EI320A(3) 深度學習使用 Python

Instructors

Tipajin Thaipisutikul (t.greentip@gmail.com)

Prof. Huang-Chia Shih (hcshih@Saturn.yzu.edu.tw)

Course Syllabus

Evaluation Criteria

Tasks	Percentage
In Class Hands-on	60
Project Proposal Presentation	10
Project Final Presentation	30
Bonus (In Class Participation)	5
Total	110/100

Week	Date	Content	Note	Total
1	2/26	Welcome to the course	Download & Install Anaconda Homework (1)	1
2	3/5	Crash Course of Python, Numpy, Pandas, and Matplotlib	In class hands-on (4)	5
3	3/12	Get to know about Data ML:Classification Models	In class hands-on (5)	10
4	3/19	ML:Regression Models	In class hands-on (5)	15
5	3/26	ML:Clustering /Apriori Models	In class hands-on (5)	20
6	4/2	Holiday		
7	4/9	Introduction to Deep Learning (ANN)	In class hands-on (5)	25
8	4/16	Convolutional Neural Network (CNN)	In class hands-on (5)	30
9	4/23	Convolutional Neural Network (CNN)	In class hands-on (5)	35
10	4/30	Recurrent Neural Network (RNN)	In class hands-on (5)	40
11	5/7	Recurrent Neural Network (RNN)	In class hands-on (5)	45
12	5/14	Project Proposal Presentation	Proposal Presentation (10)	55
13	5/21	Time series with DNN, CNN, RNN	In class hands-on (5)	60
14	5/28	Attention Neural Network	In class hands-on (5)	65
15	6/4	Generative Adversarial Network (GAN)	In class hands-on (5)	70
16	6/11	Reinforcement Learning (RL)	In class hands-on (5)	75
17	6/18	Final Project Presentation	Final Presentation (30)	105

Bonus: 5 For class participation.

Preface

ARTIFICIAL INTELLIGENCE

Science thae enables computers to mimic human imtelligence.

Subfields: Machine Learning, robotics, and computer vision

MACHINE LEARNING

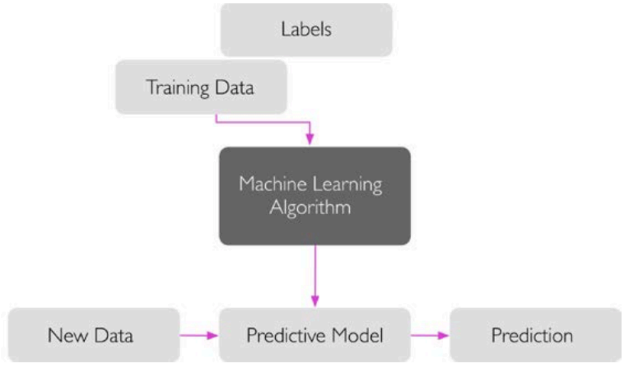
Subset of AI that enable maches to improve at tasks with experience

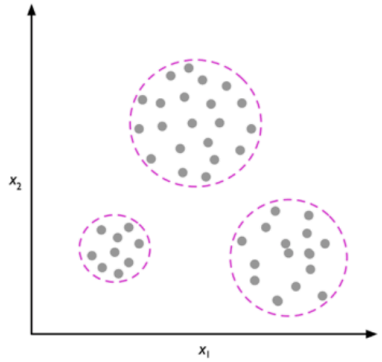
The three different types of machine learning

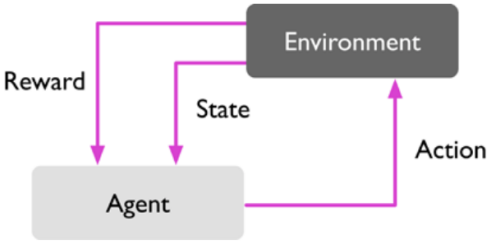
Supervised Learning
- Labeled data
- Direct feedback
- Predict outcome/future

Unsupervised Learning
- No labels
- No feedback
- Find hidden structure in data

Reinforcement Learning
- Decision process
- Reward system
- Learn series of actions



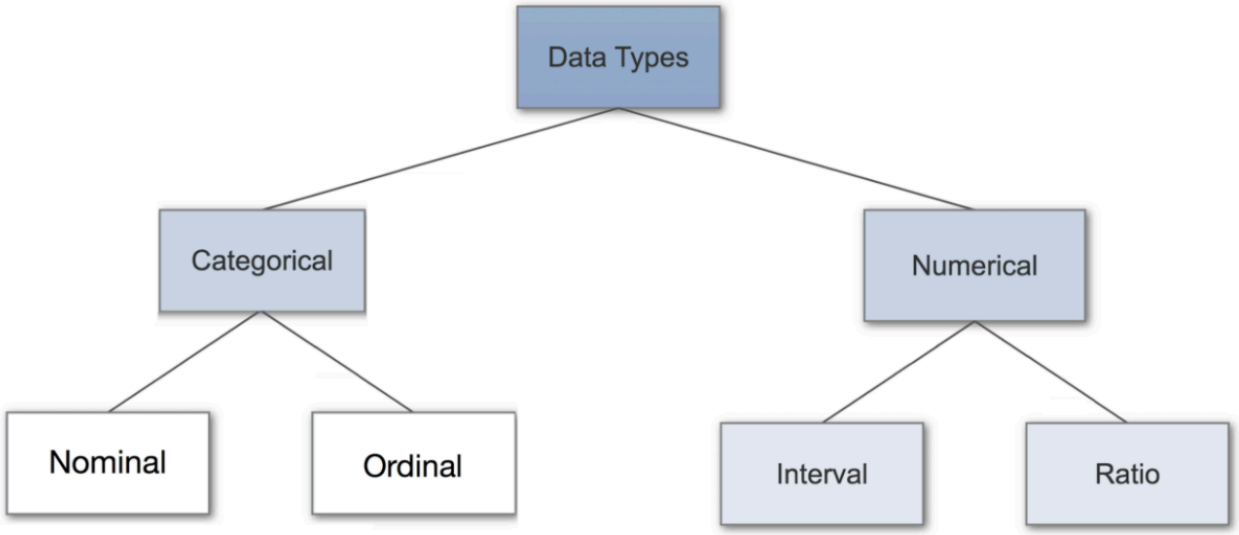




- Machine learning uses algorithms to parse data, learn from that data, and make informed decisions based on what it has learned
- Deep learning structures algorithms in layers to create an artificial "neural network" that can learn and make intelligent decisions on its own
- Deep learning is a subfield of machine learning. While both fall under the broad category of artificial intelligence, deep learning is usually what's behind the most human-like artificial intelligence

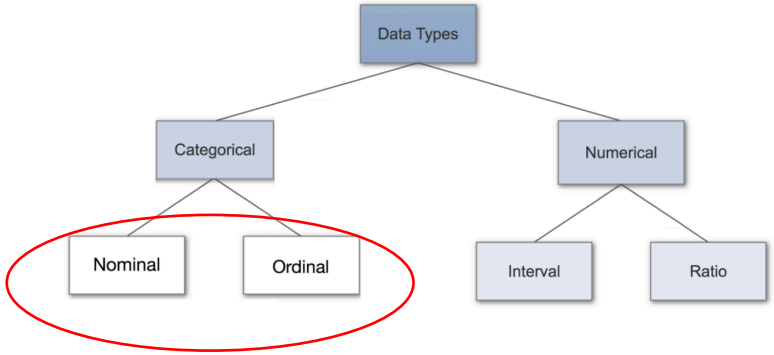
Data Types in Statistics

This section will introduce you to the different data types you need to know, to do proper exploratory data analysis (EDA), which is one of the most underestimated parts of a machine learning project.



Categorical/Discrete Data represents characteristics. Therefore it can represent things like a person's gender, language etc. Categorical data can also take on numerical values (Example: 1 for female and 0 for male).
Note that those numbers don't have mathematical meaning.
Continuous/Numerical Data represents measurements and therefore their values can't be counted but they can be measured. An example would be the height of a person, which you can describe by using intervals on the real number line.

Data Types in Statistics



Nominal values represent discrete units and are used to label variables, that have no quantitative value. Just think of them as labels. Note that nominal data that **has no order**. Therefore if you would change the order of its values, the meaning would not change. You can see two examples of nominal features below:

Are you married?

- Yes
- No

What languages do you speak?

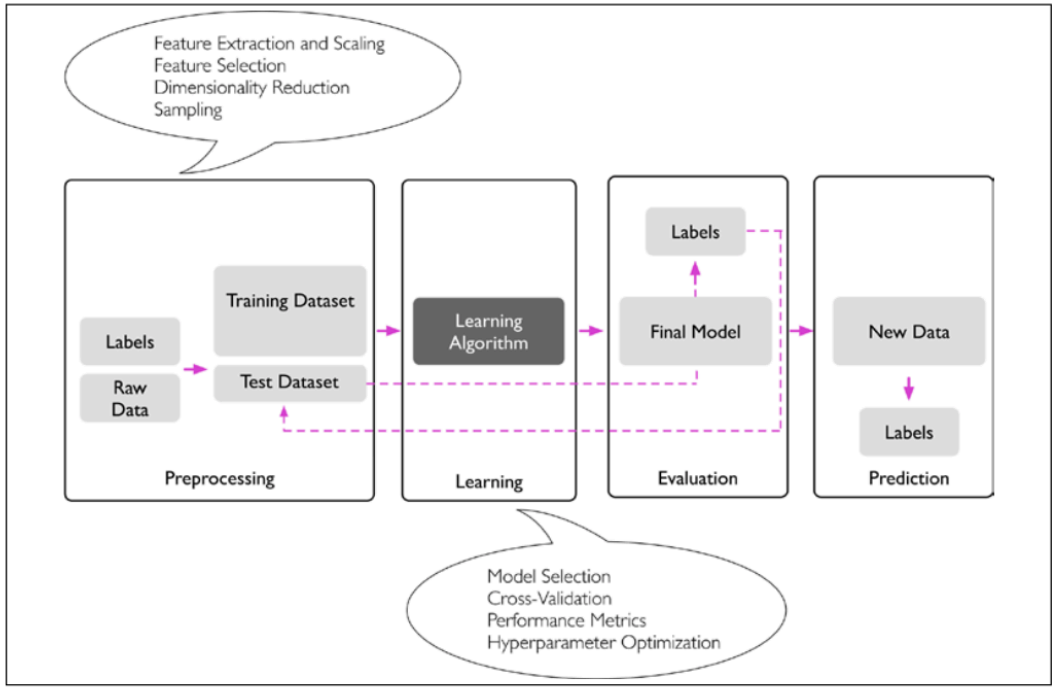
- Englisch
- French
- German
- Spanish

Ordinal values represent discrete and ordered units. It is therefore nearly the **same as nominal data, except that it's ordering matters**. Ordinal scales are usually used to measure non-numeric features like happiness, customer satisfaction and so on. You can see an example below:

What Is Your Educational Background?

- 1 - Elementary
- 2 - High School
- 3 - Undegraduate
- 4 - Graduate

A Roadmap for building ML Systems



A predictive modeling machine learning project can be broken down into **6 top-level tasks:**

1. **Define Problem**: Investigate and characterize the problem in order to better understand the goals of the project.
2. **Analyze Data**: Use **descriptive statistics** and **visualization** to better understand the data you have available.
3. **Prepare Data**: Use **data transforms** in order to better expose the structure of the prediction problem to modeling algorithms.
4. **Evaluate Algorithms:** Design a test harness to evaluate a number of standard algorithms on the data and select the top few to investigate further.
5. **Improve Results:** Use algorithm tuning and ensemble methods to get the most out of well-performing algorithms on your data.
6. **Present Results:** Finalize the model, make predictions and present results.

A Roadmap for building ML Systems

	ML Lesson
	Lesson 1: Python Ecosystem for Machine Learning. Lesson 2: Python and SciPy Crash Course. (Numpy, Pandas, Matplotlib)
Define Problem & Analyze Data	Lesson 3: Understand Data With Descriptive Statistics. Lesson 4: Understand Data With Visualization.
Prepare Data	Lesson 5: Pre-Process Data.
Evaluate Algorithms	Lesson 6: Resampling Methods. Lesson 7: Algorithm Evaluation Metrics. Lesson 8: Spot-Check Classification Algorithms. Lesson 9: Spot-Check Regression Algorithms. Lesson 10: Model Selection.
Improve Results	Lesson 11: Ensemble Methods. Lesson 12: Algorithm Parameter Tuning.
Present Results	Lesson 13: Model Finalization.

3. Understand Data With Descriptive Statistics

You must understand your data in order to get the best results.
There are 7 steps that you can use in Python to better understand your machine learning data.

3.1. Take a peek at your raw data.
3.2. Review the dimensions of your dataset.
3.3. Review the data types of attributes in your data.
3.4. Summarize the distribution of instances across classes in your dataset.
3.5. Summarize your data using descriptive statistics.
3.6. Understand the relationships in your data using correlations.
3.7. Review the skew of the distributions of each attribute.

Each step is demonstrated by loading the *Pima Indians Diabetes classification dataset* from the UCI Machine Learning repository.

3. Understand Data With Descriptive Statistics

Diabetes is a chronic condition that causes a person's blood sugar level to become too high. When this disease takes effect, one may suffer from one or many of not only uncomfortable but also dangerous and sometimes life-threatening symptoms among which are:

- High blood pressure
- Increased risk of infection
- Risk of a heart disease
- Gastroparesis
- Damaged blood vessels
- Pancreas malfunctioning, etc. (World Health Organization, 2018)

https://www.youtube.com/watch?v=oOEMKzhxEQU

3. Understand Data With Descriptive Statistics

Dataset Name: Pima Indians Diabetes Database

Number of observations: 768

Number of variables (columns): 9

Variable	Type	Description
BloodPressure	Integer	Diastolic blood pressure (mm Hg)
Skin Thickness	Integer	Triceps skinfold thickness (mm)
Insulin	Integer	2-hour serum insulin (U/ml)
BMI	Decimal	Body Mass Index (weight in kg / (height in m) ^2)
Diabetes Pedigree Function	Decimal	Diabetes pedigree function
Age	Integer	Age in years
Pregnancies	Integer	Number of times pregnant
Glucose	Integer	Plasma glucose concentration based on a 2-hour oral glucose tolerance test
Outcome	Integer (class)	Whether or not a person has diabetes (1 ┤ has diabetes; 0 - does not have diabetes)

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

3. Understand Data With Descriptive Statistics

3.1. Take a peek at your raw data

- There is no substitute for looking at the raw data.
- Looking at the raw data can reveal insights that you cannot get any other way.
- It can also plant seeds that may later grow into ideas on how to better pre-process and handle the data for machine learning tasks.
- You can review the first 20 rows of your data using the ***head()*** function on the ***Pandas DataFrame***.

```
# View first 20 rows
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
peek = data.head(20)

print(peek)
```

You can see that the first column lists the row number, which is handy for referencing a specific observation.

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
13	1	189	60	23	846	30.1	0.398	59	1
14	5	166	72	19	175	25.8	0.587	51	1
15	7	100	0	0	0	30.0	0.484	32	1
16	0	118	84	47	230	45.8	0.551	31	1
17	7	107	74	0	0	29.6	0.254	31	1
18	1	103	30	38	83	43.3	0.183	33	0
19	1	115	70	30	96	34.6	0.529	32	1

3. Understand Data With Descriptive Statistics

3.2. Dimensions of Your Data

You must have a very good handle on how much data you have, both in terms of rows and columns.

- Too many rows and algorithms may take too long to train. Too few and perhaps you do not have enough data to train the algorithms.
- Too many features and some algorithms can be distracted or suffer poor performance due to the curse of dimensionality.

You can review the shape and size of your dataset by printing the ***shape*** property on the ***Pandas DataFrame.***

```
# Dimensions of your data
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
shape = data.shape
print(shape)
```

```
(768, 9)
```

3. Understand Data With Descriptive Statistics

3.3. Data Type For Each Attribute

The type of each attribute is important.
Strings may need to be converted to floating point values or integers to represent categorical or ordinal values.
You can get an idea of the types of attributes by peeking at the raw data, as above.
You can also list the data types used by the DataFrame to characterize each attribute using the **dtypes** property.

```
# Data Types for Each Attribute
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
types = data.dtypes
print(types)
```

```
preg      int64
plas      int64
pres      int64
skin      int64
test      int64
mass    float64
pedi    float64
age       int64
class     int64
dtype: object
```

3. Understand Data With Descriptive Statistics

3.4. Descriptive Statistics

Descriptive statistics can give you great insight into the properties of each attribute. Often you can create more summaries than you have time to review. The describe() function on the Pandas DataFrame lists 8 statistical properties of each attribute. They are:

- Count.
- Mean.
- Standard Deviation.
- Minimum Value.
- 25th Percentile.
- 50th Percentile (Median).
- 75th Percentile.
- Maximum Value.

```
# Statistical Summary
from pandas import read_csv
from pandas import set_option
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
set_option('display.width', 100)
set_option('precision', 3)
description = data.describe()
print(description)
```

```
       preg    plas    pres    skin    test    mass    pedi     age   class
count  768.000 768.000 768.000 768.000 768.000 768.000 768.000 768.000 768.000
mean     3.845 120.895  69.105  20.536  79.799  31.993   0.472  33.241   0.349
std      3.370  31.973  19.356  15.952 115.244   7.884   0.331  11.760   0.477
min      0.000   0.000   0.000   0.000   0.000   0.000   0.078  21.000   0.000
25%      1.000  99.000  62.000   0.000   0.000  27.300   0.244  24.000   0.000
50%      3.000 117.000  72.000  23.000  30.500  32.000   0.372  29.000   0.000
75%      6.000 140.250  80.000  32.000 127.250  36.600   0.626  41.000   1.000
max     17.000 199.000 122.000  99.000 846.000  67.100   2.420  81.000   1.000
```

- You can see that you do get a lot of data. You will note some calls to **pandas.set option()** in the recipe to change the precision of the numbers and the preferred width of the output.
- This is to make it more readable for this example. When describing your data this way, it is worth taking some time and reviewing observations from the results.
- This might include the presence of NA values for missing data or surprising distributions for attributes.

3. Understand Data With Descriptive Statistics

3.5. Class Distribution (Classification Only)

On classification problems you need to know how balanced the class values are. Highly imbalanced problems (a lot more observations for one class than another) are common and may need special handling in the data preparation stage of your project. You can quickly get an idea of the distribution of the class attribute in Pandas.

```
# Statistical Summary
from pandas import read_csv
from pandas import set_option
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
set_option('display.width', 100)
set_option('precision', 3)
description = data.describe()
print(description)
```

You can see that there are nearly double the number of observations with class 0 (no onset of diabetes) than there are with class 1 (onset of diabetes).

```
class
0    500
1    268
```

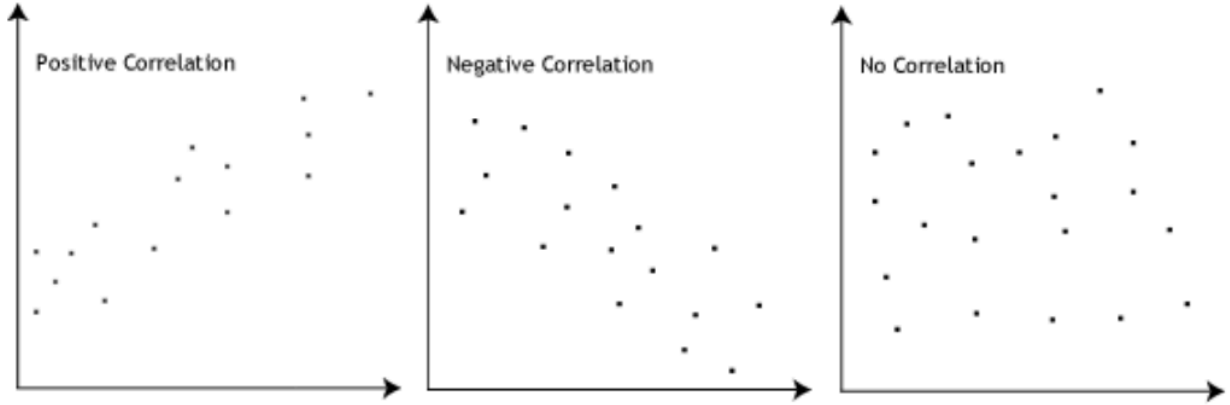
3. Understand Data With Descriptive Statistics

WHAT IS CORRELATION?

It's a measure of how well two variables are related to each other. There are positive as well as negative correlation.

1. **Positive Correlation**: It refers to the extent to which the two variables increases or decreases in parallel (think of this as **directly proportional***, one increases other will increase, one decreases other will follow the same).
2. **Negative Correlation**: It refers to the extent to which one of the two variables increases as the other decreases (think of this as **inversely proportional***, one increases other will decrease or if one decreases other will increase).

The most common correlation in statistics is the **Pearson correlation:** the measure of the strength of **linear association** between two variables.



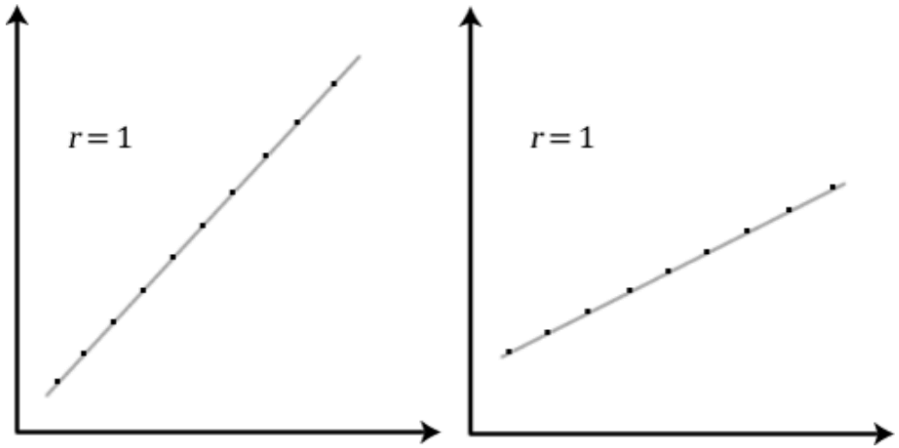
3. Understand Data With Descriptive Statistics

But what does it really represents mathematically?

