



## **MOOC Report**

**Machine Learning A-Z™: Hands-On Python & R In Data Science**

**Written by: ASSABBANE Mehdi**

**Tutor: LIM Thomas**

## Table of Contents

Introduction.....	5
Technical tools.....	5
1. Tools:.....	5
2. Main libraries: .....	5
Part 1 - Data Preprocessing .....	6
1. Describing dataset :.....	6
2. Importing Data : .....	6
3. Taking care of missing Data: .....	6
4. Encoding categorical data (matrix of features).....	7
5. Encoding categorical data (independent variable).....	7
6. Feature Scaling: .....	7
a. Splitting dataset .....	8
b. Feature Scaling .....	8
Part 2 - Regression:.....	9
1. Simple Linear Regression: .....	9
a. An example of a Simple Linear Regression.....	9
b. Ordinary Least Squares .....	10
c. Python Example .....	10
2. Multiple Linear Regression.....	12
a. Statistical significance(P-Value) .....	13
b. Build a model .....	14
c. Method 1: All-in .....	14
d. Method 2 – backward Elimination .....	14
e. Method 3 – Forward Selection .....	14
f. Method 4 - Bidirectional Elimination .....	14
g. Method 5 - All Possible Models (score comparison) .....	15
3. Polynomial Regression .....	15
4. SVR Intuition .....	15
5. Non-linear SVR .....	16
6. Decision Tree Regression .....	17
a. Most common algorithms and software for building decision trees: .....	17
b. Further more about used algorithms : .....	17
7. Random Forest Regression.....	18
8. R squared .....	18
a. How to calculate R2?.....	19
a. Interpret R-squared.....	20
9. Adjusted R-squared.....	20
Part 3 – Classification.....	21
1. Logistic regression.....	21
a. So when we are going to use classification now? .....	22
2. K-Nearest Neighbors .....	23
a. Where to use KNN.....	24

b.	The Mathematics Behind KNN .....	24
3.	Naïve Bayes .....	25
4.	Decision tree .....	26
5.	SVM Intuition .....	27
6.	Kernel SVM Intuition(linear- separability).....	27
a.	Mapping to a higher dimension .....	28
b.	The kernel trick .....	29
7.	Apriori .....	29
8.	Eclat intuition .....	31
9.	PCA:.....	32
a.	What is PCA? .....	32
b.	How Does PCA Work ? .....	32
c.	When to use PCA ? .....	32
Conclusion: .....		34
References: .....		34

## Table of Figures

Figure 1 Abstract of dataset used in Data Preprocessing Section.....	6
Figure 2 Python code example of importing data .....	6
Figure 3 Taking care of missing data .....	6
Figure 4 encode categorical data (Countries) .....	7
Figure 5 result encoding categorical data (country) .....	7
Figure 6 encode categorical data (labels) .....	7
Figure 7 splitting data into dataset and test set .....	8
Figure 8 formula used in feature scaling .....	8
Figure 9 Feature scaling python example .....	8
Figure 10 Simple Linear Regression General Formula .....	9
Figure 11 Example of a Simple Linear Regression.....	9
Figure 12 how does Simple linear regression work .....	10
Figure 13 simple linear Regression: abstract of the dataset used .....	10
Figure 14 simple linear regression python snippet code .....	11
Figure 15 simple linear regression predict new observations .....	11
Figure 16 simple linear regression visualize Training set results code .....	11
Figure 17 simple linear regression visualize Training set results (plot) .....	11
Figure 18 simple linear regression visualize Test set results code .....	12
Figure 19 simple linear regression visualize Test set results (plot) .....	12
Figure 20 dataset for multiple linear regression.....	13
Figure 21 example p-value.....	13
Figure 22 method of building a model.....	14
Figure 24 polynomial regression comparing to linear and multiple linear regression .....	15
Figure 23 example fitting polynomial regression .....	15
Figure 25: SVR example .....	16
Figure 26: example of non-linear SVR used in the course .....	17
Figure 27: Random forest steps.....	18
Figure 28: r-squared residues .....	19
Figure 29: example r-squared.....	19
Figure 30: SSres and SStot .....	20
Figure 31: calculating Adjusted R-squared .....	21
Figure 32: the problem with r-squared.....	21
Figure 33: binary classification .....	21
Figure 34: sigmoïde .....	22
Figure 35: plotting logistic regression.....	22
Figure 36: example plotting logistic regression .....	23
Figure 37: example KNN .....	24
Figure 38: The Mathematics Behind KNN.....	24
Figure 39: visualizing KNN formula .....	25
Figure 40: Diseases .....	26
Figure 41: the tree associated to diseases Dataset .....	26
Figure 42: example of SVM.....	27
Figure 43: example 2 of SVM (apples/oranges) .....	27
Figure 44: linearly separable.....	28
Figure 45: one dimension mapping .....	28
Figure 46: 2 dimension representation (SVM).....	28
Figure 47: 3D mapping (SVM) .....	28
Figure 48: radial basis function kernel.....	29
Figure 49: Apriori example .....	29
Figure 50: support example apriori .....	29
Figure 51: confident example .....	30
Figure 52: Lift (Apriori).....	30
Figure 53: Apriori steps.....	31
Figure 54: Eclat model Example data.....	31
Figure 55: calculating support of Eclat Algorithm.....	31
Figure 56: Steps of Eclat Algorithm.....	32
Figure 57: PCA EXAMPLE dataset .....	33
Figure 58: plotting PCA .....	33
Figure 59: PCA reduce dimensions .....	33

## Introduction

Data science is nowadays of great importance in today's massive information environment.

Indeed, it has been providing us with algorithms and tools to process in a better way, the huge data becoming available hence we can fully benefit from conclusions far from being intuitive.

So in order to learn more about this field I enrolled in the "Machine Learning A-Z™: Hands-On Python & R in Data Science".

This course covers many concepts and algorithms (Regressions, Classification, Clustering, Reinforcement Learning, Deep Learning, Dimensionality Reduction, Model Selection & Boosting,...etc), and for each section there is a explanation and examples with python and R.

Personally I focused on the python sections since I have a bachelor in software engineering, and my passion right now is to program web apps that can implement some Data Science concepts.

## Technical tools

In this course you can use:

### 1. Tools:

**Jupyter Notebook:** It is a server-client application that allows editing and running of notebook documents via a web browser.

**Google Colab notebooks:** allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more.

**RStudio:** is a free, open-source, cross-platform development environment for R, a programming language used for data processing and statistical analysis.

**Spyder:** Spyder is a development environment for Python. Free and multiplatform, it integrates many libraries for scientific use: Matplotlib, NumPy, SciPy and IPython.

**Anaconda:** is a free and open source distribution of the programming languages Python and R applied to the development of applications dedicated to data science and machine learning, which aims to simplify the management of packages and deployment.

### 2. Main libraries:

**Matplotlib:** It is a Python data visualization tool that supports 2D and 3D rendering, animation, UI design, event handling.

**Numpy:** It is a python library that is useful for computations and supports multi-dimensional arrays.

**Pandas:** It's the primary data analysis library in Python. Pandas is ideal for data visualization.

**Scikit learn:** It's a machine learning library in Python which features several classification, regression and clustering algorithms.

Personally I did chose to program manually on my computer using Spyder since this was the aim of this course for me (to be able to code Data science concept using python first and then any other language).

## Part 1 - Data Preprocessing

### 1. Describing dataset :

In this section (Data Preprocessing) we are using a dataset with 4 columns (Country, Age, Salary, purchased) the first 3 columns are the Matrix of futures and last col is the independent variable vector, Age, Salary and Purchased columns are missing some data also the independent variable and the first col of the Matrix of futures are not numerical.

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

Figure 1 Abstract of dataset used in Data Preprocessing Section

➔ The aim of this section is how to deal with such problems in our dataset so we can do analysis on it later.

### 2. Importing Data :

first thing to do is to import the data then create **the matrix of futures** and the **dependent variable vector** by using one of functions provided by pandas ( a library) data frame called iloc (stands for locate indexes)

```
dataset = pd.read_csv('Data.csv')  
x = dataset.iloc[:, :-1].values  
y = dataset.iloc[:, -1].values
```

Figure 2 Python code example of importing data

### 3. Taking care of missing Data:

In this section we are going to use a **SimpleImputer** a sub module from **sklearn** library) after that we need to apply this simple imputer on the matrix of futures, using the mean strategy to deal with missing data in Age and Salary since they are numerical.

```
from sklearn.impute import SimpleImputer  
imputer = SimpleImputer(missing_values= np.nan, strategy='mean')  
imputer.fit(x[:, 1:3])  
x[:, 1:3] = imputer.transform(x[:, 1:3])
```

Figure 3 Taking care of missing data

#### 4. Encoding categorical data (matrix of futures)

In Country column we have 3 categories (countries) repeated, in order to encode that we should create new columns for example France would take the vector [1,0,0], Spain [0,1,0] and Germany [0,0,1], like that we removed the numerical order problem.

```
[16] from sklearn.compose import ColumnTransformer
      from sklearn.preprocessing import OneHotEncoder
      ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
      x = np.array(ct.fit_transform(x))
```

Figure 4 encode categorical data (Countries)

As a result we have the same future matrix of futures but instead of countries we have the encoded columns:

```
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]]
```

Figure 5 result encoding categorical data (country)

#### 5. Encoding categorical data (independent variable)

The independent variable in our dataset is binary (yes or no) and we are going to convert this labels (yes, no) to zeros and ones, using the LabelEncoder class from sklearn library

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

```
print(y)
[0 1 0 0 1 1 0 1 0 1]
```

Figure 6 encode categorical data (labels)

#### 6. Feature Scaling:

Feature Scaling consist of scaling all variables (all your features actually) to make sure they all take values in the same scale in order to prevent one feature to dominate the other which therefore will be neglected by the machine learning model.

This section answers one of most frequently asked question in Data Science, do we need to apply feature scaling before or after splitting the dataset, and why?

