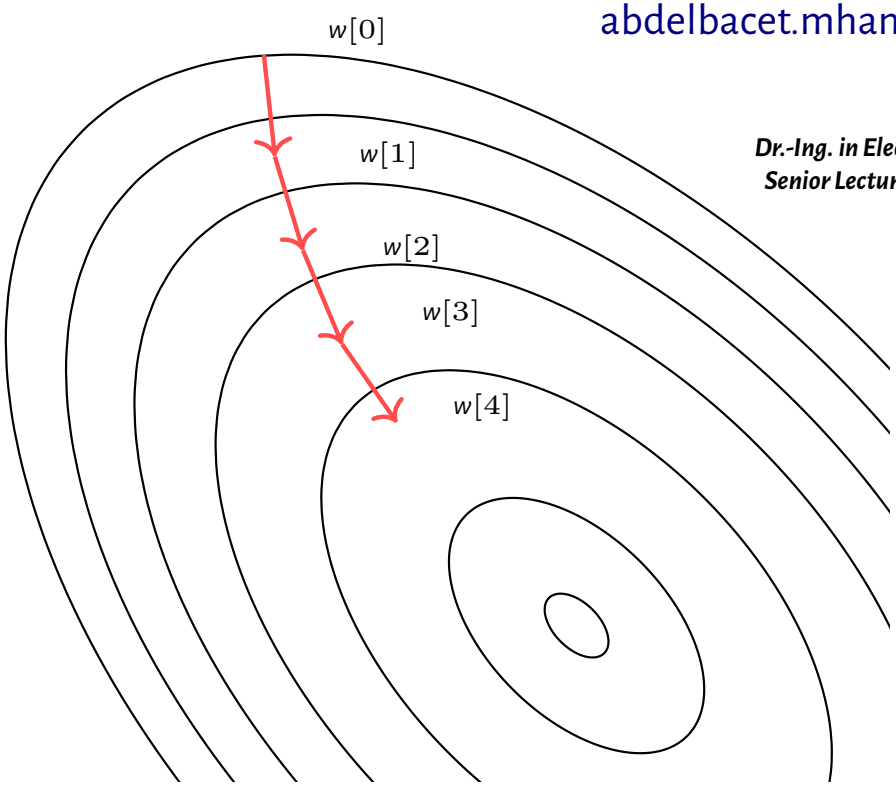
# An Introduction To Machine Learning Sorcery

## (A Demystification Draft)[1]

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

***Dr.-Ing. in Electrical Engineering***
***Senior Lecturer at ISET Bizerte***




"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...

# Next...

Consider the following multivariate equation:

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

For a particular single measurement, eq. (1) can be updated as

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

We denote hereafter by $\theta$ 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 $\theta$:

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

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 $\theta$:

$$\frac{\partial \mathcal{J}}{\partial \theta} = \frac{\partial \left(\varepsilon^T \varepsilon\right)}{\partial \theta} \tag{3}$$

$$\frac{\partial \left(\varepsilon^T \varepsilon\right)}{\partial \theta} = 2 \left(\frac{\partial \varepsilon}{\partial \theta}\right)^T \varepsilon \tag{4}$$

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

$$\frac{\partial \varepsilon}{\partial \theta} = -X \tag{5}$$

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

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

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

$$\theta_0 \leftarrow \theta_0 + \alpha \frac{1}{n} \sum_{k=1}^{n} (y_k - h_\theta(x_k))\, x_{(0,\,k)} \qquad \theta_1 \leftarrow \theta_1 + \alpha \frac{1}{n} \sum_{k=1}^{n} (y_k - h_\theta(x_k))\, x_{(1,\,k)}$$

$$\vdots$$

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

# 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)
```

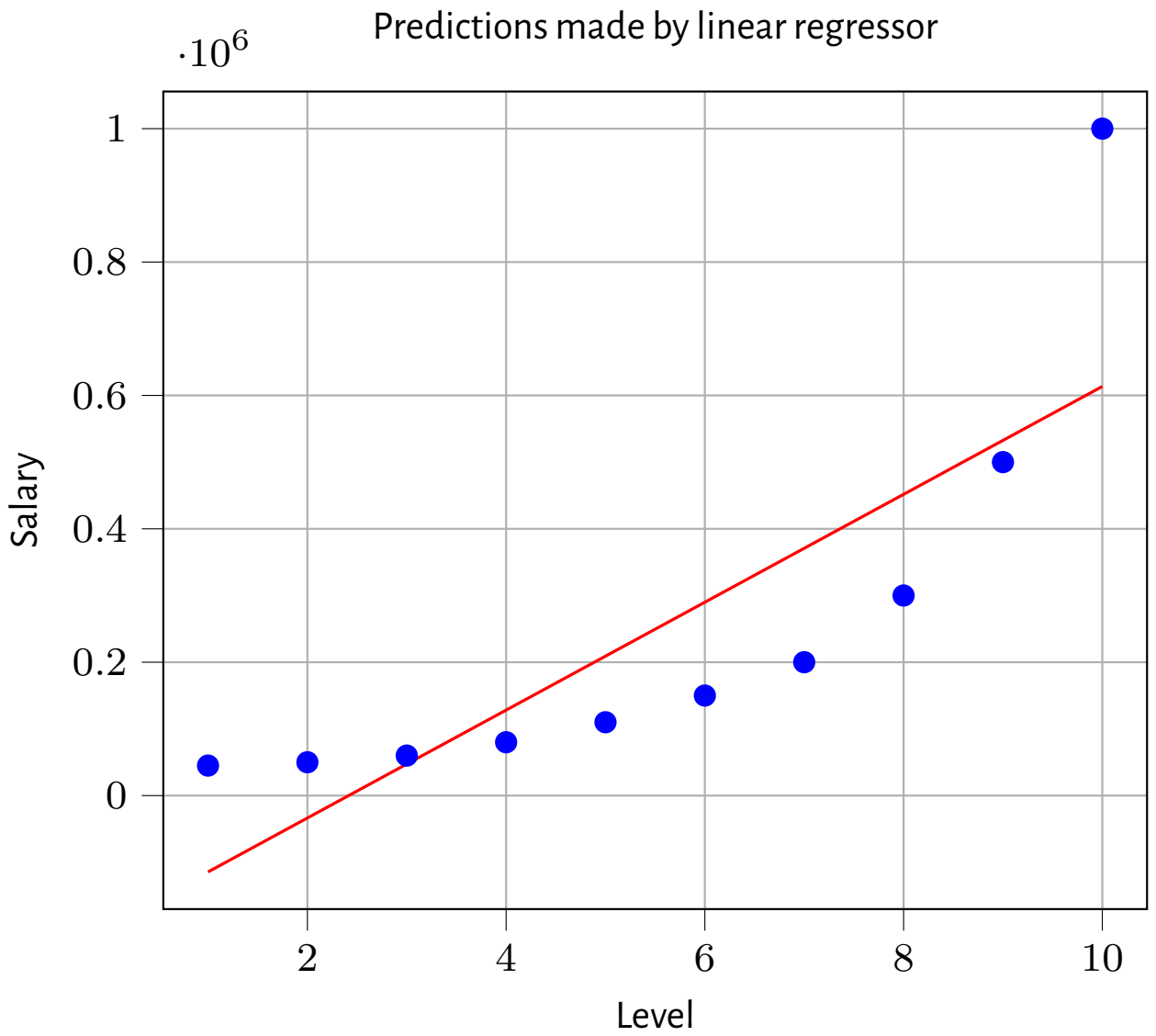
[7]:
```
LinearRegression()
```

**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.]]
```