Basically, a **Pearson Product Moment Correlation (PPMC)**attempts to draw a line to best fit through the data of the given two variables, and the Pearson correlation coefficient "**r**" indicates how far away all these data points are from the line of best fit.

The value of "**r**" ranges from +1 to -1 where:

- **r**= +1/-1 represents that al our data points lie on the line of best fit only i.e there is no data point which shows any variation from the line of best fit.

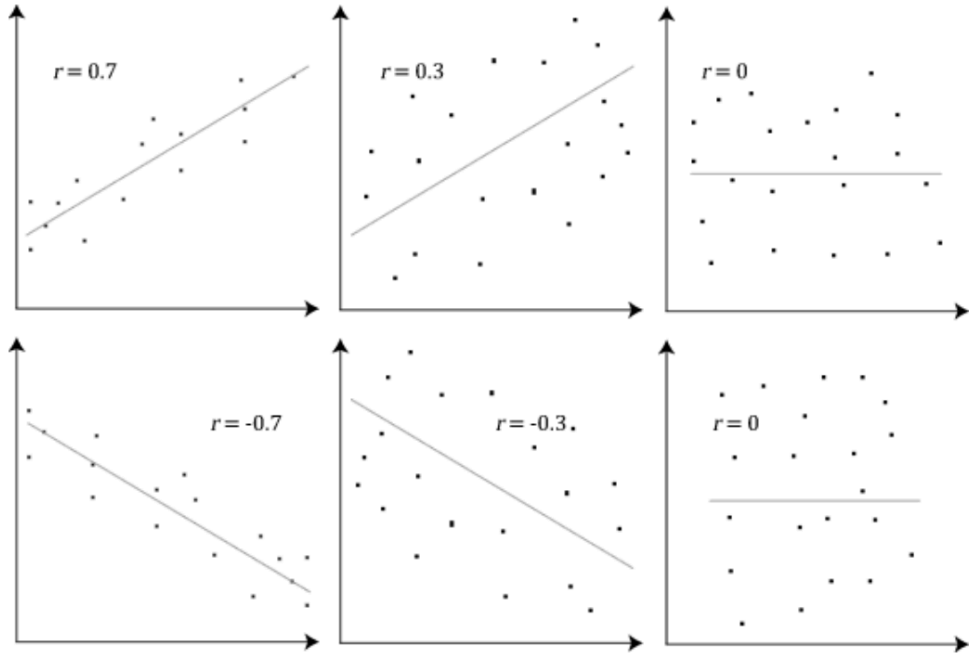


Data points with r=1

3. Understand Data With Descriptive Statistics

- Hence, the stronger the association between the two variables, the closer **r** will be to +1/-1.
- **r** = 0 means that there is no correlation between the two variables.
- The values of **r** between +1 and -1 indicate that there is a variation of data around the line.
- The closer the values of **r** to 0, the greater the variation of data points around the line of best fit.

It is also important to realize that the value of Pearson's coefficient, **r**, is not a measure of the slope of the line (i.e the line of best fit). We can see an example in the plot above with **r=1.**



Data points with **-1 < r < +1**

3. Understand Data With Descriptive Statistics

Formula of Pearson Correlation coefficient:

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[\, n\Sigma x^2 - (\Sigma x)^2 \,][\, n\Sigma y^2 - (\Sigma y)^2 \,]}}$$

An example with calculating Pearson Coefficient:

Find the value of the correlation coefficient from the following table:

SUBJECT	AGE (X)	GLUCOSE LEVEL (Y)
1	43	99
2	21	65
3	25	79
4	42	75
5	57	87
6	59	81

Age and Glucose levels of 6 subjects

We'll calculate the value of **r** using the formula mentioned above. For using that formula we need to compute **Σ(X*Y), Σ(X), Σ(Y), Σ(X²), Σ(Y²)**.

The table below shows the computed values of all the summations mentioned above.

SUBJECT	AGE (X)	GLUCOSE LEVEL (Y)	X*Y	X^2	Y^2
1	43	99	4257	1849	9801
2	21	65	1365	441	4225
3	25	79	1975	625	6241
4	42	75	3150	1764	5625
5	57	87	4959	3249	7569
6	59	81	4779	3481	6561
Σ	247	486	20485	11409	40022

3. Understand Data With Descriptive Statistics

Formula of Pearson Correlation coefficient:

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[\, n\Sigma x^2 - (\Sigma x)^2 \,][\, n\Sigma y^2 - (\Sigma y)^2 \,]}}$$

We'll calculate the value of **r** using the formula mentioned above. For using that formula we need to compute **Σ(X*Y), Σ(X), Σ(Y), Σ(X²), Σ(Y²)**.

The table below shows the computed values of all the summations mentioned above.

SUBJECT	AGE (X)	GLUCOSE LEVEL (Y)	X*Y	X^2	Y^2
1	43	99	4257	1849	9801
2	21	65	1365	441	4225
3	25	79	1975	625	6241
4	42	75	3150	1764	5625
5	57	87	4959	3249	7569
6	59	81	4779	3481	6561
Σ	247	486	20485	11409	40022

From our table we get:

- **Σ(X) = 247**
- **Σ(Y) = 486**
- **Σ(X*Y) = 20,485**
- **Σ(X²) = 11,409**
- **Σ(Y²) = 40,022**
- **n is the sample size, in our case = 6**

r = 6(20,485) — (247 × 486) / [√[[6(11,409) — (24^{72})] × [6(40,022) — 48^{62}]]]

r = 0.5298.

The range of the correlation coefficient is from **-1** to **+1**. Our result is **0.5298** or **52.98%**, which means the variables have a **moderate positive correlation**.

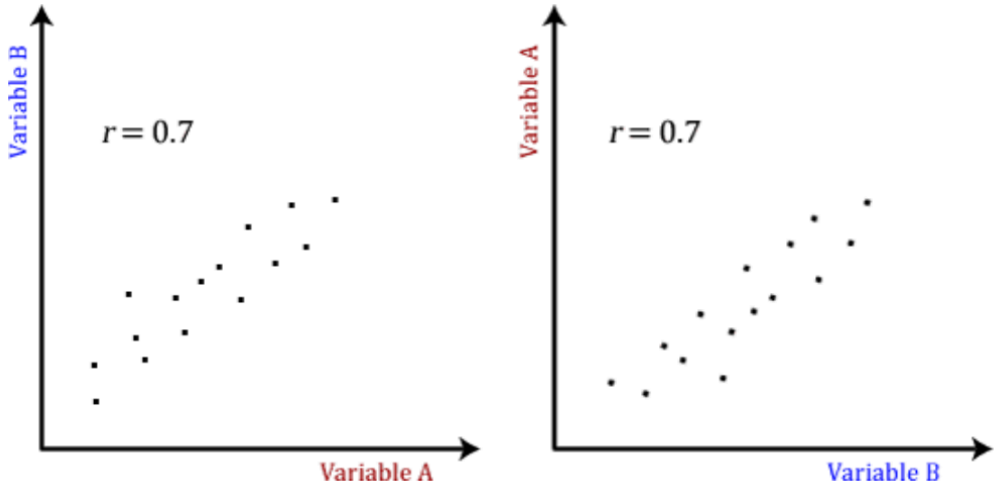
3. Understand Data With Descriptive Statistics

Problems with Pearson correlation? (Potential)

The Pearson product-moment correlation does not take into consideration whether a variable has been classified as a dependent or independent variable. It treats all variables equally.

Example-1 If we are trying to find the correlation between a high-calorie diet and diabetes, we might find a high correlation of **.8**. However, we could also get the same result with the variables switched around. In other words, we could say that diabetes causes a high-calorie diet.

Example-2 We might want to find out whether basketball performance is correlated with a person's height. We might, therefore, plot a graph of performance against height and calculate the Pearson correlation coefficient. Let's say, for example, that $r = .67$. That is, as height increases so do basketball performance. This makes sense. However, if we plotted the variables the other way around and wanted to determine whether a person's height was determined by their basketball performance (which makes no sense), we would still get $r = .67$. This is because the Pearson correlation coefficient makes no account of any theory behind why you chose the two variables to compare. This is illustrated below:



3. Understand Data With Descriptive Statistics

Quiz:

1) As a student's study time increases, so does his test average.
2) When employees make a high salary, efficiency increases.
3) A student who has many absences has a decrease in scores.
4) The older a man gets, the less hair that he has.

3. Understand Data With Descriptive Statistics

3.6. Correlations Between Attributes

--Correlation refers to the relationship between two variables and how they may or may not change together.
The most common method for calculating correlation is Pearson's Correlation Coefficient, that assumes a normal distribution of the attributes involved.

--A correlation of -1 or 1 shows a full negative or positive correlation respectively.
Whereas a value of 0 shows no correlation at all. Some machine learning algorithms like linear and logistic regression can suffer poor performance if there are highly correlated attributes in your dataset.

--As such, it is a good idea to review all of the pairwise correlations of the attributes in your dataset.
You can use the **corr() function on the Pandas DataFrame** to calculate **a correlation matrix.**

3. Understand Data With Descriptive Statistics

3.6. Correlations Between Attributes

```
# Pairwise Pearson correlations
from pandas import read_csv
from pandas import set_option
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
set_option('display.width', 100)
set_option('precision', 3)
correlations = data.corr(method='pearson')
print(correlations)
```

The matrix lists all attributes across the top and down the side, to give correlation between all pairs of attributes (twice, because the matrix is symmetrical). You can see the diagonal line through the matrix from the top left to bottom right corners of the matrix shows perfect correlation of each attribute with itself.

```
        preg   plas   pres   skin   test   mass   pedi    age  class
preg   1.000  0.129  0.141 -0.082 -0.074  0.018 -0.034  0.544  0.222
plas   0.129  1.000  0.153  0.057  0.331  0.221  0.137  0.264  0.467
pres   0.141  0.153  1.000  0.207  0.089  0.282  0.041  0.240  0.065
skin  -0.082  0.057  0.207  1.000  0.437  0.393  0.184 -0.114  0.075
test  -0.074  0.331  0.089  0.437  1.000  0.198  0.185 -0.042  0.131
mass   0.018  0.221  0.282  0.393  0.198  1.000  0.141  0.036  0.293
pedi  -0.034  0.137  0.041  0.184  0.185  0.141  1.000  0.034  0.174
age    0.544  0.264  0.240 -0.114 -0.042  0.036  0.034  1.000  0.238
class  0.222  0.467  0.065  0.075  0.131  0.293  0.174  0.238  1.000
```

3. Understand Data With Descriptive Statistics

3.7. Skew of Univariate Distributions

Skew refers to a distribution that is assumed Gaussian (normal or bell curve) that is shifted or squashed in one direction or another.
Many machine learning algorithms assume a Gaussian distribution. Knowing that an attribute has a skew may allow you to perform data preparation to correct the skew and later improve the accuracy of your models.
You can calculate the skew of each attribute using the **skew() function on the Pandas DataFrame**.

```
# Skew for each attribute
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
skew = data.skew()
print(skew)
```

The skew results show a positive (right) or negative (left) skew. Values closer to zero show less skew.

```
preg     0.901674
plas     0.173754
pres    -1.843608
skin     0.109372
test     2.272251
mass    -0.428982
pedi     1.919911
age      1.129597
class    0.635017
```

Mean, Median, Mode, Range, Interquartile Range

Mean

The ‘Mean” is the average of a set of numbers.

The "Mean" is computed by adding all of the numbers in the data together and dividing by the number of elements contained in the data set.

Example: Data Set = 2, 5, 9, 7, 5, 4, 3
Number of Elements in Data Set = 7
Mean = (2 + 5 + 9 + 7 + 5 + 4 + 3) / 7 = 5

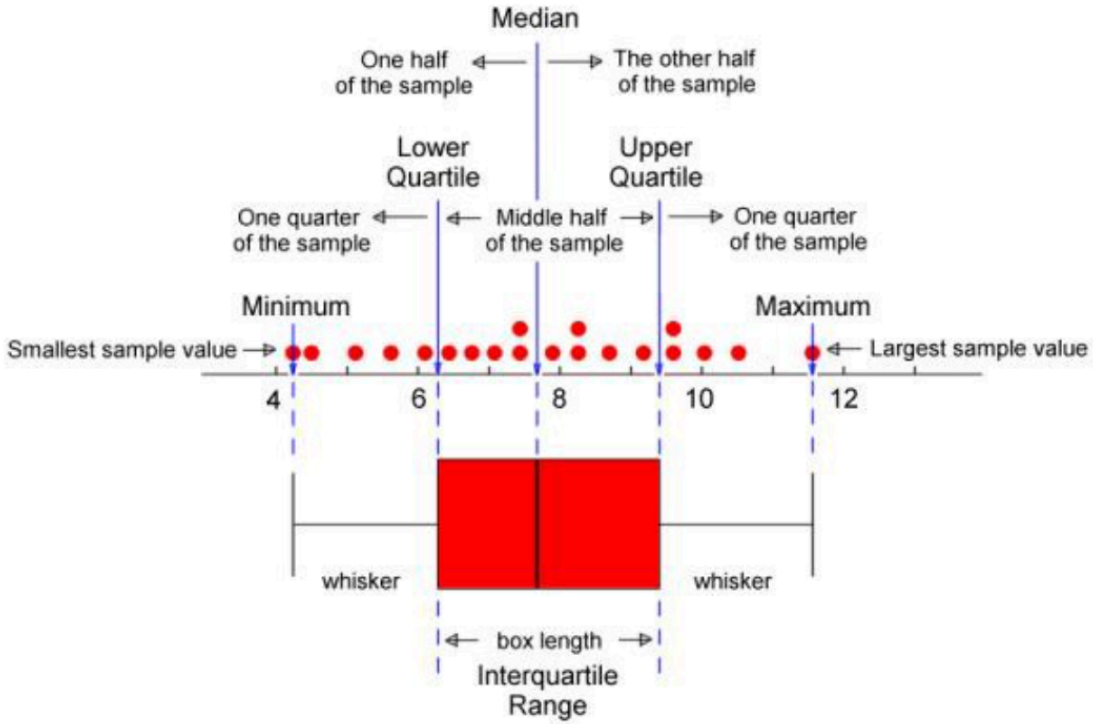
Median

The “Median” is the middle value of a set of **ordered** numbers.

The "Median" of a data set is dependent on whether the number of elements in the data set is odd or even. First reorder the data set from the smallest to the largest. If the number of elements is odd, then the Median is the element in the middle of the data set. If the number of elements is even, then the Median is the average of the two middle terms.

Example: Odd Number of Elements
Data Set = 2, 5, 9, 7, 5, 4, 3
Reordered = 2, 3, 4, 5, 5, 7, 9 - the middle term is 5
Median = 5

Example: Even Number of Elements
Data Set = 2, 5, 9, 3, 5, 4
Reordered = 2, 3, 4, 5, 5, 9 - the middle terms are 4 and 5
Median = (4 + 5) / 2 = 4.5 - the median is the average of the two middle terms



Mean, Median, Mode, Range, Interquartile Range

Mode

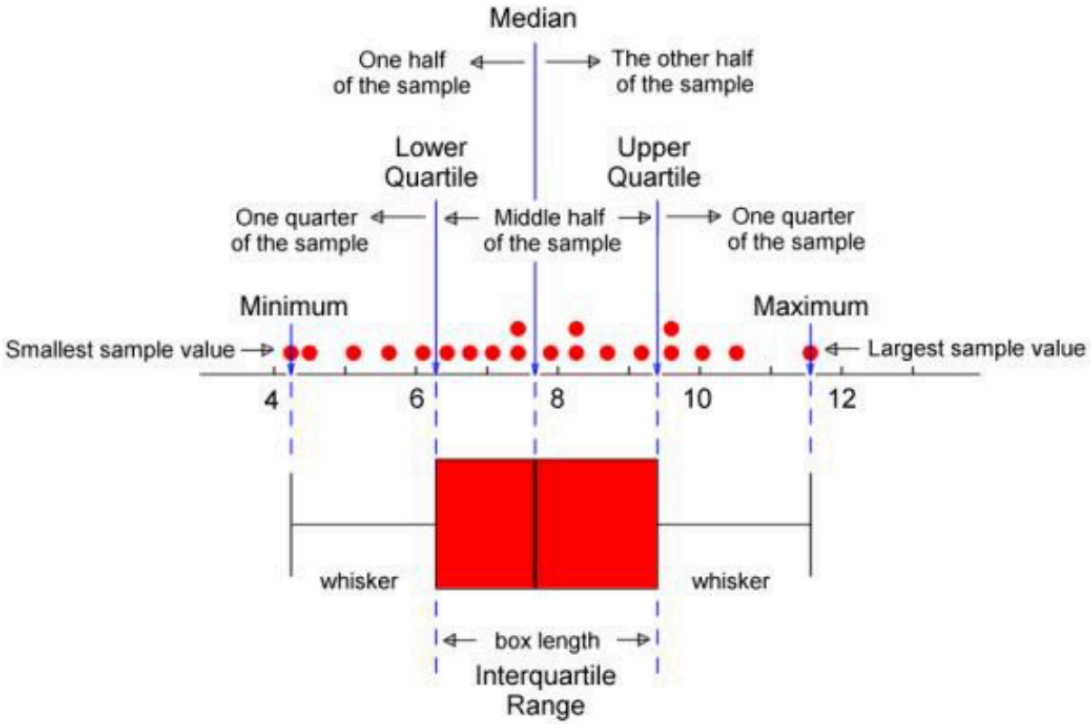
The "Mode" for a set of data is the value that occurs most often.

It is not uncommon for a data set to have more than one mode. This happens when two or more elements occur with equal frequency in the data set.

Example: One Mode
Data Set = 2, 5, 9, 7, 5, 4, 3
Mode = 5

Examples: Two Modes
Data Set = 2, 5, 2, 3, 5, 4, 7
Modes = 2 and 5

Example: Three Modes
Data Set = 2, 5, 2, 7, 5, 4, 7
Modes = 2, 5, and 7



Range

The "Range" is the difference between the largest value and smallest value in a set of data.

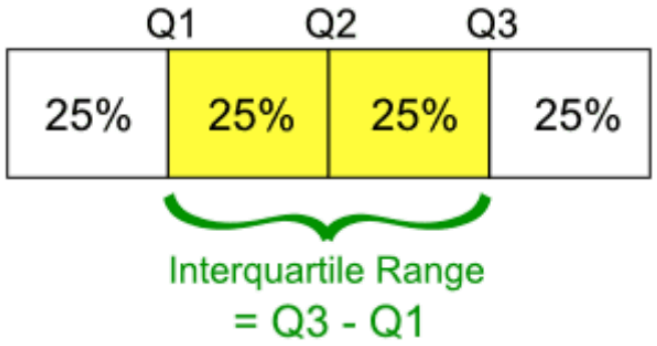
First reorder the data set from smallest to largest then subtract the first element from the last element.

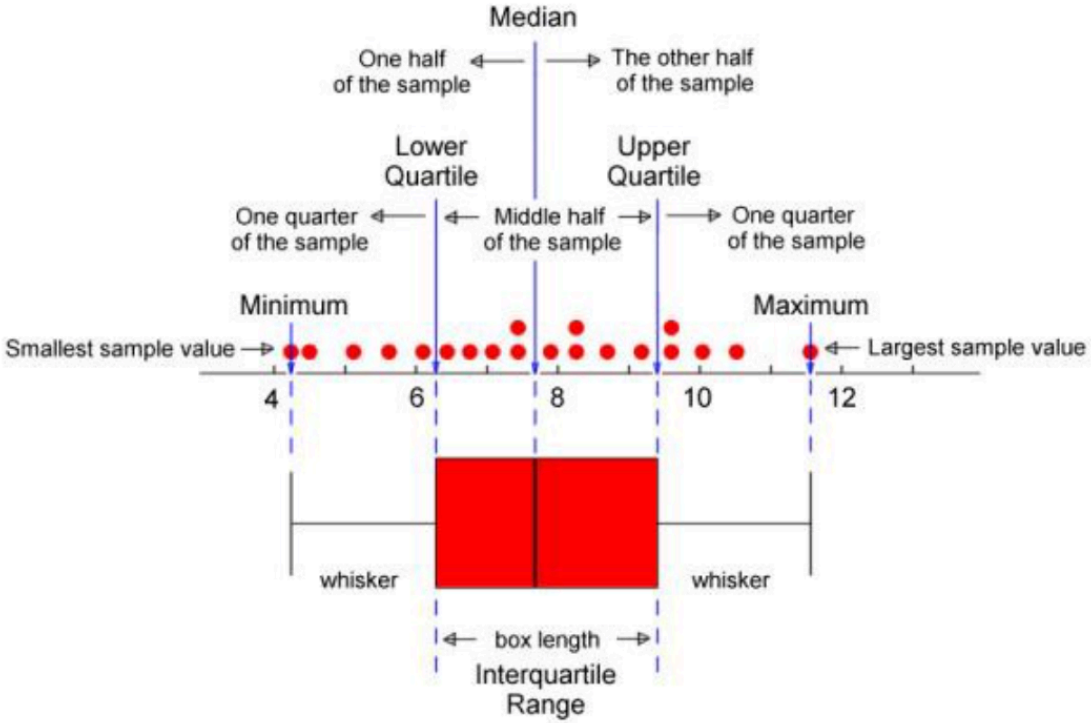
Example:
Data Set = 2, 5, 9, 7, 5, 4, 3
Reordered = 2, 3, 4, 5, 5, 7, 9
Range = (9 - 2) = 7

Interquartile Range

The “Interquartile Range” is the difference between smallest value and the largest value of the middle 50% of a set of data.

