TUTORIAL FOR MACHINE LEARNING ALGORITHMS

Assoc Prof Dr Ir Mahmud Iwan Solihin
Mechatronics Engineering, UCSI University, Malaysia

Workshop: January 2024
(Robotic Lab, Block C-Level 4)

OUTLINE

- Objectives
- Machine Learning Classification and Regression
    - Gradient Descent for Machine Learning

        1. Linear Regression
        2. Logistic Regression
        3. Decision Tree
        4. Naïve Bayes
        5. k-Nearest Neighbors
        6. Support Vector Machines (SVM)
        7. Artificial Neural Networks (ANN)
        8. Ensemble Algorithms
        9. Principal Component Analysis (PCA)
        10. Model Assessment

Objectives

- To provide fundamental overview of machine learning (ML) for further self exploration and application on the respective field.
- To explain **how** the top machine learning algorithms work and **not why** the algorithms work.
- To practice machine learning implementation using free Orange data mining software.

Introduction

Some of the most important supervised learning algorithms:

- k-Nearest Neighbors (kNN)
- Linear and Logistic Regressions
- Support Vector Machines (SVM)
- Decision Trees and Random Forests
- Naïve-Bayes
- Neural Networks* (NN)

*some NN architectures can be unsupervised

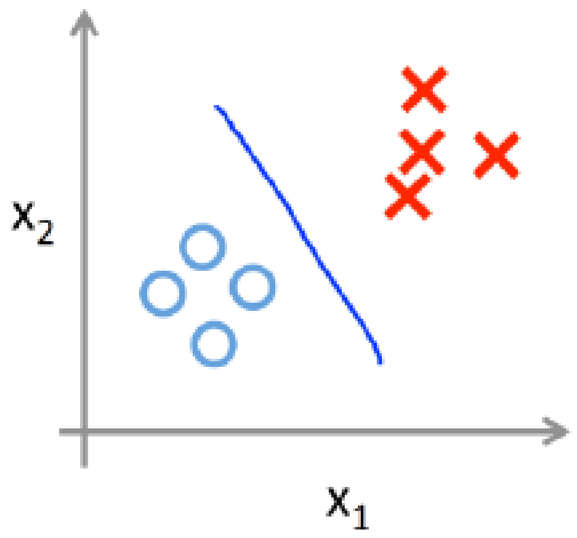
Some of the most important unsupervised learning algorithms:

- k-Means clustering
- Principal Component Analysis (PCA)
- Hierarchical Cluster Analysis (HCA)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

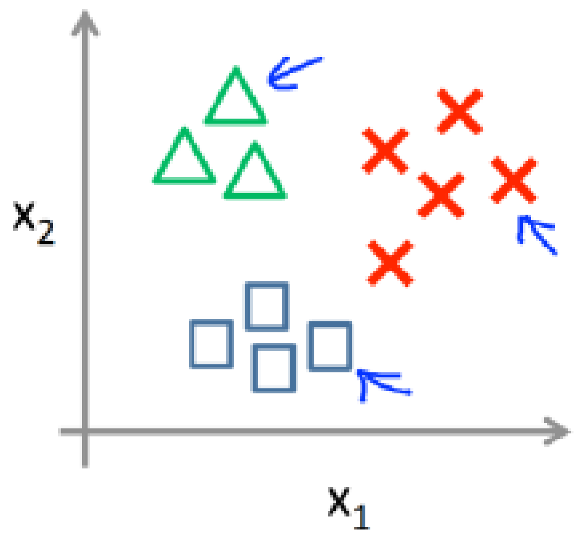
Intro to Classification

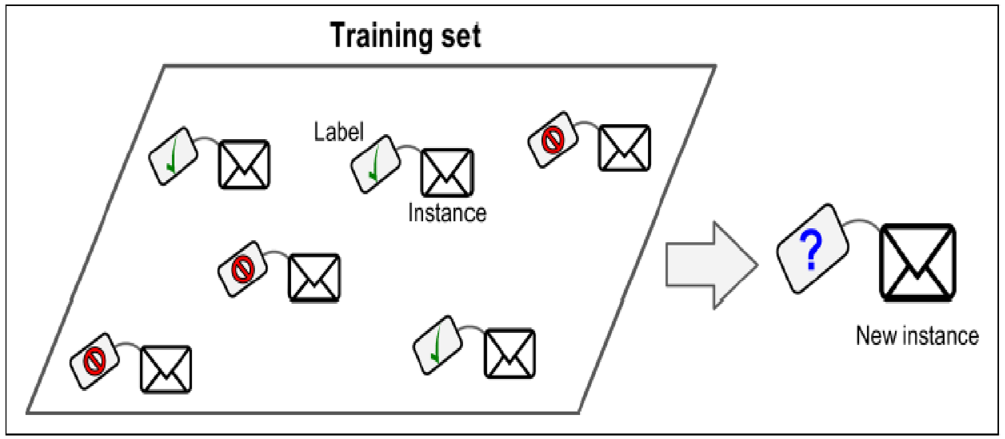
- Classification is a subcategory of supervised learning where the goal is to predict the categorical class labels of new instances, based on past observations.

Binary classification:



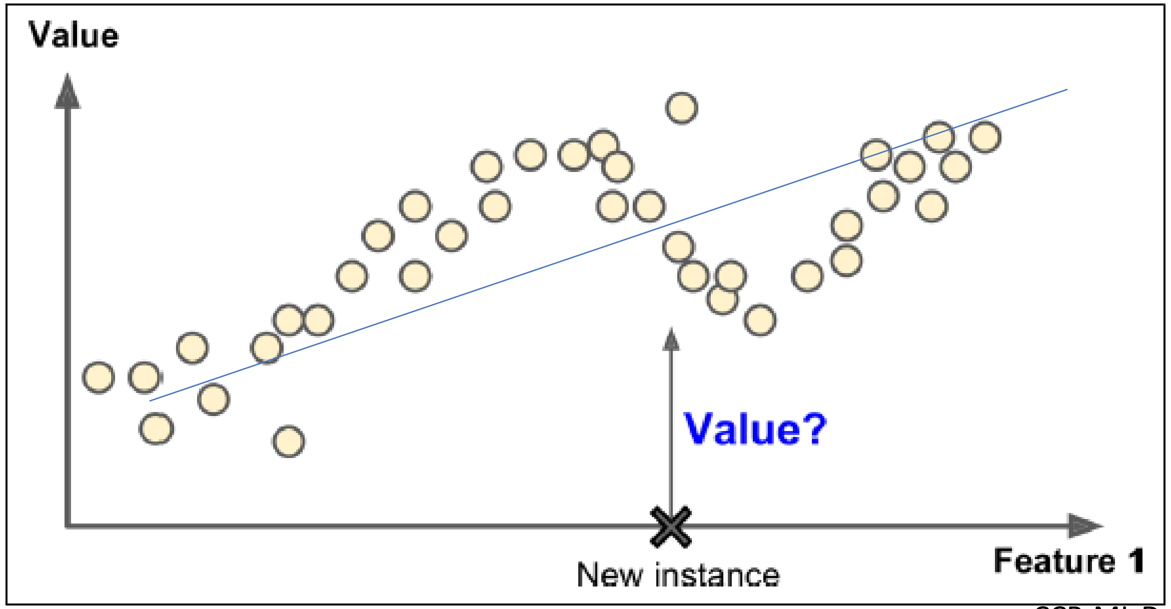
Multi-class classification:





Intro to Regression

- Regression is a type of supervised learning that involves the prediction of continuous outcomes.
- In regression, a number of predictor variables and a continuous response variable (outcome or target) are given, and the machine learning algorithm will try to find a relationship between those variables that allows it to predict an outcome



Gradient Descent for ML

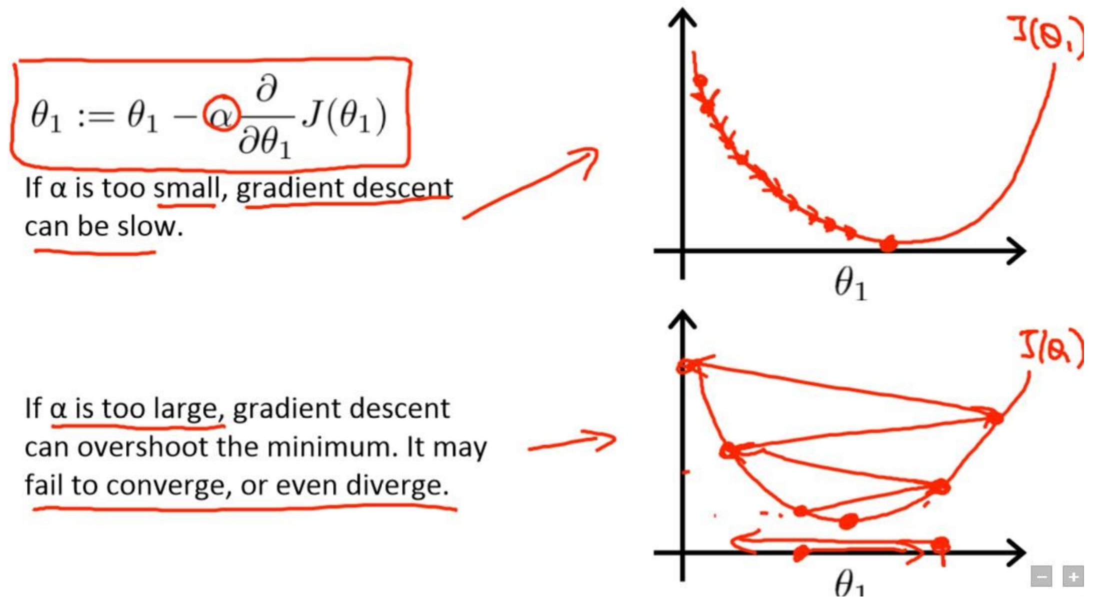
Gradient descent is an optimization algorithm used to nd the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost). Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra).

$$cost = f\,(coefficient)$$
$$cost = evaluate(f\,(coefficient))$$

$$delta = derivative(cost)$$

$$coefficient = coefficient - (alpha \times delta)$$

$$\alpha: learning\ rate$$



$$\boxed{\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial\theta_1} J(\theta_1)}$$

If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

The optimization process is repeated iteratively until the cost function/error is close to zero or no further improvements in cost can be achieved.

1. Linear Regression

Isn't linear regression a technique from statistics?

Yes, In applied machine learning we sometimes borrow, reuse and steal algorithms from many different fields, including statistics.

Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y).

$$y = B0 + B1 \times x$$

$B0: intercept$
$B1: slope$

1. Linear Regression

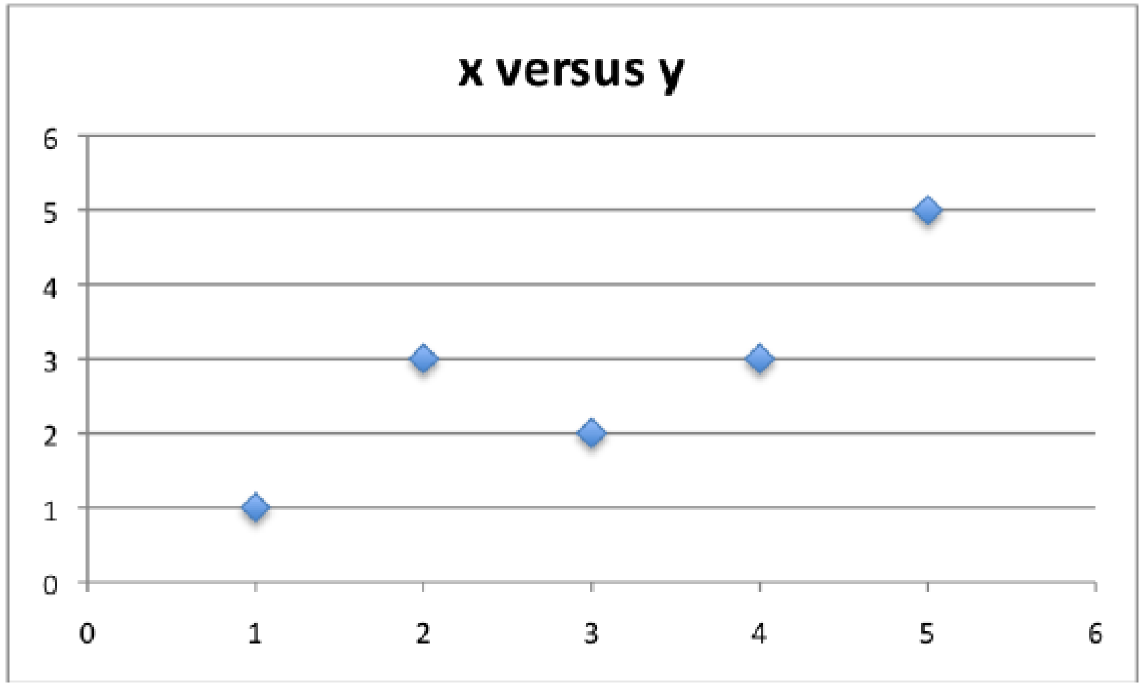
The learning task in a linear regression model is to estimate the values of coefficients used in the data model representation. There are four techniques to estimate the linear regression model:

- Conventional: With simple linear regression when we have a single input, we can use statistics properties to estimate the coefficients.
- Ordinary Least Square: When we have more than one input we can use Ordinary Least Squares to estimate the values of the coefficients. The Ordinary Least Squares procedure seeks to minimize the sum of the squared residuals.
- **Gradient Descent**: When there are one or more inputs you can use a process of optimizing the values of the coefficients by iteratively minimizing the error of the model on your training data.
- Regularized Linear Regression: This is extension of the training of the linear model called regularization methods. These seek to both minimize the sum of the squared error of the model on the training data (using Ordinary Least Squares) but also to reduce the complexity of the model. Two popular regularization procedures are Lasso Regression and Ridge Regression.

1. Linear Regression - Tutorial

Suppose that we have raw data:

x	y
1	1
2	3
4	3
3	2
5	5



We can use **gradient descent algorithm** to optimize B0 and B1. Let’s start with initial guess values B0=B1=0.

$$B0 = 0.0$$
$$B1 = 0.0$$
$$y = 0.0 + 0.0 \times x$$

1. Linear Regression - Tutorial

error = p(i) – y(i)

For first data, x=1 and y=1
p(i) = 0.0 + 0.0 × 1
p(i) = 0

error = (0 – 1)
error = −1

$B0(t+1) = 0.0 - 0.01 \times -1.0$
$B0(t+1) = 0.01$

$B1(t+1) = 0.0 - 0.01 \times -1 \times 1$
$B1(t+1) = 0.01$

The linear regression coefficients can be estimated using gradient descent:

$$Cost\ function\ (J) = \frac{1}{2}(error)^2$$

$$B0(t+1) = B0(t) - alpha \times error$$

$$B1(t+1) = B1(t) - alpha \times error \times x$$

Alpha=0.01 (learning rate)

Now we have new values of B0 and B1. This process must be repeated for the remaining 4 instances from our dataset. One pass through the training dataset is called an epoch of Iteration#1.

1. Linear Regression - Tutorial

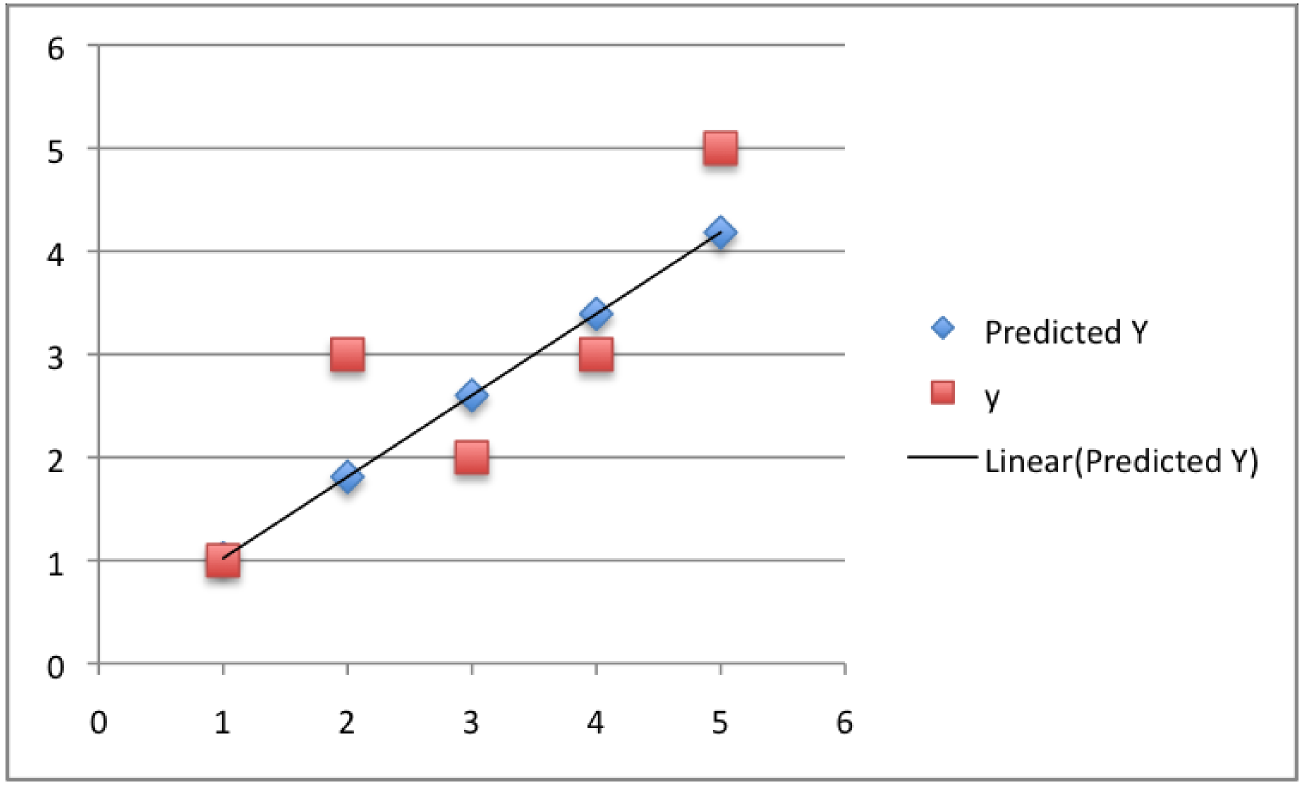
B0	B1
0.01	0.01
0.0397	0.0694
0.066527	0.176708
0.08056049	0.21880847
0.118814462	0.410078328
0.123525534	0.4147894
0.14399449	0.455727313
0.154325453	0.497051164
0.157870663	0.507686795
0.180907617	0.622871563
0.182869825	0.624833772
0.198544452	0.656183024
0.200311686	0.663251962
0.19841101	0.657549935
0.213549404	0.733241901
0.21408149	0.733773988
0.227265196	0.760141398
0.224586888	0.749428167
0.219858174	0.735242025
0.230897491	0.79043861

Here is a list of all of the values for the coefficients over the 20 iterations.

You can see that our final coefficients have the values B0 = 0.230897491 and B1 = 0.79043861.

x	Prediction	y
1	1.021336101	1
2	1.811774711	3
4	3.392651932	3
3	2.602213322	2
5	4.183090542	5

1. Linear Regression - Tutorial



$$R^2 = \frac{\sum(p_i - \bar{y_i})^2}{\sum(y_i - \bar{y_i})^2}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - y_i)^2}{n}}$$

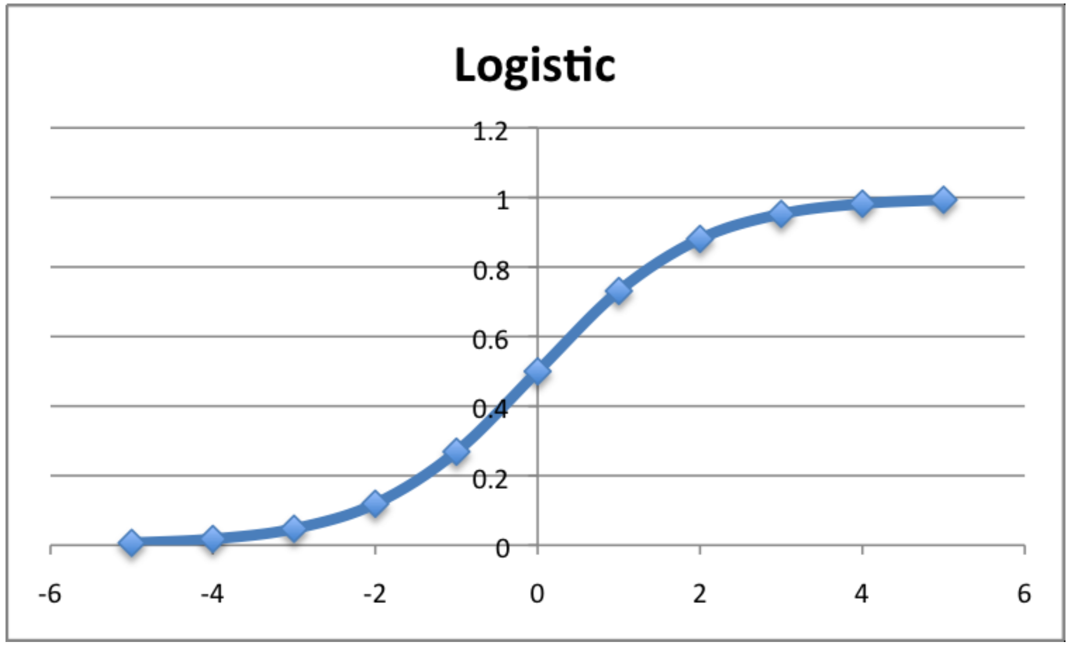
R^2=0.7322
(determination coefficient)

RMSE = 0.7206
(Root mean squared error)

2. Logistic Regression

Logistic regression is another technique borrowed by machine learning from the field of statistics. It is a method used for binary classification problems (problems with two class values). The name is regression but it is used for binary (two-class) classification actually.

Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function or often called sigmoid function.



$$\frac{1}{1+e^{-value}}$$

2. Logistic Regression

A key difference of logistic regression from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.

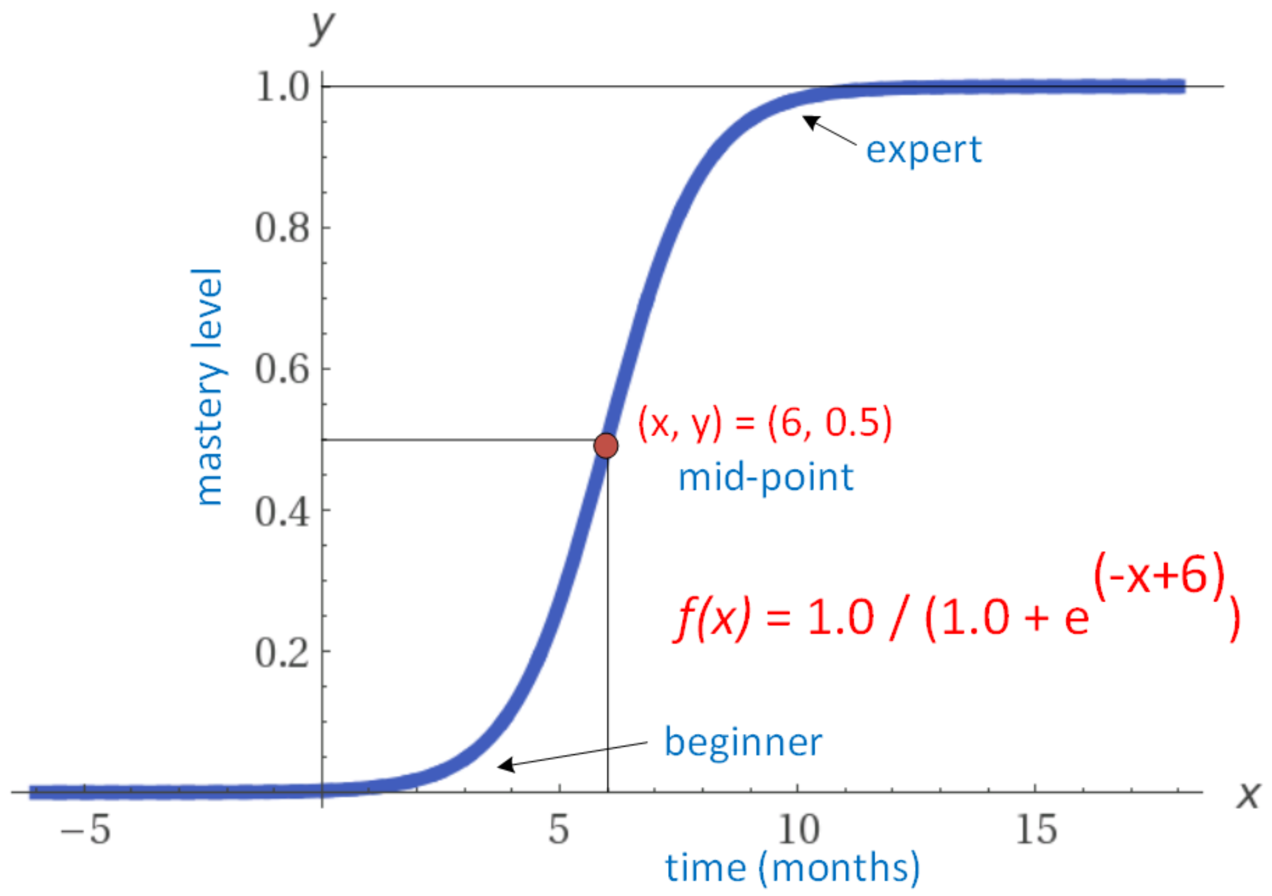
Below is an example logistic regression equation:

$$y = \frac{1}{1+e^{B0+B1\times x}}$$

Where y is the predicted output, B0 is the bias or intercept term and B1 is the coefficient for the single input value (x).

2. Logistic Regression

Example of making prediction with a logistic regression:

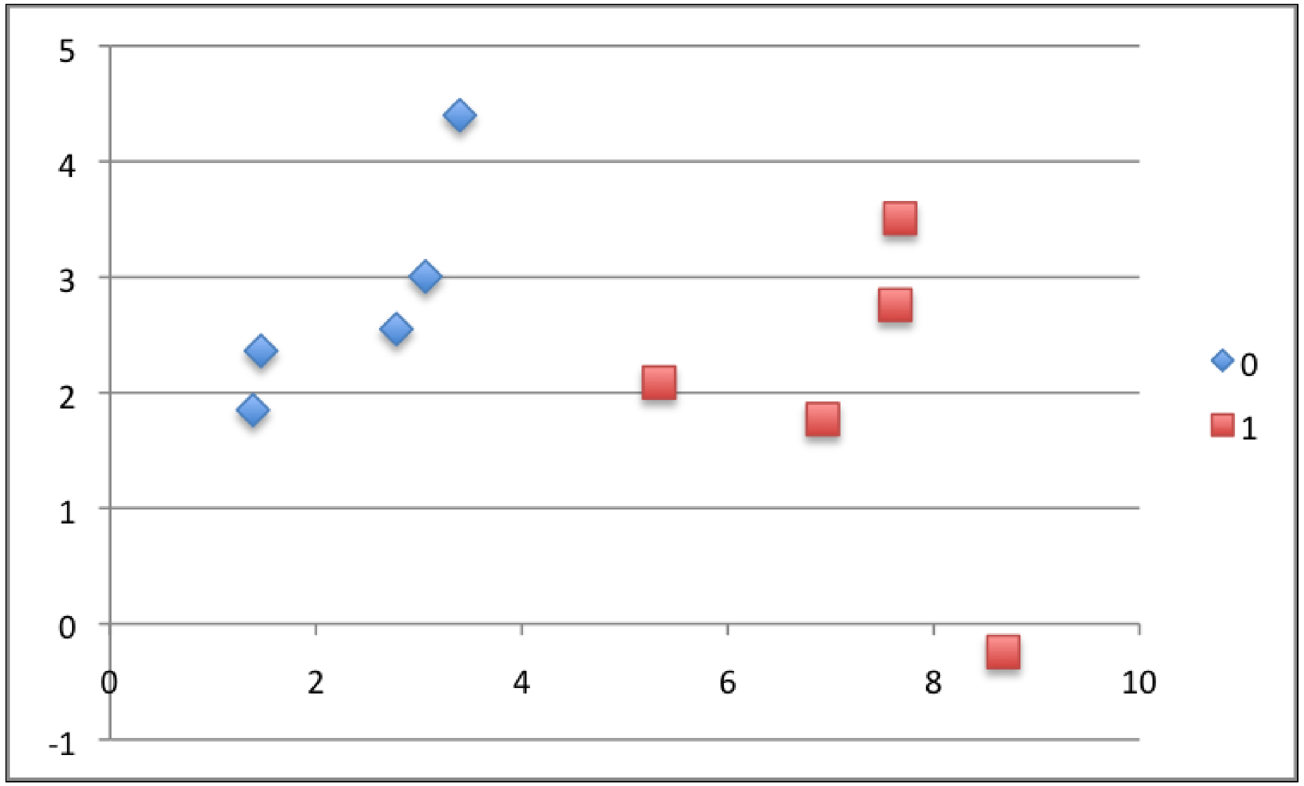


The coefficients of logistic regression coefficient must be estimated from the training data using a certain learning algorithm such as gradient descent.

2. Logistic Regression - Tutorial

Suppose that we have dataset for binary classification with two input variables (X1 and X2) and one output (Y):

X1	X2	Y
2.7810836	2.550537003	0
1.465489372	2.362125076	0
3.396561688	4.400293529	0
1.38807019	1.850220317	0
3.06407232	3.005305973	0
7.627531214	2.759262235	1
5.332441248	2.088626775	1
6.922596716	1.77106367	1
8.675418651	-0.242068655	1
7.673756466	3.508563011	1



2. Logistic Regression - Tutorial

The logistic regression model takes real-valued inputs and makes a prediction as for instance:
If the prediction output is greater than 0.5 we can take the output as a prediction for the default class (class 0), otherwise the prediction is for the other class (class 1).

$$prediction = \frac{1}{1 + e^{-(B0+B1\times X1+B2\times X2)}}$$

The job of the learning algorithm will be to discover the best values for the coefficients (B0, B1 and B2) based on the training data.

Let's start off by setting initial value for B0=B1=B2=0. The first training instance is:
X1 = 2:7810836, X2 = 2:550537003, Y = 0, resulting the prediction:

$$prediction = \frac{1}{1 + e^{-(0.0+0.0\times 2.7810836+0.0\times 2.550537003)}}$$
$$prediction = 0.5$$

2. Logistic Regression - Tutorial

We can calculate the new coefficient values using a simple update equation based on gradient descent.

$$b = b + alpha \times (y - prediction) \times prediction \times (1 - prediction) \times x$$

The learning rate (α) controls how much the coefficients changes or learns each time it is iteratively updated. Good values might be in the range 0.1 to 0.3. Let's use a value of α =0.3.

$$B0 = B0 + 0.3 \times (0 - 0.5) \times 0.5 \times (1 - 0.5) \times 1.0$$
$$B1 = B1 + 0.3 \times (0 - 0.5) \times 0.5 \times (1 - 0.5) \times 2.7810836$$
$$B2 = B2 + 0.3 \times (0 - 0.5) \times 0.5 \times (1 - 0.5) \times 2.550537003$$