- ➔ The answer is we should apply the feature scaling after splitting the data into training set and test set, because the test set supposed to be a brand new set on which we are going to evaluate our machine learning model, having that feature scaling is a technic that will get the mean understanding deviation of our feature so if we apply the feature scaling before the split then we will get the mean understanding deviation of all values including the test set and since the test set is something you are not supposed to have then you must not use feature scaling before splitting or that will cause some information leakage

### a. Splitting dataset

While splitting is very recommended to have 20% of data as test set and 80% as training set, so we are going to use “train\_test\_split” class which takes as parameter the matrix of features the independent variable, the size of the test set and since taking data from the source has some random factors and in order to have the same result each time I added “random\_state” parameter

```
[22] from sklearn.model_selection import train_test_split
      x_train,x_test,y_train,y_test = train_test_split(x,y, test_size= 0.2,random_state =1)
```

Figure 7 splitting data into dataset and test set

### b. Feature Scaling

In feature scaling we are going to use this formula:

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

Figure 8 formula used in feature scaling

The method fit will only compute the mean and the standard deviation of all the features, and then we have to apply the transform method which will apply the Standardization formula so all values will be on the same scale.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_train[:,3:] = sc.fit_transform(x_train[:,3:])
x_test[:,3:] = sc.transform(x_test[:,3:])
print(x_train)

[[0.0 0.0 1.0 -0.1915918438457855 -1.0781259408412427]
 [0.0 1.0 0.0 -0.014117293757057819 -0.07013167641635407]
 [1.0 0.0 0.0 0.5667085065333239 0.6335624327104546]
 [0.0 0.0 1.0 -0.30453019390224867 -0.307866172742979]
 [0.0 0.0 1.0 -1.901801144700799 -1.4204636155515822]
 [1.0 0.0 0.0 1.1475343068237056 1.2326533634535488]
 [0.0 1.0 0.0 1.4379472069688966 1.5749910381638883]
 [1.0 0.0 0.0 -0.7401495441200352 -0.5646194287757336]]
```

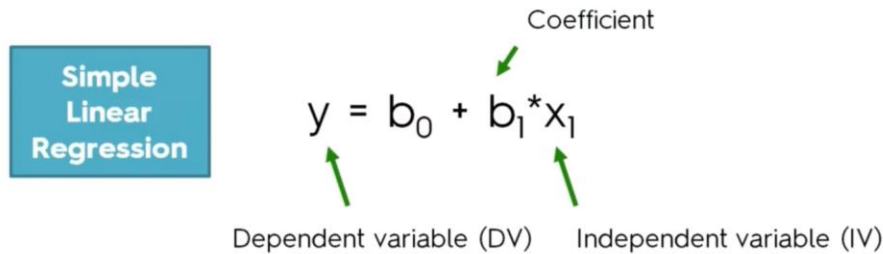
Figure 9 Feature scaling python example

Note: test set should be scaled using the same scaler used in the training set



## Part 2 - Regression:

### 1. Simple Linear Regression:



The diagram shows the general formula for simple linear regression:  $y = b_0 + b_1 * x_1$ . A blue box on the left contains the text "Simple Linear Regression". Green arrows point from labels to parts of the formula: "Coefficient" points to  $b_1$ , "Dependent variable (DV)" points to  $y$ , and "Independent variable (IV)" points to  $x_1$ .

Figure 10 Simple Linear Regression General Formula

In a Simple linear Regression Y (the dependent variable) is something that we want to explain to understand, on the other side X1 which will be noted as just X in case if its unique (like the example above "Figure 10") is the variable that you are assuming that is causing the dependent variable to change or sometimes it doesn't have much impact, and  $b_1$  is the coefficient which indicate how much the unit change of X1 in our example impact a unit change in Y, and finally  $b_0$  is a constant I will explain it with an example in the next section

#### a. An example of a Simple Linear Regression

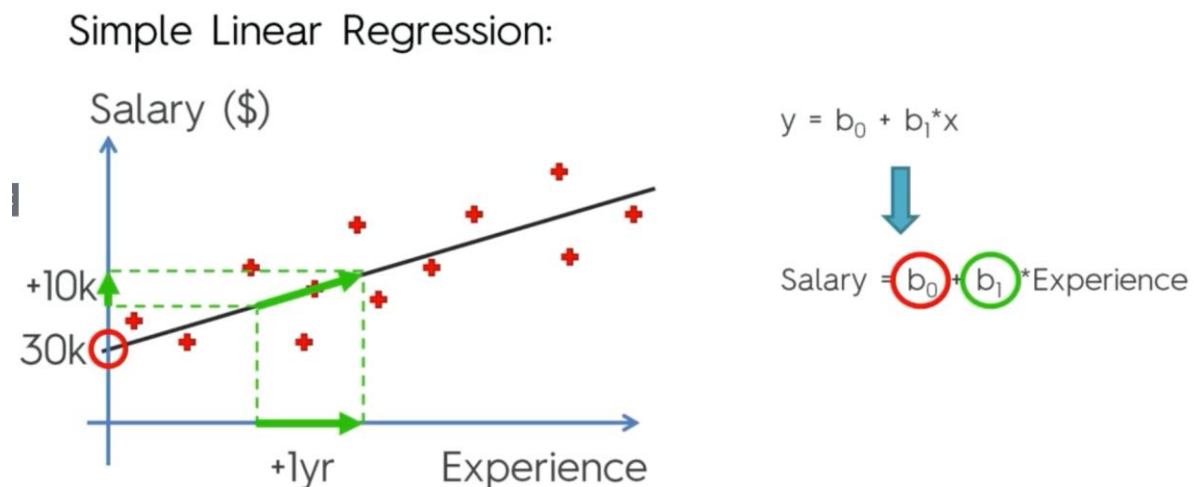


Figure 11 Example of a Simple Linear Regression

In this example above the red + are the observations (how salaries distributed in depend of employee's experience) and the Line is the result of the formula on the right in other words it's just putting a line into the chart that will be the best fitting of data ( we will explain best fitting in next sections)

So Salary is Y (dependent variable) and  $b_0$  is the constant which represent the intersection between the line (the aim of the simple linear regression) and Y axis in other words when we have no experience the second term of the formula became 0 so the Line should start from  $b_0$  ( with no experience the employee will have 30K salary), so if an employee wants to know his salary after 1 year experience he will project from X axis to the line and then to Y axis (in one year experience 10K is added to his salary )

## b. Ordinary Least Squares

This section explains how the simple linear regression finds the line in (Figure 11).

So as we have said the red + are the observations or the facts see the example bellow:

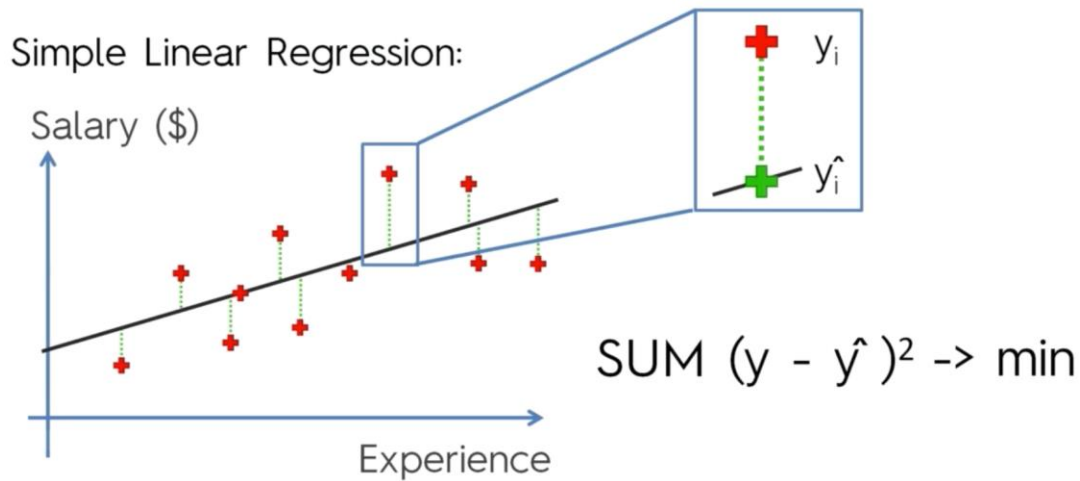


Figure 12 how does Simple linear regression work

So the green + is where the employee is among salaries and the red + is where is he actually and the green line is the difference between what is observed and what is model, so in order to have **the best fitting line** we take all the green lines then we calculate  $\text{sum}(y - \hat{y})^2$  which give us the minimum distance this method is called Ordinary Least Squares.

➔ To sum-up the best fitting line is the line that pass through the  $\text{SUM } (y - \hat{y})^2$  minimum sum of squares and that called the ordinary least squares method.

## c. Python Example

In order to implement the Simple Linear Regression am going to use a dataset described as the following:

It's a dataset containing as you can see in the figure below 30 observations and two columns, with one Feature "YearExperience" and one independent variable "Salary"

Salary_Data.csv		
	A	B
1	YearsExperience	Salary
2	1.1	39343
3	1.3	46205
4	1.5	37731
5	2	43525
6	2.2	39891
7	2.9	56642
8	3	60150

Figure 13 simple linear Regression: abstract of the dataset used

So the aim of this example is to build a simple linear regression Model that will be trained to understand the correlation between “YearExperience” and the “Salary” so it can predict the salary of a new employee who have several years of experiences

So as I described at part 1 the python snippets codes of importing libraries, dataset, split dataset into training set and test set are always the same so what's new now is the linear regression code, predicting data and visualization of results

```
[4] from sklearn.linear_model import LinearRegression
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
```

*Figure 14 simple linear regression python snippet code*

So as showed above we do linear regression on the training set using “LinearRegression” class

Then we I fitted the model on the training data, next step is to predict new observations.

