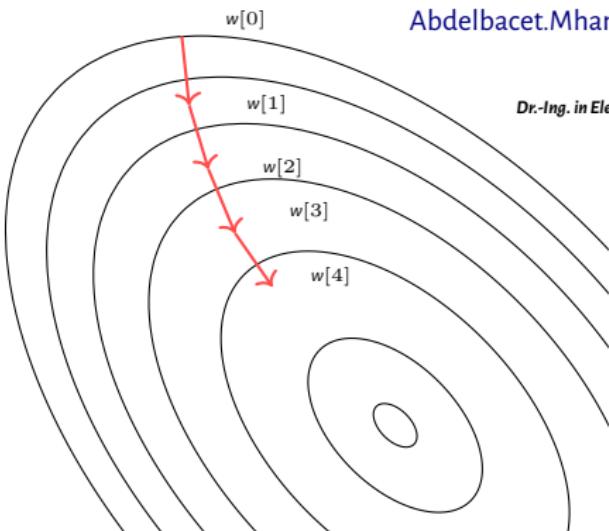


Machine Learning

(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

¹Available @ <https://github.com/a-mhamdi/isetbz/>



Disclaimer

- 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.

- 1 An overview
- 2 Supervised Learning
- 3 Unsupervised Learning
- 4 Deep Learning

Next...

