))\, x_{(0,k)}$$

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{1}{n} \sum_{k=1}^{n} (y_k - h_\theta(x_k))\, x_{(1,k)}$$

$$\vdots$$

$$\theta_m \leftarrow \theta_m - \alpha \frac{1}{n} \sum_{k=1}^{n} (y_k - h_\theta(x_k))\, x_{(m,k)}$$

Polynomial Regression (1/8)

Importing the libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing the dataset

[2]:
```
df = pd.read_csv('datasets/Position_Salaries.csv')
df.head()
```

[2]:

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000

[3]:
```
X = df.iloc[:, 1:-1].values
y = df.iloc[:, -1].values
```

[4]:
```
print(type(X), X[:5], sep='\n')
```

Polynomial Regression (2/8)

```
<class 'numpy.ndarray'>
[[1]
 [2]
 [3]
 [4]
 [5]]
```

[5]:
```
print(type(y), y[:5], sep='\n')
```

```
<class 'numpy.ndarray'>
[ 45000  50000  60000  80000 110000]
```

Training the linear regression model on the whole dataset

[6]:
```
from sklearn.linear_model import LinearRegression
```

Training the linear regression model on the whole dataset

[7]:
```
lr_1 = LinearRegression()
lr_1.fit(X, y)
```

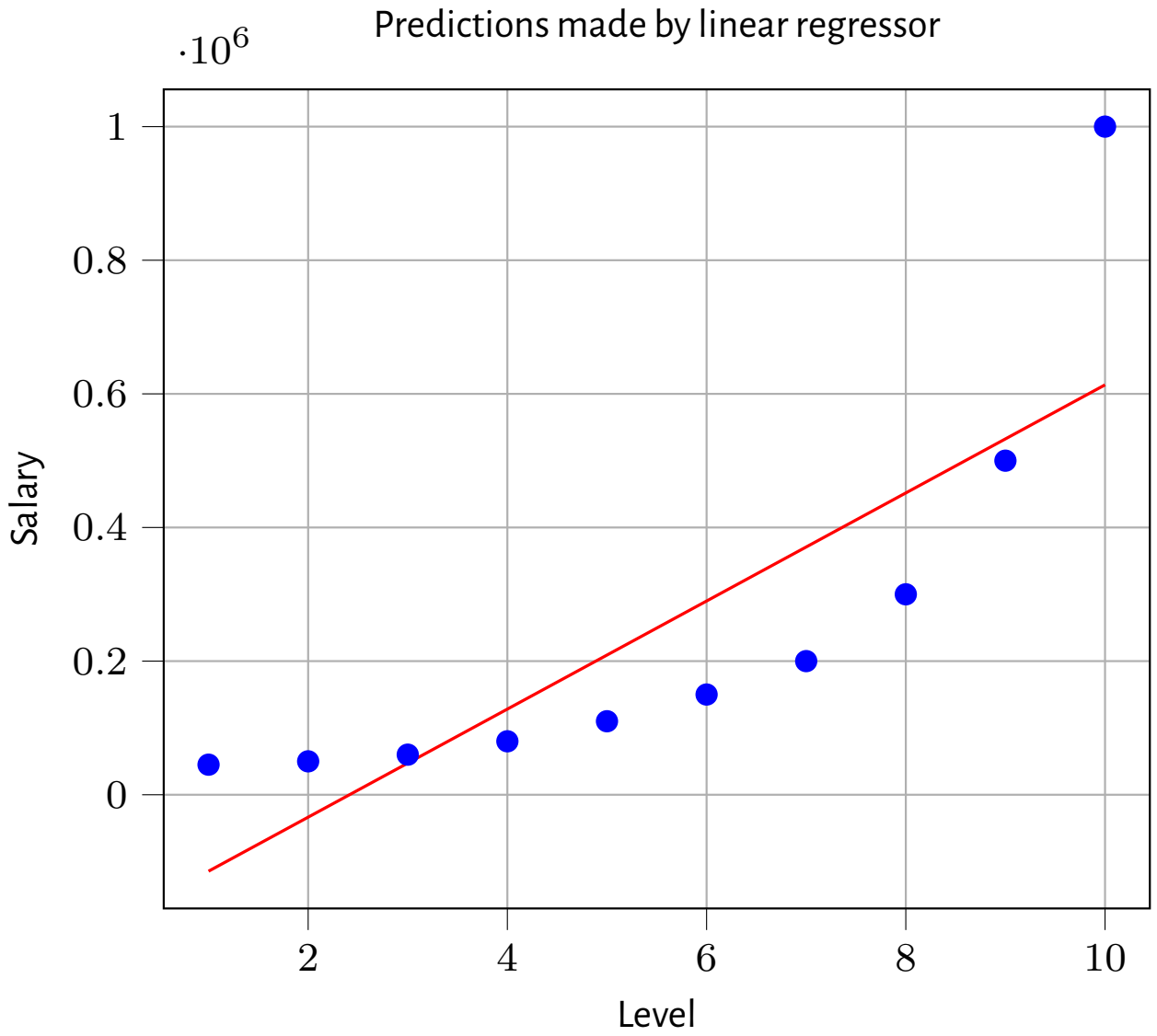
[7]:
```
LinearRegression()
```

Visualising the linear regression predictions

Polynomial Regression (3/8)