```
[5] y_pred = regressor.predict(X_test)
```

*Figure 15 simple linear regression predict new observations*

So x\_test is the new observations that we didn't have we did the model and just that simple with calling the function predict from our linear regressor which will predict the new salaries according to x\_test and store them in the variable y\_pred (y predicted).

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

*Figure 16 simple linear regression visualize Training set results code*



So the x axis is the years experience from 1 to 10 and the Y axis is the salaries, this plot is showing the real salaries (red points) and the predicted ones (blue lines)

This graph is realized by the library matplotlib names plt as shortcut.

And next I will visualize the test set results:

*Figure 17 simple linear regression visualize Training set results (plot)*

```

▶ plt.scatter(X_test, y_test, color = 'red')
  plt.plot(X_train, regressor.predict(X_train), color = 'blue')
  plt.title('Salary vs Experience (Test set)')
  plt.xlabel('Years of Experience')
  plt.ylabel('Salary')
  plt.show()

```

Figure 18 simple linear regression visualize Test set results code

You can notice that the first line in figure 18 is the only change comparing to the other example (the scatter)

The next line is actually kind of an answer to a trick question, since at the beginning I thought that `x_train` should be replaced by the `x_test` but since the predictions are already on the line no need to change the `x_train` since the model is coming from one same equation and therefore we end up with the same regression line.



Figure 19 simple linear regression visualize Test set results (plot)

We can see in the figure above that the new observations are close to the blue line which means that our simple linear regression model was able to do a wonderful job at predicting new observations.

However the reason why we get such a good regression line is simply because there was a linear relationship in the data set between the features and the dependent variable, in other words our dataset is a perfectly linear dataset (dataset with linear correlations), and therefore there is no guarantee that those would happen for any dataset (datasets that require a non-linear model)

## 2. Multiple Linear Regression

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

So in this section we are going to work with the following dataset:

Profit	R&D Spend	Admin	Marketing	State	Dummy Variables	
					New York	California
192,261.83	165,349.20	136,897.80	471,784.10	New York	1	0
191,792.06	162,597.70	151,377.59	443,898.53	California	0	1
191,050.39	153,441.51	101,145.55	407,934.54	California	0	1
182,901.99	144,372.41	118,671.85	383,199.62	New York	1	0
166,187.94	142,107.34	91,391.77	366,168.42	California	0	1

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * x_3 + b_4 * D_1$$



Figure 20 dataset for multiple linear regression

So as always the dependent variable Y is the profit and we are going to try explain the profit using the other columns (the explanatory variables), and as you can see the last column is a categorical variable, and in order to include it in the equation we have to convert it to binary variable or what we call dummy variable, then we add only one dummy variable in this case since its binary it works like a light switch (new York or not new York), and if we add both dummy variables first that's duplicating the same variable and because  $D_2 = 1 - D_1$  that will cause what we call **multicollinearity** as an effect the model cannot distinguish between the effects of  $D_1$  from the effects of  $D_2$

→ This is called the dummy variable trap

### a. Statistical significance(P-Value)

the course try to explain the p-value using the example of a coin that you flip it and then for an example you have the front side on the first and second time nothing suspicious but imagine having that same result repeatedly on the 10 th or 50 th time ,well now you start getting suspicious maybe something is not right in fact the probability to have the front side is 50% on the first try 25% on second try and so on the probability keeps dropping until it's not likely to have the front side, so what is the relation of that and p-value, if we put an  $\alpha=0.05$  and it will make us accept or reject the fact that we are using a fair coin look at the figure bellow that why alpha should be chosen very wisely, in our example after the line of alpha we reject the hypothesis

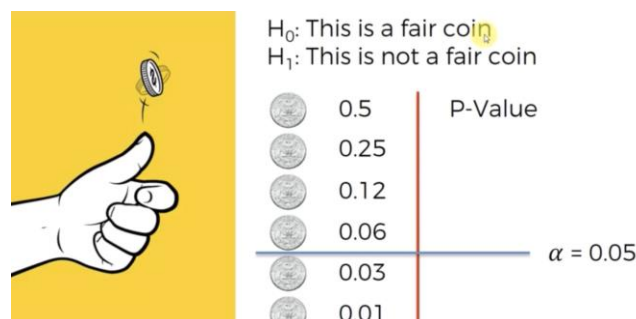



Figure 21 example p-value

### **b. Build a model**

In multiple linear regression we have many variable and this may not be very helpful since at the end on the day you have to explain hundreds of variables and their relationships and it may not make the model reliable or efficient that's why we have 5 methods to make our model strong

## **5 methods of building models:**

1. All-in
  2. Backward Elimination
  3. Forward Selection
  4. Bidirectional Elimination
  5. Score Comparison
- 
- Stepwise Regression

*Figure 22 methode of building a model*

### **c. Method 1: All-in**

In fact "ALL in" is not a technical word it's called like that since the concept is to throw in all your variables, if you have a prior knowledge of the exact variables that are your true predictors (you don't have to build anything you already know the necessary variables for a reason or another)

### **d. Method 2 – backward Elimination**

Step 1: the first thing here is to set a significance level to stay in the model (  $sl = 0.05$  )

Step 2: fill the model of all possible predictors, in order to get rid of some of them later

Step 3: consider the predictor of the highest P-value, if  $P > sl$  , go to step 4 , go to FIN

Step 4: Remove that predictor

Step 5: Fit the model without this variable

Like that you keep removing variables until in a moment the highest p-value is still less than the significance level then we don't go to step 4 anymore we do the FIN (finish ) it means the model is ready, and that was how the backward elimination work

### **e. Method 3 – Forward Selection**

Step1: set the significance level to enter the model  $sl=0.05$

Step 2: fit all simple regression model  $y \sim X_n$  Select the lowest p-value.

Step3: keep this variable and fit all possible models with one extra predictor added to the one(s)

Step4: consider the predictor with the lowest p-value. If  $p < sl$ , go to step 3, otherwise go to FIN

And in FIN we keep the previous model because in the last model we added the variable which leaded us to the FIN step (insignificant variable)

### **f. Method 4 - Bidirectional Elimination**

Step 1: select a significant level to enter and to stay in the model

Example :  $SLENTER=0.05$ ,  $SLSTAY =0.05$

Step 2: perform the next step of forward selection (new variable must have  $p < \text{SLENTER}$  to enter)

Step 3: perform all steps of backward elimination (old vars must have  $p < \text{SLStay}$  to stay)

Step 4: no new variables can enter and no old variables can exit -> FIN

Note: some people call this method step-wise regression

### **g. Method 5 - All Possible Models (score comparison)**

Step 1: select a criterion of goodness of fit ( example AKAIKE CRITERION)

Step 2: Construct all possible models:  $(2^N - 1)$  total combinations

Step3: Select one of this models with the best criterion

FIN: your model is ready

This method is really time and resource consuming since with only 10 columns we have 1023 possible models

## **3. Polynomial Regression**

In the screenshot bellow we have the equations of the regression i have talked about until now, and it's obvious that Polynomial linear regression is very similar to the multiple linear regression, we have the same variables but this time in different powers :

Simple Linear Regression	$y = b_0 + b_1x_1$
Multiple Linear Regression	$y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$
Polynomial Linear Regression	$y = b_0 + b_1x_1 + b_2x_1^2 + \dots + b_nx_1^n$

Figure 24 polynomial regression comparing to linear and multiple linear regression

As you can see on the figure on the right the polynomial linear regression fits perfectly thanks to the term  $b_2x_1^2$  who gives the parabolic form to the curve, and if we apply a simple linear regression here it won't fit well since the line won't be near to most of dots .