The "Interquartile Range" is from Q1 to Q3:



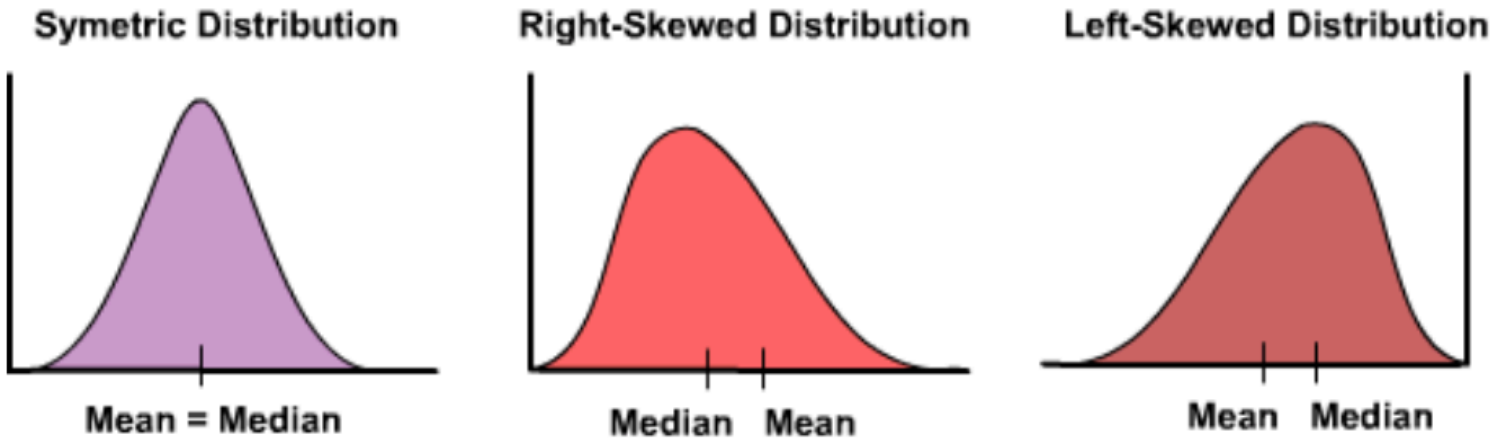


To find the interquartile range of a set of data:

- First put the list of numbers in order
- Then cut the list into four equal parts
- The quartiles are the “cuts”
- The interquartile range is the distance between the two middle sets of data

3. Understand Data With Descriptive Statistics

3.7. Skew of Univariate Distributions



If tail is on the right as that of the second image in the figure, it is right skewed data. It is also called **positive skewed data.** If the tail is to the left of data, then it is called left skewed data. It is also called **negatively skewed data.**

4. Understand Your Data With Visualization

4.1. Univariate Plots

Histograms

A fast way to get an idea of the distribution of each attribute is to look at histograms. Histograms group data into bins and provide you a count of the number of observations in each bin. From the shape of the bins you can quickly get a feeling for whether an attribute is Gaussian, skewed or even has an exponential distribution. It can also help you see possible outliers.

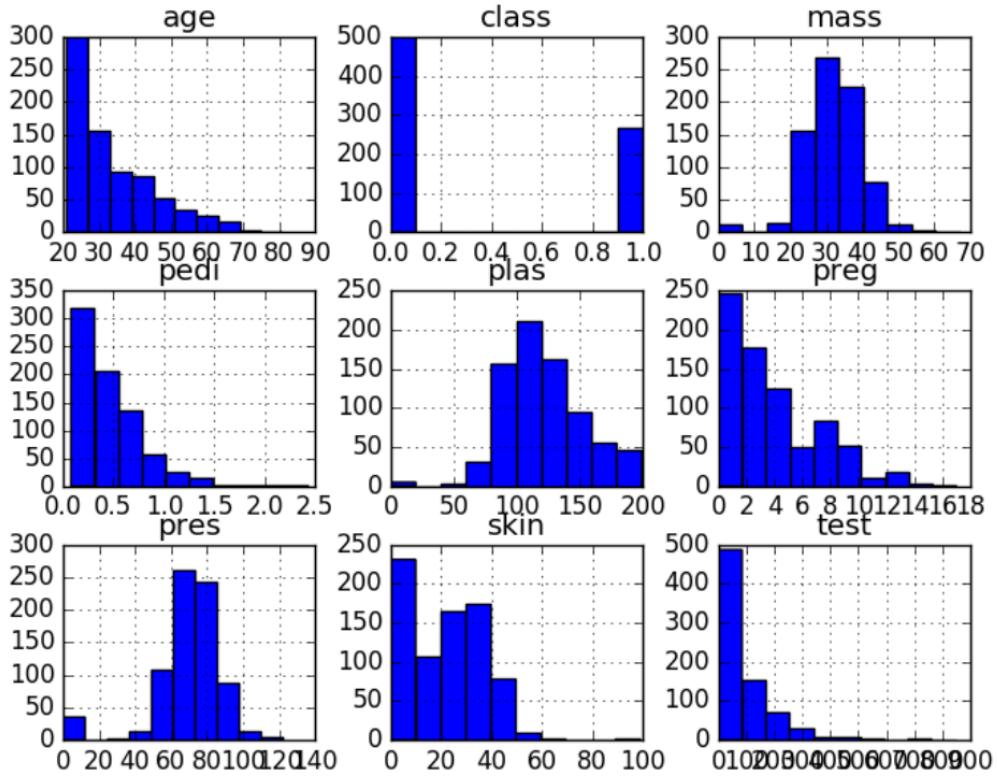
```
# Univariate Histograms
from matplotlib import pyplot
from pandas import read_csv
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data.hist()
pyplot.show()
```

4. Understand Your Data With Visualization

4.1. Univariate Plots

Histograms

We can see that perhaps the attributes `age`, `pedi` and `test` may have an exponential distribution. We can also see that perhaps the `mass` and `pres` and `plas` attributes may have a Gaussian or nearly Gaussian distribution. This is interesting because many machine learning techniques assume a Gaussian univariate distribution on the input variables.



4. Understand Your Data With Visualization

4.1. Univariate Plots

Density Plots

Density plots are another way of getting a quick idea of the distribution of each attribute. The plots look like an abstracted histogram with a smooth curve drawn through the top of each bin, much like your eye tried to do with the histograms.

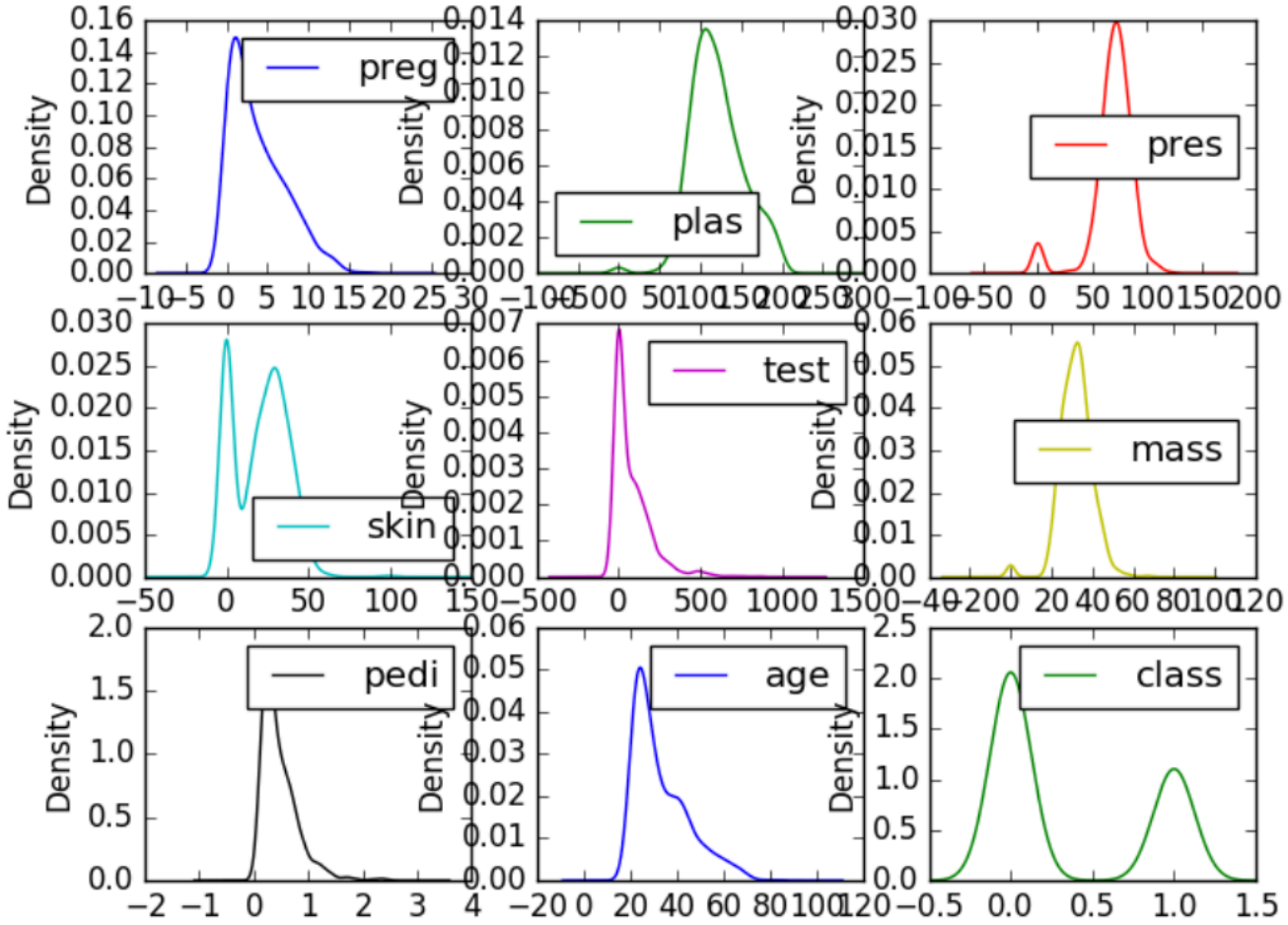
```
# Univariate Density Plots
from matplotlib import pyplot
from pandas import read_csv
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data.plot(kind='density', subplots=True, layout=(3,3), sharex=False)
pyplot.show()
```

4. Understand Your Data With Visualization

4.1. Univariate Plots

We can see the distribution for each attribute is clearer than the histograms.

Density Plots



4. Understand Your Data With Visualization

4.1. Univariate Plots

Box and Whisker Plots

Another useful way to review the distribution of each attribute is to use Box and Whisker Plots or boxplots for short. Boxplots summarize the distribution of each attribute, drawing a line for the median (middle value) and a box around the 25th and 75th percentiles (the middle 50% of the data). The whiskers give an idea of the spread of the data and dots outside of the whiskers show candidate outlier values (values that are 1.5 times greater than the size of spread of the middle 50% of the data).

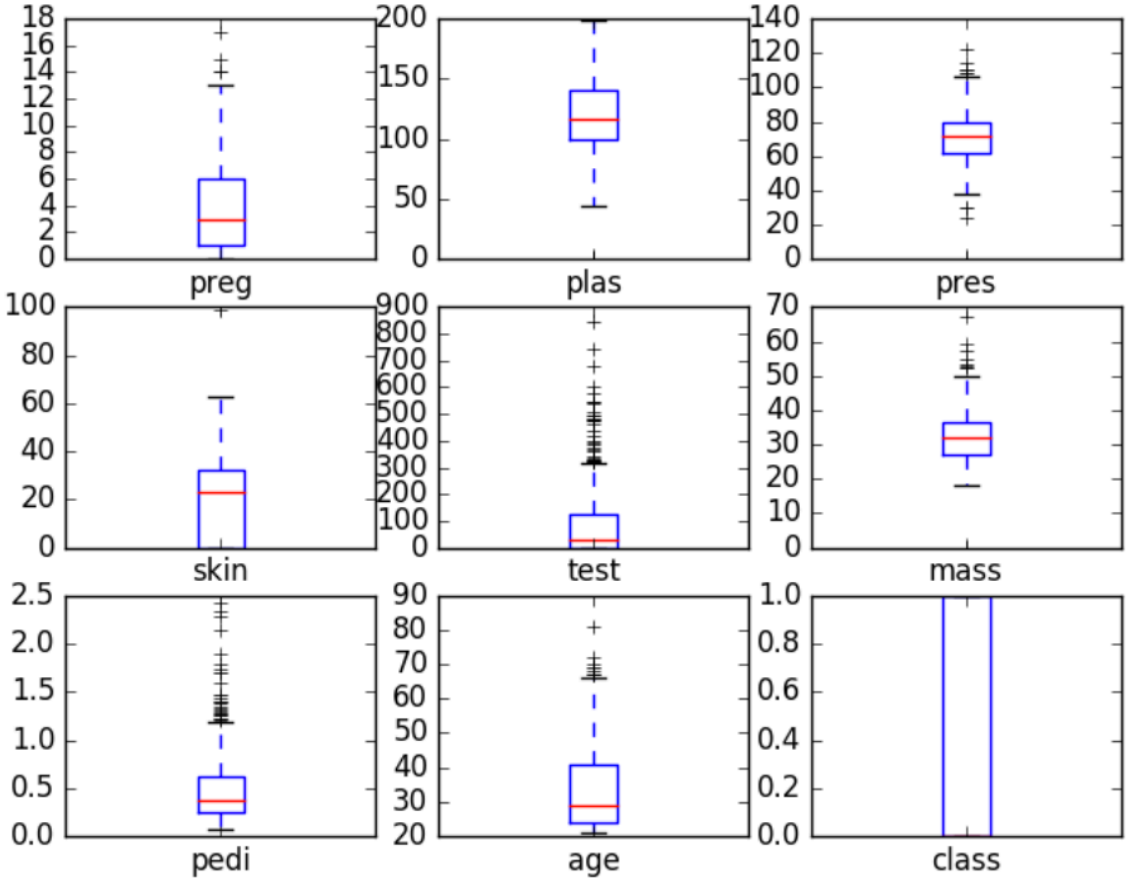
```
# Box and Whisker Plots
from matplotlib import pyplot
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False)
pyplot.show()
```

4. Understand Your Data With Visualization

4.1. Univariate Plots

Box and Whisker Plots

We can see that the spread of attributes is quite different. Some like `age`, `test` and `skin` appear quite skewed towards smaller values.



4. Understand Your Data With Visualization

4.2. Multivariate Plots

This section provides examples of two plots that show the interactions between multiple variables in your dataset. Correlation Matrix Plot, Scatter Plot Matrix.

Correlation Matrix Plot

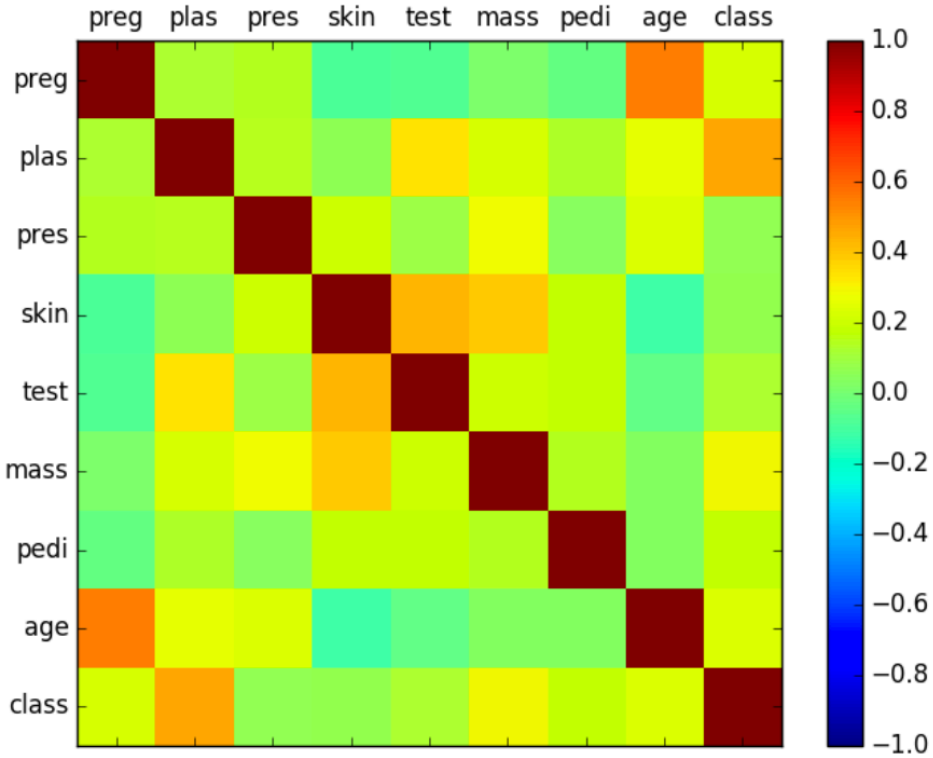
Correlation gives an indication of how related the changes are between two variables. If two variables change in the same direction they are positively correlated. If they change in opposite directions together (one goes up, one goes down), then they are negatively correlated. You can calculate the correlation between each pair of attributes. This is called a correlation matrix. You can then plot the correlation matrix and get an idea of which variables have a high correlation with each other. This is useful to know, because some machine learning algorithms like linear and logistic regression can have poor performance if there are highly correlated input variables in your data.

4. Understand Your Data With Visualization

4.2. Multivariate Plots

Correlation Matrix Plot

```
# Correlation Matrix Plot
from matplotlib import pyplot
from pandas import read_csv
import numpy
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
correlations = data.corr()
# plot correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```



4. Understand Your Data With Visualization

4.2. Multivariate Plots

This section provides examples of two plots that show the interactions between multiple variables in your dataset. Correlation Matrix Plot, Scatter Plot Matrix.

Scatter Plot Matrix

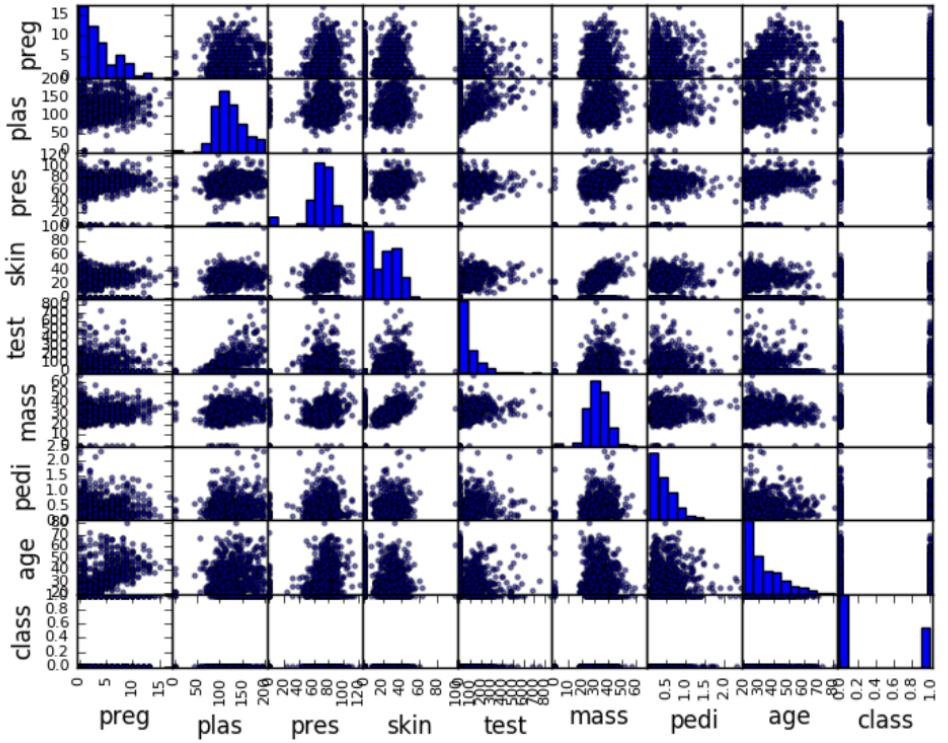
A scatter plot shows the relationship between two variables as dots in two dimensions, one axis for each attribute. You can create a scatter plot for each pair of attributes in your data. Drawing all these scatter plots together is called a scatter plot matrix. Scatter plots are useful for spotting structured relationships between variables, like whether you could summarize the relationship between two variables with a line. Attributes with structured relationships may also be correlated and good candidates for removal from your dataset.

4. Understand Your Data With Visualization

4.2. Multivariate Plots

Scatter Plot Matrix

```
# Scatterplot Matrix
from matplotlib import pyplot
from pandas import read_csv
from pandas.plotting import scatter_matrix
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
scatter_matrix(data)
pyplot.show()
```



A Roadmap for building ML Systems

	ML Lesson
	Lesson 1: Python Ecosystem for Machine Learning. Lesson 2: Python and SciPy Crash Course. (Numpy, Pandas, Matplotlib)
Define Problem & Analyze Data	Lesson 3: Understand Data With Descriptive Statistics. Lesson 4: Understand Data With Visualization.
Prepare Data	Lesson 5: Pre-Process Data.
Evaluate Algorithms	Lesson 6: Resampling Methods. Lesson 7: Algorithm Evaluation Metrics. Lesson 8: Spot-Check Classification Algorithms. Lesson 9: Spot-Check Regression Algorithms. Lesson 10: Model Selection.
Improve Results	Lesson 11: Ensemble Methods. Lesson 12: Algorithm Parameter Tuning.
Present Results	Lesson 13: Model Finalization.

5. Prepare Your Data For Machine Learning

- Many machine learning algorithms make assumptions about your data.
- It is often a very good idea to prepare your data in such a way to best expose the structure of the problem to the machine learning algorithms that you intend to use.
- In this section you will discover how to **prepare your data for machine learning in Python** using **scikit-learn.**
- After completing this lesson you will know how to:
    1. Rescale data.
    2. Standardize data.
    3. Normalize data.
    4. Binarize data.

5.1. Rescale Data

When your data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale.

Often this is referred to as normalization and attributes are often rescaled into the range between 0 and 1.

This is useful for optimization algorithms used in the core of machine learning algorithms like gradient descent.

It is also useful for algorithms that weight inputs like regression and neural networks and algorithms that use distance measures like k-Nearest Neighbors.

You can rescale your data using **scikit-learn** using the **MinMaxScaler** class.

5. Prepare Your Data For Machine Learning

5.1. Rescale Data

You can rescale your data using **scikit-learn** using the **MinMaxScaler** class.