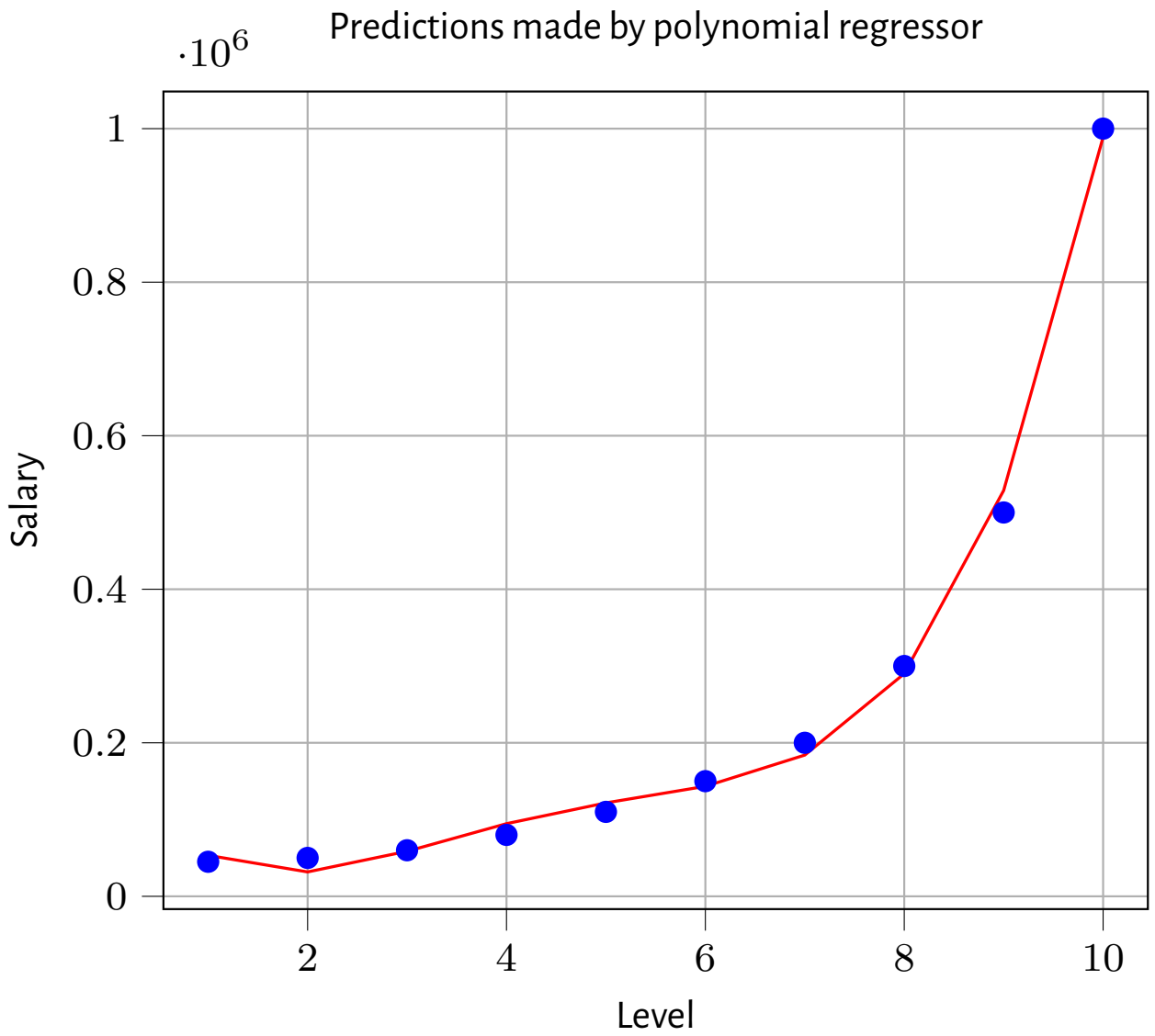
[10]:
```
LinearRegression()
```

**Visualising the polynomial regression predictions**

# Polynomial Regression (6/8)

[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 #1

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 $\theta_0$, $\theta_1$.

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 0x7f4a79d726e0>]
>>> plt.plot(vect[1, :], label=r'$\hat{\theta}_1$')
[<matplotlib.lines.Line2D object at 0x7f4a79d72950>]
>>> plt.legend()
<matplotlib.legend.Legend object at 0x7f4a79d715a0>
>>> plt.grid()
>>> plt.show()