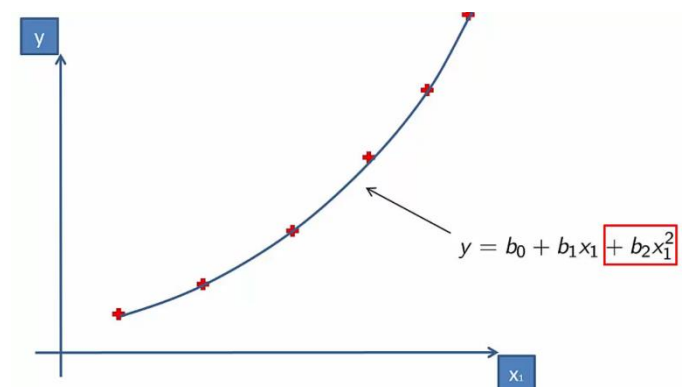


Figure 23 example fitting polynomial regression

## **4. SVR Intuition**

-So in order to understand SVR we have taken the same data on the left it's a simple linear regression using OLS method and on the right we applied SVR (FIGURE 25 bellow).



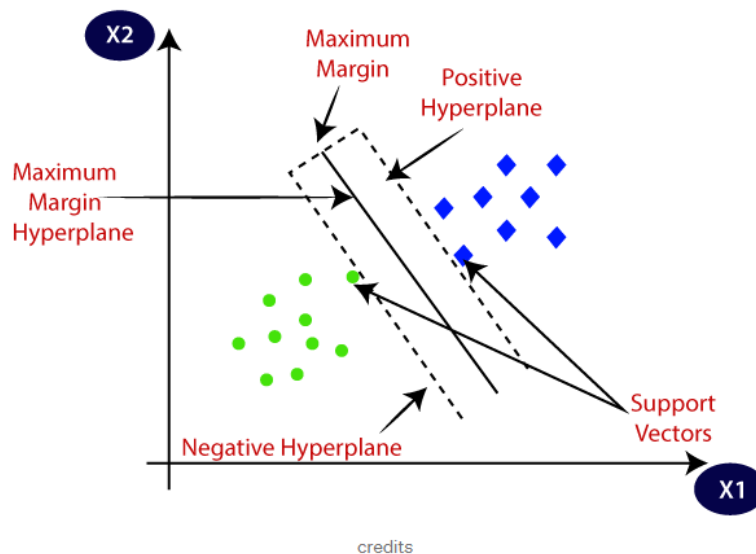
So on the left nothing unusual we have the best simple linear model based on OLS and the distances between the dots and the line are having the least distance possible.

And on the right even the line is different in fact we don't care much about the line we care about the tube (yellow tube) and the epsilon between the line and the border of the tube is the error that we accept in our model so if the points are inside the tube its ok but the real errors the important ones are the errors out of the tube.

This method is named after SUPPORT VECTOR REGRESSION since every point of the plot are vectors (blue lines), and the highlighted point (in red) are called support vectors since they are dictating how this tube is great , so basically they are supporting the structure or formation of this tube and that's why they called support.

## 5. Non-linear SVR

**Non-linear SVR** is a little bit complicated the previous example was linear and the differences are first we need to go on 3D for plotting we need to do some calculations and come back and in order to understand better SVR we need to know about **SVM** and in the course they just want to say there is something called **SVR** and they didn't really give a full explanation, so I did some research and come up with some definitions.



**Support Vector Machines(SVM)** are one of the state-of-the-art machine learning algorithm based on **Maximal Margin Classifier**.

SVM support linear as well as non-linear regression called **Support Vector Regression(SVR)**.

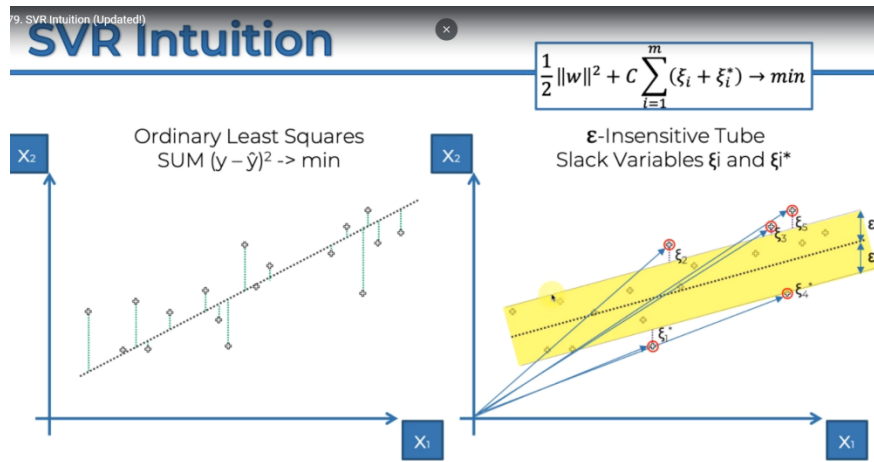


Figure 25: SVR example



In case of SVM (to be continued in Part-3 classification):

- A point above the line gets a label `1`
- A point below the line gets a label `0`
- A point close to the line return a value close to `0` and the point may be difficult to classify.
- If the magnitude of value is large for a test data point, the model will have more confidence in prediction.

Thus, comes the concept of maximizing the margin.

Section on SVM:

- SVM Intuition

Section on Kernel SVM:

- Kernel SVM Intuition
- Mapping to a higher dimension
- The Kernel Trick
- Types of Kernel Functions
- Non-linear Kernel SVR

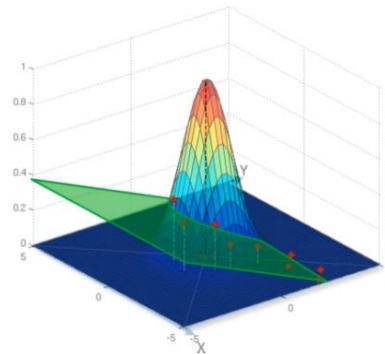


Figure 26: example of non-linear SVR used in the course

## 6. Decision Tree Regression

Regression problems are usually those for which we seek to predict values of a continuous variable from one or more categorical and/or continuous predictive variables. For example, if we seek to predict the sale price of apartments in a residential area (this is a continuous dependent variable) from a number of continuous predictors (for example, the area in square metres) but also from categorical predictors (e.g. building style [modern, old], postal code of the commune, etc. ...;

→ Regression trees are used to explain and/or predict values taken by a quantitative dependent variable, based on quantitative and/or qualitative explanatory variables.

### a. Most common algorithms and software for building decision trees:

- **CHAID:** Chi-Square Automatic Interaction Detection (1975)
- **CART:** Classification And Regression Trees (Breiman et al., 1984)
- **Knowledge seeker**

### b. Further more about used algorithmes :

For each node of the tree:

-For each of the X explanatory variables, matching the modalities from the chi-carré test.

-After matching the modalities for each X, Selection of the X variable most strongly bound to the target variable Y, i.e. p-value of the smallest chi-carred test.

For each next thread node, resuming steps (a) and (b)

- The process stops when at all shaft nodes, the chi-two tests between variables X and Y are all non-significant, i.e.  $p\text{-value} > \alpha$

## 7. Random Forest Regression

- Random forests are methods for obtaining predictive models for classification and regression. The method implements binary decision trees, including CART trees proposed by Breiman et al. (1984).
- The general idea behind the method is that instead of trying to get an optimized method at once, we generate several predictors before sharing their different predictions.

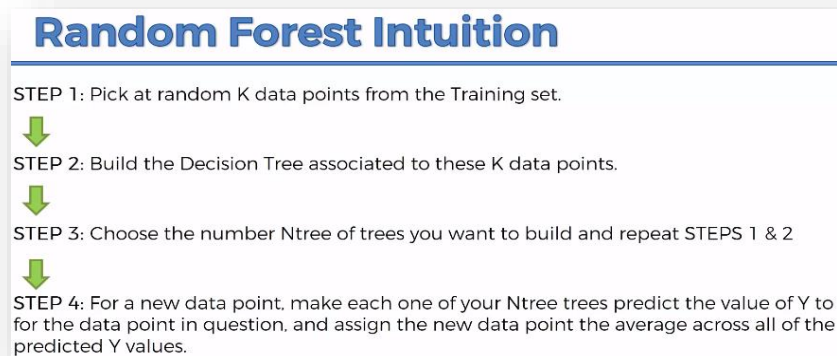


Figure 27: Random forest steps

- The difference between decision trees and random forests is the number of estimators: number of trees and instead of getting a prediction we will get a lot of prediction.
- To evaluate our model, and to know whether it is a good model or not, it is absolutely necessary to see the R-squared parameter.

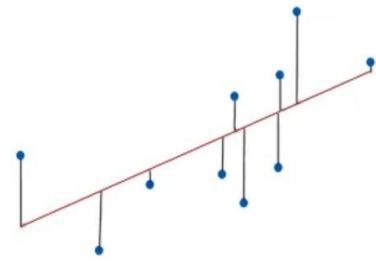
## 8. R squared

R to square is a measure of adjustment quality for linear regression models.

This statistic indicates the percentage of the variance of the dependent variable that the independent variables collectively explain.

R to square measures the strength of the relationship between your model and the dependent variable on a practical scale from 0 to 100%

The linear regression identifies the equation which produces the smallest difference between all observed values and their adjusted values. To be precise, linear regression finds the smallest sum of squares of residues possible for the data set.



Les résidus sont la distance entre la valeur observée et la valeur ajustée.

Figure 28: r-squared residues

Statisticians say that a regression model corresponds well to the data if the differences between observations and predicted values are small and unbiased. In this context, impartial means that the adjusted values are not systematically too high or too low anywhere in the observation space.

However, before evaluating numerical measurements of adjustment quality, such as R to square, you must evaluate the residual lines. Residual tracing can expose a biased model much more efficiently than digital output by displaying problematic models in residues. If your model is biased, you cannot trust the results. If your residual graphs look good, go ahead and evaluate your R-carré and other statistics.

#### a. How to calculate R2?

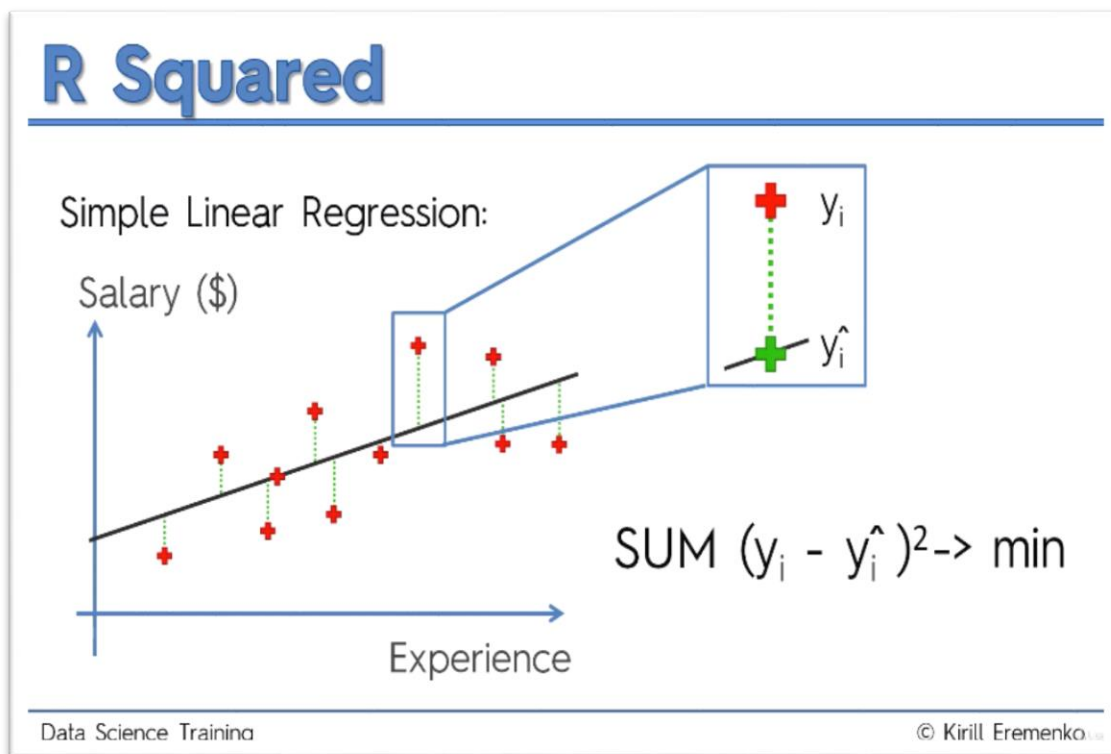
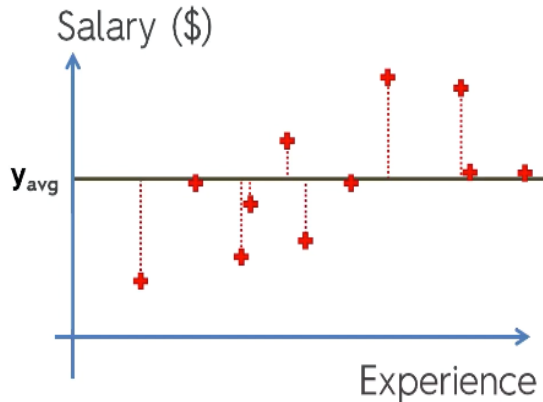


Figure 29: example r-squared

# R Squared

Simple Linear Regression:



$$SS_{res} = \text{SUM } (y_i - \hat{y}_i)^2$$

$$SS_{tot} = \text{SUM } (y_i - y_{avg})^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Data Science Training

© Kirill Eremenko

Figure 30: SSres and SStot

## a. Interpret R-squared

This coefficient is between 0 and 1 and increases with the adequacy of the regression to the model

- If the R2 is close to zero, then the regression line sticks to 0% with all the given points.
- If the R2 of a model is 0,50, then half of the variation observed in the calculated model can be explained by the points
- If the R2 is 1, then the regression determines 100% of the point distribution. In practice, it is impossible to obtain an R2 of 1 from empirical data. A square R is considered to be high when it is between 0.85 and 1

## 9. Adjusted R-squared

The problem with R-squared is that it will remain the same or will increase with the addition of more variables, even if they have no relationship with the output variables. This is where the "adjusted R square" helps. The adjusted R square penalizes you for adding variables that do not improve your existing model.

Therefore, if you build a linear regression over multiple variables, it is always suggested to use the adjusted R- squared to judge the quality of the model. If you have only one input variable, R- squared and R adjusted to the square would be exactly the same.

As a general rule, the more non-significant variables you add to the model, the difference between the square R and the adjusted square R increases.

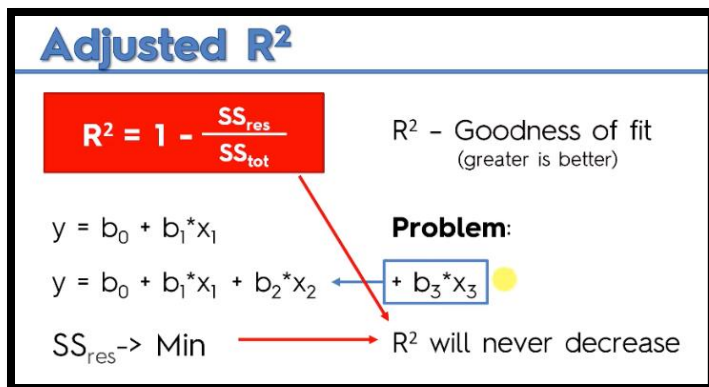


Figure 31: calculating Adjusted R-squared

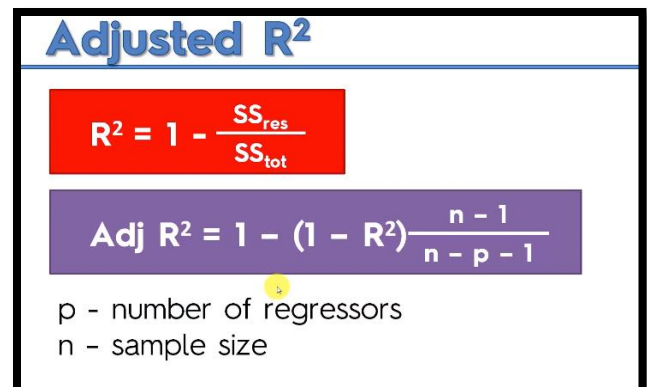