```
# Rescale data (between 0 and 1)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

After rescaling you can see that all of the values are in the range between 0 and 1.

```
[[ 0.353 0.744 0.59  0.354 0.    0.501 0.234 0.483]
 [ 0.059 0.427 0.541 0.293 0.    0.396 0.117 0.167]
 [ 0.471 0.92  0.525 0.    0.    0.347 0.254 0.183]
 [ 0.059 0.447 0.541 0.232 0.111 0.419 0.038 0.   ]
 [ 0.    0.688 0.328 0.354 0.199 0.642 0.944 0.2 ]]
```

5. Prepare Your Data For Machine Learning

5.2. Standardize Data

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data, such as linear regression, logistic regression and linear discriminate analysis.

You can standardize data using **scikit-learn with the StandardScaler class**

```
# Standardize data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

The values for each attribute now have a mean value of 0 and a standard deviation of 1.

```
[[ 0.64   0.848  0.15   0.907 -0.693  0.204  0.468  1.426]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.684 -0.365 -0.191]
 [ 1.234  1.944 -0.264 -1.288 -0.693 -1.103  0.604 -0.106]
 [-0.845 -0.998 -0.161  0.155  0.123 -0.494 -0.921 -1.042]
 [-1.142  0.504 -1.505  0.907  0.766  1.41   5.485 -0.02 ]]
```

5. Prepare Your Data For Machine Learning

5.3. Normalize Data

Normalizing in scikit-learn refers to rescaling each observation (row) to have a length of 1
(called a unit norm or a vector with the length of 1 in linear algebra).
This pre-processing method can be useful for sparse datasets (lots of zeros) with attributes of varying scales when using algorithms that weight input values such as neural networks and algorithms that use distance measures such as k-Nearest Neighbors.
You can normalize data in **Python with scikit-learn using the Normalizer class**

```
# Normalize data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(normalizedX[0:5,:])
```

The rows are normalized to length 1.

```
[[ 0.034 0.828 0.403 0.196 0.    0.188 0.004 0.28 ]
 [ 0.008 0.716 0.556 0.244 0.    0.224 0.003 0.261]
 [ 0.04  0.924 0.323 0.    0.    0.118 0.003 0.162]
 [ 0.007 0.588 0.436 0.152 0.622 0.186 0.001 0.139]
 [ 0.    0.596 0.174 0.152 0.731 0.188 0.01  0.144]]
```

5. Prepare Your Data For Machine Learning

5.4. Binarize Data (Make Binary)

You can transform your data using a binary threshold.

All values above the threshold are marked 1 and all equal to or below are marked as 0.

This is called binarizing your data or thresholding your data. It can be useful when you have probabilities that you want to make into crisp values. It is also useful when feature engineering and you want to add new features that indicate something meaningful.

You can create new binary attributes in **Python using scikit-learn with the Binarizer class**

```
# binarization
from sklearn.preprocessing import Binarizer
from pandas import read_csv
```

```
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
binarizer = Binarizer(threshold=0.0).fit(X)
binaryX = binarizer.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(binaryX[0:5,:])
```

You can see that all values equal or less than 0 are marked 0 and all of those above 0 are marked 1.

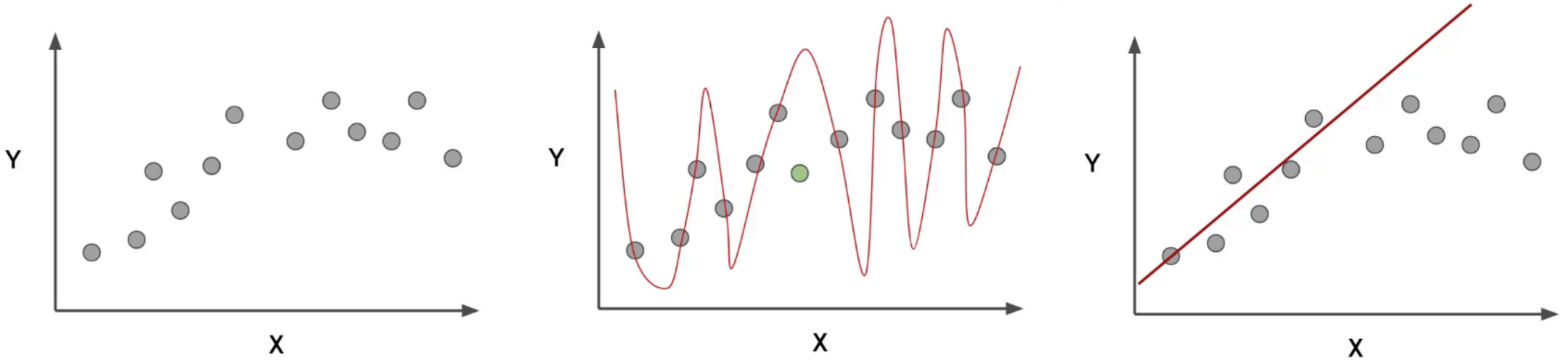
```
[[ 1.  1.  1.  1.  0.  1.  1.  1.]
 [ 1.  1.  1.  1.  0.  1.  1.  1.]
 [ 1.  1.  1.  0.  0.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.]
 [ 0.  1.  1.  1.  1.  1.  1.  1.]]
```

A Roadmap for building ML Systems

	ML Lesson
	Lesson 1: Python Ecosystem for Machine Learning. Lesson 2: Python and SciPy Crash Course. (Numpy, Pandas, Matplotlib)
Define Problem & Analyze Data	Lesson 3: Understand Data With Descriptive Statistics. Lesson 4: Understand Data With Visualization.
Prepare Data	Lesson 5: Pre-Process Data.
Evaluate Algorithms	Lesson 6: Resampling Methods. Lesson 7: Algorithm Evaluation Metrics. Lesson 8: Spot-Check **Classification** Algorithms. Lesson 9: Spot-Check Regression Algorithms. Lesson 10: Model Selection.
Improve Results	Lesson 11: Ensemble Methods. Lesson 12: Algorithm Parameter Tuning.
Present Results	Lesson 13: Model Finalization.

6. Evaluate the Performance of Machine Learning Algorithms with Resampling

Why can’t you prepare your machine learning algorithm on your training dataset and use predictions from this same dataset to evaluate performance? The simple answer is overfitting.

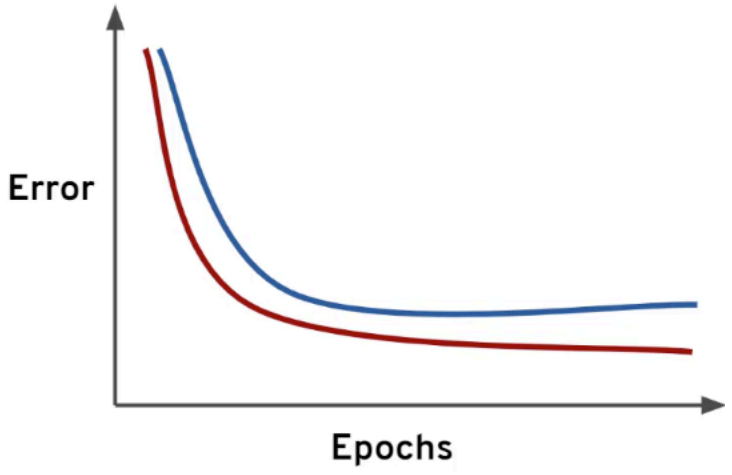


Overfitting

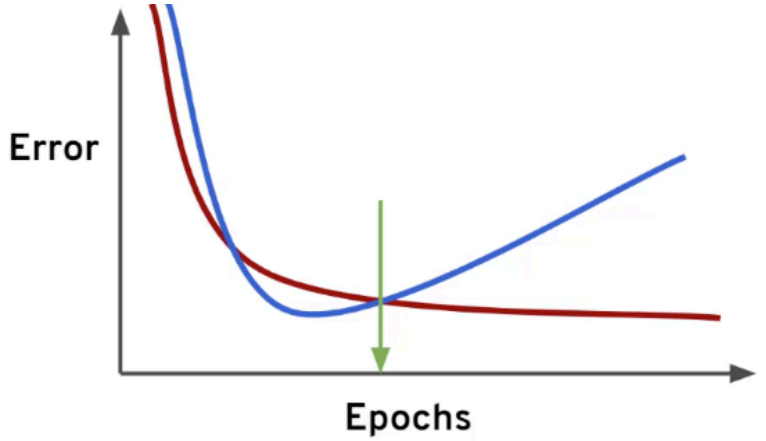
Underfitting

6. Evaluate the Performance of Machine Learning Algorithms with Resampling

Why can’t you prepare your machine learning algorithm on your training dataset and use predictions from this same dataset to evaluate performance? The simple answer is overfitting.



Train/Test Ideally



Cut Off Point at training (Overfitting)

6. Evaluate the Performance of Machine Learning Algorithms with Resampling

6.1. Split into Train and Test Sets

The simplest method that we can use to evaluate the performance of a machine learning algorithm is to use different training and testing datasets. We can take our original dataset and split it into two parts. Train the algorithm on the first part, make predictions on the second part and evaluate the predictions against the expected results. The size of the split can depend on the size and specifics of your dataset, although it is common to use ~70% of the data for training and the remaining ~30% for testing.

```
# Evaluate using a train and a test set
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result*100.0))
```

```
Accuracy: 75.591%
```

6. Evaluate the Performance of Machine Learning Algorithms with Resampling

6.2. K-fold Cross-Validation

Cross-validation is an approach that you can use to estimate the performance of a machine learning algorithm with less variance than a single train-test set split. It works by splitting the dataset into k-parts (e.g. k = 5 or k = 10). Each split of the data is called a fold. The algorithm is trained on k − 1 folds with one held back and tested on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the held back test set. After running cross-validation you end up with k different performance scores that you can summarize using a mean and a standard deviation.

```
# Evaluate using Cross Validation
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = LogisticRegression(solver='liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 76.951% (4.841%)
```

7. Machine Learning Algorithm Performance Metrics

The metrics that you choose to evaluate your machine learning algorithms are very important.
Choice of metrics influences how the performance of machine learning algorithms is measured and compared.
They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose.

Classification problems are perhaps the most common type of machine learning problem and as such there is a myriad of metrics that can be used to evaluate predictions for these problems. In this section we will review how to use the following metrics:

1. Classification Accuracy
2. Confusion Matrix (Precision, Recall, F-Measure)
3. Classification Report

7. Machine Learning Algorithm Performance Metrics

1. Classification Accuracy

Classification accuracy is the number of correct predictions made as a ratio of all predictions made.

This is the most common evaluation metric for classification problems, it is also the most misused.

It is really only suitable when there are an equal number of observations in each class (which is rarely the case) and that all predictions and prediction errors are equally important, which is often not the case.

Example: Imagine we had 99 images of dogs and 1 image of cat. If our model was simply predicted dog we would get 99% accuracy!

Below is an example of calculating classification accuracy.

```
# Cross Validation Classification Accuracy
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = LogisticRegression(solver='liblinear')
scoring = 'accuracy'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
```

```
Accuracy: 0.770 (0.048)
```

7. Machine Learning Algorithm Performance Metrics

2. Confusion Matrix (Precision, Recall, F-Measure)

The confusion matrix is a handy presentation of the accuracy of a model with two or more classes. The table presents predictions on the x-axis and true outcomes on the y-axis. The cells of the table are the number of predictions made by a machine learning algorithm. For example, a machine learning algorithm can predict 0 or 1 and each prediction may actually have been a 0 or 1. Predictions for 0 that were actually 0 appear in the cell for prediction = 0 and actual = 0, whereas predictions for 0 that were actually 1 appear in the cell for prediction = 0 and actual = 1. And so on.

		predicted condition	
	total population	prediction positive	prediction negative
true condition	condition positive	**True Positive (TP)**	**False Negative (FN)** (type II error)
	condition negative	**False Positive (FP)** (Type I error)	**True Negative (TN)**

Precision = TruePositives / (TruePositives + FalsePositives)

Recall = TruePositives / (TruePositives + FalseNegatives)

F-Measure = (2 * Precision * Recall) / (Precision + Recall)

7. Machine Learning Algorithm Performance Metrics

2. Confusion Matrix (Precision, Recall, F-Measure)

	Predicted class POSITIVE (spam ✉)	**Predicted class NEGATIVE (normal 📧)**	
Actual class POSITIVE (spam ✉)	TRUE POSITIVE (TP) ✉ ✉ **320**	FALSE NEGATIVE (FN) ✉ 📧 **43**	*Recall* $= \frac{TP}{TP + FN}$ $= \frac{320}{320 + 43} = 0.882$
Actual class NEGATIVE (normal ✉)	FALSE POSITIVE (FP) ✉ ✉ **20**	TRUE NEGATIVE (TN) ✉ 📧 **538**	
	Precision $= \frac{TP}{TP + FP}$ $= \frac{320}{320 + 20} = 0.941$		

Summarize Confusion Matrix in 3 Mins
Group of 2-3 persons

7. Machine Learning Algorithm Performance Metrics

2. Confusion Matrix (Precision, Recall, F-Measure)

Below is an example of calculating a confusion matrix for a set of predictions by a Logistic Regression on the Pima Indians onset of diabetes dataset.

```
# Cross Validation Classification Confusion Matrix
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)

print(matrix)
```

Although the array is printed without headings, you can see that the majority of the predictions fall on the diagonal line of the matrix (which are correct predictions).

```
[[141  21]
 [ 41  51]]
```

7. Machine Learning Algorithm Performance Metrics

3. Classification Report

The scikit-learn library provides a convenience report when working on classification prob- lems to give you a quick idea of the accuracy of a model using a number of measures. The classification report() function displays the precision, recall, F1-score and support for each class.

The example below demonstrates the report on the binary classification problem.

```
# Cross Validation Classification Report
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

You can see good precision and recall for the algorithm.

	precision	recall	f1-score	support
0.0	0.77	0.87	0.82	162
1.0	0.71	0.55	0.62	92
avg / total	0.75	0.76	0.75	254

8. Spot-Check Classification Algorithms

Spot-checking is a way of discovering which algorithms perform well on your machine learning problem. You cannot know which algorithms are best suited to your problem beforehand. You must trial a number of methods and focus attention on those that prove themselves the most promising.

8.1. Logistic Regression

Logistic regression assumes a Gaussian distribution for the numeric input variables and can model binary classification problems. You can construct a logistic regression model using the LogisticRegression class

```
# Logistic Regression Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = LogisticRegression(solver='liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example prints the mean estimated accuracy.

```
0.76951469583
```

8. Spot-Check Classification Algorithms

8.2. Linear Discriminant Analysis

Linear Discriminant Analysis or LDA is a statistical technique for binary and multiclass classification. It too assumes a Gaussian distribution for the numerical input variables.

You can construct an LDA model using the LinearDiscriminantAnalysis class

```
# LDA Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = LinearDiscriminantAnalysis()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example prints the mean estimated accuracy.

```
0.773462064252
```

8. Spot-Check Classification Algorithms

8.3. k-Nearest Neighbors

The k-Nearest Neighbors algorithm (or KNN) uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction.

You can construct a KNN model using the KNeighborsClassifier class

```
# KNN Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = KNeighborsClassifier()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example prints the mean estimated accuracy.

```
0.726555023923
```

8. Spot-Check Classification Algorithms

8.4. Naive Bayes

Naive Bayes calculates the probability of each class and the conditional probability of each class given each input value. These probabilities are estimated for new data and multiplied together, assuming that they are all independent (a simple or naive assumption). When working with real-valued data, a Gaussian distribution is assumed to easily estimate the probabilities for input variables using the Gaussian Probability Density Function.

You can construct a Naive Bayes model using the GaussianNB class

```
# Gaussian Naive Bayes Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = GaussianNB()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example prints the mean estimated accuracy.

```
0.75517771702
```

8. Spot-Check Classification Algorithms

8.5. Decision Tree

Classification and Regression Trees (CART or just decision trees) construct a binary tree from the training data. Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function (like the Gini index). You can construct a CART model using the DecisionTreeClassifier class

```
# CART Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = DecisionTreeClassifier()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example prints the mean estimated accuracy.

```
0.692600820232
```

8. Spot-Check Classification Algorithms

8.6. Support Vector Machines

Support Vector Machines (or SVM) seek a line that best separates two classes. Those data instances that are closest to the line that best separates the classes are called support vectors and influence where the line is placed. SVM has been extended to support multiple classes. Of particular importance is the use of different kernel functions via the kernel parameter. A powerful Radial Basis Function is used by default. You can construct an SVM model using the SVC class

```
# SVM Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = SVC()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example prints the mean estimated accuracy.

```
0.651025290499
```

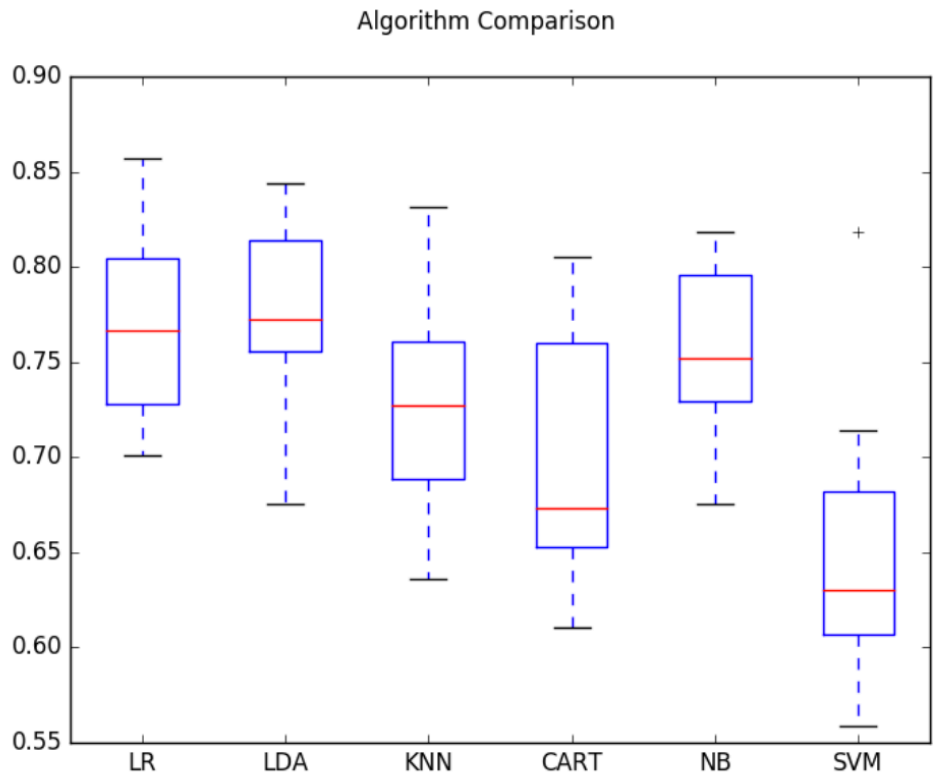
10. Compare Machine Learning Algorithms

It is important to compare the performance of multiple different machine learning algorithms consistently. In this chapter you will discover how you can create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn. You can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare.

```
# Compare Algorithms
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
# load dataset
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# prepare models
models = []
models.append(('LR', LogisticRegression(solver='liblinear')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
  kfold = KFold(n_splits=10, random_state=7, shuffle=True)
  cv_results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

Running the example provides a list of each algorithm short name, the mean accuracy and the standard deviation accuracy.

```
LR: 0.769515 (0.048411)
LDA: 0.773462 (0.051592)
KNN: 0.726555 (0.061821)
CART: 0.695232 (0.062517)
NB: 0.755178 (0.042766)
SVM: 0.651025 (0.072141)
```



From these results, it would suggest that both logistic regression and linear discriminant analysis are perhaps worthy of further study on this problem.

A Roadmap for building ML Systems

	ML Lesson
	Lesson 1: Python Ecosystem for Machine Learning. Lesson 2: Python and SciPy Crash Course. (Numpy, Pandas, Matplotlib)
Define Problem & Analyze Data	Lesson 3: Understand Data With Descriptive Statistics. Lesson 4: Understand Data With Visualization.
Prepare Data	Lesson 5: Pre-Process Data.
Evaluate Algorithms	Lesson 6: Resampling Methods. Lesson 7: Algorithm Evaluation Metrics. Lesson 8: Spot-Check **Classification** Algorithms. Lesson 9: Spot-Check Regression Algorithms. Lesson 10: Model Selection.
Improve Results	Lesson 11: Ensemble Methods. Lesson 12: Algorithm Parameter Tuning.
Present Results	Lesson 13: Model Finalization.

11. Improve Performance with Ensembles

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms. It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data. The predictions of the sub-models can be weighted, but specifying the weights for classifiers manually or even heuristically is difficult. More advanced methods can learn how to best weight the predictions from sub-models, but this is called stacking (stacked aggregation) and is currently not provided in scikit-learn.
You can create a voting ensemble model for classification using the VotingClassifier class.
The code below provides an example of combining the predictions of logistic regression, classification and regression trees and support vector machines together for a classification problem.

11. Improve Performance with Ensembles

```
# Voting Ensemble for Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
# create the sub models
estimators = []
model1 = LogisticRegression(solver='liblinear')
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC(gamma='auto')
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

Running the example provides a mean estimate of classification accuracy.

```
0.729049897471
```

12. Improve Performance with Algorithm Tuning

Machine learning models are parameterized so that their behavior can be tuned for a given problem.
Models can have many parameters and finding the best combination of parameters can be treated as a search problem.