1 An overview

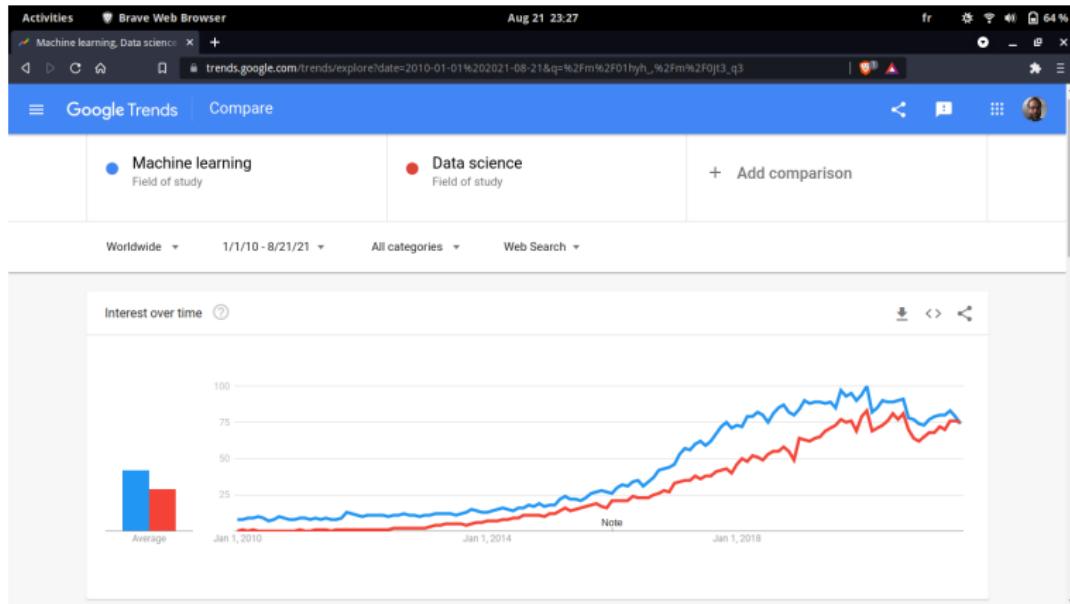
2 Supervised Learning

3 Unsupervised Learning

4 Deep Learning

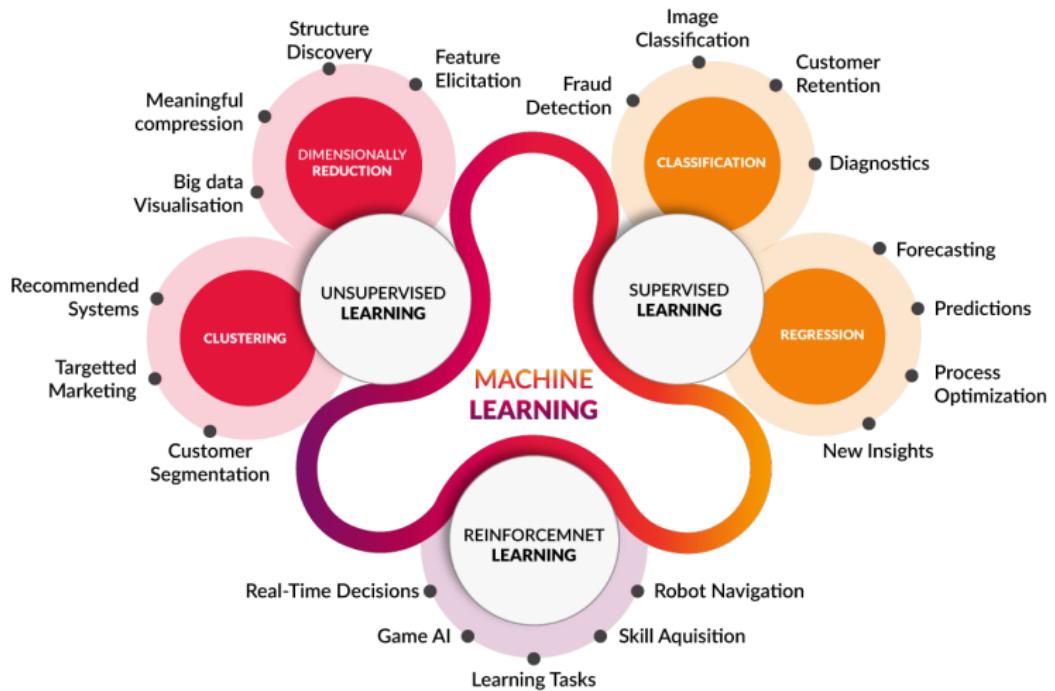
Top uses



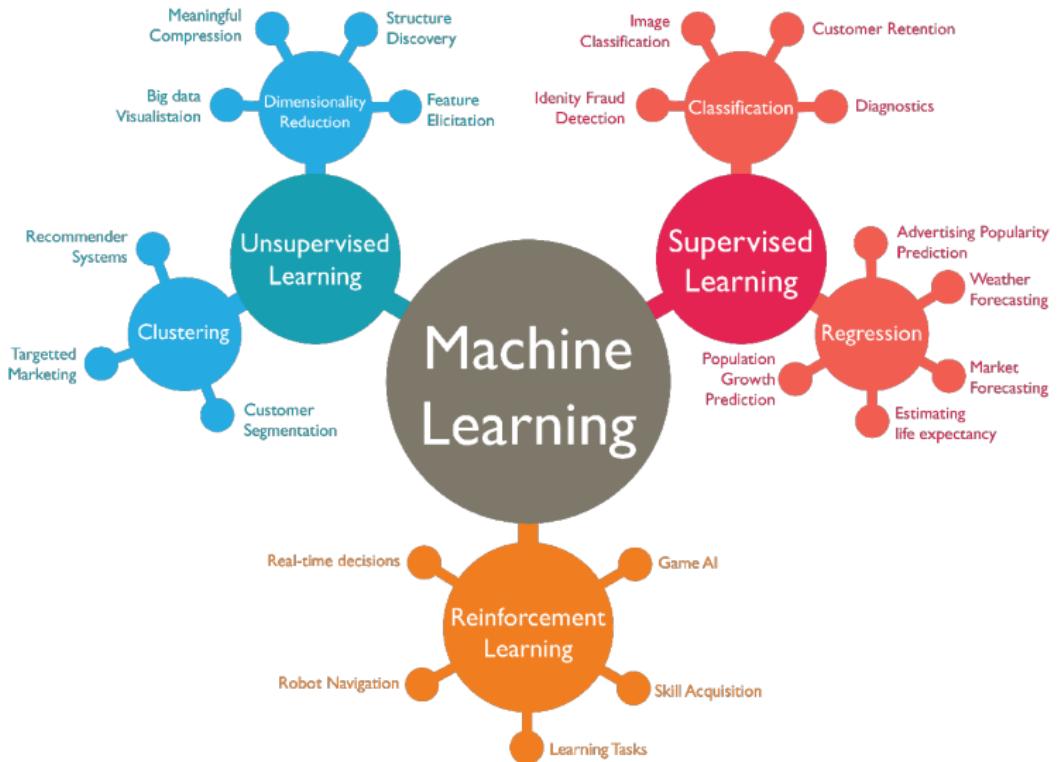


“Numbers represent search interest relative to the highest point on the chart for the given region and time.