Figure 32: the problem with r-squared

## Part 3 – Classification

Unlike regression where you predict a continuous number, you use classification to predict a category. There is a wide variety of classification applications from medicine to marketing. Classification models include linear models like Logistic Regression, SVM, and nonlinear ones like K-NN, Kernel SVM and Random Forests.

In this part, you will understand and learn how to implement the following Machine Learning Classification models:

- Logistic Regression
- K-Nearest Neighbors (K-NN)
- Support Vector Machine (SVM)
- Kernel SVM
- Naive Bayes
- Decision Tree Classification
- Random Forest Classification

### 1. Logistic regression

Logistic regression is a highly used classification model in statistics and in particular in machine learning. Moreover, it is an excellent introduction to several key machine learning concepts.

A logistic regression model also predicts the probability of an event (value of 1) or not (value of 0) from the

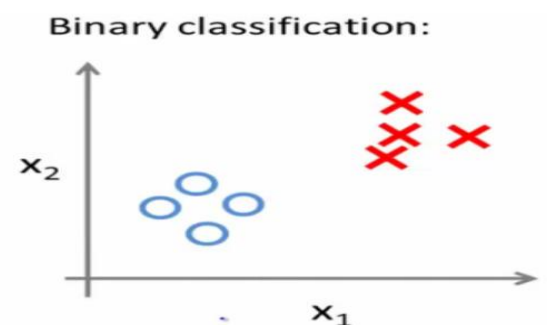


Figure 33: binary classification

optimization of regression coefficients. This result always varies between 0 and 1. Where the predicted value is above a threshold, the event may occur, whereas when the value is below the same threshold, it is not.

The goal of the game is to find a line (Boundary Decision) separating the two groups (circles and squares).

Mathematically, how does it translate/it's written?

Consider an entry  $X = x_1 \ x_2 \ x_3 \ \dots \ x_n$ , the logistic regression aims to find a  $h$  function such that we can calculate:

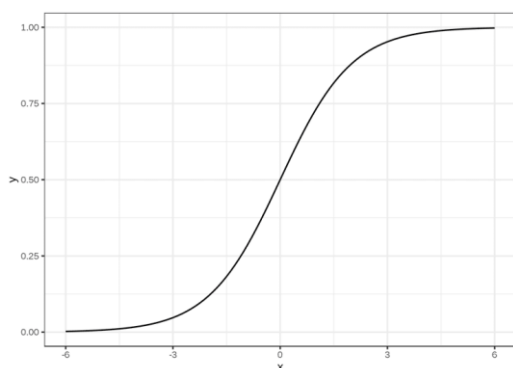
$$Y = \{1 \text{ if } hX \geq \text{margin}, 0 \text{ si } hX < \text{margin} \}$$

We therefore understand that we expect our function to be a probability between 0 and 1, set by  $=1 \ 2 \ 3 \ n$  to optimize, and that the threshold we define corresponds to our classification criterion, usually it is taken as worth 0.5.

The function that best fulfils these conditions is the sigmoid function, defined on  $\mathbb{R}$  with values in  $[0,1]$ . It is written as follows:

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

Figure 34: sigmoïde



Graphically, this corresponds to a S-shaped curve with limits of -1 and 1 when  $x$  tends to  $-\infty$  and  $+\infty$  passing through  $y = 0.5$  in  $x = 0$  respectively.

Figure 35: plotting logistic regression

### **a. So when we are going to use classification now?**

The  $h$  function defining logistic regression is written as:

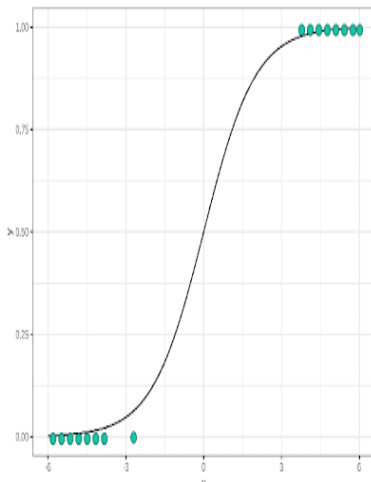
$$\forall (X \in \mathbb{R}^n) \quad h(X) = \sigma(\Theta X)$$

i.e.

$$\forall (X \in \mathbb{R}^n) \quad h(X) = \frac{1}{1 + e^{-\sum_{i=1}^n \theta_i x_i}}$$

The entire problem of logistic regression classification then appears to be a simple optimization problem where, from data, we try to obtain the best set of parameters? Allowing our sigmoid curve to best stick to the data. It is in this step that our automatic learning takes place.

Once this step has been completed, here is an overview of the result that can be obtained:



From the defined threshold, it only remains to classify the points according to their positions in relation to the regression and our classification is made.

Figure 36: example plotting logistic regression

## 2. K-Nearest Neighbors

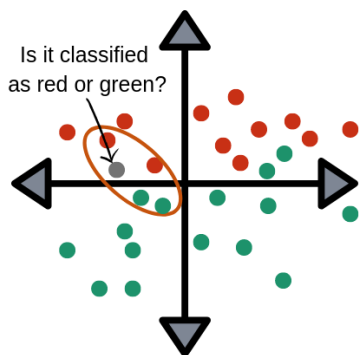
What is KNN? KNN is a model that classifies data points based on the points that are most similar to it. It uses test data to make an “educated guess” on what an unclassified point should be classified as.

### Pros:

- Easy to use.
- Quick calculation time.
- Does not make assumptions about the data.

### Cons:

- Accuracy depends on the quality of the data.
- Must find an optimal k value (number of nearest neighbors).
- Poor at classifying data points in a boundary where they can be classified one way or another.



KNN is an algorithm that is considered both non-parametric and an example of lazy learning. What do these two terms mean exactly?

Non-parametric means that it makes no assumptions. The model is made up entirely from the data given to it rather than assuming its structure is normal.

Lazy learning means that the algorithm makes no generalizations. This means that there is little training involved when using this method. Because of this, all of the training data is also used in testing when using KNN.

### a. Where to use KNN

KNN is often used in simple recommendation systems, image recognition technology, and decision-making models. It is the algorithm companies like Netflix or Amazon use in order to recommend different movies to watch or books to buy. Netflix even launched the Netflix Prize competition, awarding \$1 million to the team that created the most accurate recommendation algorithm!

You might be wondering, “But how do these companies do this?” Well, these companies will apply KNN on a data set gathered about the movies you’ve watched or the books you’ve bought on their website. These companies will then input your available customer data and compare that to other customers who have watched similar movies or bought similar books. This data point will then be classified as a certain profile based on their past using KNN. The movies and books recommended will then depend on how the algorithm classifies that data point.

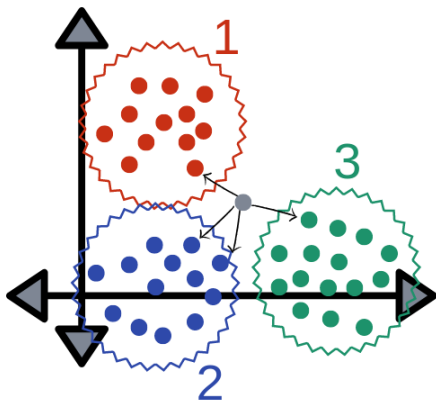


Figure 37: example KNN

The image above visualizes how KNN works when trying to classify a data point based a given data set. It is compared to its nearest points and classified based on which points it is closest and most similar to. Here you can see the point  $X_j$  will be classified as either  $W_1$  (red) or  $W_3$  (green) based on its distance from each group of points.

### b. The Mathematics Behind KNN

Just like almost everything else, KNN works because of the deeply rooted mathematical theories it uses. When implementing KNN, the first step is to transform data points into feature vectors, or their mathematical value. The algorithm then works by finding the distance between the mathematical values of these points. The most common way to find this distance is the Euclidean distance, as shown below.

$$\begin{aligned} d(\mathbf{p}, \mathbf{q}) &= d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \\ &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}. \end{aligned}$$

Figure 38: The Mathematics Behind KNN

KNN runs this formula to compute the distance between each data point and the test data. It then finds the probability of these points being similar to the test data and classifies it based on which points share the highest probabilities.

To visualize this formula, it would look something like this:



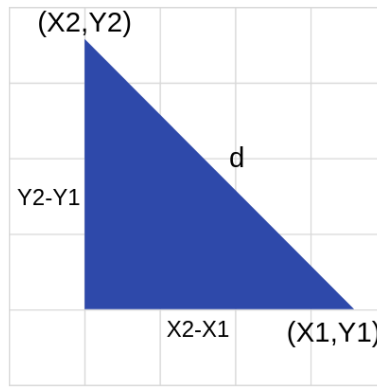


Figure 39: visualizing KNN formula

### 3. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable  $y$  and dependent feature vector  $x_1$  through  $x_n$ :

$$P(y | x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n | y)}{P(x_1, \dots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

for all  $i$ , this relationship is simplified to

$$P(y | x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i | y)}{P(x_1, \dots, x_n)}$$

Since  $P(x_1, \dots, x_n)$  is constant given the input, we can use the following classification rule:

$$P(y | x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i | y)$$

$$\Downarrow$$

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate  $P(y)$  and  $P(x_i | y)$ ; the former is then the relative frequency class  $y$  in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of  $P(x_i | y)$ .

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from `predict_proba` are not to be taken too seriously.

GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The parameters  $\sigma_y$  and  $\mu_y$  are estimated using maximum likelihood.

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.naive_bayes import GaussianNB
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(X_train, y_train).predict(X_test)
>>> print("Number of mislabeled points out of a total %d points : %d"
...      % (X_test.shape[0], (y_test != y_pred).sum()))
Number of mislabeled points out of a total 75 points : 4
```