Grid Search Parameter Tuning

Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. You can perform a grid search using the GridSearchCV class

The example below evaluates different alpha values for the Ridge Regression algorithm on the standard diabetes dataset. This is a one-dimensional grid search.

```
# Grid Search for Algorithm Tuning
import numpy
from pandas import read_csv
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import GridSearchCV
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
alphas = numpy.array([1,0.1,0.01,0.001,0.0001,0])
param_grid = dict(alpha=alphas)
model = RidgeClassifier()
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3)
grid.fit(X, Y)
print(grid.best_score_)
print(grid.best_estimator_.alpha)
```

Running the example lists out the optimal score achieved and the set of parameters in the grid that achieved that score. In this case the `alpha` value of 1.0.

```
0.7708333333333334
1.0
```

A Roadmap for building ML Systems

	ML Lesson
	Lesson 1: Python Ecosystem for Machine Learning. Lesson 2: Python and SciPy Crash Course. (Numpy, Pandas, Matplotlib)
Define Problem & Analyze Data	Lesson 3: Understand Data With Descriptive Statistics. Lesson 4: Understand Data With Visualization.
Prepare Data	Lesson 5: Pre-Process Data.
Evaluate Algorithms	Lesson 6: Resampling Methods. Lesson 7: Algorithm Evaluation Metrics. Lesson 8: Spot-Check **Classification** Algorithms. Lesson 9: Spot-Check Regression Algorithms. Lesson 10: Model Selection.
Improve Results	Lesson 11: Ensemble Methods. Lesson 12: Algorithm Parameter Tuning.
Present Results	Lesson 13: Model Finalization.

13. Model Finalization

Save and Load Machine Learning Models: How to use pickle to serialize and deserialize machine learning models.

```
# Save Model Using Pickle
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from pickle import dump
from pickle import load
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=7)
# Fit the model on 33%
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
# save the model to disk
filename = 'finalized_model.sav'
dump(model, open(filename, 'wb'))

# some time later...

# load the model from disk
loaded_model = load(open(filename, 'rb'))
result = loaded_model.score(X_test, Y_test)
print(result)
```

Running the example saves the model to `finalized_model.sav` in your local working directory. Load the saved model and evaluating it provides an estimate of accuracy of the model on unseen data.

```
0.755905511811
```

More Details on Classification Methods

1. Logistic Regression Classification

Linear Regression:

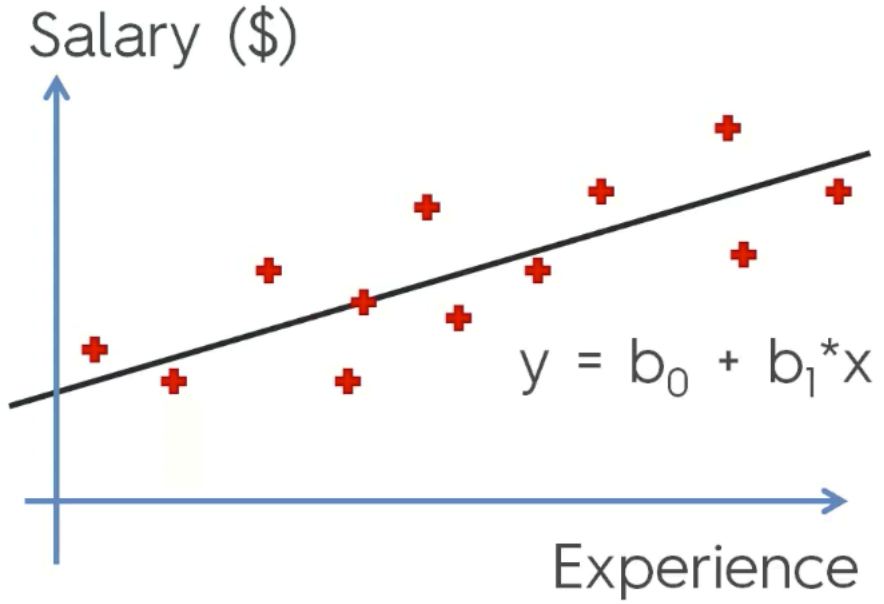
- **Simple:**

$$y = b_0 + b_1 * x$$

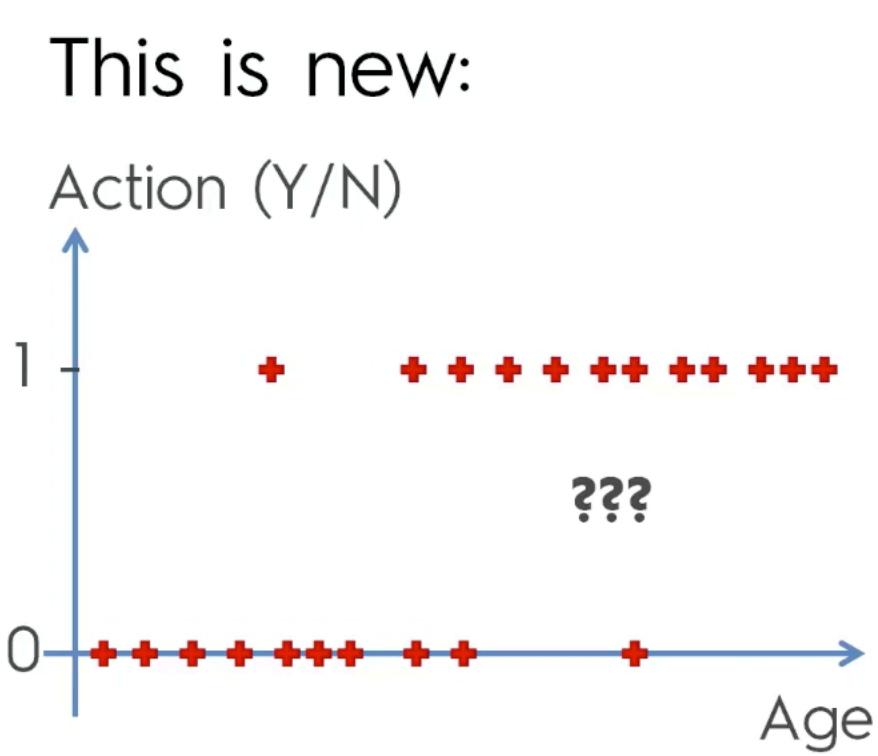
- **Multiple:**

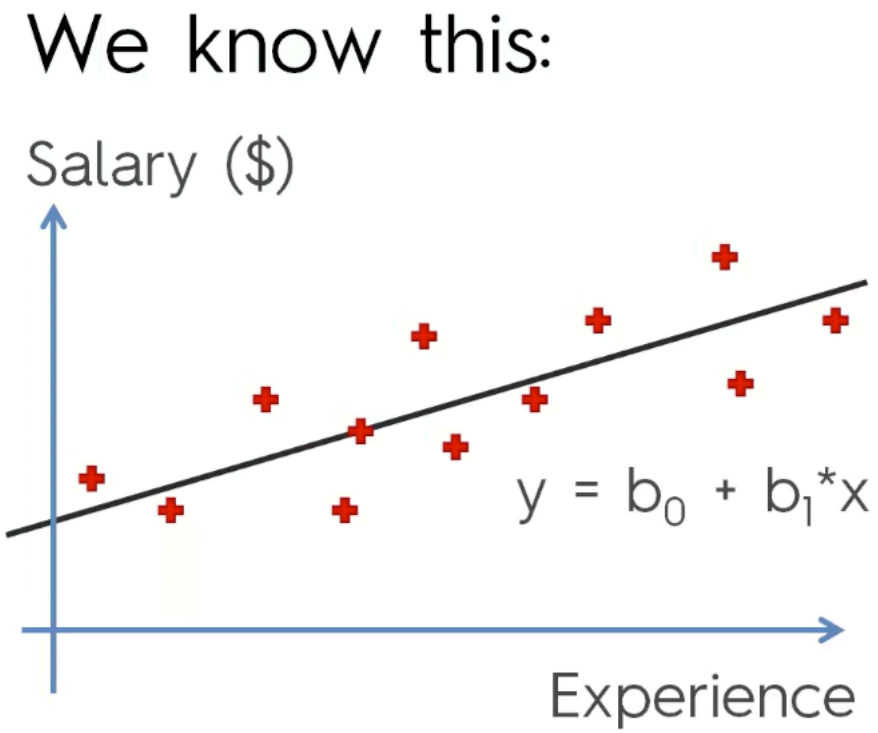
$$y = b_0 + b_1 * x_1 + \ldots + b_n * x_n$$

We know this:

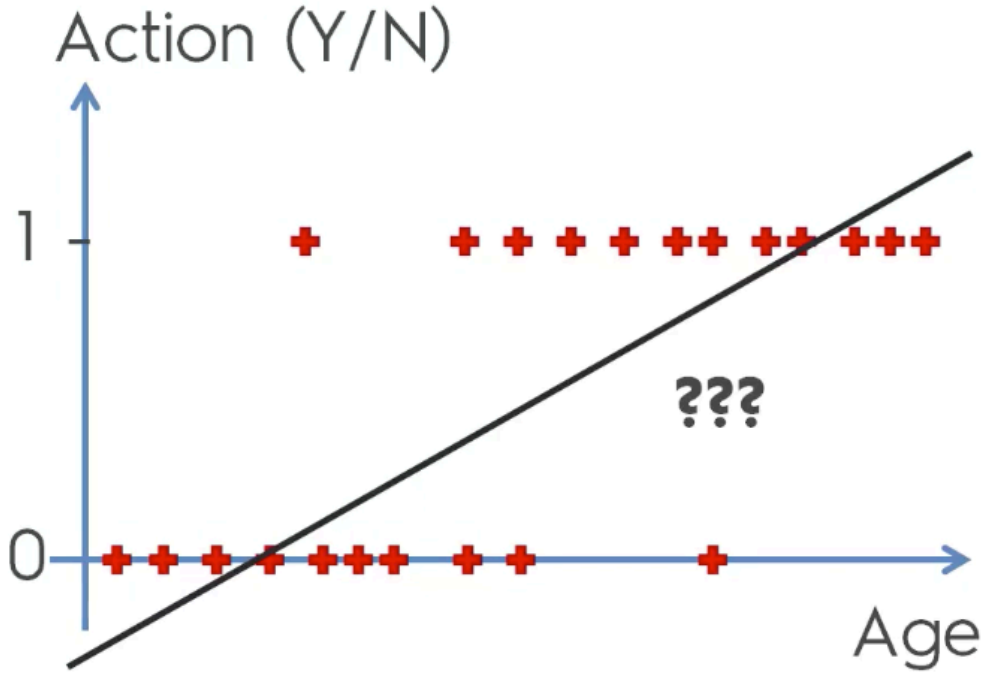


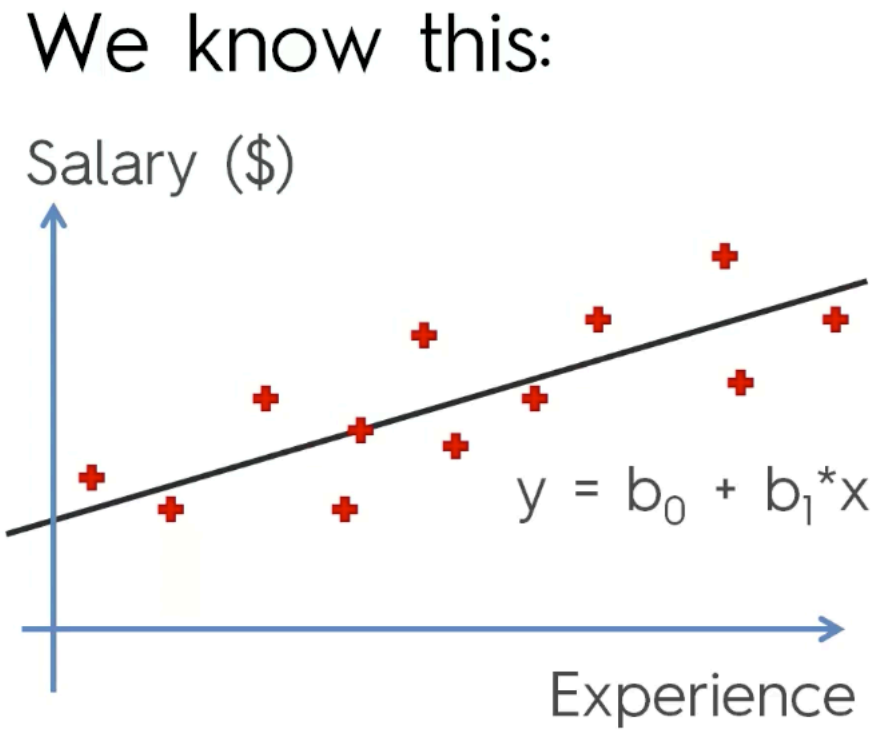
1. Logistic Regression Classification



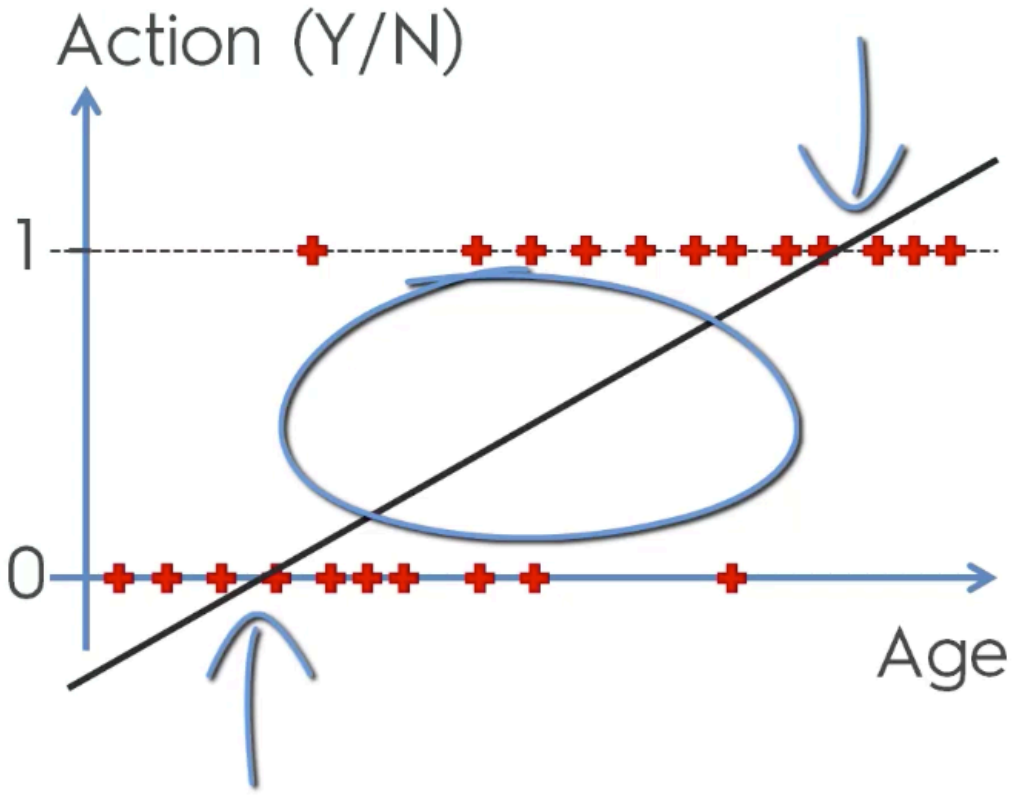


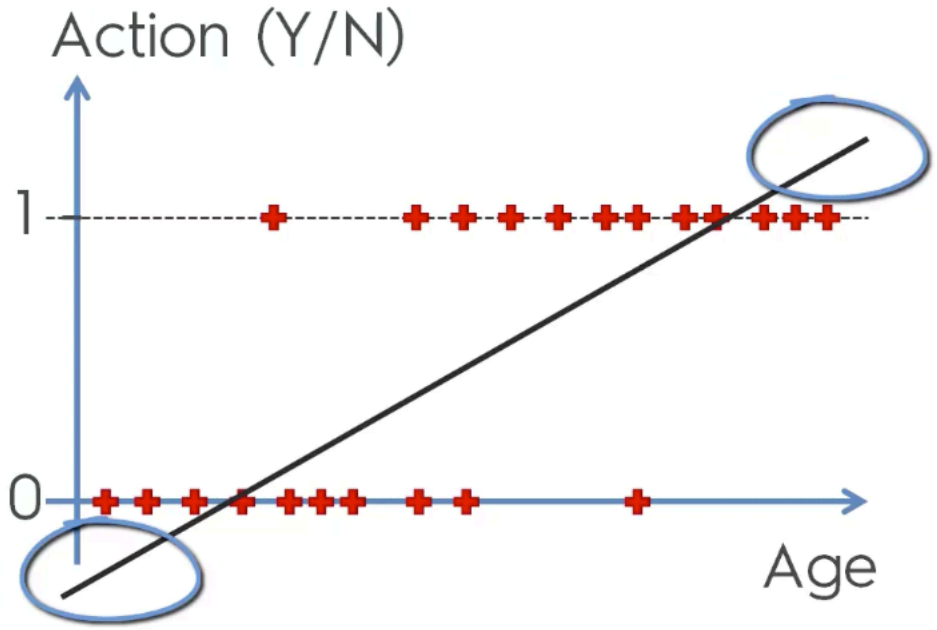
1. Logistic Regression Classification



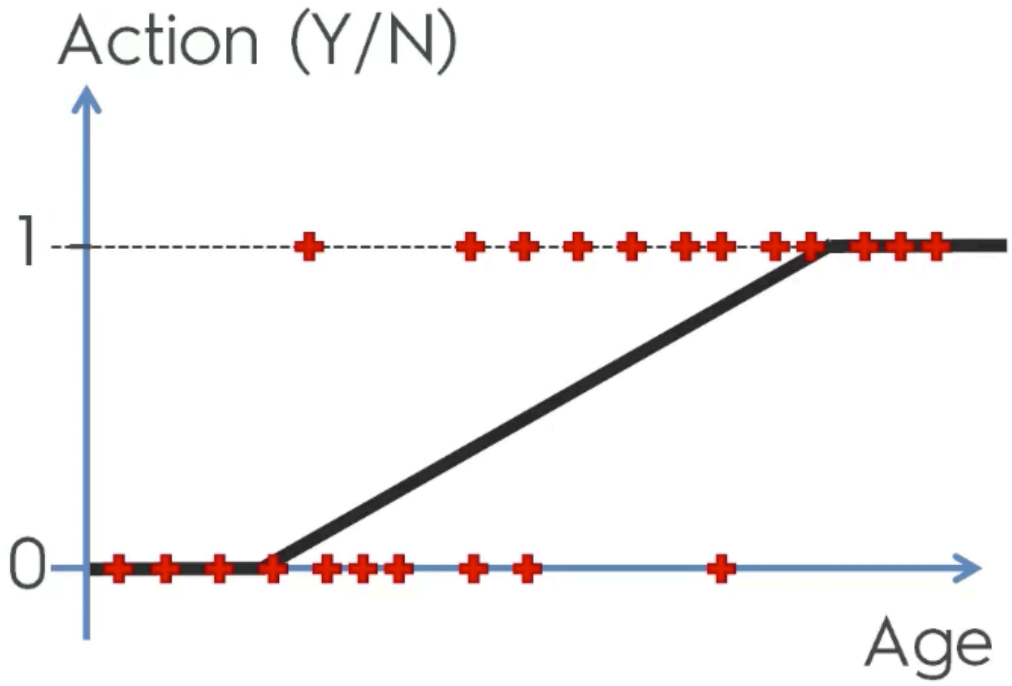


1. Logistic Regression Classification





1. Logistic Regression Classification



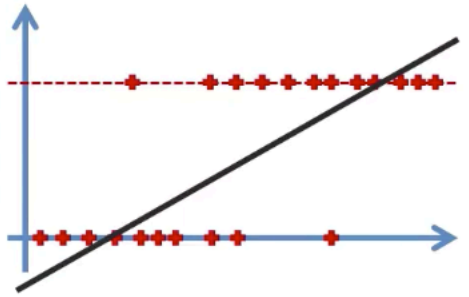
1. Logistic Regression Classification

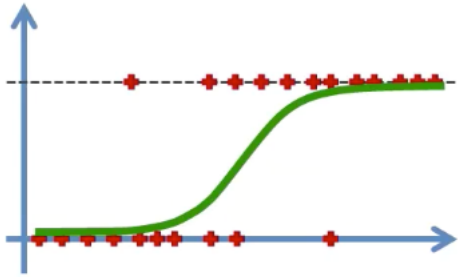
$$y = b_0 + b_1 * x$$

Sigmoid Function

$$p = \frac{1}{1 + e^{-y}}$$

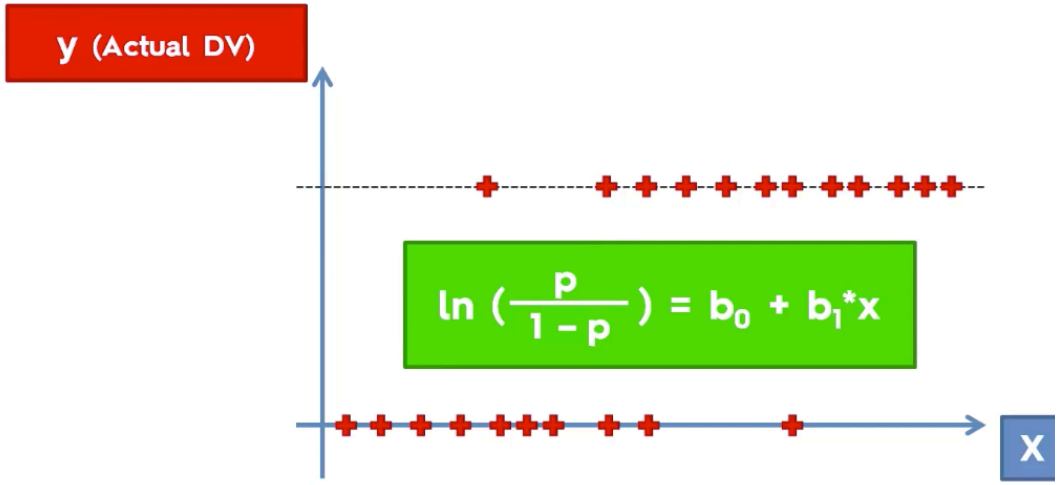
$$\ln\left(\frac{p}{1-p}\right) = b_0 + b_1 * x$$