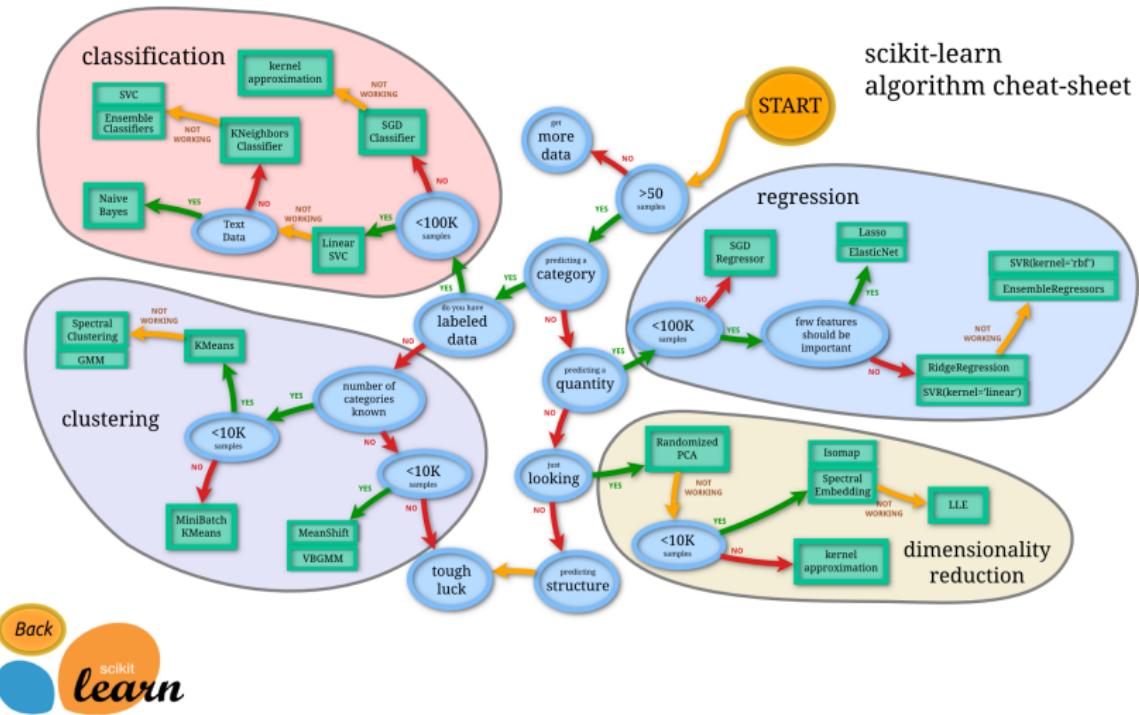
- A value of 100 is the peak popularity for the term;
- A value of 50 means that the term is half as popular;
- A score of 0 means there was not enough data for this term.”



<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

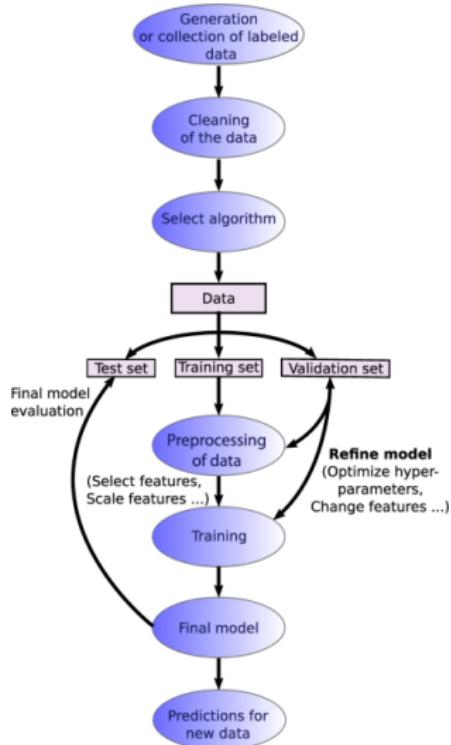
Regression | Classification | Clustering