$$B0 = -0.0375$$
$$B1 = -0.104290635$$
$$B2 = -0.095645138$$

Derivative of Sigmoid function

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

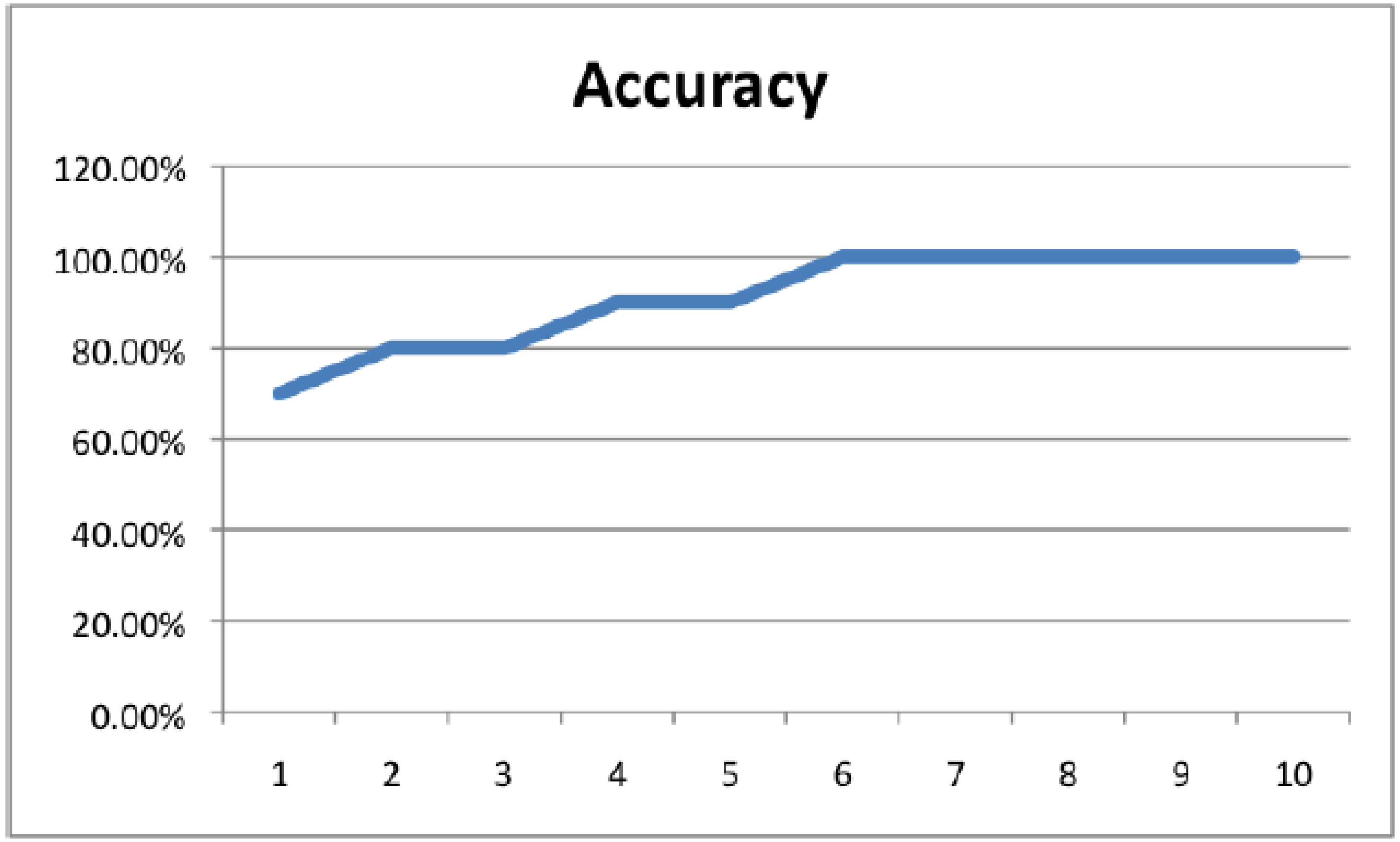
$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = y(1-y)$$

2. Logistic Regression - Tutorial

We repeat this process and update for each training instance in the dataset (called 1 epoch). After 10 epochs, the updated coefficient are:

$$B0 = -0.406605464$$
$$B1 = 0.852573316$$
$$B2 = -1.104746259$$



Accuracy vs Iteration (epoch)

2. Logistic Regression - Tutorial

Now we have the trained model, we can use it to make prediction (testing). Realistically, testing should use new dataset. But let's make prediction using training dataset.

X1	X2	Prediction
2.7810836	2.550537003	0.298756986
1.465489372	2.362125076	0.145951056
3.396561688	4.400293529	0.085333265
1.38807019	1.850220317	0.219737314
3.06407232	3.005305973	0.247059
7.627531214	2.759262235	0.954702135
5.332441248	2.088626775	0.862034191
6.922596716	1.77106367	0.971772905
8.675418651	-0.242068655	0.999295452
7.673756466	3.508563011	0.905489323

IF (out<0.5) THEN 0 ELSE 1

$\Longrightarrow$

Prediction	Crisp
0.298756986	0
0.145951056	0
0.085333265	0
0.219737314	0
0.247059	0
0.954702135	1
0.862034191	1
0.971772905	1
0.999295452	1
0.905489323	1

$$ccuracy = \frac{Correct Predictions}{Total Predictions} \times 100$$

$$ccuracy = \frac{10}{10} \times 100$$

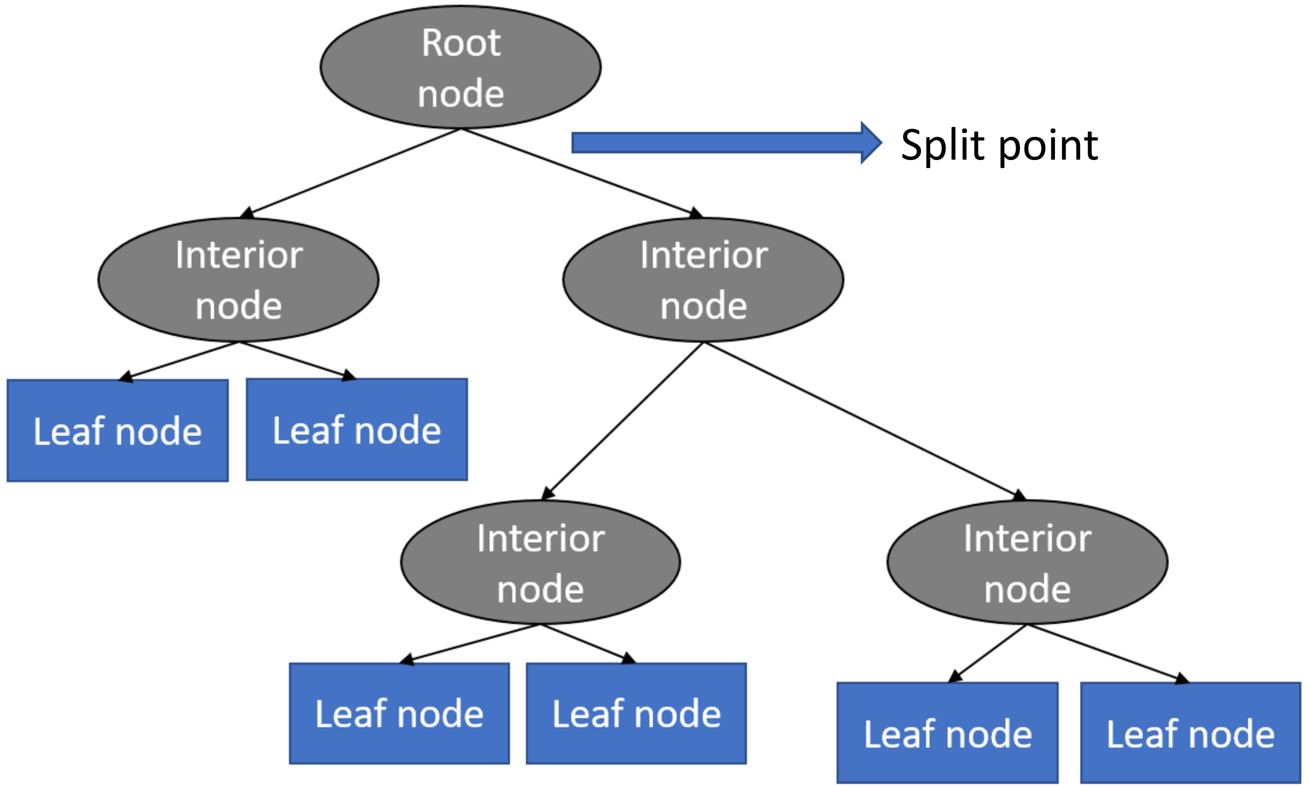
100% Accuracy

3. Decision Tree

Decision Trees are important type of algorithm in predictive modeling using ML. The classical decision tree algorithms have been around for 3 decades and modern variations like **Random Forest** is among the most powerful techniques.

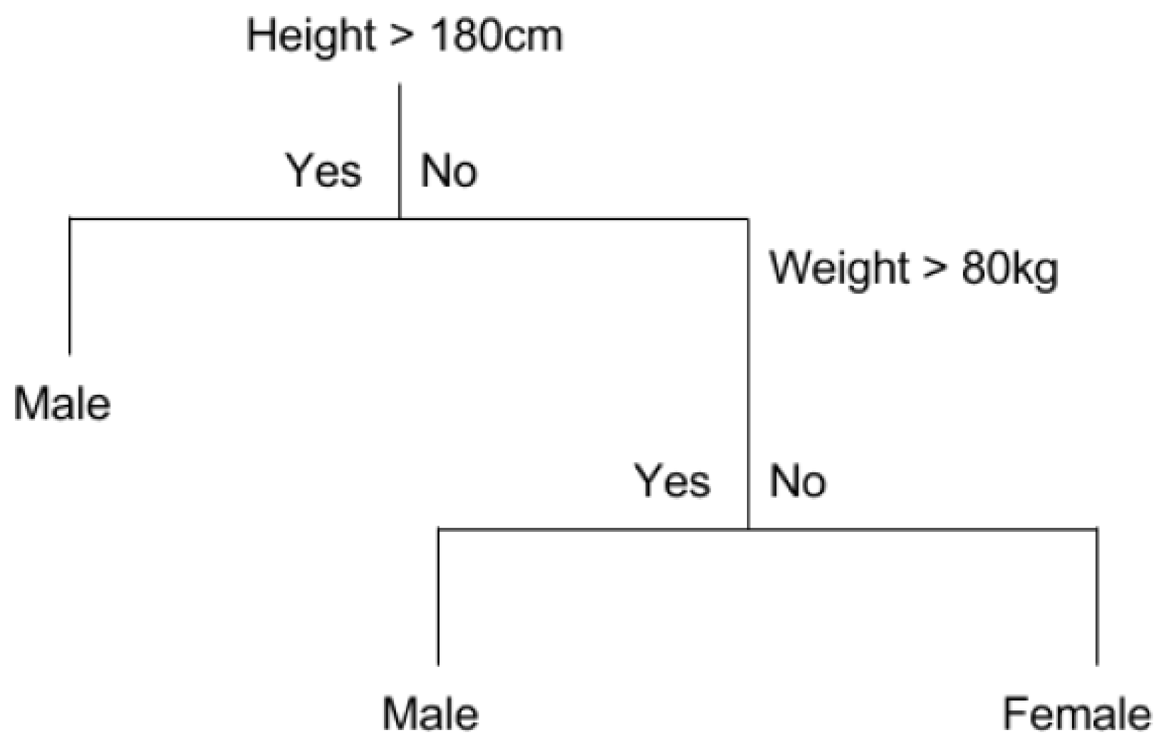
Here, we will discuss about a humble decision tree algorithm named **CART** (Classification and Regression Trees).

In CART, model representation is a binary tree, nothing too fancy.



3. Decision Tree

```
If Height > 180 cm Then Male
If Height <= 180 cm AND Weight > 80 kg Then Male
If Height <= 180 cm AND Weight <= 80 kg Then Female
Make Predictions With CART Models
```



Example of a rule representation in a decision tree.