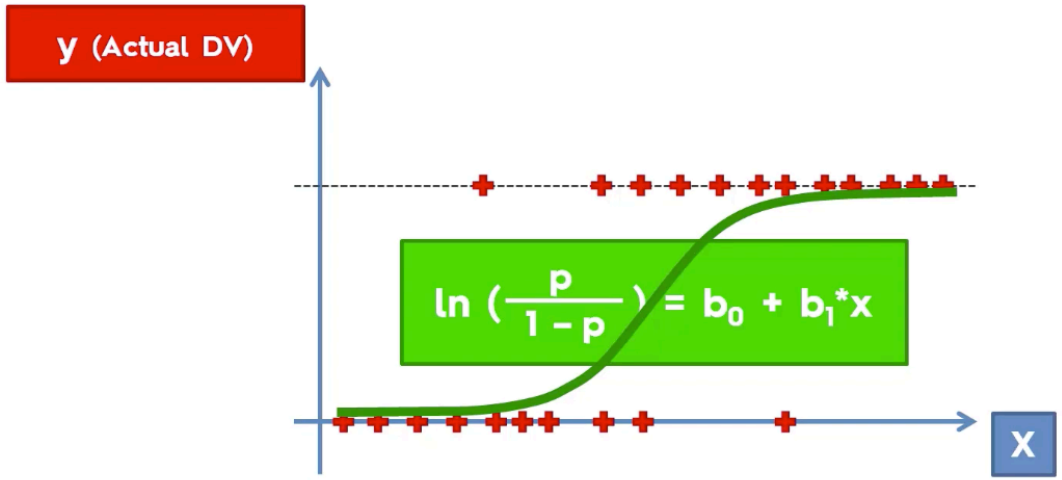




1. Logistic Regression Classification

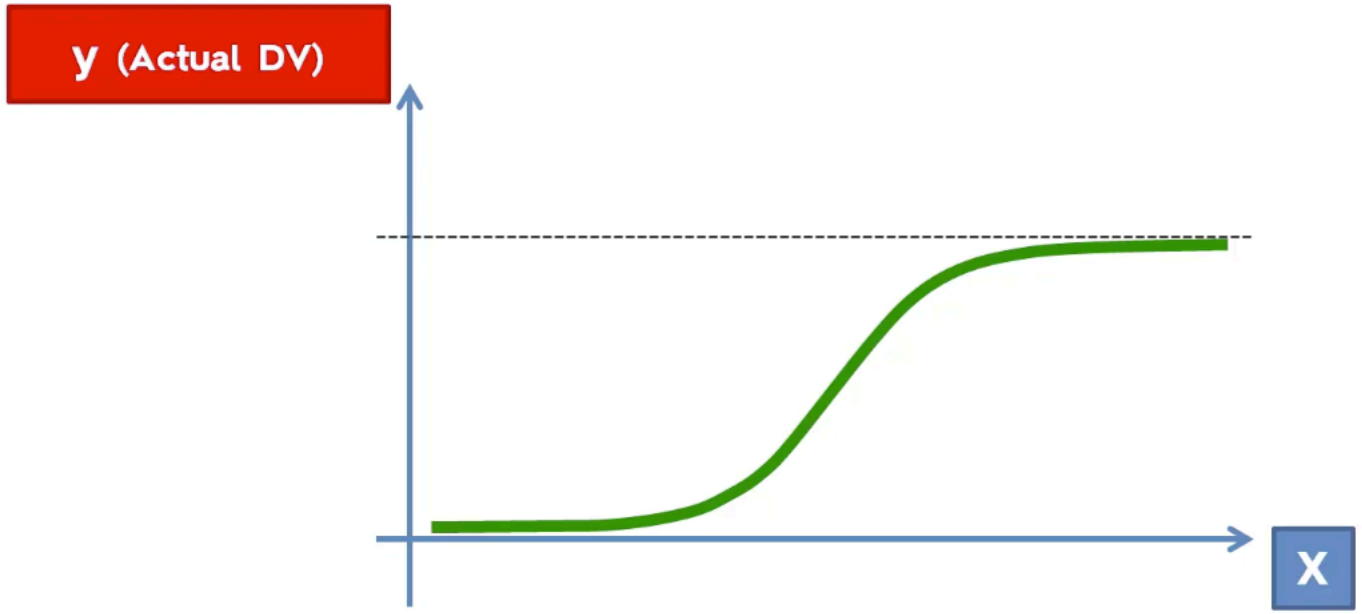
DV = Dependent Variable



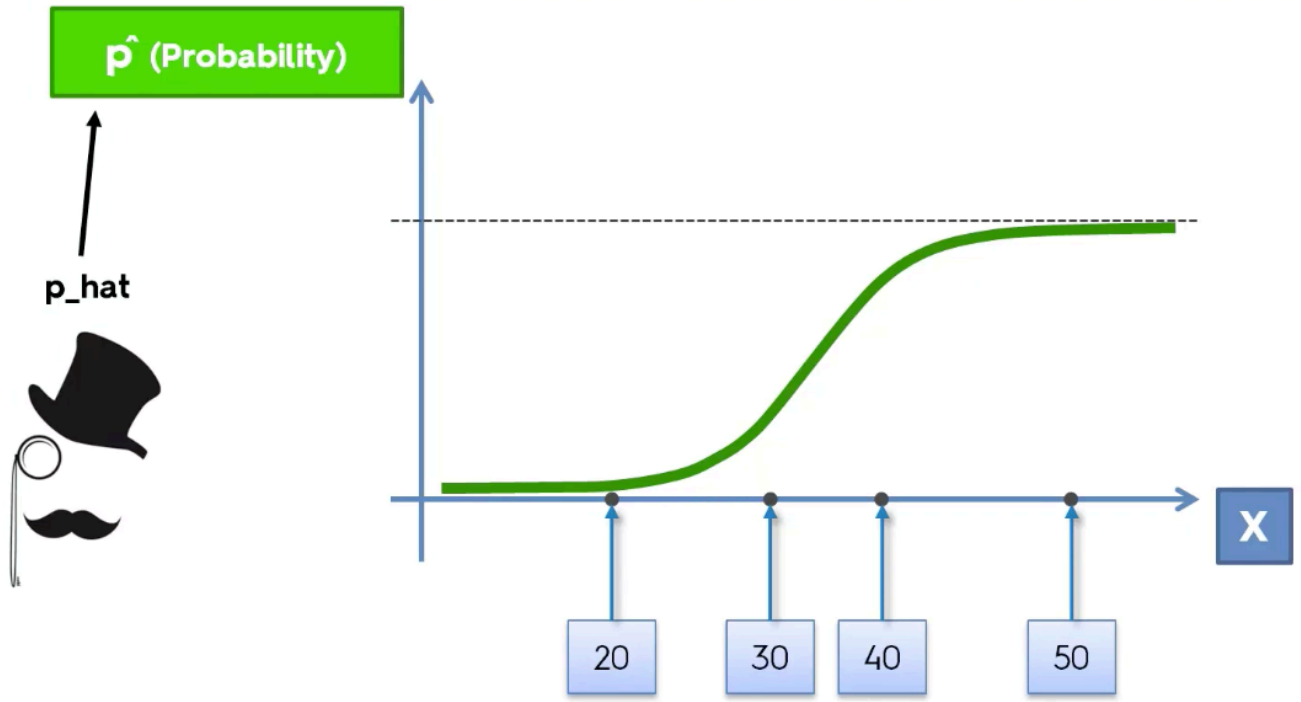


1. Logistic Regression Classification

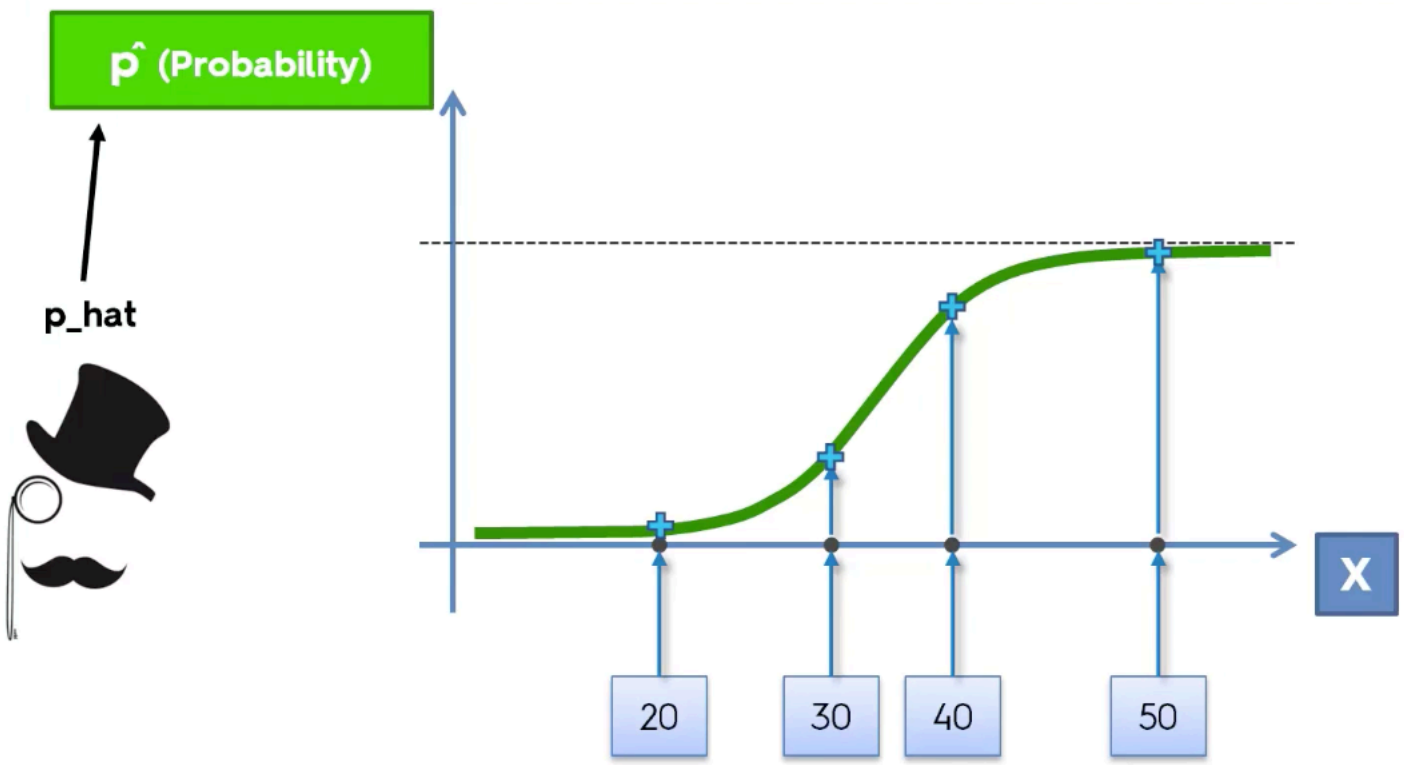
Get the pre-trained model After Training



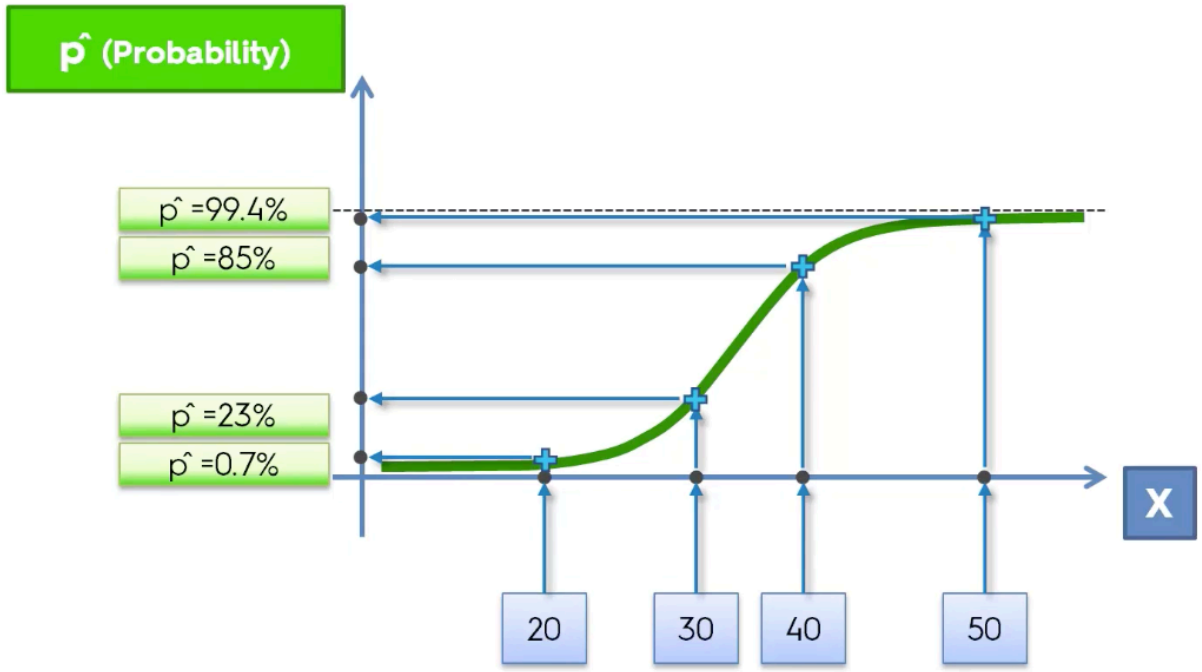
1. Logistic Regression Classification

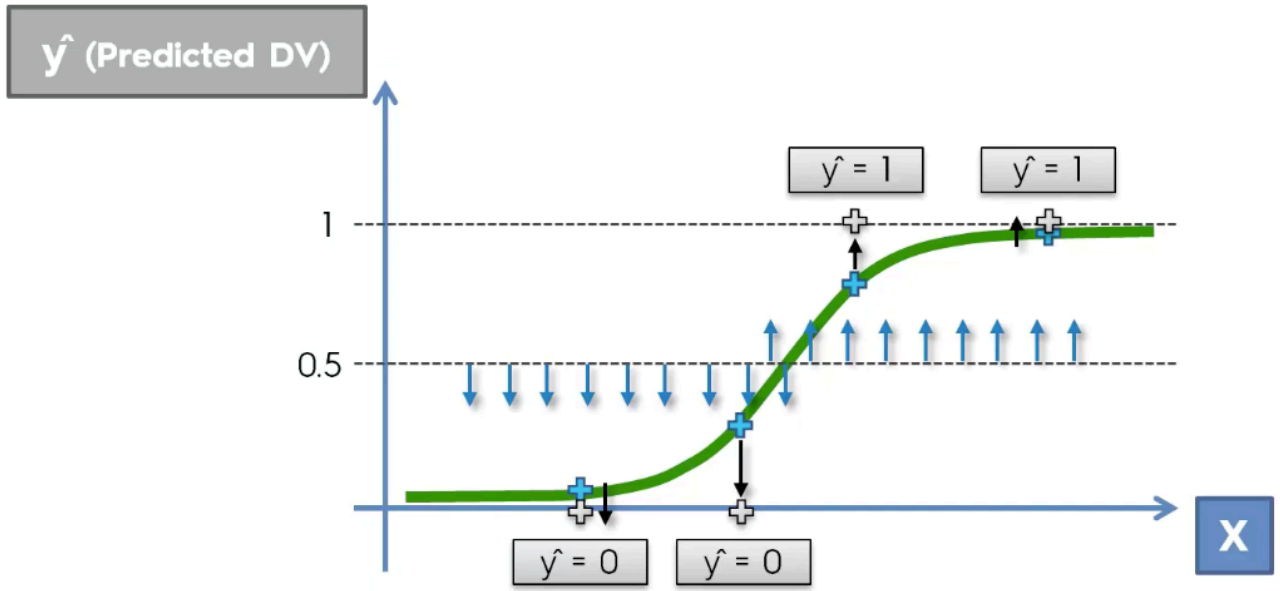


1. Logistic Regression Classification

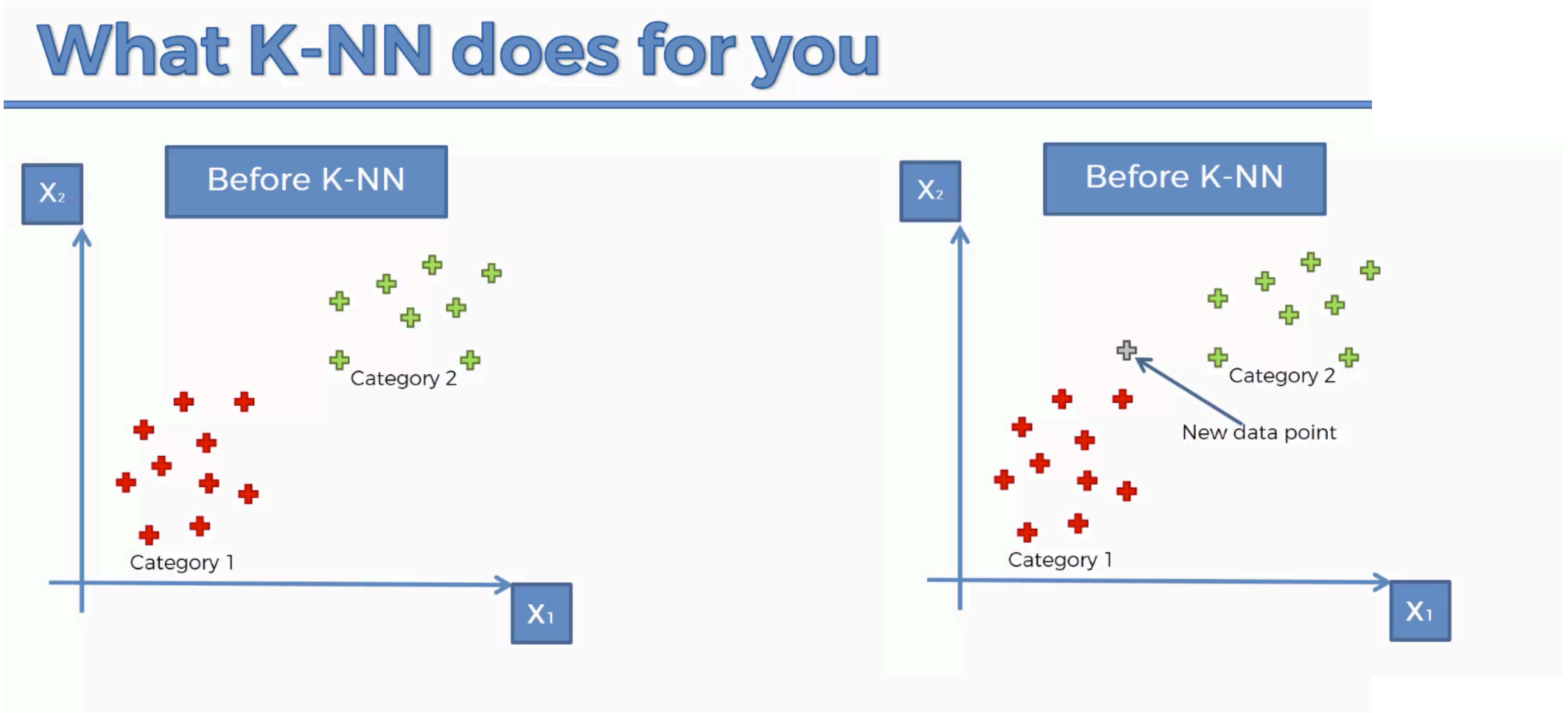


1. Logistic Regression Classification

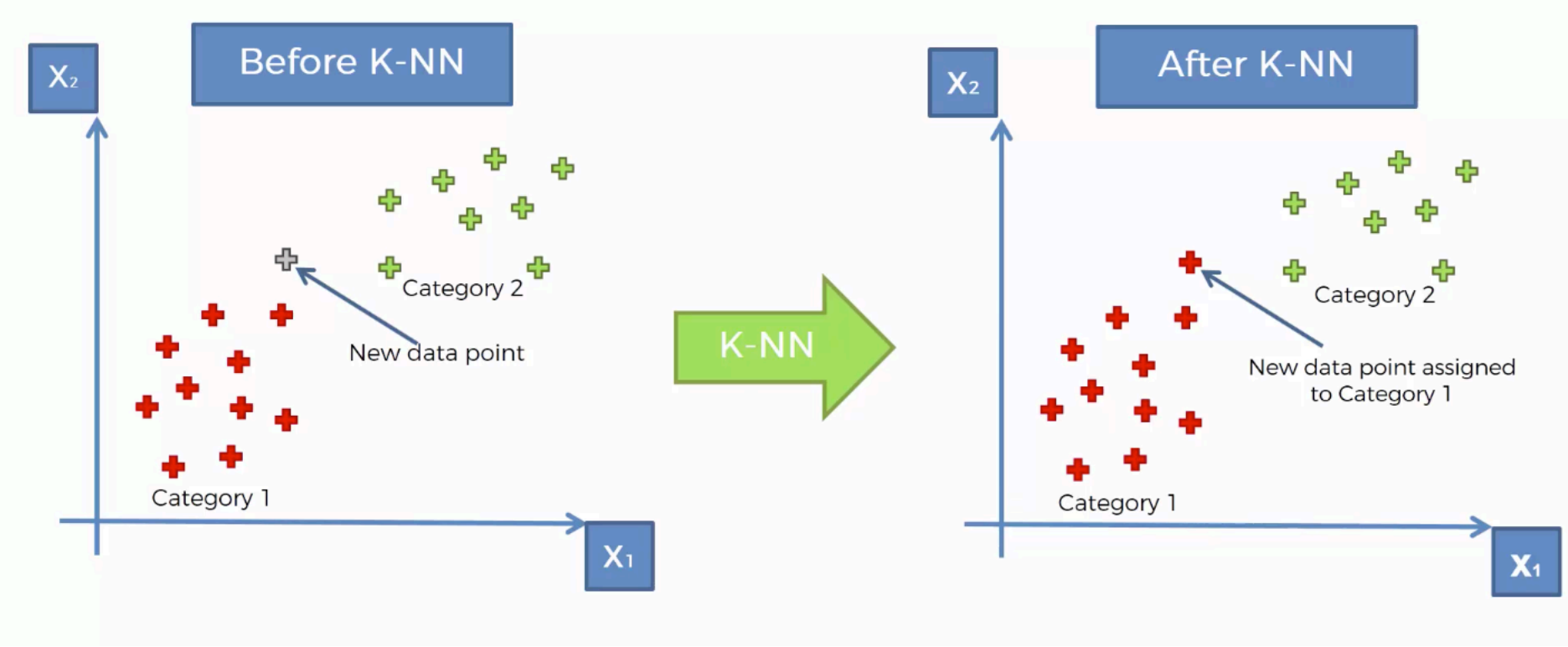




2. K-Nearest Neighbors (K-NN)



2. K-Nearest Neighbors (K-NN)



2. K-Nearest Neighbors (K-NN)

How did it do that ?