>>> plt.plot(lost)
[<matplotlib.lines.Line2D object at 0x7f4a48c4a3e0>]
>>> plt.grid()
>>> plt.show()
```

**F1-Score**, **Accuracy**, **Recall** and **Precision** are calculated as follow:

$$f1 - \text{score} = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$$

$f1 -$ score denotes the *Harmonic Mean of Recall & Precision*

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

It denotes the ratio of how much we got right over all cases. Recall, on the other hand, designates the ratio of how much positives do we got right over all actual positive cases.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

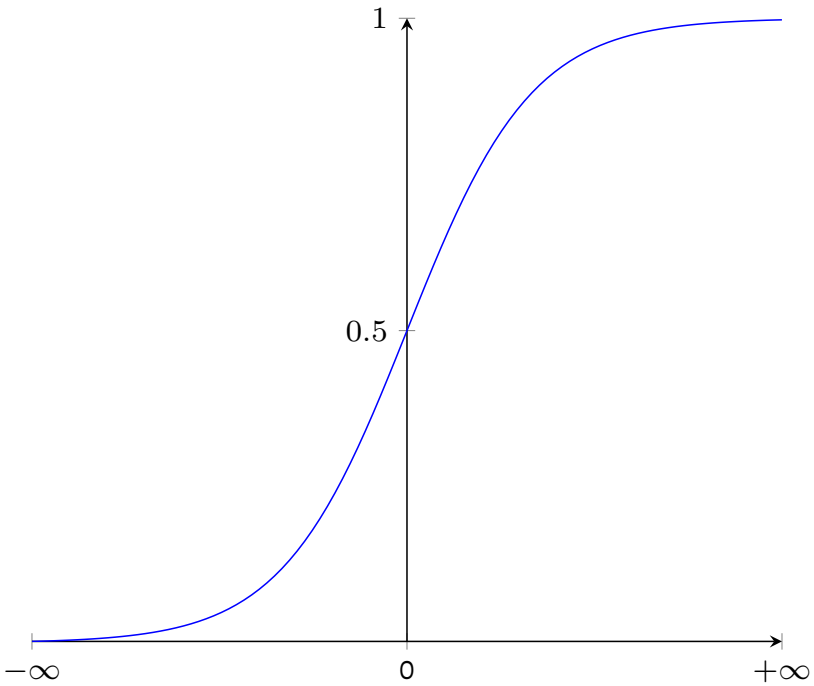
Precision, at last, is how much positives we got right over all positive predictions. It is given by:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

| Method | Pros | | Cons | |
|---|---|---|---|---|
| *Logistic Regression* | ▲ | Probabilistic | ▼ | almost linearly separable data |
| *K-NN* | ▲ | Simple | ▼ | Number of neighbors *k* |
| | ▲ | Fast | | |
| | ▲ | Efficient | | |
| *SVM* | ▲ | *** | ▼ | *** |
| | ▲ | *** | ▼ | *** |
| *Kernel SVM* | ▲ | *** | ▼ | *** |
| | ▲ | *** | ▼ | *** |
| *Naive Bayes* | ▲ | *** | ▼ | *** |
| | ▲ | *** | ▼ | *** |
| *Decision Tree Classification* | ▲ | *** | ▼ | *** |
| | ▲ | *** | ▼ | *** |
| *Random Forest Classification* | ▲ | *** | ▼ | *** |
| | ▲ | *** | ▼ | *** |

# Logistic or S-shaped function $\sigma$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$




- $\sigma$ squashes range of distance from $]-\infty, +\infty[$ to $[0, 1]$
- $\sigma$ is differentiable and easy to compute:

$$\dot{\sigma} = \sigma \times (1 - \sigma)$$

# Decision boundary

$$y = \sigma\left(\theta_1 x_{(1)} + \theta_2 x_{(2)} + \cdots + \theta_m x_{(m)}\right)$$

$$y = \frac{1}{1 + \mathrm{e}^{-\theta^T x}}$$

Hypothesis:

$$h_\theta(x) = \frac{1}{1 + \mathrm{e}^{-\theta^T x}} \qquad h_\theta(x_k) = \frac{1}{1 + \mathrm{e}^{-\theta^T x_k}}$$

Cost function:

$$\mathcal{J} = \begin{cases} -\ln\left(h_\theta(x)\right) & \text{if} \quad y = 1 \\ -\ln\left(1 - h_\theta(x)\right) & \text{if} \quad y = 0 \end{cases}$$

$$\boxed{\mathcal{J} = -y \ln\left(h_\theta(x)\right) - (1 - y) \ln\left(1 - h_\theta(x)\right)}$$

# Gradient Descent

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

Generalizing $\mathcal{J}$ yields: $\mathcal{J} = -\frac{1}{n}\sum_{k=1}^{n}\left(y_k \ln\left(h_\theta(x_k)\right) + (1-y_k)\ln\left(1-h_\theta(x_k)\right)\right)$

$$\Longrightarrow \quad \frac{\partial \mathcal{J}}{\partial \theta_i} = -\frac{1}{n}\sum_{k=1}^{n}\left(y_k - h_\theta(x_k)\right)x_{(i,\,k)}$$

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

$$\theta_0 \leftarrow \theta_0 + \alpha\frac{1}{n}\sum_{k=1}^{n}\left(y_k - h_\theta\left(x_k\right)\right)x_{(0,\,k)} \qquad \theta_1 \leftarrow \theta_1 + \alpha\frac{1}{n}\sum_{k=1}^{n}\left(y_k - h_\theta\left(x_k\right)\right)x_{(1,\,k)}$$

$$\vdots$$

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

# Logistic 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/Social_Network_Ads.csv')
df.head()
```

[2]:

| | Age | EstimatedSalary | Purchased |
|---|---|---|---|
| 0 | 19 | 19000 | 0 |
| 1 | 35 | 20000 | 0 |
| 2 | 26 | 43000 | 0 |
| 3 | 27 | 57000 | 0 |
| 4 | 19 | 76000 | 0 |

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

### Splitting the dataset into the Training set and Test set

# Logistic Regression (2/8)

[4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪25, random_state=0)
```

**Feature Scaling**

[5]:
```
from sklearn.preprocessing import StandardScaler
```

[6]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

**Training the Logistic Regression model on the Training set**

[7]:
```
from sklearn.linear_model import LogisticRegression
```

[8]:
```
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

[8]:
```
LogisticRegression(random_state=0)
```

**Predicting a new result**

# Logistic Regression (3/8)

[9]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

**Predicting the Test set results**

[10]:
```
y_pred = classifier.predict(X_test)
#print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.
↪reshape(len(y_test),1)),1))
```

**Making the Confusion Matrix**

[11]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

[12]:
```
confusion_matrix(y_test, y_pred)
```

[12]:
```
array([[65,  3],
       [ 8, 24]])
```

[13]:
```
accuracy_score(y_test, y_pred)
```

[13]:
```
0.89
```

**Map of Training set results**

# Logistic Regression (4/8)

[14]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =␣
↪X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop␣
↪= X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.
↪ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =␣
↪ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.grid()
```

# Logistic Regression (5/8)

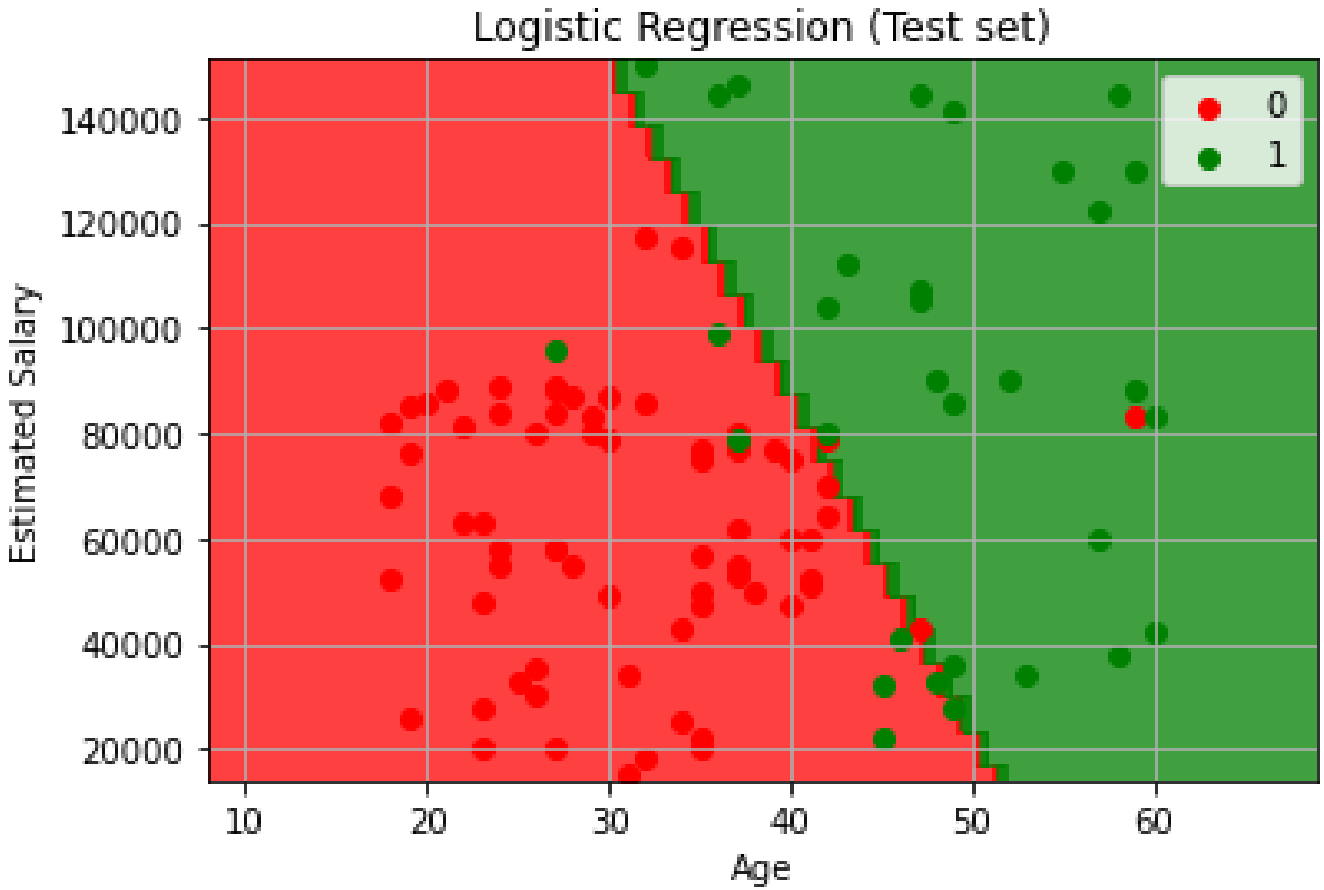
# Logistic Regression (6/8)

**Map of Test set results**

# Logistic Regression (7/8)

[ ]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =␣
↪X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop␣
↪= X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.
↪ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =␣
↪ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.grid()
```