Split point=180cm

3. Decision Tree

In CART, the cost function to be minimized to choose split points is called **Gini Index**. For classification, the Gini Index function indicates how pure the **leaf nodes** are.

$$G = \sum_{k=1}^{n} p_k \times (1 - p_k)$$

Where G is Gini Index over all classes and p_k is proportion of training instances with class k. A node that has all classes of the same type (perfect class purity/perfect classification) will have G=0.

For binary classification problem, the Gini Index can be written:

$$G = 1 - ({p_1}^2 + {p_2}^2)$$

Where p1 is the proportion of instances in the node with class 1 and p2 is the proportion of instances in the node for class 2.

3. Decision Tree

The Gini index calculation for each node is weighted by the total number of instances in the root node. The Gini score for a chosen **split point** in a binary classification problem is calculated as follows:

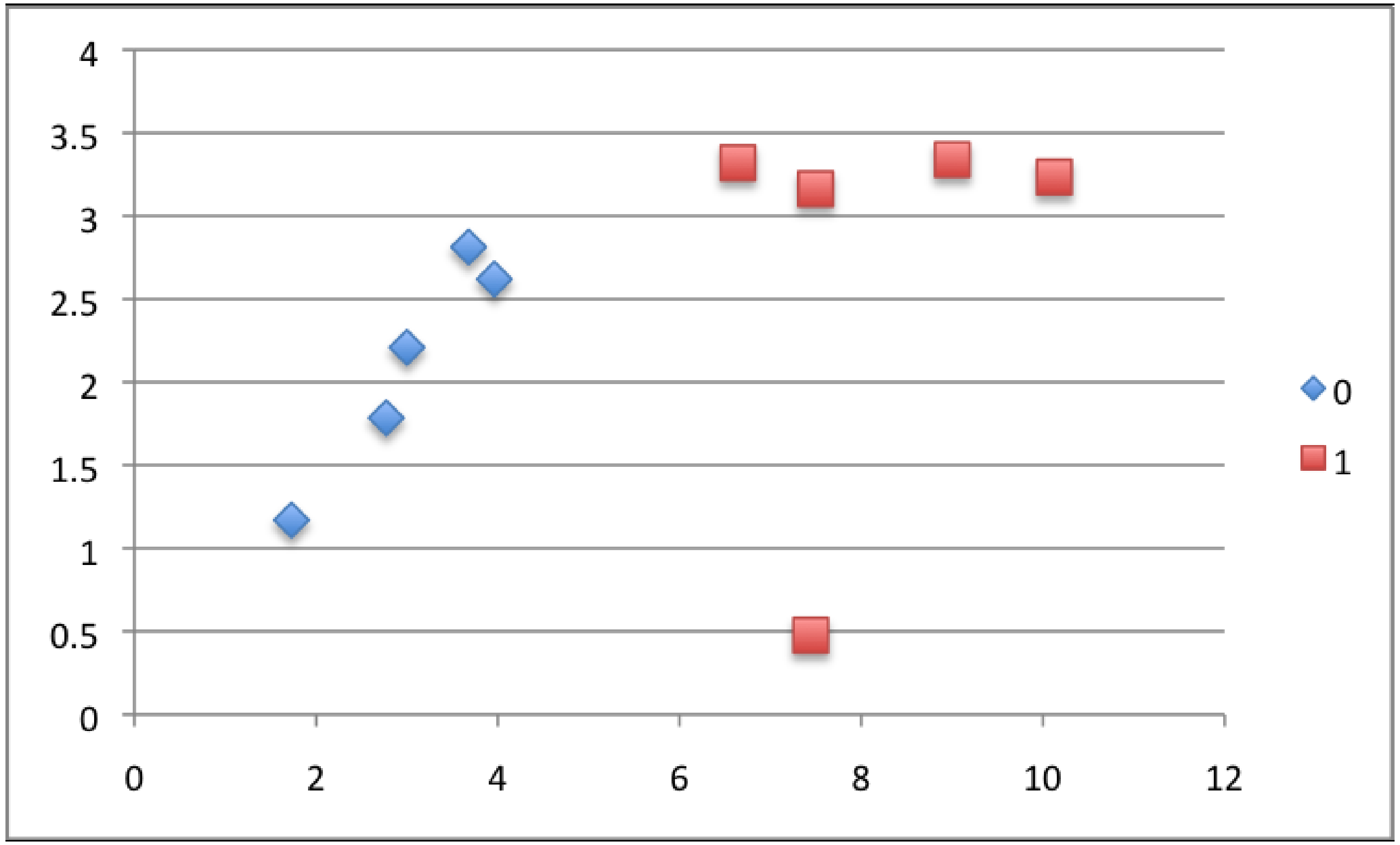
$$G = ((1 - ({g1_1}^2 + {g1_2}^2)) \times \frac{n_{g1}}{n}) + ((1 - ({g2_1}^2 + {g2_2}^2)) \times \frac{n_{g2}}{n})$$

Where G is Gini Index for the split point, $g1_1$ is the proportion of instances in group 1 for class 1, $g1_2$ in group 1 for class 2, $g2_1$ in group 2 for class 1, $g2_2$ in group 2 for class 2, n_{g1} and n_{g2} are the total number of instances in group 1 and 2 and n are the total number of instances we are trying to group (classify) from the root node.

3. Decision Tree - Tutorial

In this tutorial we will work through a simple binary (two-class) classification problem for CART. To keep things simple we will work with a two input variables (X1 and X2) and a single output variable (Y).

X1	X2	Y
2.771244718	1.784783929	0
1.728571309	1.169761413	0
3.678319846	2.81281357	0
3.961043357	2.61995032	0
2.999208922	2.209014212	0
7.497545867	3.162953546	1
9.00220326	3.339047188	1
7.444542326	0.476683375	1
10.12493903	3.234550982	1
6.642287351	3.319983761	1



3. Decision Tree - Tutorial

We calculate the Gini index for a **leaf node** as follows:

$$G = 1 - ({p_1}^2 + {p_2}^2)$$

We will always have two groups, a LEFT and RIGHT group because we are using a binary tree. And we know from our dataset that we only have two classes.

For example, if a LEFT group has 3 instances with class 0 and 4 instances with class 1, then the proportion of data instances with class 0 would be: $\frac{3}{7}$ = 0.428571429.

To get a feeling for Gini index scores for leaf nodes, take a look at the table below. It provides 7 different scenarios for mixes of 0 and 1 classes in a single group.

3. Decision Tree - Tutorial

Sample of Gini Index calculation for leaf nodes (for a single group)

Class 0	Class 1	Count	Class 0/Count	Class 1/Count	Gini
10	10	20	0.5	0.5	0.5
19	1	20	0.95	0.05	0.095
1	19	20	0.05	0.95	0.095
15	5	20	0.75	0.25	0.375
5	15	20	0.25	0.75	0.375
11	9	20	0.55	0.45	0.495
20	0	20	1	0	0

You can see when the group has a 50-50 mix in the 1st row, that Gini is 0.5. This is the worst possible split.

You can also see a case where the group only has data instances with class 0 on the last row and a Gini index of 0. This is an example of a perfect split.

3. Decision Tree - Tutorial

The first step is to choose a split that will become the root node of our decision tree. We will start with the first candidate split point which is the X1 attribute and the value of X1 in the first instance: X1 = 2.7712.

IF $X1 < 2.7712$ THEN LEFT

IF $X1 \geq 2.7712$ THEN RIGHT

Let's apply this rule to each X1 value in our training dataset. See the answer we get for each numbered instance in the dataset:

X1	Y	Group
2.771244718	0	RIGHT
1.728571309	0	LEFT
3.678319846	0	RIGHT
3.961043357	0	RIGHT
2.999208922	0	RIGHT
7.497545867	1	RIGHT
9.00220326	1	RIGHT
7.444542326	1	RIGHT
10.12493903	1	RIGHT
6.642287351	1	RIGHT

3. Decision Tree - Tutorial

How good was this split?
The LEFT group only has one member, where as the RIGHT group has 9 members. We can calculate the proportion of training instances that have each class, for LEFT and RIGHT group respectively:

- $Y = 0$: $g1_1 = \frac{1}{1}$
- $Y = 1$: $g1_2 = \frac{0}{1}$

(FOR LEFT)

- $Y = 0$: $g2_1 = \frac{4}{9}$
- $Y = 1$: $g2_2 = \frac{5}{9}$

(FOR RIGHT)

We now have enough information to calculate the Gini index for this split:

$$G = ((1 - ({g1_1}^2 + {g1_2}^2)) \times \frac{n_{g1}}{n}) + ((1 - ({g2_1}^2 + {g2_2}^2)) \times \frac{n_{g2}}{n})$$

$$Gini(X1 = 2.7712) = ((1 - (\frac{1}{1}^2 + \frac{0}{1}^2)) \times \frac{1}{10}) + ((1 - (\frac{4}{9}^2 + \frac{5}{9}^2)) \times \frac{9}{10})$$

A bad split!!

$$Gini(X1 = 2.7712) = 0.4444$$

3. Decision Tree - Tutorial

We can evaluate each candidate split point using the process above with the values from X1 and X2. If we look at the graph of the data, we can see that we can probably draw a vertical line to separate the classes. This would translate to a split point for X1 with a value around 0.5. A close t would be the value for X1 in the last instance: X1 = 6.6422.

$$\text{IF } X1 < 6.6422 \text{ THEN LEFT}$$

$$\text{IF } X1 \geq 6.6422 \text{ THEN RIGHT}$$

X1	Y	Group
2.771244718	0	LEFT
1.728571309	0	LEFT
3.678319846	0	LEFT
3.961043357	0	LEFT
2.999208922	0	LEFT
7.497545867	1	RIGHT
9.00220326	1	RIGHT
7.444542326	1	RIGHT
10.12493903	1	RIGHT
6.642287351	1	RIGHT

$$Gini(X1 = 6.6422) = ((1 - (\frac{5}{5}^2 + \frac{0}{5}^2)) \times \frac{5}{10}) + ((1 - (\frac{0}{5}^2 + \frac{5}{5}^2)) \times \frac{5}{10})$$

$$Gini(X1 = 6.6422) = 0.0$$

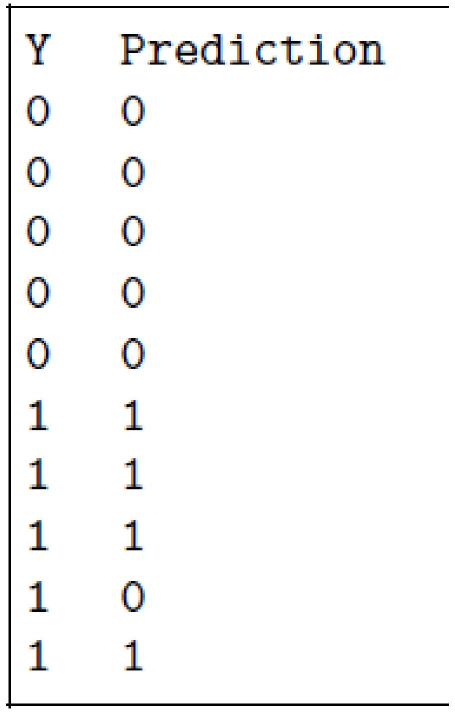
A perfect split.