## 4. Decision tree

A tree is a graph that is divided into three categories:

- Root node (access to the tree is via this node),
- Internal nodes: nodes which have descendants (or children), which are in turn nodes,
- Terminal nodes (or leaves): nodes that have no descendant.

Decision trees are a category of trees used in data mining and business intelligence. They employ a hierarchical representation of the data structure in the form of sequences of decisions (tests) for the prediction of an outcome or a class. Each individual (or observation), which must be assigned to a class, is described by a set of variables which are tested in the nodes of the tree. Testing takes place in internal nodes and decisions are made in leaf nodes.

Example of a problem adapted to a decision tree approach:

	Toux	Fièvre	Poids	Douleur
Marie	non	oui	normal	gorge
Fred	non	oui	normal	abdomen
Julie	oui	oui	maigre	aucune
Elvis	oui	non	obese	poitrine

Figure 40: Diseases

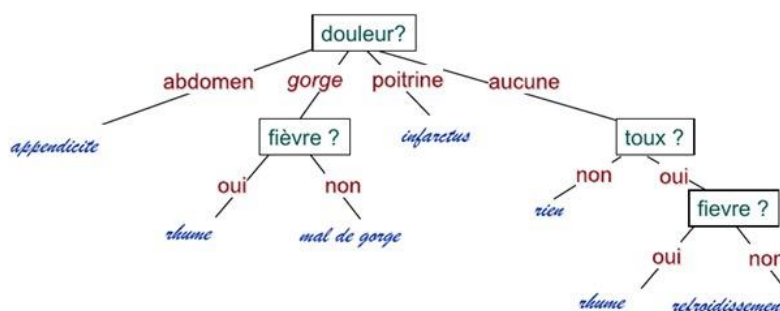


Figure 41: the tree associated to diseases Dataset

## 5. SVM Intuition

in Classification SVM is a powerful algorithm that can separate between variables, and by looking at the example below the green points and the red dots are separated by the tube in the middle so if we add another point it will fall in the green Area or the red area, however the support vectors are calibrating our model, this means that other points are not contributing in the result of the algorithm, the line in the middle called maximum margin hyper-plan, the green line is the positive hyper plan, and the red dotted line is the negative hyper-plan, and of course the naming order is not important but in general the right part is the positive hyper-plan.

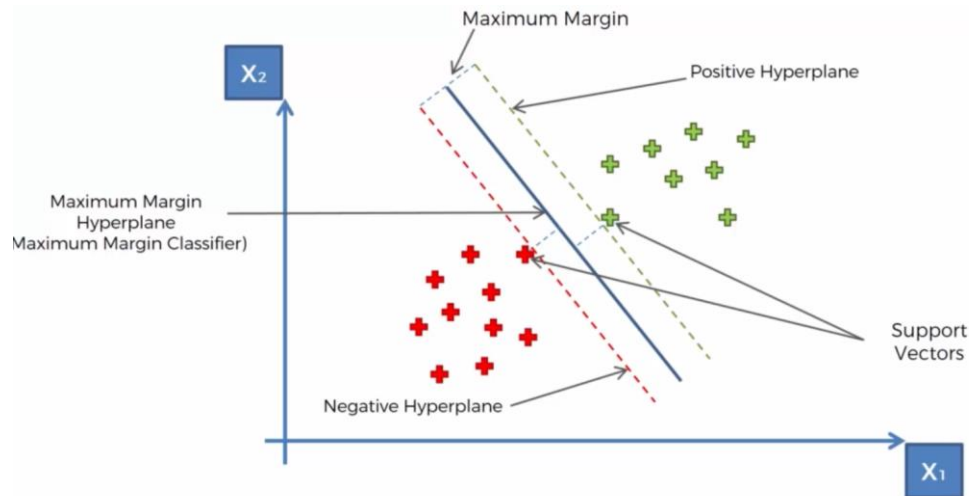


Figure 42: example of SVM

So let's take a concrete example in the example bellow we have the apples on the left and oranges on the right, so what happen is that the machine will learn what is an orange and what is an apple and the it can separate between them so when you give another object it will know if it's an apple an orange or something else.

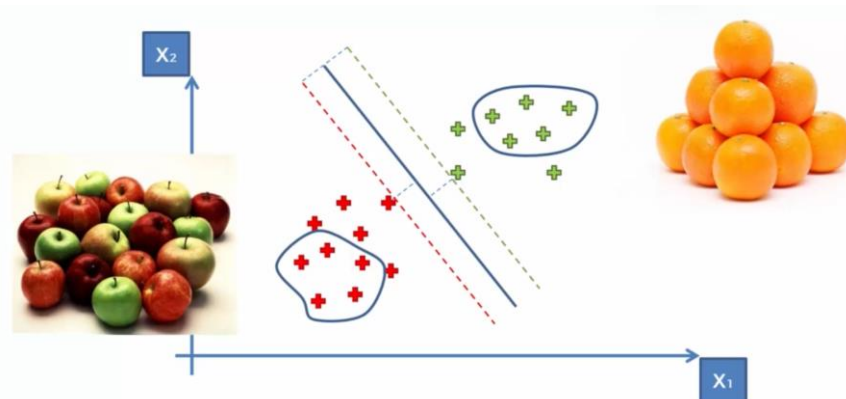


Figure 43: example 2 of SVM (apples/oranges)

## 6. Kernel SVM Intuition(linear- separability)

In the figure bellow on the right the points are well separated with a straight line, but on the right we cannot just draw a line through the points, well that because the points are Not Linearly Separable.

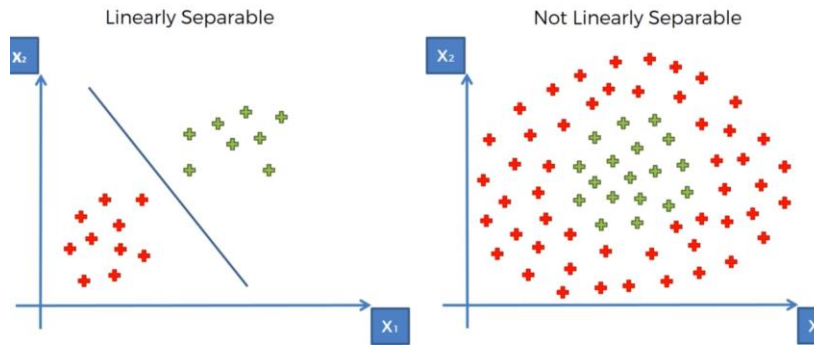


Figure 44: linearly separable

### a. Mapping to a higher dimension

In the figure bellow, if we put the dots in one dimension it's impossible to separate red dots and green dots.