# Logistic Regression (8/8)



# K Nearest Neighbors (1/10)

### 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/Social_Network_Ads.csv')
df.head()
```

[2]:

| | Age | EstimatedSalary | Purchased |
|---|---|---|---|
| 0 | 19 | 19000 | 0 |
| 1 | 35 | 20000 | 0 |
| 2 | 26 | 43000 | 0 |
| 3 | 27 | 57000 | 0 |
| 4 | 19 | 76000 | 0 |

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

### Splitting the dataset into the Training set and Test set

# K Nearest Neighbors (2/10)

[4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪25, random_state=0)
```

**Feature Scaling**

[5]:
```
from sklearn.preprocessing import StandardScaler
```

[6]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

**Training the K-NN model on the Training set**

[7]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[8]:
```
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski',␣
↪p=2)
```

[9]:
```
classifier.fit(X_train, y_train)
```

[9]:
```
KNeighborsClassifier()
```

# K Nearest Neighbors (3/10)

### Predicting a new result

[10]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

### Predicting the Test set results

[11]:
```
y_pred = classifier.predict(X_test)
#print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.
↪reshape(len(y_test),1)),1))
```

### Displaying the Confusion Matrix

[12]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

[13]:
```
confusion_matrix(y_test, y_pred)
```

[13]:
```
array([[64,  4],
       [ 3, 29]])
```

[14]:
```
accuracy_score(y_test, y_pred)
```

# K Nearest Neighbors (4/10)

```
[14]: 0.93
```

**Map of Training set results**

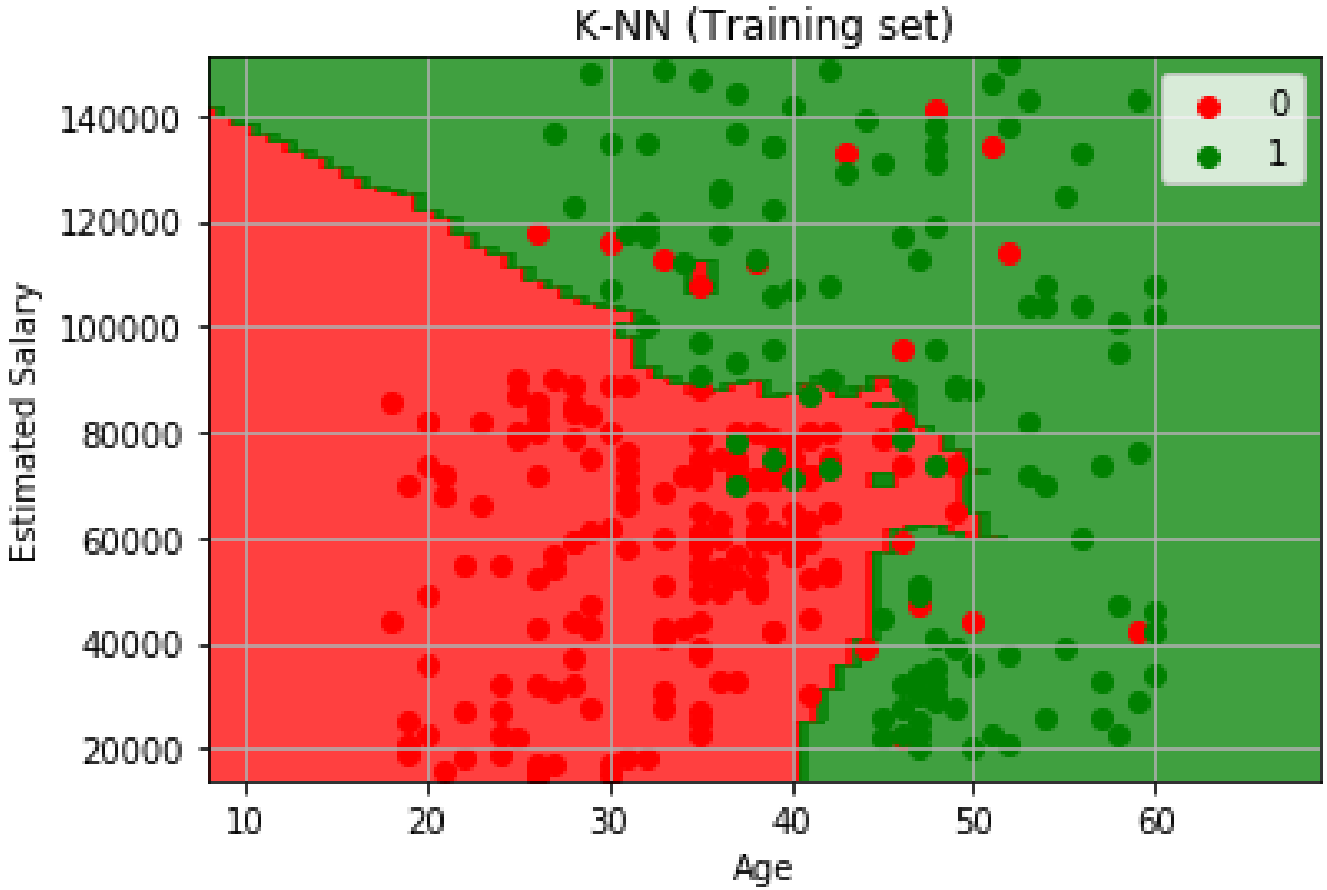
# K Nearest Neighbors (5/10)