STEP 1: Choose the number K of neighbors

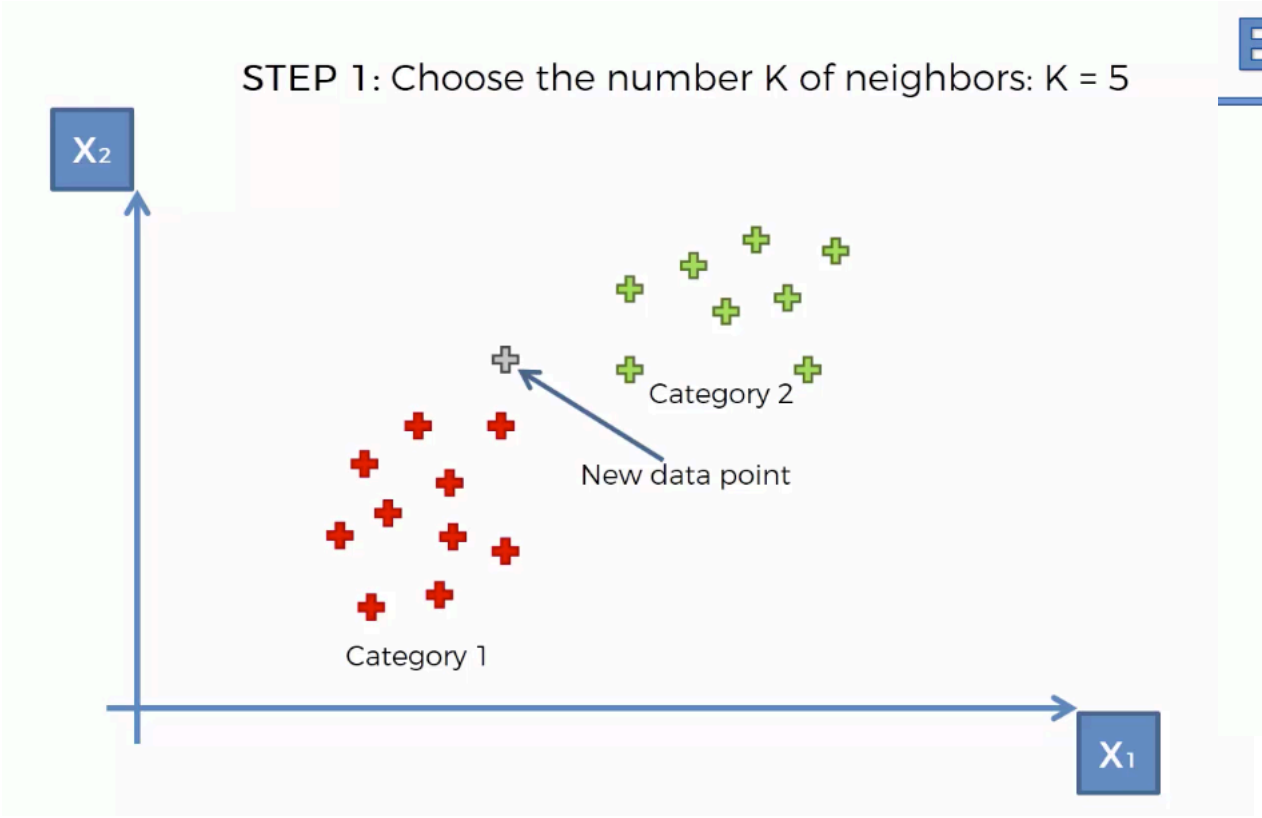
STEP 2: Take the K nearest neighbors of the new data point, according to the Euclidean distance

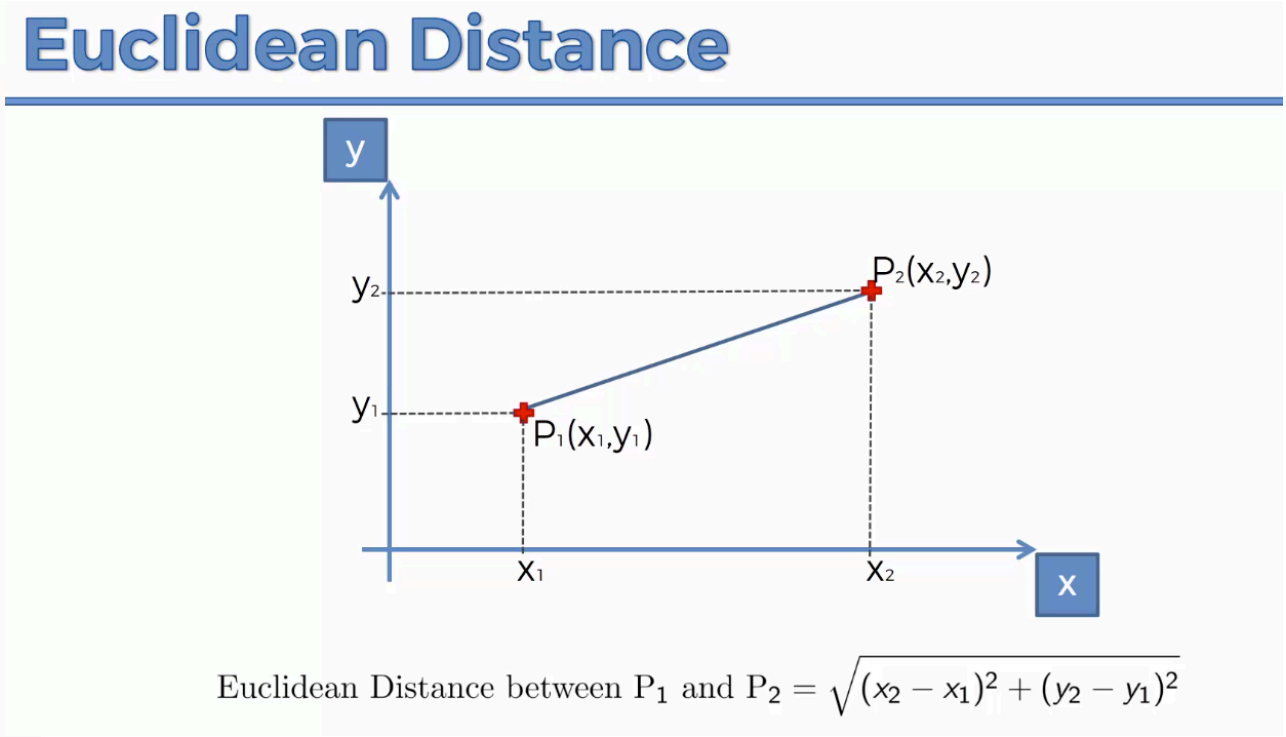
STEP 3: Among these K neighbors, count the number of data points in each category

STEP 4: Assign the new data point to the category where you counted the most neighbors

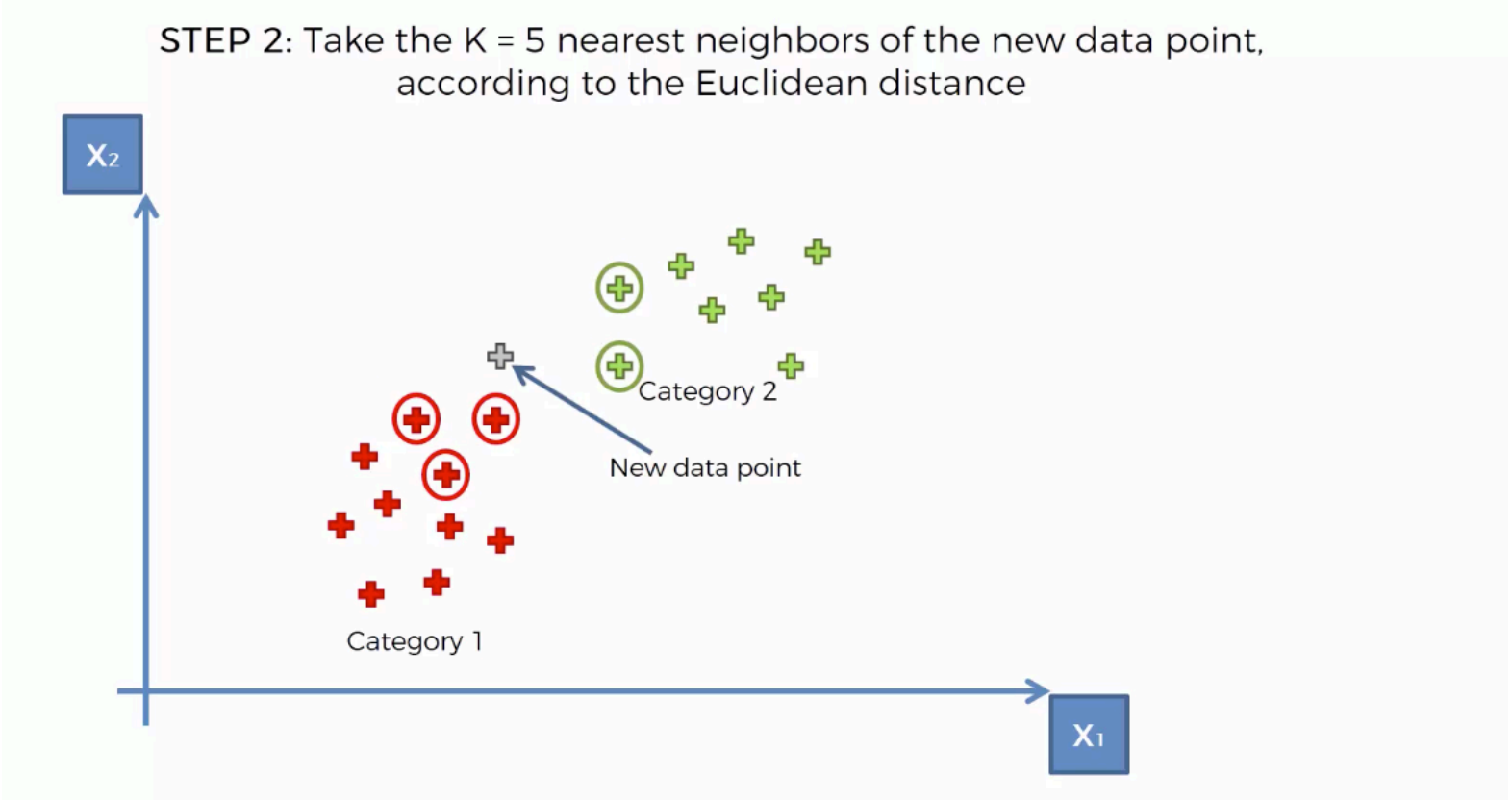
Your Model is Ready

2. K-Nearest Neighbors (K-NN)

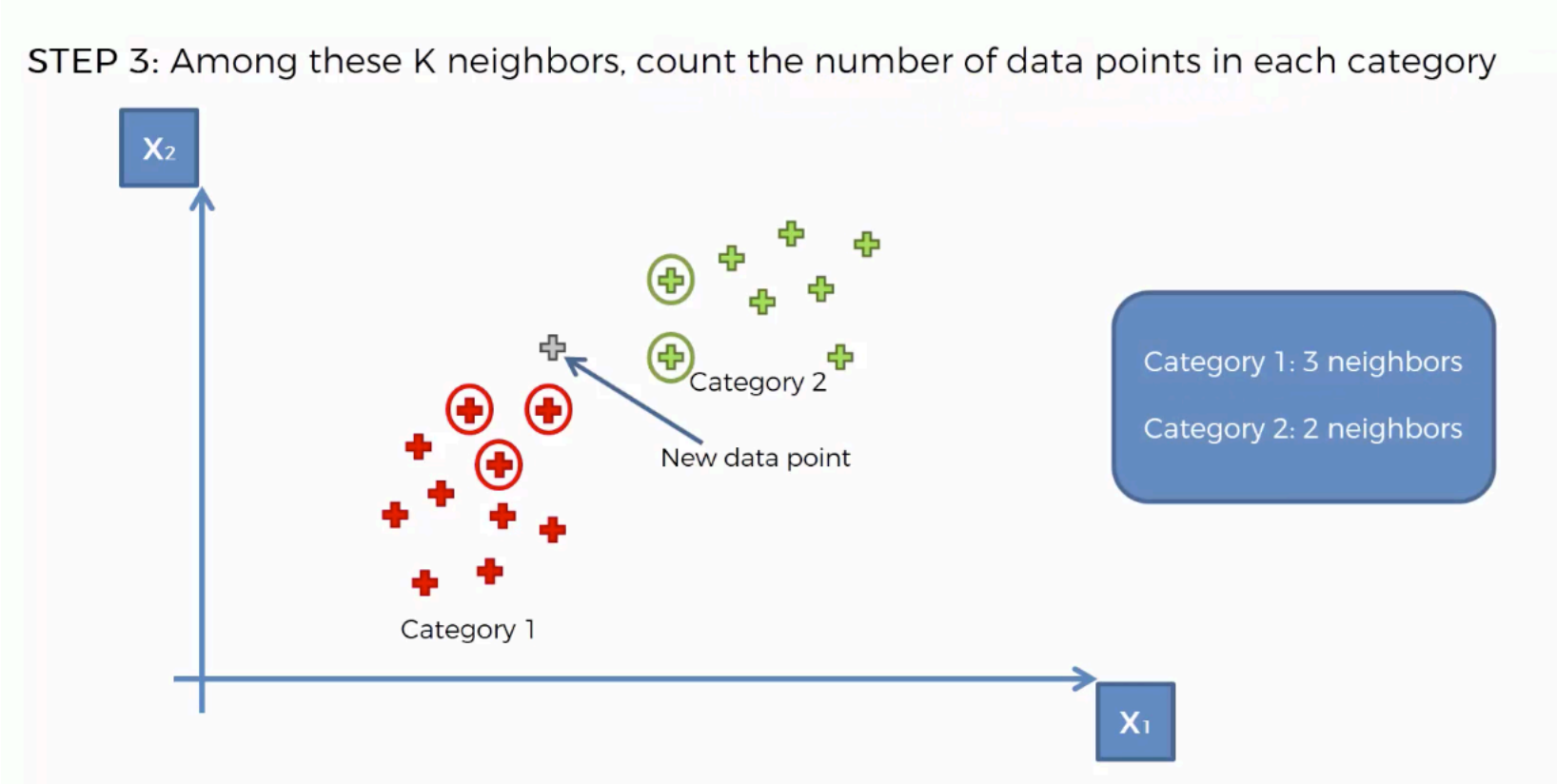




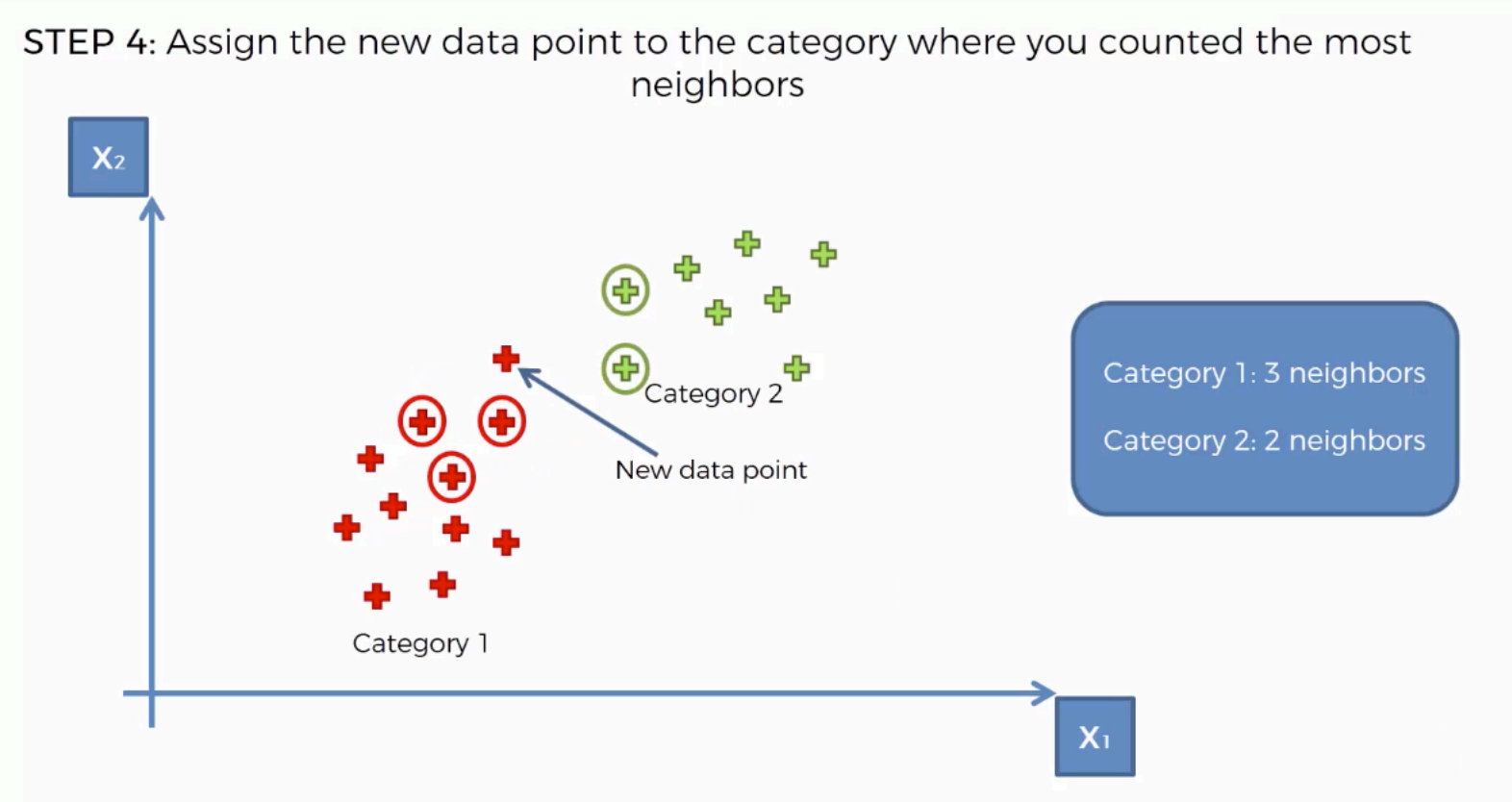
2. K-Nearest Neighbors (K-NN)



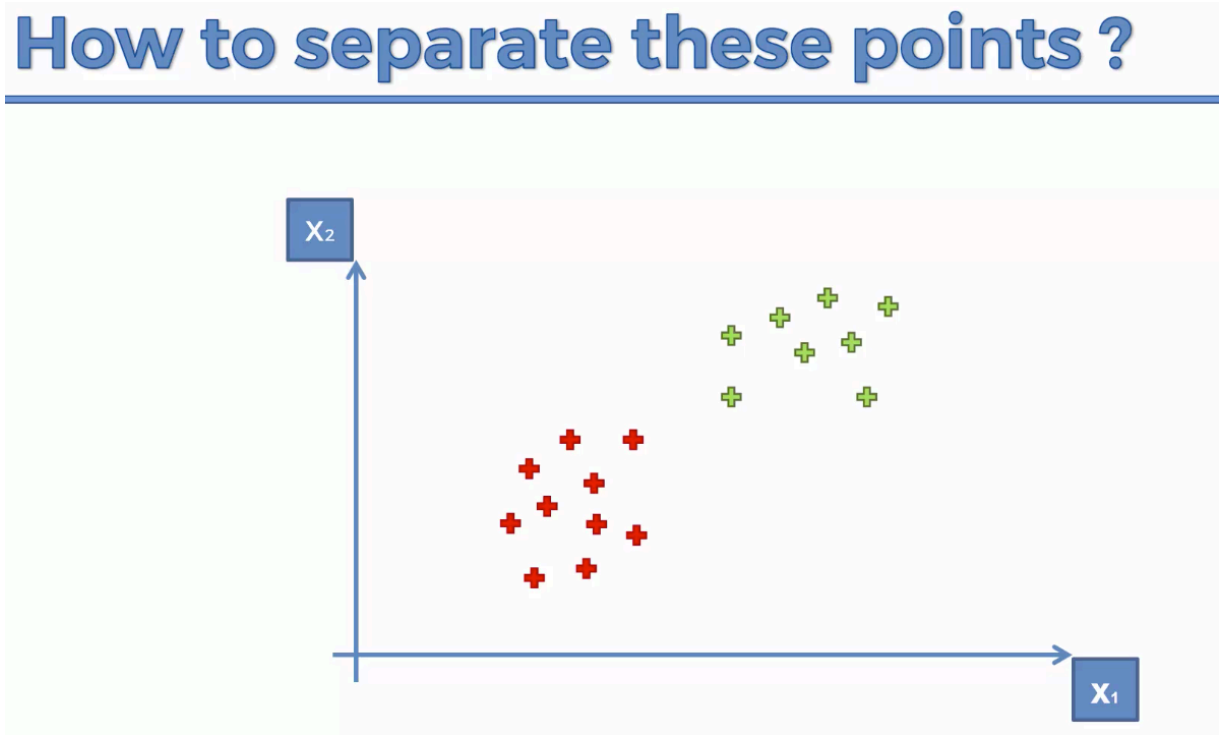
2. K-Nearest Neighbors (K-NN)