Figure 45: one dimension mapping

Now if we take the same dots from the previous figure and put it in 2 dimensions space, it's now possible to have a line that can separate data.

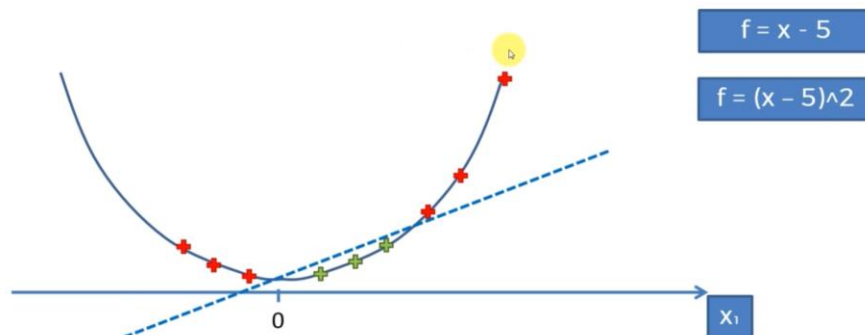


Figure 46: 2 dimension representation (SVM)

Furthermore if we chose to use a 3 dimensions representation its always possible to separate points again, however it's still possible to separate them in 2D using projection (Figure bellow) but this is not always possible.

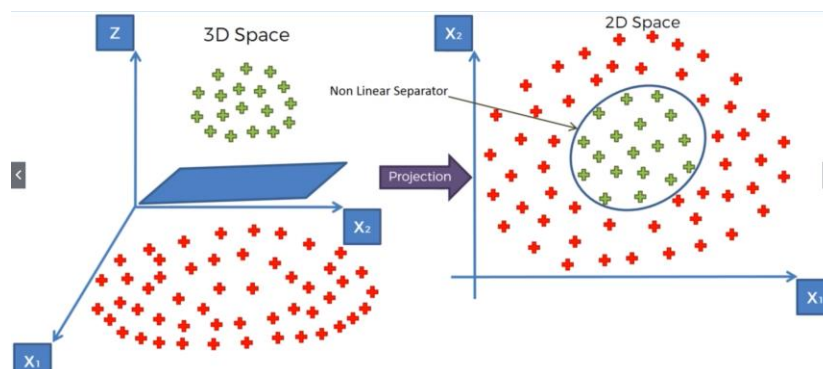


Figure 47: 3D mapping (SVM)

There is a problem with this algorithm is that is very highly compute intensive, so it may require a lot of computation, a lot of processing power, and the larger your dataset is the more it cause problems, therefore this approach isn't the best. Therefore the trick used here is what we call in mathematics the kernel trick and that's approach is going to help is a very similar results, but without having to go to a higher dimension.

## b. The kernel trick

So I following figure is the Gaussian, radial basis function kernel:

$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x} - \vec{l}^i\|^2}{2\sigma^2}}$$

Figure 48: radial basis function kernel

In machine learning, kernel machines are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets.

## 7. Apriori

In the figure bellow people who have seen movie 1 most them have seen movie 2 and then who have seen movie 1 and to are most likely to see movie 4, so we can from this data we can extract some informations, and then we don't have to ask people do you like a particular movie in fact we can recommend movies for people

User ID	Movies liked
46578	Movie1, Movie2, Movie3, Movie4
98989	Movie1, Movie2
71527	Movie1, Movie2, Movie4
78981	Movie1, Movie2
89192	Movie2, Movie4
61557	Movie1, Movie3

Potential Rules:	Movie1	→	Movie2
	Movie2	→	Movie4
	Movie1	→	Movie3

Figure 49: Apriori example

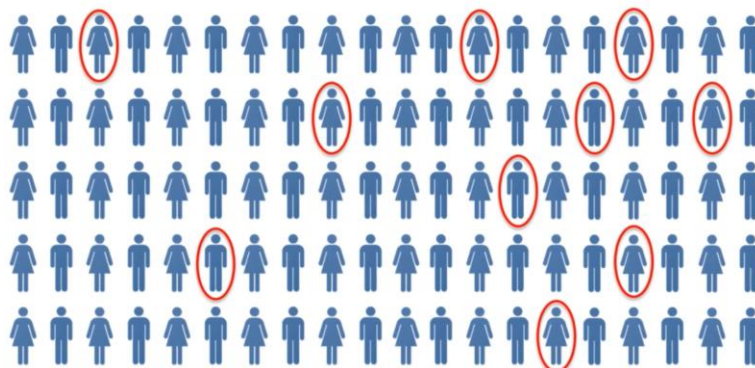


Figure 50: support example apriori



- So in the figure above 10 people of 100 have watched a movie so the support here is:

$$\text{Support} = 10/100 = 10\%$$

- Next step is to find the confidence:

In this step we have a hypothesis people who have seen movie 1 are most likely to see movie 2

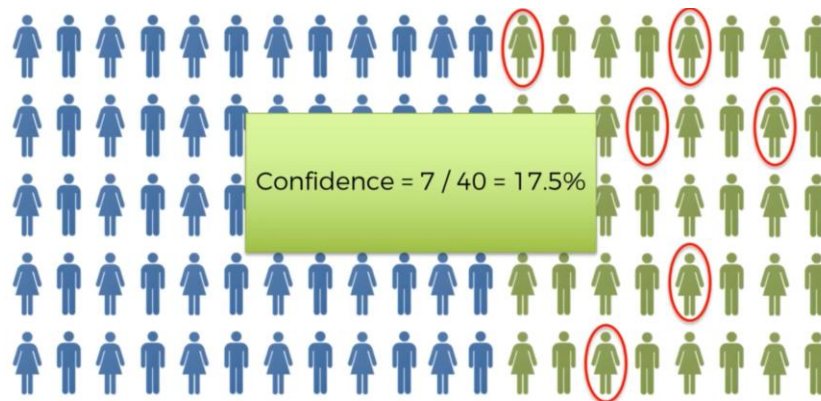


Figure 51: confident example

In the example above the green people have seen movie 2 and only the selected green people have seen movie 1 like the blue people

So now the confidence is equal to:  $7/40 = 17,5\%$

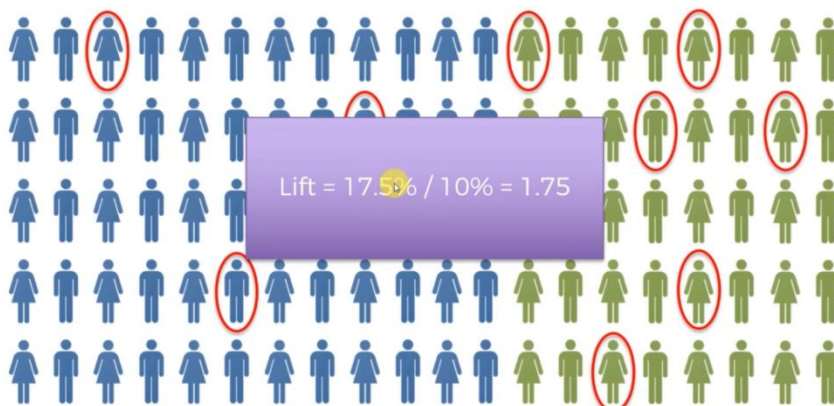


Figure 52: Lift (Apriori)

And then we should calculate the lift which is simply the confidence divided by the support, so if we randomly chose a person what is the likelihood to like a chosen movie, a recommended movie for them, more precisely what is we take a random population from the 100 people from previous examples and we recommended the first movie, now the question is what is likelihood they will like it?

To answer the question we have to rely on the previous knowledge (that 10%) liked the first movies and we know that the green people have liked the second movie but only 17.5% of them have seen the first movie (that why the algorithm is called Apriori since we have a prior knowledge), so the lift is the improvement of your prediction :

$$\text{Lift} = 17.5\% / 10\% = 1.75$$

Finally the Apriori have several steps:

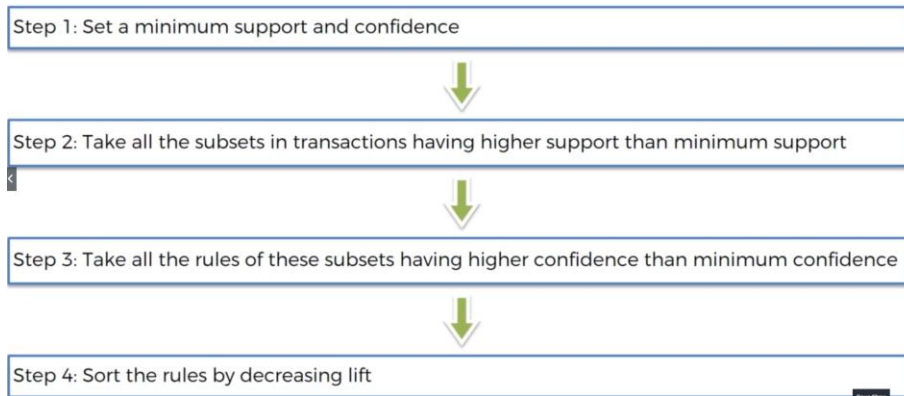


Figure 53: Apriori steps

Furthermore Apriori is kind of slow algorithm since it goes through all this steps (combinations) it combines a lot not just pairs and not triplets like it combines four, five six, seven items in one set and so on, it gets quite big and therefore you need to set some kind of limitations like a minimum support (for example do not look at product that have a support less than 20%)

## 8. Eclat intuition

Eclat algorithm as Apriori talks about “who bought also bought ...”, so in Apriori we have rules towards an output, and based on the lift we could judge the strength of a tool.