3. Decision Tree - Tutorial

We can now use this decision tree to make some predictions. Let's classify some new data generated for each class using the same distribution (the test dataset):

X1	X2	Y
2.343875381	2.051757824	0
3.536904049	3.032932531	0
2.801395588	2.786327755	0
3.656342926	2.581460765	0
2.853194386	1.052331062	0
8.907647835	3.730540859	1
9.752464513	3.740754624	1
8.016361622	3.013408249	1
6.58490395	2.436333477	1
7.142525173	3.650120799	1

Using the decision tree with a single split at, X1 = 6.6422 we can classify the test instances as follows:



Y	Prediction
0	0
0	0
0	0
0	0
0	0
1	1
1	1
1	1
1	0
1	1

The result is a nearly perfect classification, accuracy of 90%.

4. Naïve Bayes

Naive Bayes is a simple but surprisingly powerful algorithm for predictive modeling.

Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge. Bayes' Theorem is stated as:

$$P(h|d) = \frac{P(d|h) \times P(h)}{P(d)}$$

Where:
P (h|d) is the probability of hypothesis h given the data d. This is called the **posterior probability**.
P (d|h) is the probability of data d given that the hypothesis h was true.
P (h) is the probability of hypothesis h being true (regardless of the data). This is called **the prior probability** of h.

4. Naïve Bayes

After calculating the posterior probability for a number of different hypotheses, you can select the hypothesis with the highest probability.

This is the maximum probable hypothesis and may formally be called the **maximum a posteriori** (MAP) hypothesis. This can be written as:

$$MAP(h) = max(P(h|d))$$

$$MAP(h) = max(\frac{P(d|h) \times P(h)}{P(d)})$$

$$MAP(h) = max(P(d|h) \times P(h))$$

The P (d) is a normalizing term which allows us to calculate the probability and it is constant so we can drop it.

4. Naïve Bayes - Tutorial

In this tutorial, we use dataset having two categorical input variables and an output variable with two values of class.

Weather	Car	Class
sunny	working	go-out
rainy	broken	go-out
sunny	working	go-out
sunny	working	go-out
sunny	working	go-out
rainy	broken	stay-home
rainy	broken	stay-home
sunny	working	stay-home
sunny	broken	stay-home
rainy	broken	stay-home

Weather: sunny = 1, rainy = 0

Car: working = 1, broken = 0

Class: go-out = 1, stay-home = 0

Weather	Car	Class
1	1	1
0	0	1
1	1	1
1	1	1
1	1	1
0	0	0
0	0	0
1	1	0
1	0	0
0	0	0

4. Naïve Bayes - Tutorial

There are two types of quantities that need to be calculated from the dataset for the Naïve Bayes model: Class probabilities and Conditional probabilities.

- The Class probabilities for classes 0 and 1 are:

$$P(class = 1) = \frac{count(class = 1)}{count(class = 0) + count(class = 1)}$$

$$P(class = 0) = \frac{count(class = 0)}{count(class = 0) + count(class = 1)}$$

$$P(class = 1) = \frac{5}{5 + 5}$$

$$P(class = 0) = \frac{5}{5 + 5}$$

This works out to be a probability of 0.5 for any given data instance belonging to class 0 or class 1.

4. Naïve Bayes - Tutorial

- The Conditional probabilities for the dataset can be calculated as:

For Weather Input Variable

$$P(weather = sunny|class = \text{go-out}) = \frac{count(weather = sunny \wedge class = \text{go-out})}{count(class = \text{go-out})}$$

$$P(weather = rainy|class = \text{go-out}) = \frac{count(weather = rainy \wedge class = \text{go-out})}{count(class = \text{go-out})}$$

$$P(weather = sunny|class = \text{stay-home}) = \frac{count(weather = sunny \wedge class = \text{stay-home})}{count(class = \text{stay-home})}$$

$$P(weather = rainy|class = \text{stay-home}) = \frac{count(weather = rainy \wedge class = \text{stay-home})}{count(class = \text{stay-home})}$$

$$P(weather = sunny|class = \text{go-out}) = 0.8$$
$$P(weather = rainy|class = \text{go-out}) = 0.2$$
$$P(weather = sunny|class = \text{stay-home}) = 0.4$$
$$P(weather = rainy|class = \text{stay-home}) = 0.6$$

$\Lambda \;\; is\ shorthand\ for\ conjuction\ (AND)$

4. Naïve Bayes - Tutorial

For Car Input Variable

$$P(car = working|class = \text{go-out}) = \frac{count(car = working \wedge class = \text{go-out})}{count(class = \text{go-out})}$$

$$P(car = broken|class = \text{go-out}) = \frac{count(car = broken \wedge class = \text{go-out})}{count(class = \text{go-out})}$$

$$P(car = working|class = \text{stay-home}) = \frac{count(car = working \wedge class = \text{stay-home})}{count(class = \text{stay-home})}$$

$$P(car = broken|class = \text{stay-home}) = \frac{count(car = broken \wedge class = \text{stay-home})}{count(class = \text{stay-home})}$$

$$P(car = working|class = \text{go-out}) = 0.8$$
$$P(car = broken|class = \text{go-out}) = 0.2$$
$$P(car = working|class = \text{stay-home}) = 0.2$$
$$P(car = broken|class = \text{stay-home}) = 0.8$$

We now have done the Naive Bayes model to make predictions.

4. Naïve Bayes - Tutorial

We can make predictions using Bayes Theorem using MAP.
For instance, what should be the predicted output when: *weather=sunny*, *car=working*.

We plug the probabilities in our model for both classes:

$$\begin{aligned}\text{go-out} = {} & P(weather = sunny|class = \text{go-out})\times \\ & P(car = working|class = \text{go-out})\times \\ & P(class = \text{go-out})\end{aligned}$$

$$\text{go-out} = 0.8 \times 0.8 \times 0.5$$
$$\text{go-out} = 0.32$$

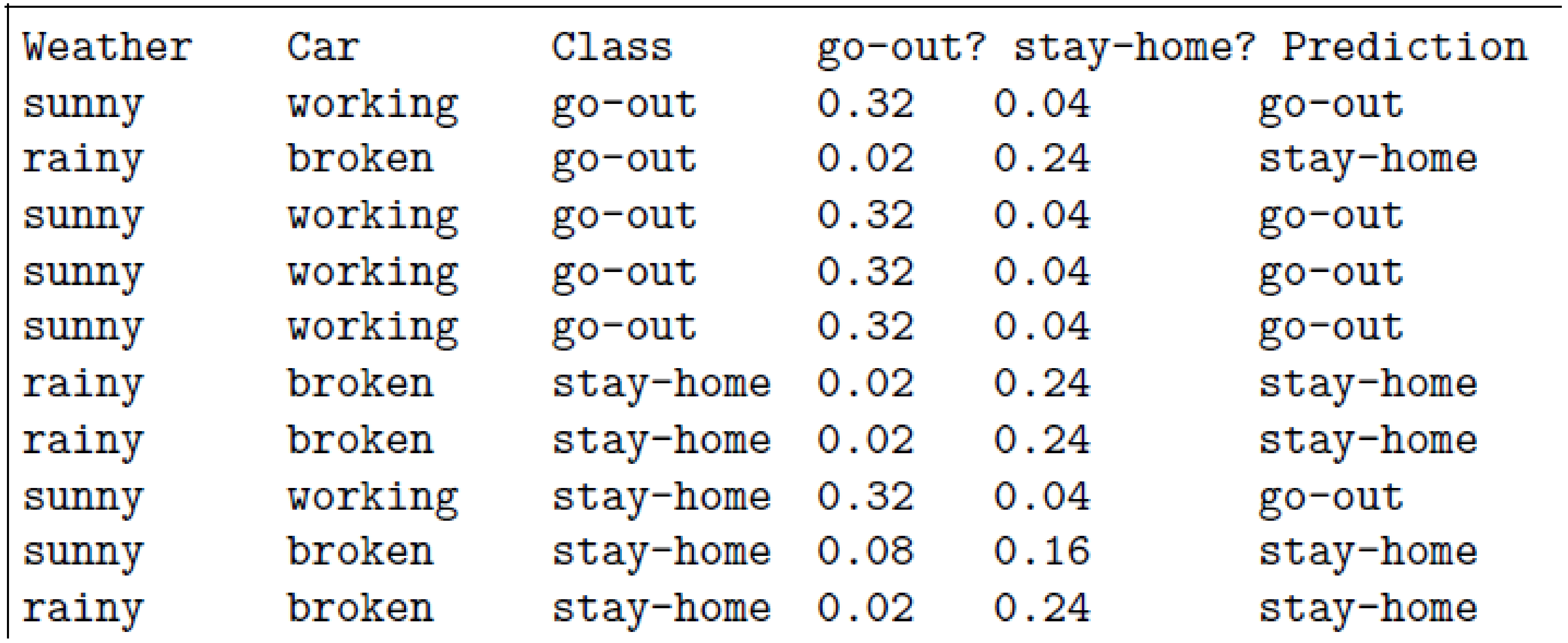
$$\begin{aligned}\text{stay-home} = {} & P(weather = sunny|class = \text{stay-home})\times \\ & P(car = working|class = \text{stay-home})\times \\ & P(class = \text{stay-home})\end{aligned}$$

$$\text{stay-home} = 0.4 \times 0.2 \times 0.5$$
$$\text{stay-home} = 0.04$$

We can see that 0.32 is greater than 0.04, therefore we predict *go-out* for this instance, which is correct. We can repeat this operation for the entire dataset:

4. Naïve Bayes - Tutorial

Prediction Results:



Weather	Car	Class	go-out?	stay-home?	Prediction
sunny	working	go-out	0.32	0.04	go-out
rainy	broken	go-out	0.02	0.24	stay-home
sunny	working	go-out	0.32	0.04	go-out
sunny	working	go-out	0.32	0.04	go-out
sunny	working	go-out	0.32	0.04	go-out
rainy	broken	stay-home	0.02	0.24	stay-home
rainy	broken	stay-home	0.02	0.24	stay-home
sunny	working	stay-home	0.32	0.04	go-out
sunny	broken	stay-home	0.08	0.16	stay-home
rainy	broken	stay-home	0.02	0.24	stay-home

If we tally up the predictions compared to the actual class values, we get an accuracy of 80%, which is excellent given that there are conflicting examples in the dataset.

5. k-Nearest Neighbors (KNN)

The k-Nearest Neighbors (KNN) algorithm is very simple and very effective. It uses the entire training dataset that is stored such that no learning is required.

To determine which of the k instances in the training dataset are most similar to a new input, a distance measure is used such as Euclidean Distance. Euclidean distance is calculated as the square root of the sum of the squared differences between a point a and point b across all input attributes i.

$$EuclideanDistance(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

There are other popular distance measures including Hamming Distance, Manhattan Distance, etc.