[15]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =␣
↪X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop␣
↪= X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.
↪ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =␣
↪ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.grid()
```

# K Nearest Neighbors (6/10)

```
*c* argument looks like a single numeric RGB or RGBA sequence, which␣
↪should be
avoided as value-mapping will have precedence in case its length␣
↪matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D␣
↪array with a
single row if you intend to specify the same RGB or RGBA value for all␣
↪points.
*c* argument looks like a single numeric RGB or RGBA sequence, which␣
↪should be
avoided as value-mapping will have precedence in case its length␣
↪matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D␣
↪array with a
single row if you intend to specify the same RGB or RGBA value for all␣
↪points.
m
```

# K Nearest Neighbors (7/10)



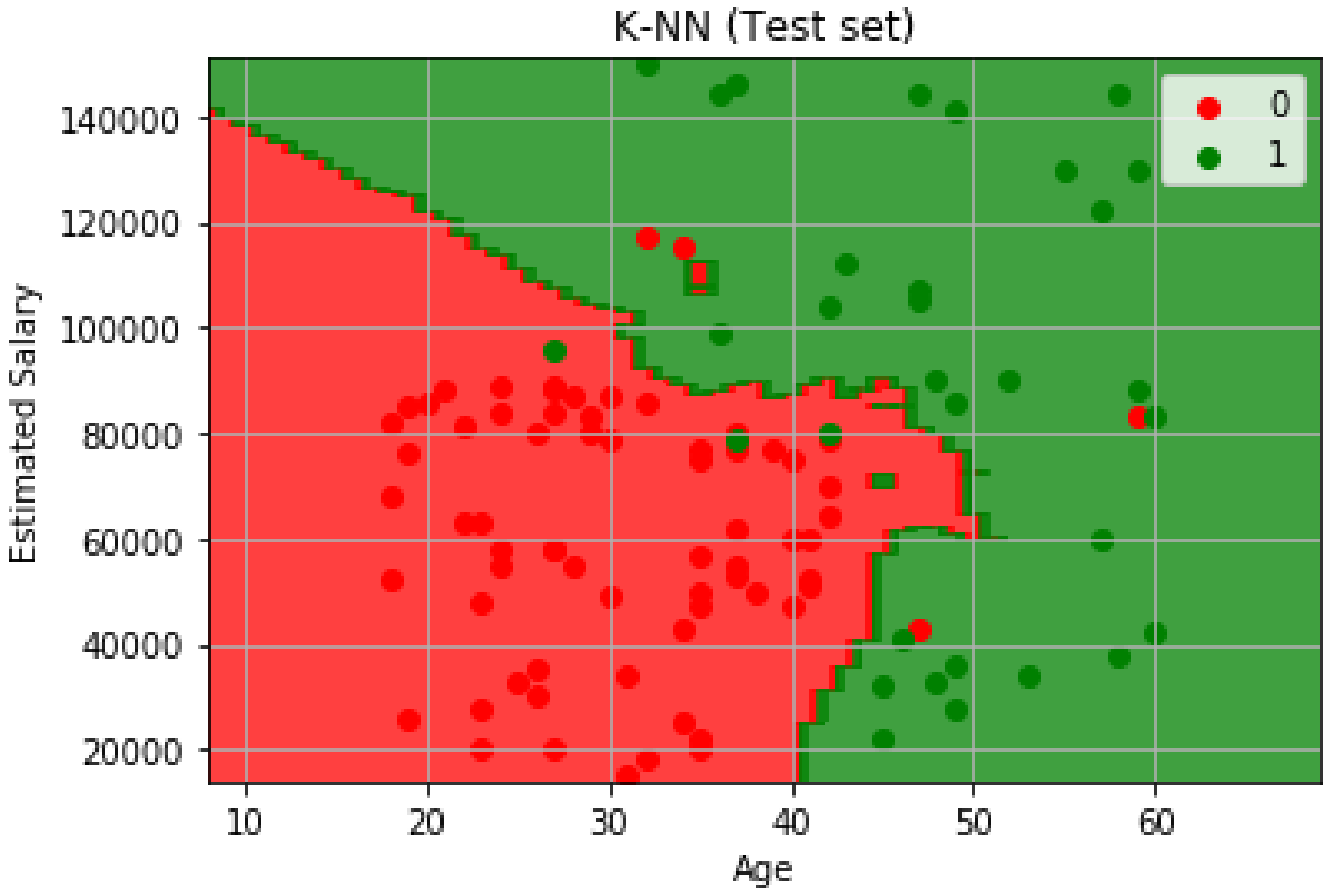
# K Nearest Neighbors (8/10)

**Map of Test set results**

# K Nearest Neighbors (9/10)

[16 ]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =␣
↪X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop␣
↪= X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.
↪ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =␣
↪ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.grid()
```