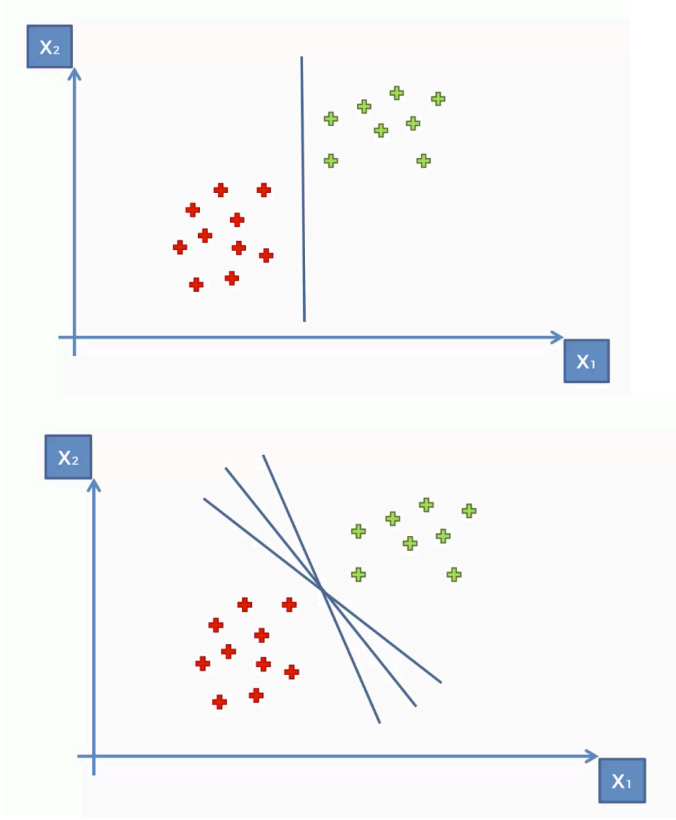


2. K-Nearest Neighbors (K-NN)



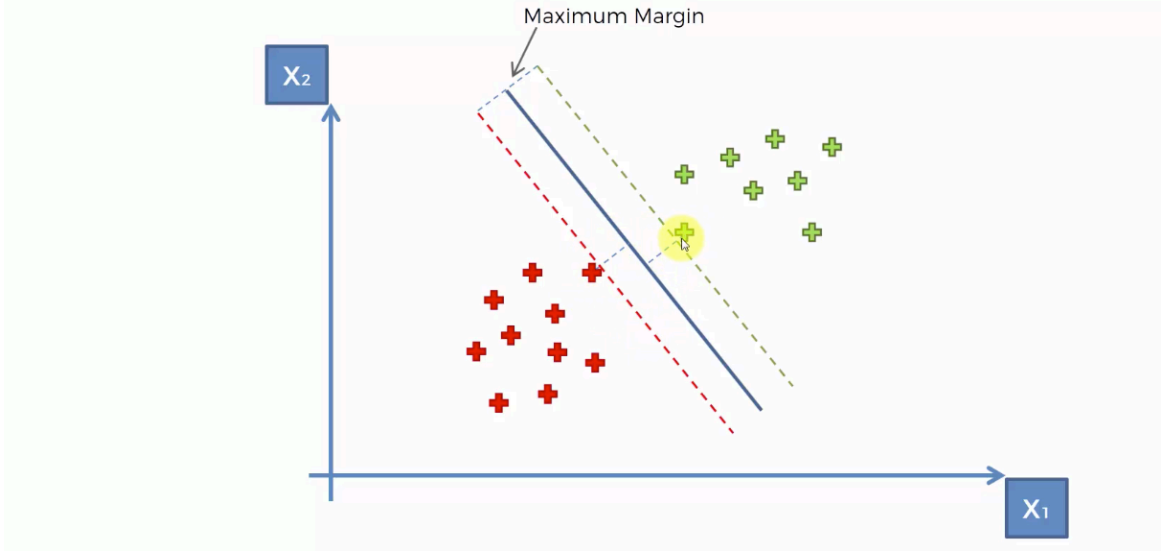
3. Support Vector Machine (SVM)



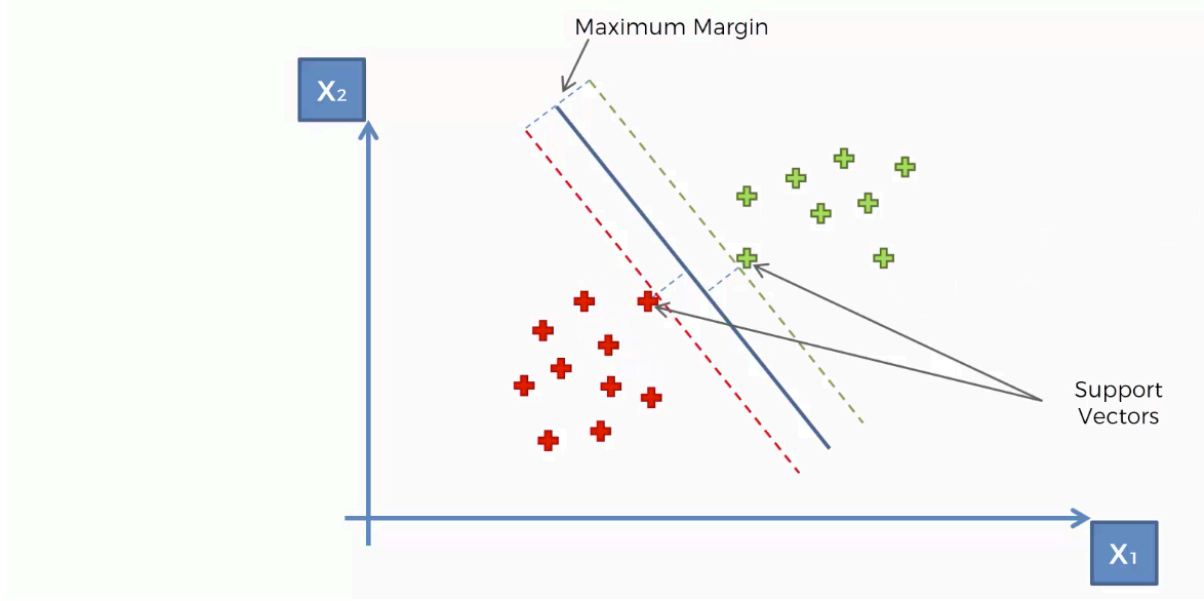


3. Support Vector Machine (SVM)

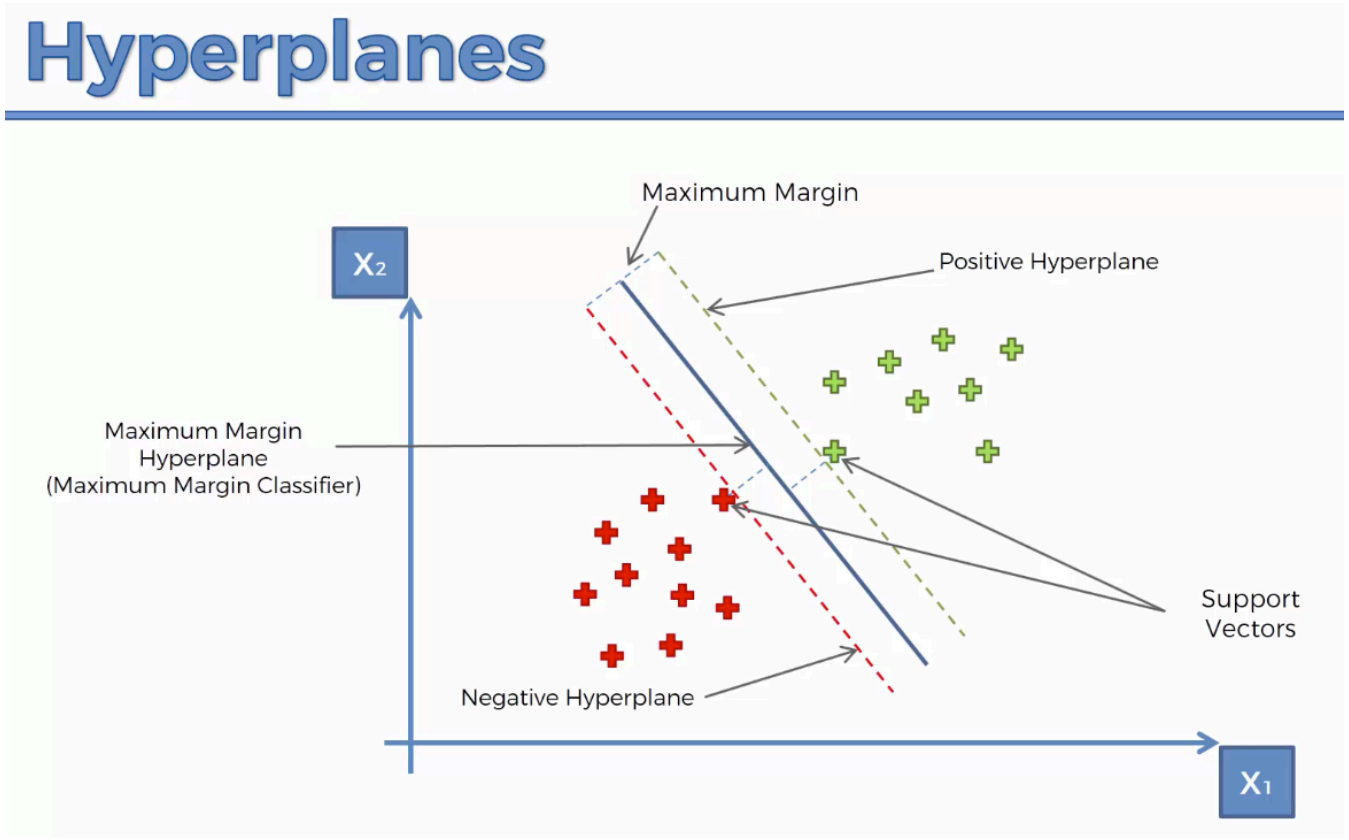
Maximum Margin



Support Vectors

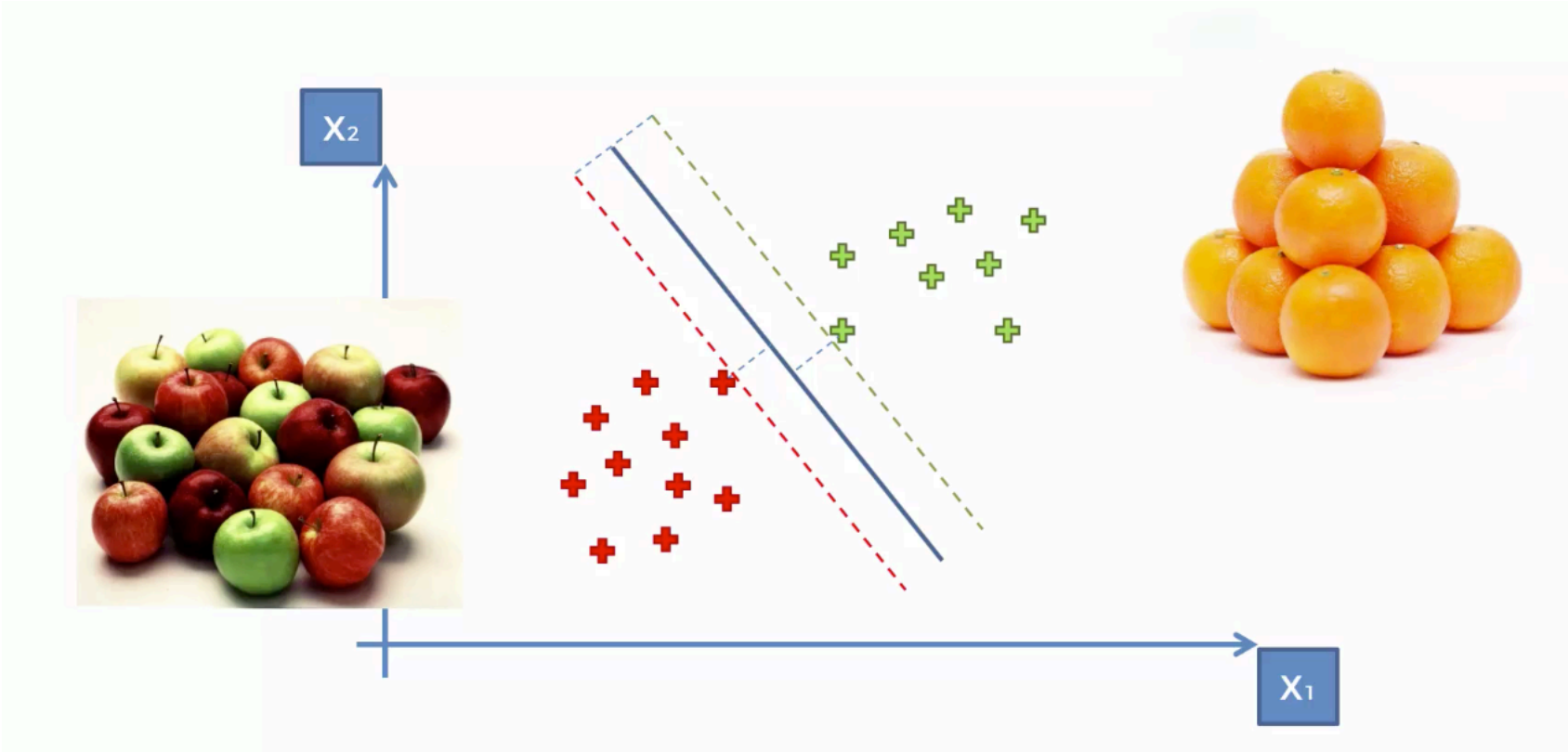


3. Support Vector Machine (SVM)

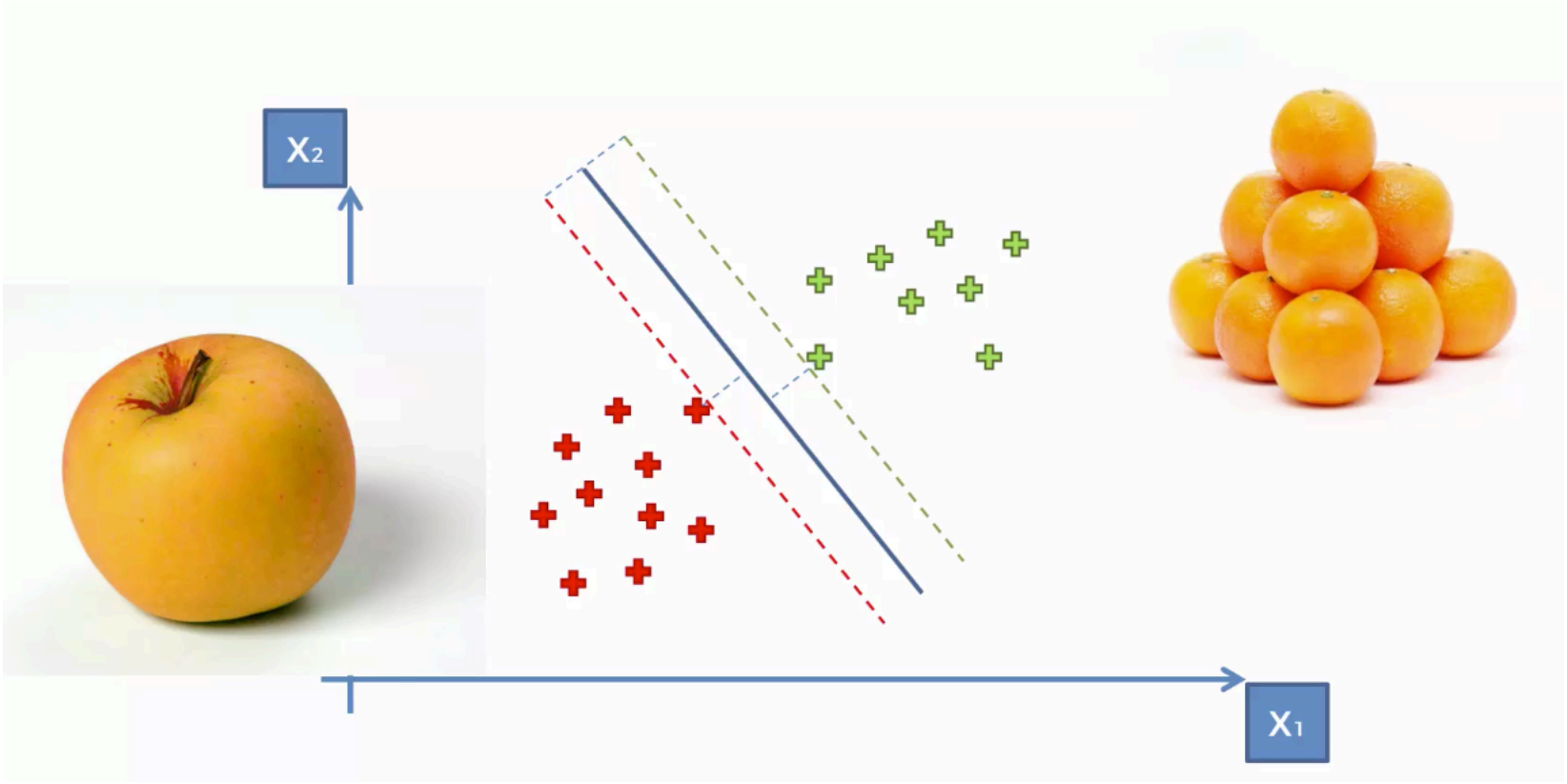


3. Support Vector Machine (SVM)

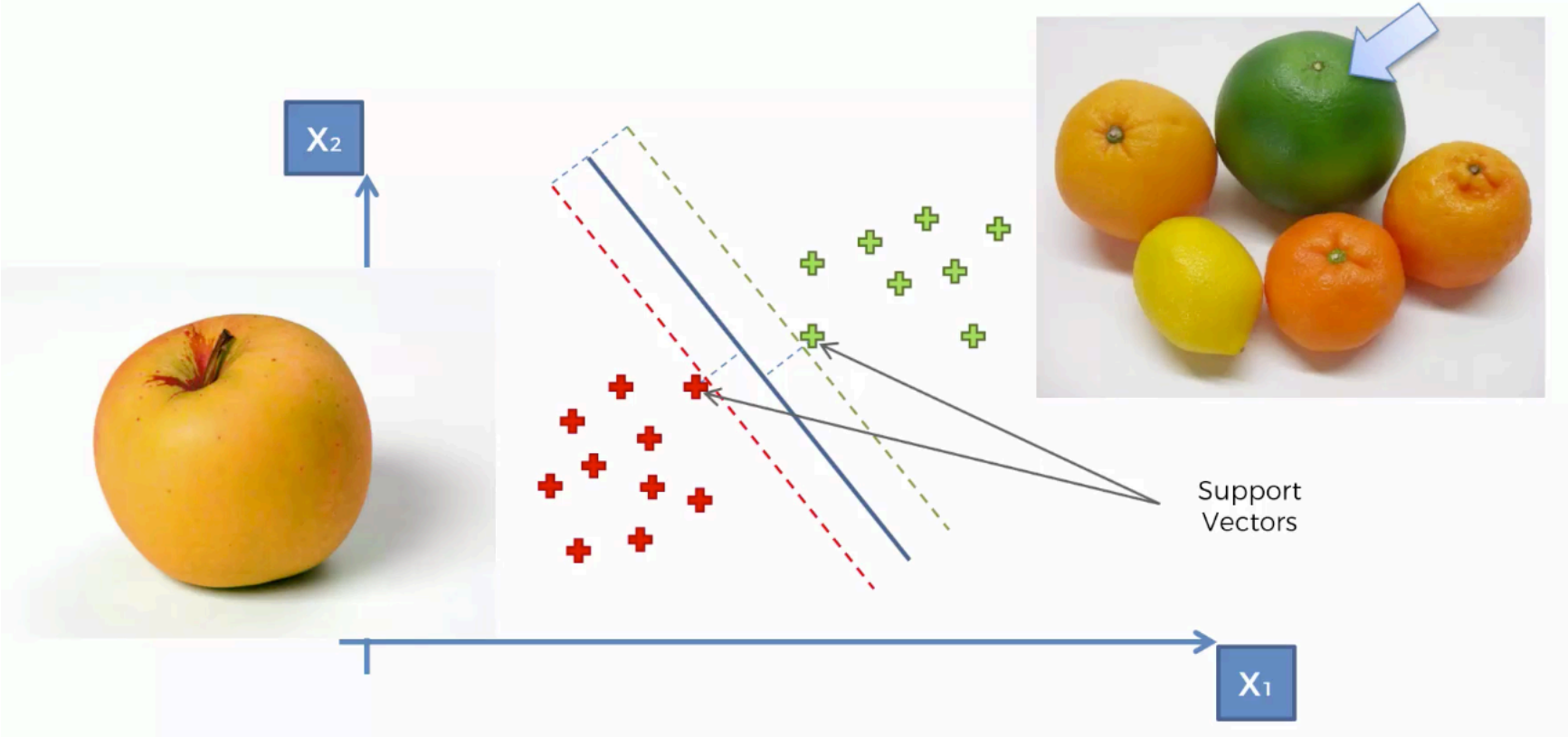
3. Support Vector Machine (SVM)



3. Support Vector Machine (SVM)



3. Support Vector Machine (SVM)



Questions:

1. **What method should I use for prediction? How is the accuracy? Precision? Recall?**

https://www.kaggle.com/c/titanic/overview

Classification Example 3 (Kaggle Competition)

Data Descriptions: Here is what each feature represents. Understanding the data is must before it's manipulation and analysis.

1. PassengerID
2. Survival: 0 = No, 1 = Yes
3. pclass (Ticket class): 1 = 1st, 2 = 2nd, 3 = 3rd
4. Name
5. sex: Sex
6. Age: Age in years
7. sibsp: number of siblings/spouses aboard the Titanic
8. parch: number of parents/children aboard the Titanic
9. ticket: Ticket number
10. fare: Passenger fare
11. cabin: Cabin number
12. embarked: Port of Embarkation, C = Cherbourg, Q = Queenstown, S = Southampton

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

https://www.kaggle.com/c/titanic/overview

Classification Example 3 (Kaggle Competition)