Whereas in Eclat we are going to talk about sets, we will start with some data :

Transaction ID	Products purchased
46578	Burgers, French Fries, Vegetables
98989	Burgers, French Fries, Ketchup
71527	Vegetables, Fruits
78981	Pasta, Fruits, Butter, Vegetables
89192	Burgers, Pasta, French Fries
61557	Fruits, Orange Juice, Vegetables
87923	Burgers, French Fries, Ketchup, Mayo

Potential Rules:	Burgers	⇒	French Fries
	Vegetables	⇒	Fruits
	Burgers, French Fries	⇒	Ketchup

Figure 54: Eclat model Example data

You can already notice that it's similar to the data used in Apriori, so the same in Eclat we have support factor, comparing again with the last algorithm in Apriori we have support + confidence + lift, and in Eclat model we have only support and the next figure explains how to calculate the support :

Movie Recommendation:

$$\text{support}(\mathbf{M}) = \frac{\# \text{ user watchlists containing } \mathbf{M}}{\# \text{ user watchlists}}$$

Market Basket Optimisation:

$$\text{support}(\mathbf{I}) = \frac{\# \text{ transactions containing } \mathbf{I}}{\# \text{ transactions}}$$

Figure 55: calculating support of Eclat Algorithm

So in Eclat we see how frequently a set of items occurs, so in the figure above M is a set of two movies and the result is what percentage of watch lists of movies people like contain both movies together, so if hypothetically speaking if 100% of the lists that you have in a large dataset contain both movies together that implies that everybody who watched the first movie will like the second, even if 75% of list have both movies then if someone random watched the first movie there is 75% chance that he will like the second one.

This Algorithm is much faster than Apriori and those are the steps of the algorithm:

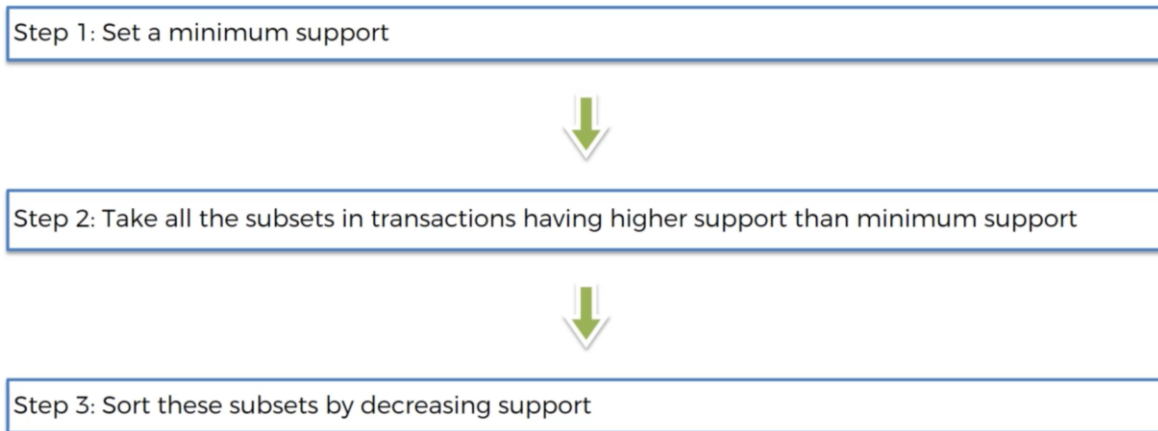


Figure 56: Steps of Eclat Algorithm

## 9. PCA:

### a. What is PCA?

Principal Component Analysis is an unsupervised algorithm that reduces a dataset's dimension through feature extraction. Indeed, PCA is able to transform any continuous or categorical data into another dataset containing less features than the original, for the same number samples. The new set returned contains only linearly independent features.

### b. How Does PCA Work ?

PCA is a very useful technique when dealing with a multidimensional dataset. It computes the covariance matrix for the dataset as well as the eigenvectors and eigenvalues associated to it in order to determine the best unique features that describe the samples we have.

An iterative approach to this would first find the center of your data, based off its numeric features. then, it would search for the direction that has the most variance or widest spread of values, and that direction is the principal component vector, so it is added to a list. By searching for more directions of maximal variance that are orthogonal to all previously computed vectors, more principal component can then be added to the list. This set of vectors form a new feature space that can represent our samples with.

Moreover, PCA makes sure that all the features are ordered by importance. It then drops the least important features allowing us to represent our dataset with minimal loss of information.

### c. When to use PCA ?

PCA is able to distinguish the natural categories that rely within our unlabeled data. These categories that it uncovers are the principal components, which are the best possible, linearly independent combination of features that represents our data. It is mainly used with high dimensionality dataset. It allows us to get rid of redundant and irrelevant features in order to have a better perspective of our data. Through Lab assignment 2 of the Module 4, We were able to get more familiar with using PCA. We had a UCI's Chronic Kidney Disease data set, a collection of samples taken from patients in India



over a two month period, some of whom were in the early stages of the disease. And we experimented with PCA on this dataset:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	__	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
id																					
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	__	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
9	53.0	90.0	1.020	2.0	0.0	abnormal	abnormal	present	notpresent	70.0	__	29	12100	3.7	yes	yes	no	poor	no	yes	ckd
11	63.0	70.0	1.010	3.0	0.0	abnormal	abnormal	present	notpresent	380.0	__	32	4500	3.8	yes	yes	no	poor	yes	no	ckd
14	68.0	80.0	1.010	3.0	2.0	normal	abnormal	present	present	157.0	__	16	11000	2.6	yes	yes	yes	poor	yes	no	ckd
20	61.0	80.0	1.015	2.0	0.0	abnormal	abnormal	notpresent	notpresent	173.0	__	24	9200	3.2	yes	yes	yes	poor	yes	yes	ckd
22	48.0	80.0	1.025	4.0	0.0	normal	abnormal	notpresent	notpresent	95.0	__	32	6900	3.4	yes	no	no	good	no	yes	ckd
27	69.0	70.0	1.010	3.0	4.0	normal	abnormal	notpresent	notpresent	264.0	__	37	9600	4.1	yes	yes	yes	good	yes	no	ckd
48	73.0	70.0	1.005	0.0	0.0	normal	normal	notpresent	notpresent	70.0	__	29	18900	3.5	yes	yes	no	good	yes	no	ckd
58	73.0	80.0	1.020	2.0	0.0	abnormal	abnormal	notpresent	notpresent	253.0	__	33	7200	4.3	yes	yes	yes	good	no	no	ckd

Figure 57: PCA EXAMPLE dataset

We reduced the dimensionality of the dataset down to two components:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
T = pca.fit_transform(df)
T
array([[ -1.77599523e+03, -6.50884111e+00],
       [ 3.62374523e+03, -7.73043491e+01],
       [-3.97481492e+03,  2.66180054e+02],
       [ 2.52413939e+03,  1.45475306e+01],
       [ 7.24227321e+02,  3.84761986e+01],
       [-1.57609407e+03, -2.93866865e+01],
       [ 1.12462435e+03,  1.27705575e+02],
       [ 1.04236792e+04, -1.07270983e+02],
       [-1.27540082e+03,  1.27281567e+02],
       [ 6.12413081e+03,  4.67849769e+00],
       [-2.07593939e+03,  6.81590301e+00],
       [-2.27591987e+03,  1.16936466e+01],
       [-4.67614757e+03, -3.47166692e+01],
       [ 1.32469291e+03,  1.42823140e+02],
       [ 4.02435810e+03,  6.09177690e+01],
       [-2.87553505e+03,  1.00337891e+02],
       [-1.47521371e+03,  1.70168344e+02],
       [ 6.72392658e+03, -4.29710106e+01],
       [-3.47550712e+03,  1.07986121e+02],
```

Figure 59: PCA reduce dimensions

```
: # Since we transformed via PCA, we no longer have column names; but we know we
# are in 'principal-component' space, so we'll just define the coordinates accordingly:
ax = drawVectors(T, pca.components_, df.columns.values, plt, scaleFeatures)
T = pd.DataFrame(T)

T.columns = ['component1', 'component2']
T.plot.scatter(x='component1', y='component2', marker='o', c=labels, alpha=0.75, ax=ax)
plt.show()
```

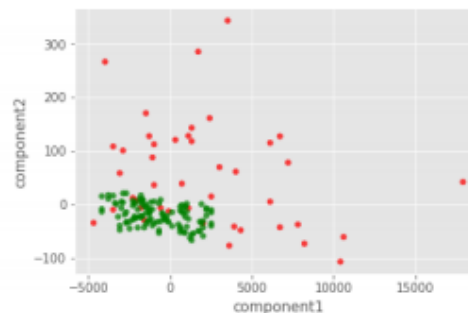


Figure 58: plotting PCA

## Conclusion:

Taking part of the “**Machine Learning A-Z™: Hands-On Python & R In Data Science**” course, has been an insightful and fruitful journey. It allowed me to have a clear and distinct perspective on what can be achievable within the machine learning field and data science in general through relatively accessible algorithms. The course is put together in an orderly way, through various videos, frequent lab assignments and review questions so as to make the learning process easier and pedagogic. All along this MOOC, I have been able to learn theoretical concepts, methods and algorithms that are extremely useful in data analysis. The lab assignments included in this course allowed me to get more familiar with Jupyter notebook, spyder, google collab and expand my skills in R and Python coding language. Eventhough these exercices were sometimes tedious and quite challenging, I was able to complete most of them successfully. All in all, enrolling in this course has made me eager to pursue further in the data science field in the future and put my newly aquired machine learning skills into practice.

## References:

<https://drive.google.com/drive/folders/1OFNnrHRZPZ3unWdErjLHod8lbv2FfG1d?usp=sharing>

<https://www.udemy.com/course/machinelearning/>

<https://iaml.it/blog/optimizing-sklearn-pipelines>

<https://www.analyticsvidhya.com/blog/2017/08/audio-voice-processing-deep-learning/>

[https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html)