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



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 the 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 63777.7777777778]
 [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 -0.30786617274297867]
 [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]]
```

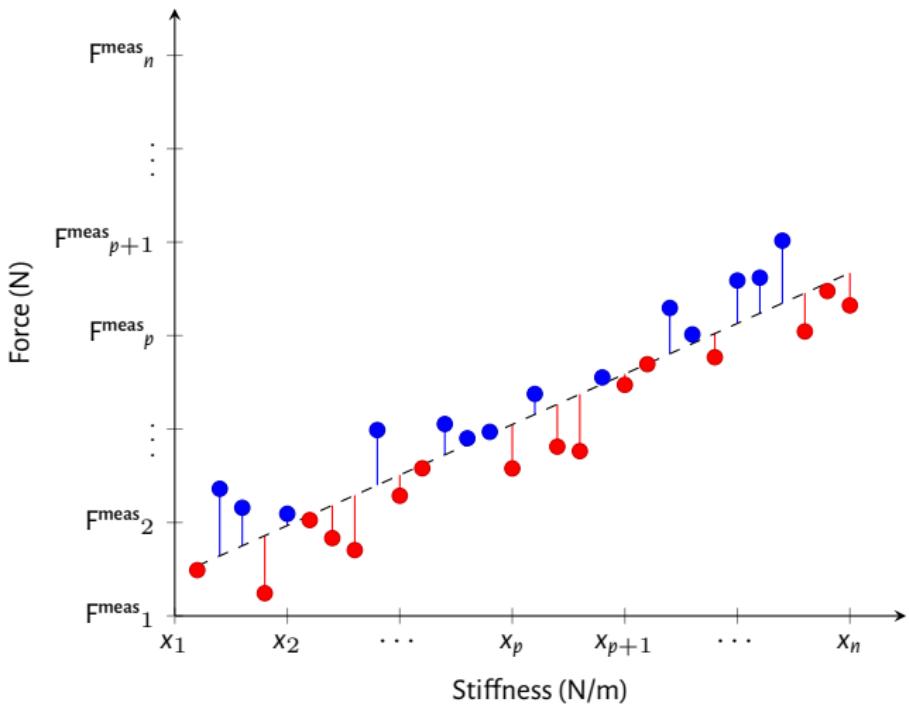
[Next...](#)

1 An overview

2 Supervised Learning

3 Unsupervised Learning

4 Deep Learning



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*):

$$F = kx \quad (1)$$

Restoring force is proportional to displacement.

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

x_i	x_1	\dots	x_p	\dots	x_n
F^{meas}_i	F^{meas}_1	\dots	F^{meas}_p	\dots	F^{meas}_n

$$\begin{aligned} F^{\text{meas}}_i &= F_i + \varepsilon_i \\ &= kx_i + \varepsilon_i, \end{aligned} \quad (2)$$

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:

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

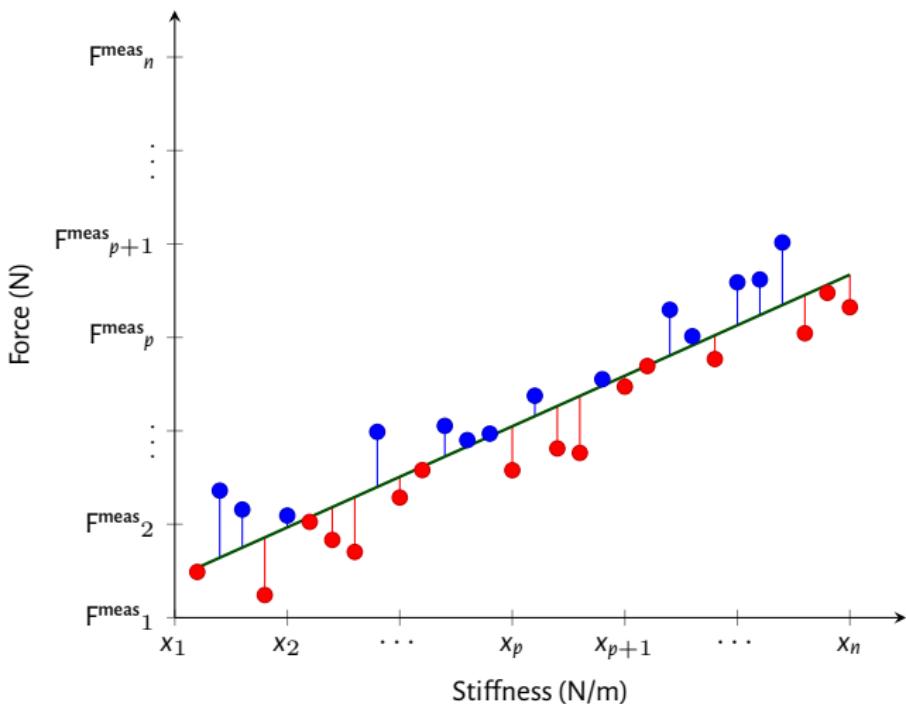
Regression

$$\frac{\partial \mathcal{J}}{\partial k} = 0 \quad (3)$$

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