# K Nearest Neighbors (10/10)



# Next...

# KMeans (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/Mall_Customers.csv')
df.head()
```

[2]:

| | CustomerID | Genre | Age | Annual Income (k$) | Spending Score (1-100) |
|---|---|---|---|---|---|
| 0 | 1 | Male | 19 | 15 | 39 |
| 1 | 2 | Male | 21 | 15 | 81 |
| 2 | 3 | Female | 20 | 16 | 6 |
| 3 | 4 | Female | 23 | 16 | 77 |
| 4 | 5 | Female | 31 | 17 | 40 |

[3]:
```
X = df.iloc[:, [3, 4]].values
```

### Import `KMeans` class

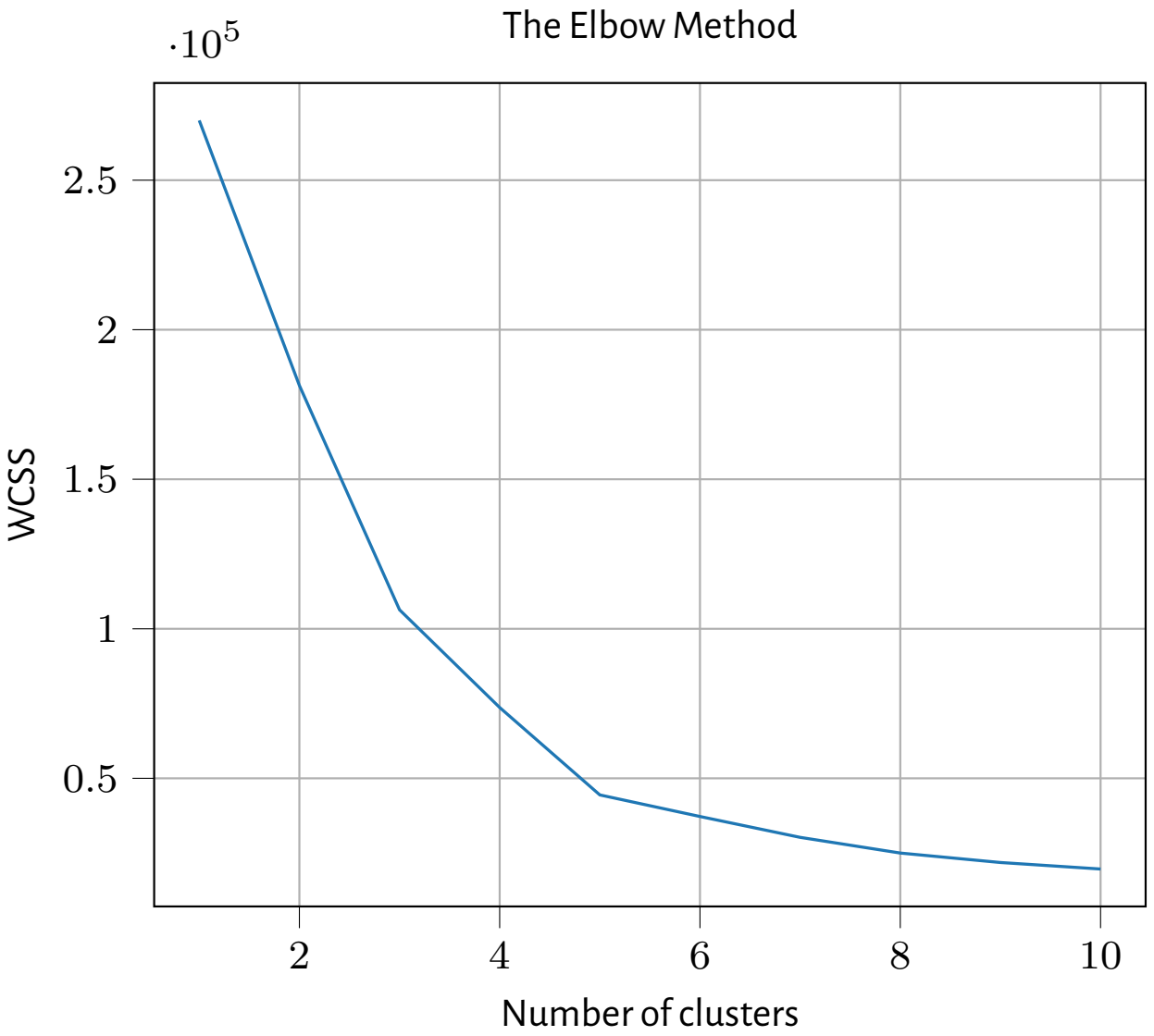
[4]:
```
from sklearn.cluster import KMeans
```

# KMeans (2/6)

**Using the elbow method to find the optimal number of clusters**

[5]:
```
wcss = []
for i in range(1, 11):
kmeans = KMeans(n_clusters=i,
    init='k-means++', # Init method
    random_state=42) # Random seed for reproducibility
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.grid()
```

# KMeans (3/6)


The Elbow Method
$\cdot 10^5$
2.5
2
1.5
1
0.5
WCSS
2
4
6
8
10
Number of clusters

# KMeans (4/6)

**Training the K-Means model on the dataset**

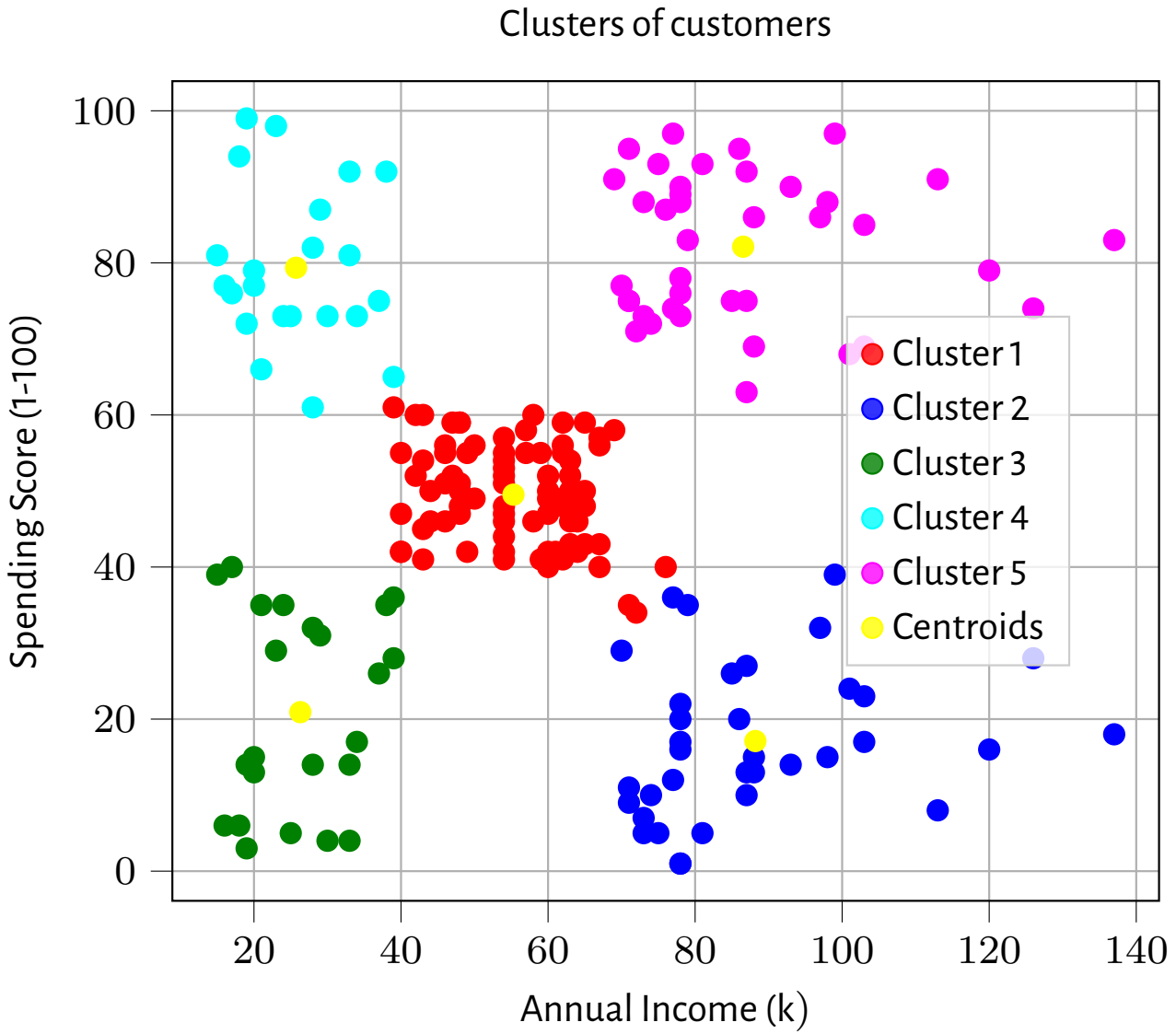
[6]:
```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