[8]:
```
plt.scatter(X, y, color = 'blue')
plt.plot(X, lr_1.predict(X), color = 'red')
plt.title('Predictions made by linear regressor')
plt.xlabel('Level')
plt.ylabel('Salary')
plt.grid()
```

Polynomial Regression (4/8)



Polynomial Regression (5/8)

[9]:
```
from sklearn.preprocessing import PolynomialFeatures
```

[10]:
```
poly_reg = PolynomialFeatures(degree=4)
X_poly = poly_reg.fit_transform(X)
print(X_poly[:5])
lr_2 = LinearRegression()
lr_2.fit(X_poly, y)
```

```
[[  1.   1.   1.   1.   1.]
 [  1.   2.   4.   8.  16.]
 [  1.   3.   9.  27.  81.]
 [  1.   4.  16.  64. 256.]
 [  1.   5.  25. 125. 625.]]
```

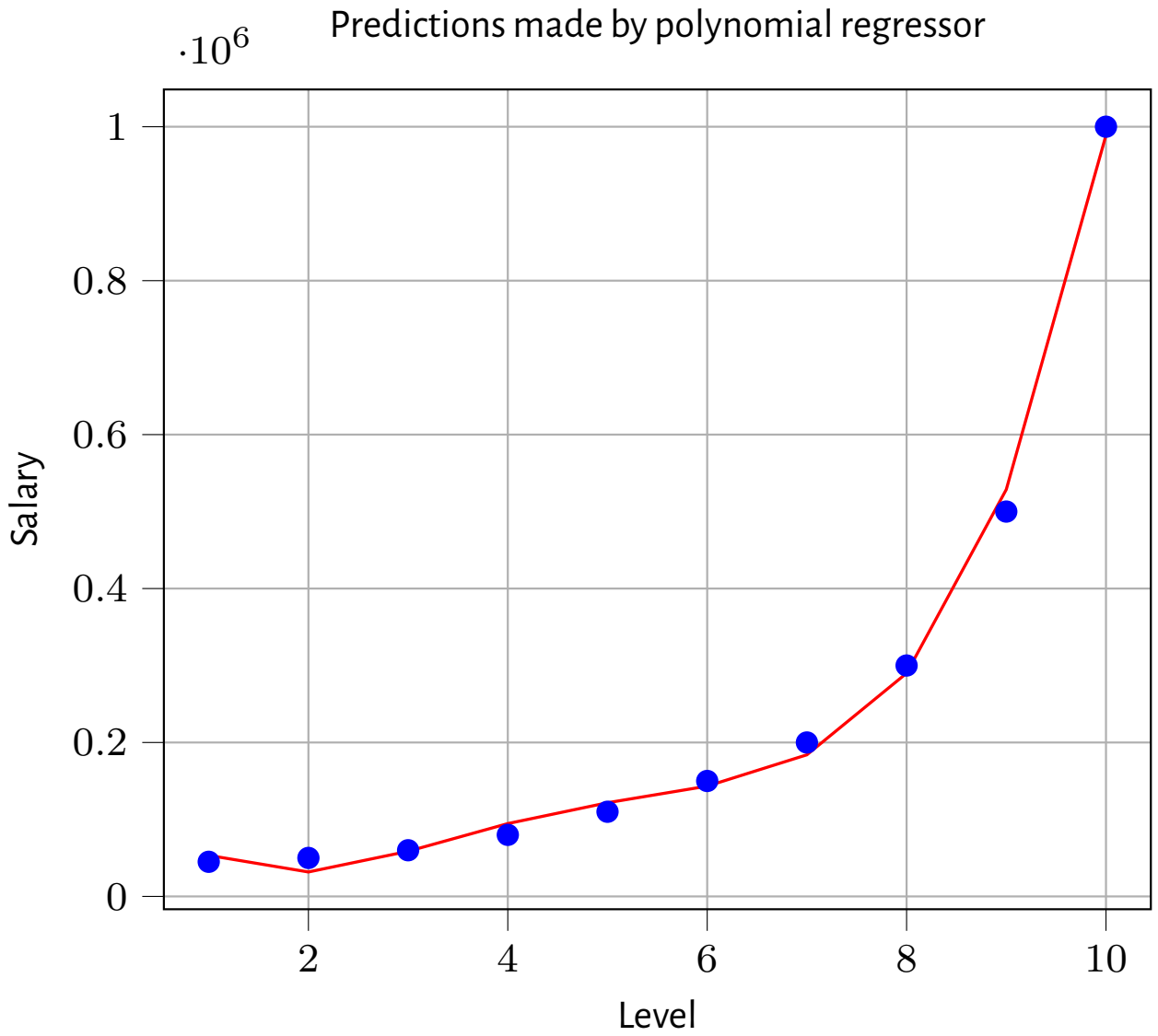
Polynomial Regression (6/8)