$$\sum_{i=1}^n (\mathbf{F}^{\text{meas}}_i - kx_i) \sum_{i=1}^n x_i = 0$$

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



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 the 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

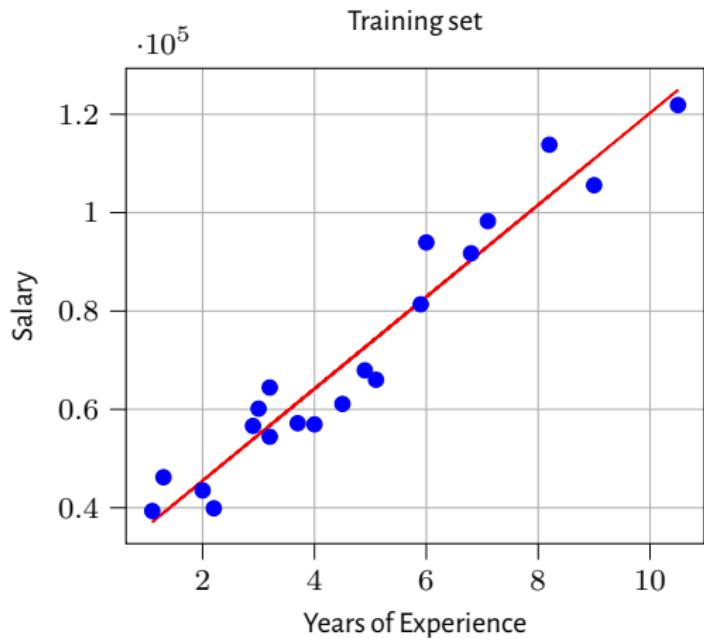
```
[8]: y_pred = regressor.predict(X_test)
```

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)

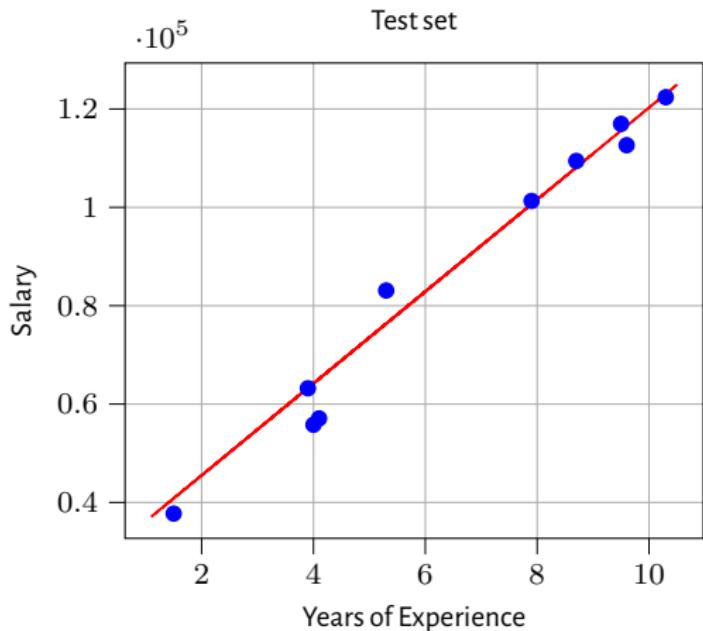


Simple Linear Regression (5/6)

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:

$$y = y_0 + vt \quad (4)$$

Table: Measurements of position y

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

$$\begin{aligned} y^{\text{meas}}_i &= y_i + \varepsilon_i \\ &= y_0 + vt_i + \varepsilon_i, \end{aligned} \quad (5)$$

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 $[y_0, v]^T$, 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 $\varepsilon_i, \forall i \geq 1$:

$$\varepsilon = [\varepsilon_1 \quad \cdots \quad \varepsilon_n]^T$$

$$\frac{\partial \mathcal{J}}{\partial \begin{bmatrix} y_0 \\ v \end{bmatrix}} = 0 \quad (6)$$

Multiple Linear 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/50_Startups.csv')
df.head()
```