**Visualizing the clusters**

# KMeans (5/6)

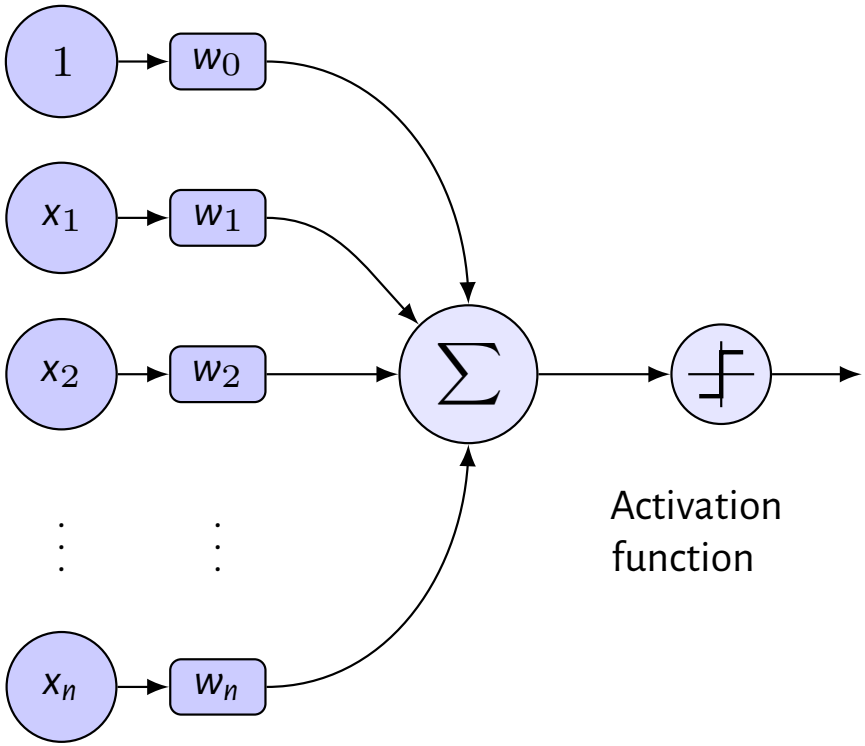
[7]:
```
plt.scatter(X[y_kmeans==0,0], X[y_kmeans==0,1],
    s=100, c='red', label='Cluster 1')
plt.scatter(X[y_kmeans==1,0], X[y_kmeans==1,1],
    s=100, c='blue', label='Cluster 2')
plt.scatter(X[y_kmeans==2,0], X[y_kmeans==2,1],
    s=100, c='green', label='Cluster 3')
plt.scatter(X[y_kmeans==3,0], X[y_kmeans==3,1],
    s=100, c='cyan', label='Cluster 4')
plt.scatter(X[y_kmeans==4,0], X[y_kmeans==4,1],
    s=100, c='magenta', label='Cluster 5')

plt.scatter(kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1],
    s=300, c='yellow', label='Centroids')

plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.grid()
```

# KMeans (6/6)

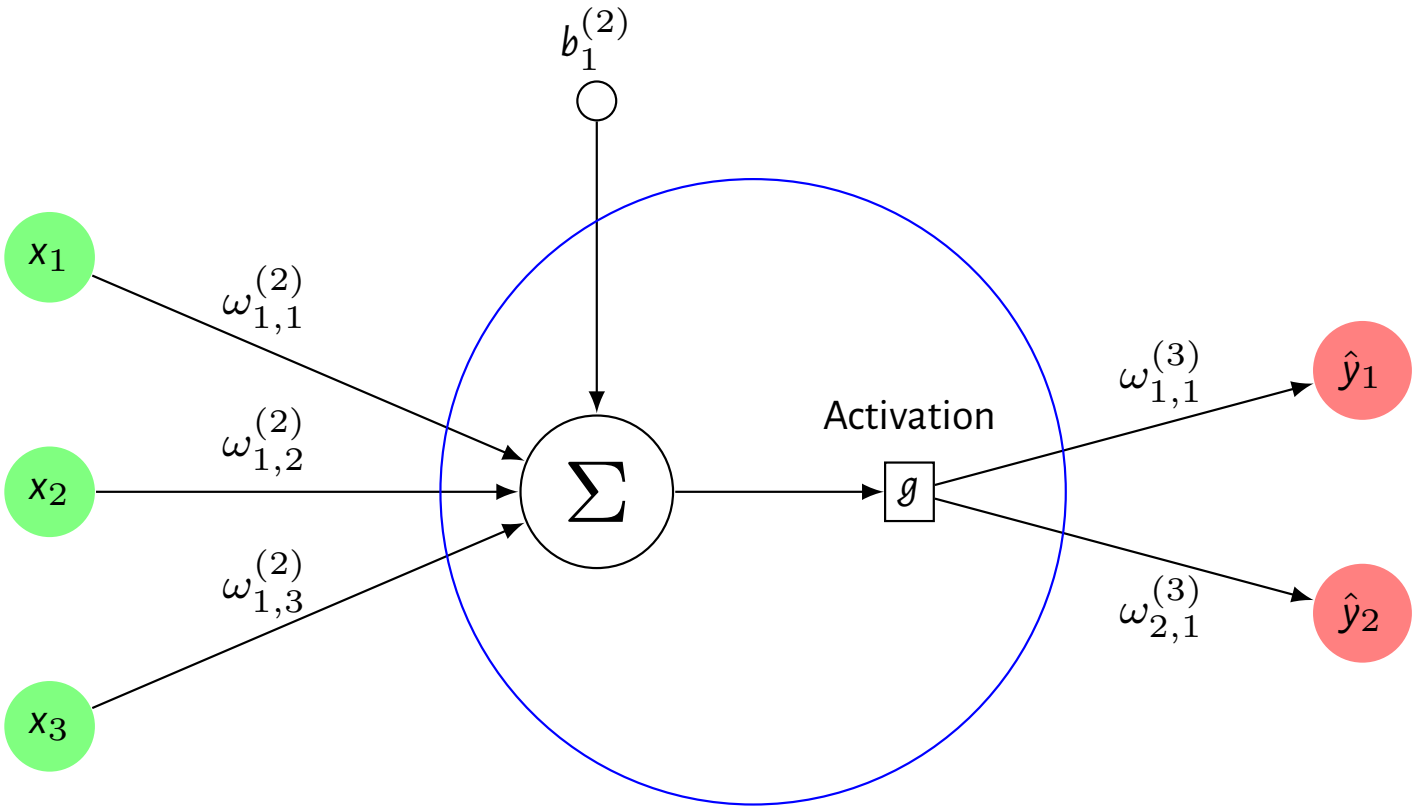

Clusters of customers
Spending Score (1-100)
Annual Income (k)
100
80
60
40
20
0
20
40
60
80
100
120
140
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids

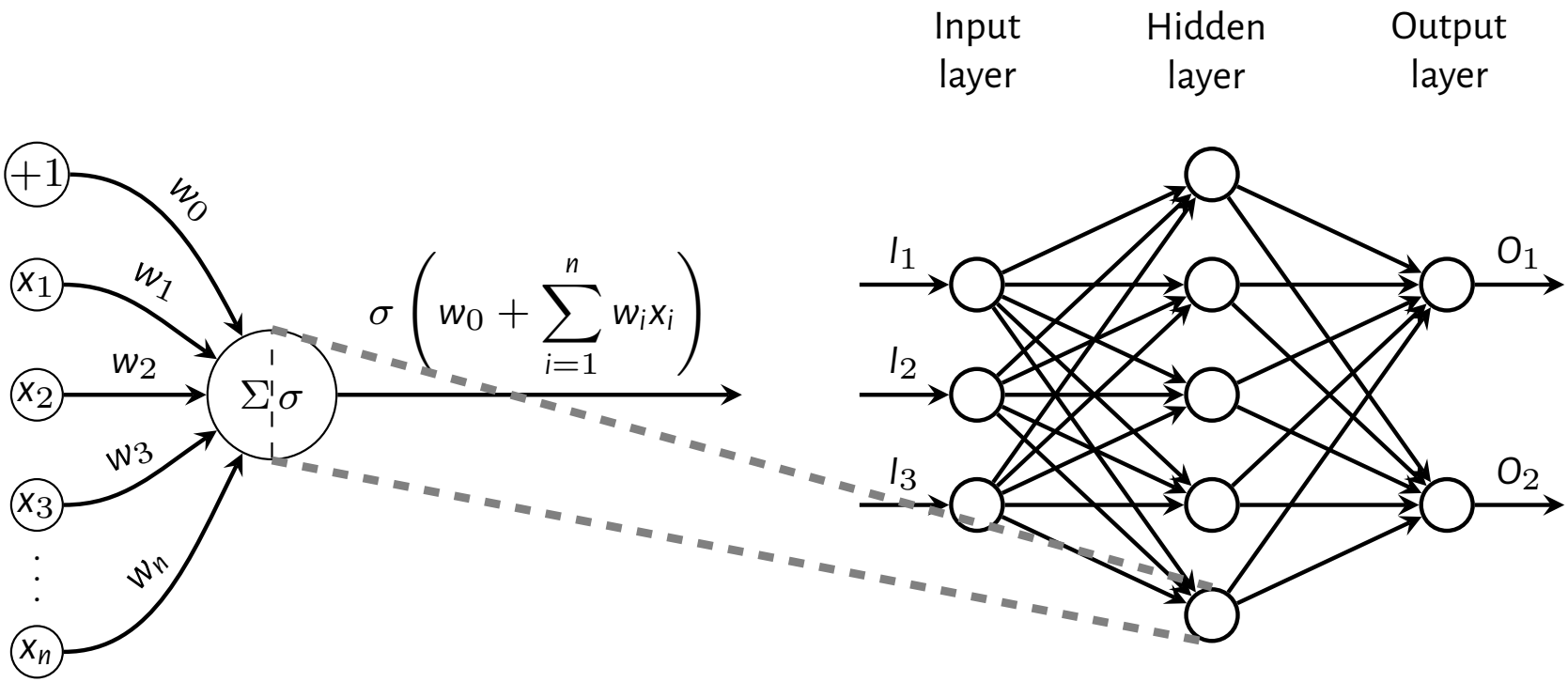
# Next...

# Fundamental unit of a neural network $(1/2)$

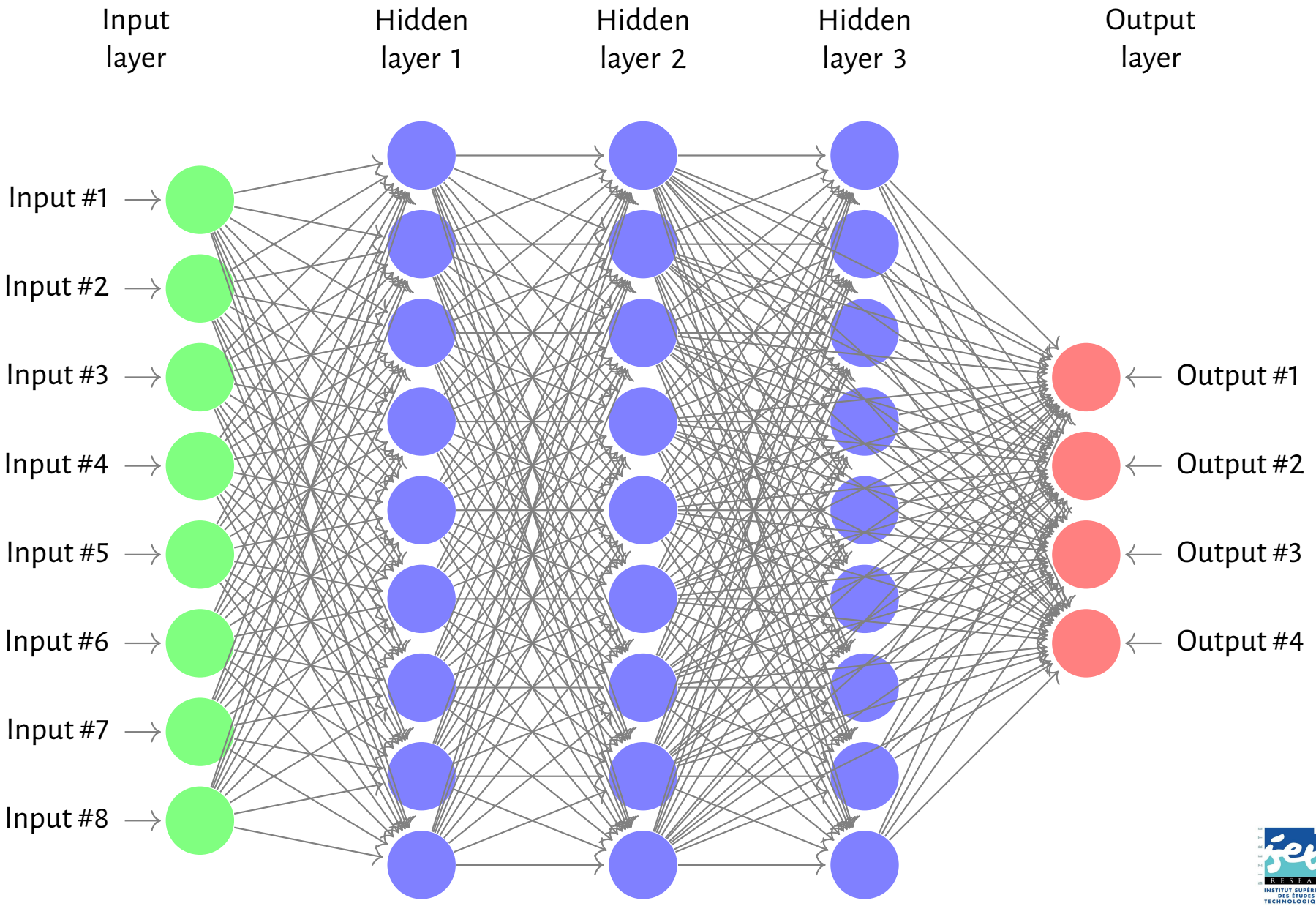


# Fundamental unit of a neural network $(2/2)$




Input layer
Hidden layer
Output layer
+1
$w_0$
$x_1$
$w_1$
$x_2$
$w_2$
$x_3$
$w_3$
$x_n$
$w_n$
$\Sigma$
$\sigma$
$\sigma\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$
$I_1$
$I_2$
$I_3$
$O_1$
$O_2$

# Multilayer Perceptron (MLP)



# Further Reading


[ENM15] 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.

[Sch+19] 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.