[10]:
```
LinearRegression()
```

Visualising the polynomial regression predictions

[11]:
```
plt.scatter(X, y, color='blue')
plt.plot(X, lr_2.predict(poly_reg.fit_transform(X)), color='red')
plt.title('Predictions made by polynomial regressor')
plt.xlabel('Level')
plt.ylabel('Salary')
plt.grid()
```

Polynomial Regression (7/8)



Polynomial Regression (8/8)

Predicting a new result using the linear regressor

[12]:
```
lr_1.predict([[6.5]])
```

[12]:
```
array([330378.78787879])
```

Predicting a new result using the polynomial regressor

[13]:
```
lr_2.predict(poly_reg.fit_transform([[6.5]]))
```

[13]:
```
array([158862.45265155])
```

Task #2

The yield y of a chemical process is a random variable whose value is considered to be a linear function of the temperature x. The following data of corresponding values of x and y is found:

Temperature in °C (x)	0	25	50	75	100
Yield in grams (y)	14	38	54	76	95

The linear regression model $y = \theta_0 + \theta_1 x$ is used. Determine the values of θ_0, θ_1.

1. Using normal equation,
2. Using gradient descent for 5 iterations.

$$y = \begin{bmatrix} 14 \\ 38 \\ 54 \\ 76 \\ 95 \end{bmatrix} \quad \text{et} \quad X = \begin{bmatrix} 1 & 0 \\ 1 & 25 \\ 1 & 50 \\ 1 & 75 \\ 1 & 100 \end{bmatrix} \Longrightarrow X^T X = \begin{bmatrix} 5 & 250 \\ 250 & 18750 \end{bmatrix}$$

$$\hat{\theta} = \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \end{bmatrix} = \begin{bmatrix} 15.4 \\ 0.8 \end{bmatrix}$$

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np

>>> X = np.array([[1,0], [1,25], [1,50], [1,75], [1,100]], dtype=np.float32)
>>> y = np.array([[14], [38], [54], [76], [95]])

>>> # NORMAL EQUATION
>>> XtX = X.T.dot(X)
>>> invXtX = np.linalg.inv(XtX)
>>> t_ne = invXtX.dot(np.matmul(X.T, y))

>>> print(t_ne)
[[15.39999944]
 [ 0.79999982]]

>>> X[:,1]= (X[:,1]-X[:,1].min())/X[:,1].max()
>>> y = (y-y.min())/y.max()

>>> # GRADIENT DESCENT
>>> t_gd = np.array([[1], [1]])
>>> alpha = .1
>>> vect = np.zeros(shape=(2, 1001))
>>> vect[:,0] = t_gd[[0,1],[0]]
>>> lost = []
```

```
>>> for k in range(1000):
...     eps = y-np.matmul(X, t_gd)
...     lost.append(1/(2*len(y))*eps.T.dot(eps)[[0],[0]][0])
...     t_gd = t_gd+alpha*1/len(y)*np.matmul((eps).T, X).T
...     vect[:,k+1] = t_gd[[0,1],[0]]
...
>>> print(vect[:, -1])
[0.01474565 0.84208938]

>>> plt.plot(vect[0, :], label=r'$\hat{\theta}_0$')
[<matplotlib.lines.Line2D object at 0x7fde144cdb80>]
>>> plt.plot(vect[1, :], label=r'$\hat{\theta}_1$')
[<matplotlib.lines.Line2D object at 0x7fde144cdf10>]
>>> plt.legend()
<matplotlib.legend.Legend object at 0x7fde144e41f0>
>>> plt.grid()
>>> plt.show()

>>> plt.plot(lost)
[<matplotlib.lines.Line2D object at 0x7fde000bcf40>]
>>> plt.grid()
>>> plt.show()
```

Next...

Next...

Further Reading

I. El Naqa and M. J. Murphy. "What Is Machine Learning?" In: *Machine Learning in Radiation Oncology: Theory and Applications*. Ed. by I. El Naqa, R. Li, and M. J. Murphy. Cham: Springer International Publishing, 2015, pp. 3–11. DOI: 10.1007/978-3-319-18305-3_1.

J. Schmidt et al. "Recent advances and applications of machine learning in solid-state materials science". In: *npj Computational Materials* 5.1 (Aug. 2019). DOI: 10.1038/s41524-019-0221-0 (cit. on p. 15).