	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)
```

Multiple Linear Regression (2/8)

```
[[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']
 [131876.9 99814.71 362861.36 'New York']
 [134615.46 147198.87 127716.82 'California']
 [130298.13 145530.06 323876.68 'Florida']
 [120542.52 148718.95 311613.29 'New York']
 [123334.88 108679.17 304981.62 'California']
 [101913.08 110594.11 229160.95 'Florida']
 [100671.96 91790.61 249744.55 'California']
 [93863.75 127320.38 249839.44 'Florida']
 [91992.39 135495.07 252664.93 'California']
 [119943.24 156547.42 256512.92 'Florida']
 [114523.61 122616.84 261776.23 'New York']
 [78013.11 121597.55 264346.06 'California']
 [94657.16 145077.58 282574.31 'New York']
 [91749.16 114175.79 294919.57 'Florida']
 [86419.7 153514.11 0.0 'New York']
 [76253.86 113867.3 298664.47 'California']
 [78389.47 153773.43 299737.29 'New York']
 [73994.56 122782.75 303319.26 'Florida']]
```

Multiple Linear Regression (3/8)

```
[67532.53 105751.03 304768.73 'Florida']
[77044.01 99281.34 140574.81 'New York']
[64664.71 139553.16 137962.62 'California']
[75328.87 144135.98 134050.07 'Florida']
[72107.6 127864.55 353183.81 'New York']
[66051.52 182645.56 118148.2 'Florida']
[65605.48 153032.06 107138.38 'New York']
[61994.48 115641.28 91131.24 'Florida']
[61136.38 152701.92 88218.23 'New York']
[63408.86 129219.61 46085.25 'California']
[55493.95 103057.49 214634.81 'Florida']
[46426.07 157693.92 210797.67 'California']
[46014.02 85047.44 205517.64 'New York']
[28663.76 127056.21 201126.82 'Florida']
[44069.95 51283.14 197029.42 'California']
[20229.59 65947.93 185265.1 'New York']
[38558.51 82982.09 174999.3 'California']
[28754.33 118546.05 172795.67 'California']
[27892.92 84710.77 164470.71 'Florida']
[23640.93 96189.63 148001.11 'California']
[15505.73 127382.3 35534.17 'New York']
[22177.74 154806.14 28334.72 'California']
[1000.23 124153.04 1903.93 'New York']
```

Multiple Linear Regression (4/8)

```
[1315.46 115816.21 297114.46 'Florida']
[0.0 135426.92 0.0 'California']
[542.05 51743.15 0.0 'New York']
[0.0 116983.8 45173.06 'California']]
```

Encoding categorical data

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

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

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

Multiple Linear Regression (5/8)