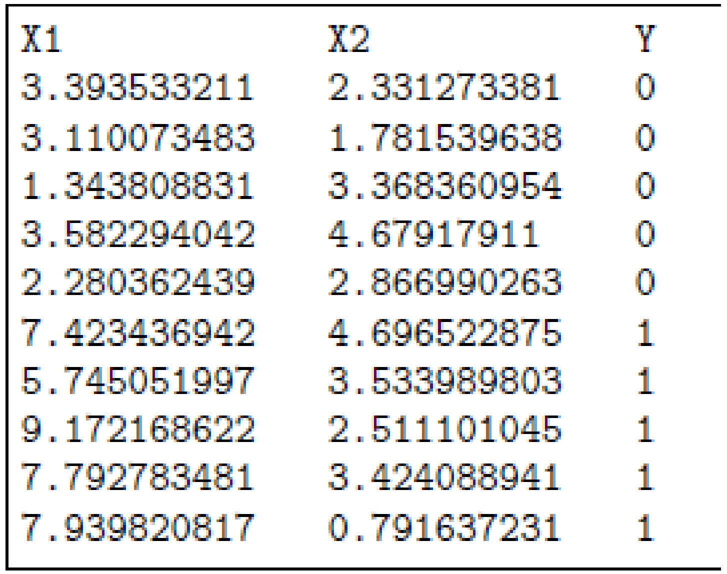
5. k-Nearest Neighbors (KNN)

When KNN is used for classification, the output can be calculated as the class with highest frequency from the k-most similar instances. Each instance votes for their class and the class with most votes is taken as the prediction. Therefore, it is a good idea that k is set to be odd number to avoid tie in the voting for even number of class (e.g. 2 in binary classification) and vise versa.

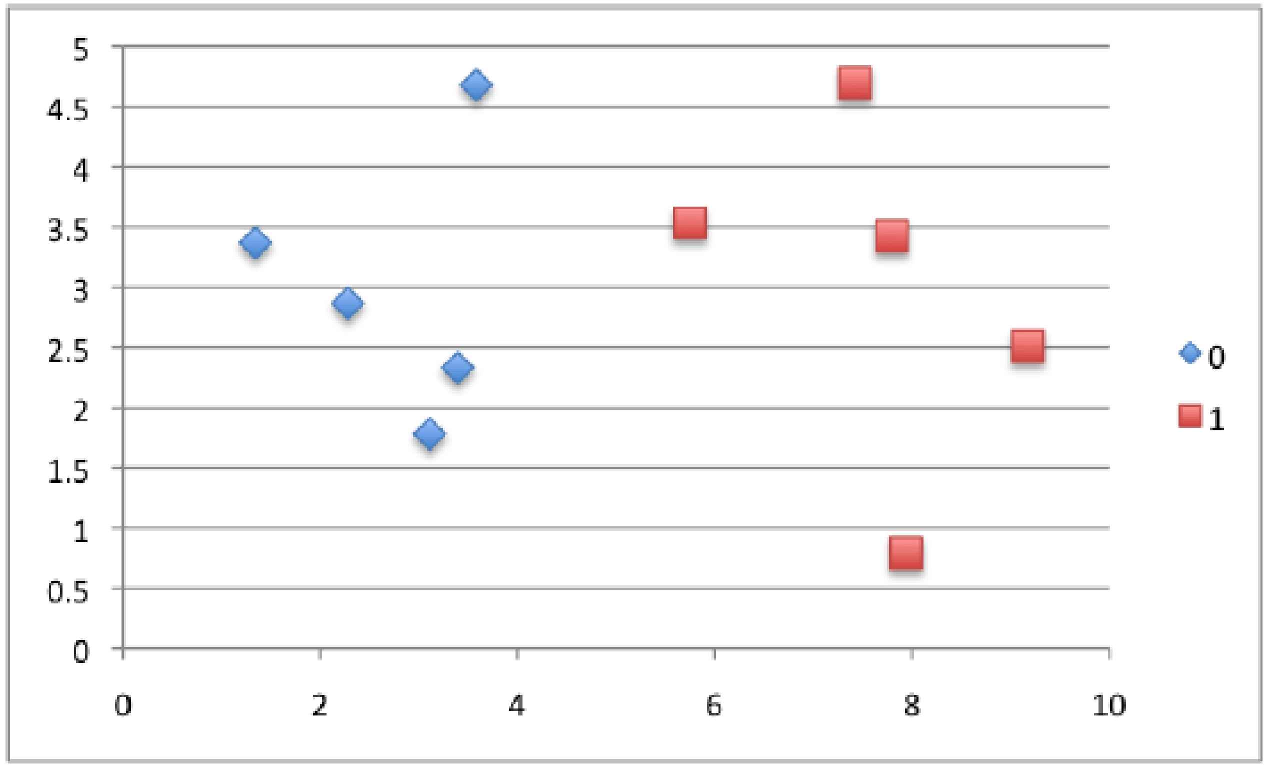
However, KNN works well with small number of input variables, but struggles when the number of inputs is very large, i.e. curse of dimensionality.

5. k-Nearest Neighbors – Tutorial

Suppose that the problem is a binary (two-class) classification problem.
The dataset contains two input variables (X1 and X2) and the class output variable with the values 0 and 1. The dataset contains 10 records, 5 that belong to each class.



X1	X2	Y
3.393533211	2.331273381	0
3.110073483	1.781539638	0
1.343808831	3.368360954	0
3.582294042	4.67917911	0
2.280362439	2.866990263	0
7.423436942	4.696522875	1
5.745051997	3.533989803	1
9.172168622	2.511101045	1
7.792783481	3.424088941	1
7.939820817	0.791637231	1



5. k-Nearest Neighbors – Tutorial

We will work through the calculation of the Euclidean distance for two instances from our dataset.

Instance	X1	X2
1	3.393533211	2.331273381
2	3.110073483	1.781539638

The 1st step is to calculate the squared difference for each attribute:

$$SquaredDifference1 = (X1_1 - X1_2)^2$$
$$SquaredDifference2 = (X2_1 - X2_2)^2$$

$$SquaredDifference1 = (3.393533211 - 3.110073483)^2$$
$$SquaredDifference2 = (2.331273381 - 1.781539638)^2$$

$$SquaredDifference1 = 0.08034941698$$
$$SquaredDifference2 = 0.3022071889$$

5. k-Nearest Neighbors – Tutorial

We calculate the sum of these squared differences as:

$$SumSquaredDifference = SquaredDifference1 + SquaredDifference2$$
$$SumSquaredDifference = 0.080349417 + 0.302207188$$
$$SumSquaredDifference = 0.382556606$$

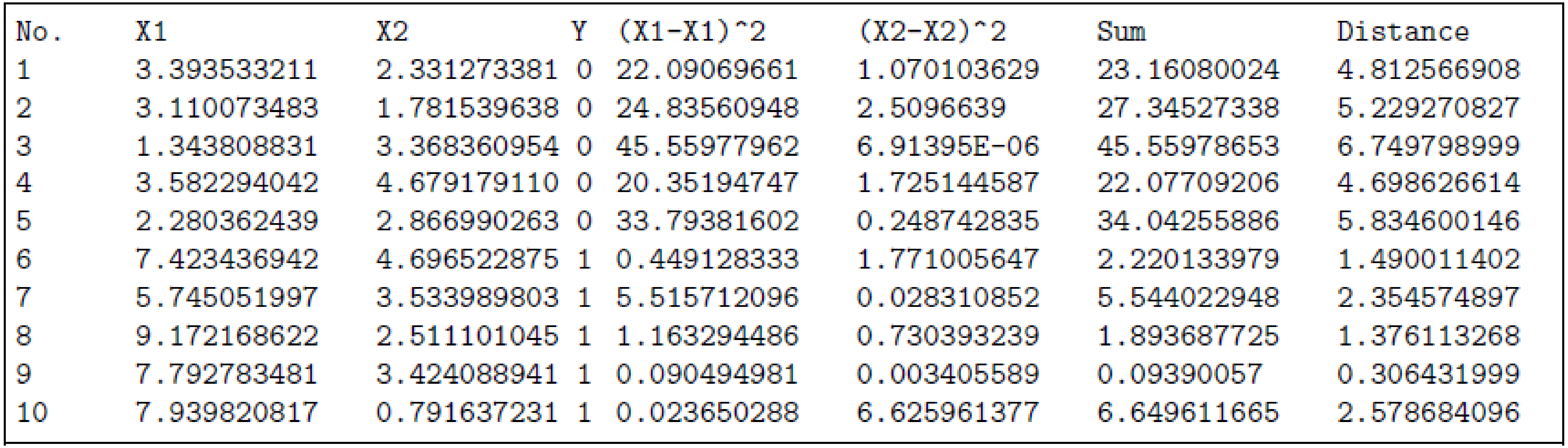
$$Distance = \sqrt{SumSquaredDifference}$$
$$Distance = 0.618511605$$

We can calculate this distance for entire data.
Now we have made a model, then we can make prediction for new data instance.
Let’s say a new data is: X1 = 8:093607318, X2 = 3:365731514, Y = 1.

5. k-Nearest Neighbors – Tutorial

A new data for prediction (testing): X1 = 8:093607318, X2 = 3:365731514, Y = 1.

The 1st step is to calculate the Euclidean distance between the new data and All data instances in the training dataset.

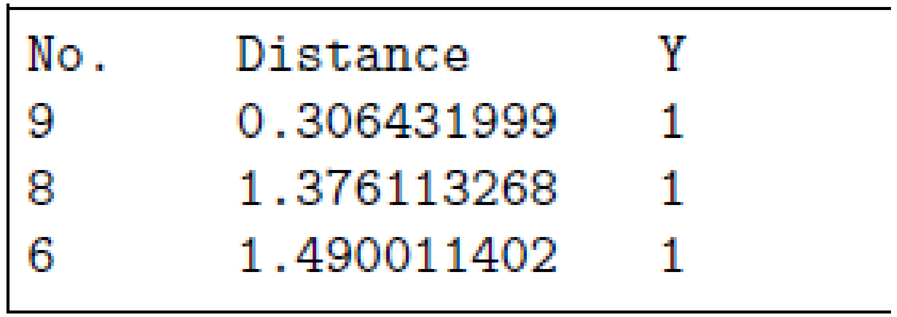


No.	X1	X2	Y	(X1-X1)^2	(X2-X2)^2	Sum	Distance
1	3.393533211	2.331273381	0	22.09069661	1.070103629	23.16080024	4.812566908
2	3.110073483	1.781539638	0	24.83560948	2.5096639	27.34527338	5.229270827
3	1.343808831	3.368360954	0	45.55977962	6.91395E-06	45.55978653	6.749798999
4	3.582294042	4.679179110	0	20.35194747	1.725144587	22.07709206	4.698626614
5	2.280362439	2.866990263	0	33.79381602	0.248742835	34.04255886	5.834600146
6	7.423436942	4.696522875	1	0.449128333	1.771005647	2.220133979	1.490011402
7	5.745051997	3.533989803	1	5.515712096	0.028310852	5.544022948	2.354574897
8	9.172168622	2.511101045	1	1.163294486	0.730393239	1.893687725	1.376113268
9	7.792783481	3.424088941	1	0.090494981	0.003405589	0.09390057	0.306431999
10	7.939820817	0.791637231	1	0.023650288	6.625961377	6.649611665	2.578684096

5. k-Nearest Neighbors – Tutorial

With setting k=3 and therefore choosing 3 most similar (least distance) neighbors to the training data. The k=3 is small and easy to use and it is add to avoid tie on the vole of the class output.