```
[[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]
 [0.0 0.0 1.0 131876.9 99814.71 362861.36]
 [1.0 0.0 0.0 134615.46 147198.87 127716.82]
 [0.0 1.0 0.0 130298.13 145530.06 323876.68]
 [0.0 0.0 1.0 120542.52 148718.95 311613.29]
 [1.0 0.0 0.0 123334.88 108679.17 304981.62]
 [0.0 1.0 0.0 101913.08 110594.11 229160.95]
 [1.0 0.0 0.0 100671.96 91790.61 249744.55]
 [0.0 1.0 0.0 93863.75 127320.38 249839.44]
 [1.0 0.0 0.0 91992.39 135495.07 252664.93]
 [0.0 1.0 0.0 119943.24 156547.42 256512.92]
 [0.0 0.0 1.0 114523.61 122616.84 261776.23]
 [1.0 0.0 0.0 78013.11 121597.55 264346.06]
 [0.0 0.0 1.0 94657.16 145077.58 282574.31]
 [0.0 1.0 0.0 91749.16 114175.79 294919.57]
 [0.0 0.0 1.0 86419.7 153514.11 0.0]
 [1.0 0.0 0.0 76253.86 113867.3 298664.47]
 [0.0 0.0 1.0 78389.47 153773.43 299737.29]
 [0.0 1.0 0.0 73994.56 122782.75 303319.26]]
```

Multiple Linear Regression (6/8)

```
[0.0 1.0 0.0 67532.53 105751.03 304768.73]
[0.0 0.0 1.0 77044.01 99281.34 140574.81]
[1.0 0.0 0.0 64664.71 139553.16 137962.62]
[0.0 1.0 0.0 75328.87 144135.98 134050.07]
[0.0 0.0 1.0 72107.6 127864.55 353183.81]
[0.0 1.0 0.0 66051.52 182645.56 118148.2]
[0.0 0.0 1.0 65605.48 153032.06 107138.38]
[0.0 1.0 0.0 61994.48 115641.28 91131.24]
[0.0 0.0 1.0 61136.38 152701.92 88218.23]
[1.0 0.0 0.0 63408.86 129219.61 46085.25]
[0.0 1.0 0.0 55493.95 103057.49 214634.81]
[1.0 0.0 0.0 46426.07 157693.92 210797.67]
[0.0 0.0 1.0 46014.02 85047.44 205517.64]
[0.0 1.0 0.0 28663.76 127056.21 201126.82]
[1.0 0.0 0.0 44069.95 51283.14 197029.42]
[0.0 0.0 1.0 20229.59 65947.93 185265.1]
[1.0 0.0 0.0 38558.51 82982.09 174999.3]
[1.0 0.0 0.0 28754.33 118546.05 172795.67]
[0.0 1.0 0.0 27892.92 84710.77 164470.71]
[1.0 0.0 0.0 23640.93 96189.63 148001.11]
[0.0 0.0 1.0 15505.73 127382.3 35534.17]
[1.0 0.0 0.0 22177.74 154806.14 28334.72]
[0.0 0.0 1.0 1000.23 124153.04 1903.93]
```

Multiple Linear Regression (7/8)

```
[0.0 1.0 0.0 1315.46 115816.21 297114.46]  
[1.0 0.0 0.0 0.0 135426.92 0.0]  
[0.0 0.0 1.0 542.05 51743.15 0.0]  
[1.0 0.0 0.0 0.0 116983.8 45173.06]]
```

Splitting the dataset into the Training set and Test set

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

```
[9]: 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

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

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

```
[11]: LinearRegression()
```

Predicting the Test set results

Multiple Linear Regression (8/8)

```
[12]: y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.
    ↪reshape(len(y_test),1)),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]]
```

[Next...](#)

- 1 An overview
- 2 Supervised Learning
- 3 Unsupervised Learning
- 4 Deep Learning

Next...

- 1 An overview
- 2 Supervised Learning
- 3 Unsupervised Learning
- 4 Deep Learning



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](https://doi.org/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](https://doi.org/10.1038/s41524-019-0221-0) (cit. on p. 11).