The 3 most similar neighbors to the new data instance are:



No.	Distance	Y
9	0.306431999	1
8	1.376113268	1
6	1.490011402	1

All outputs are 1, therefore the prediction of this new test data is 1 (i.e., the new data belong to class 1).

6. Support Vector Machines (SVM)

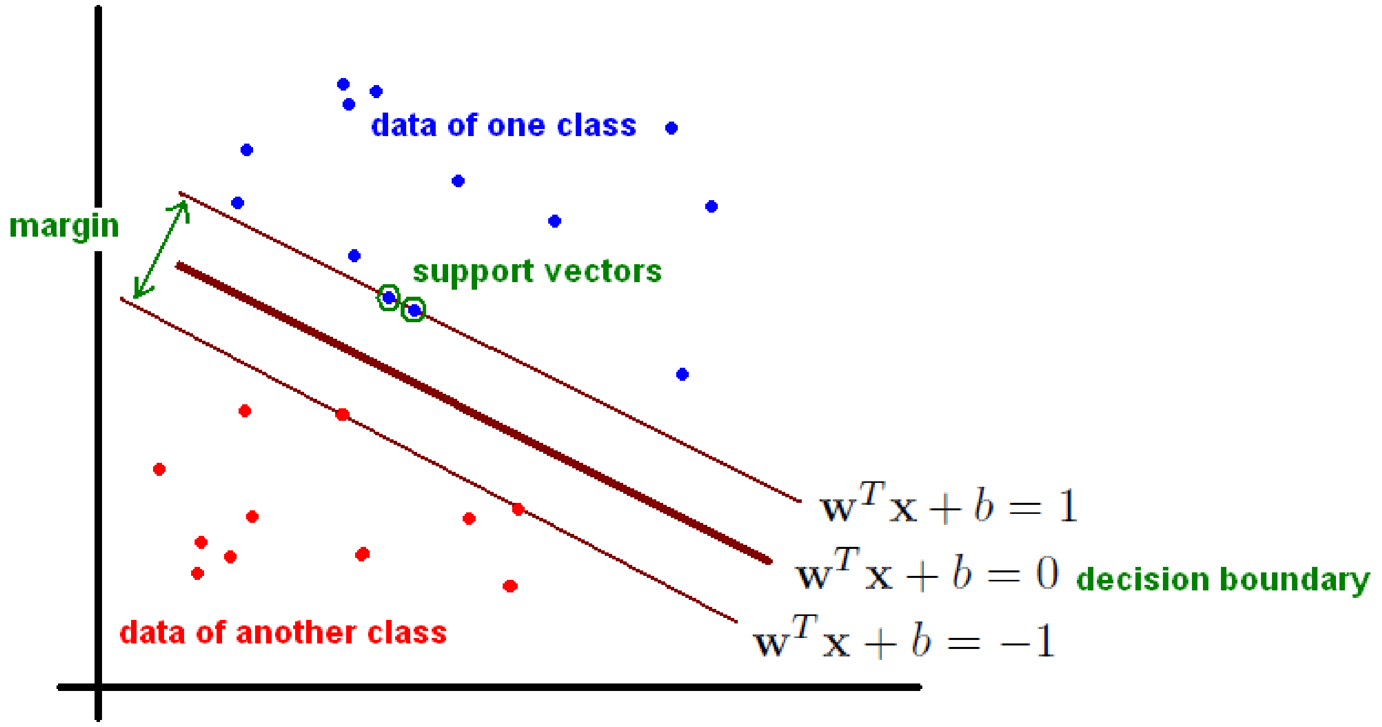
Support Vector Machines are a flexible **nonparametric** (does not make prior assumption about the model structure) machine learning algorithm and perhaps one of the most popular and talked about machine learning algorithms.
SVMs were extremely popular around the time they were developed in the 1990s and continue to become high-performing algorithms with little tuning.

Applications of SVM in image processing field:

- https://www.youtube.com/watch?v=itmV7druy9Y

- https://www.youtube.com/watch?v=cyNKepIPEP4

6. Support Vector Machines (SVM)



A sample of **linearly separable data** classified

Here is a simplified expression of what SVM do:

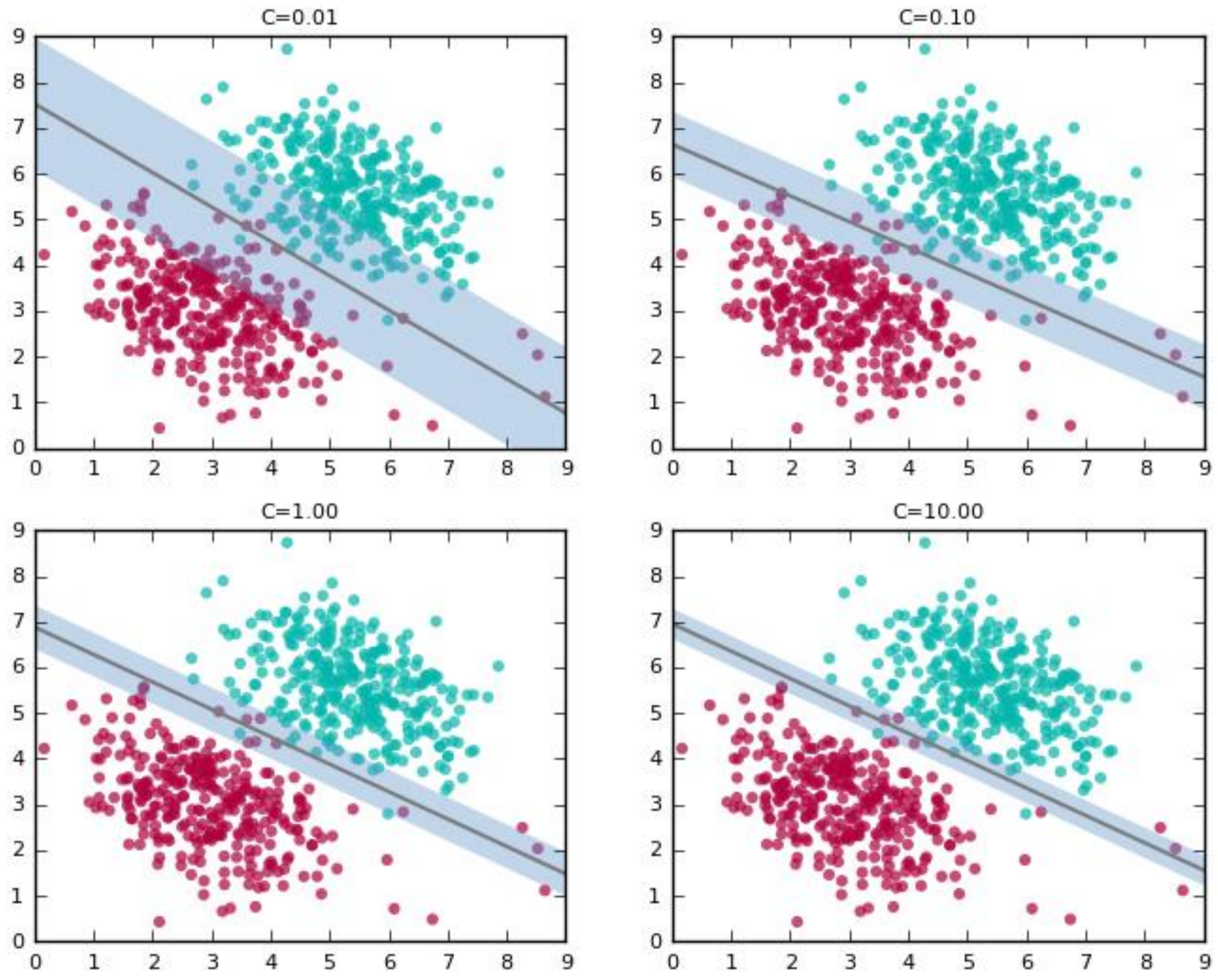
- Find lines that correctly classify the training data
- Among all such lines, pick the one that has the greatest distance to the points closest to it.

The closest points that identify this line are known as *support vectors*. The region they define around the line is known as the *margin.*

6. Support Vector Machines (SVM)

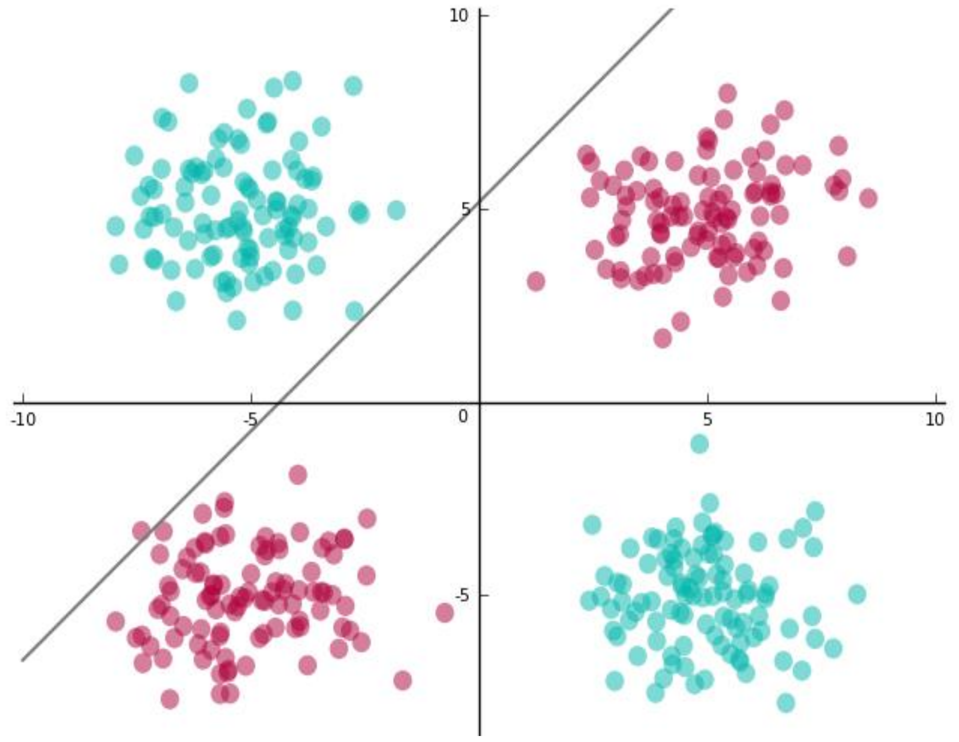
In practice, real data is messy and cannot be separated perfectly with a hyperplane. The constraint of maximizing the margin of the line that separates the classes must be relaxed. This is often called the **soft margin classifier**.
This change allows some points in the training data to violate the separating line.

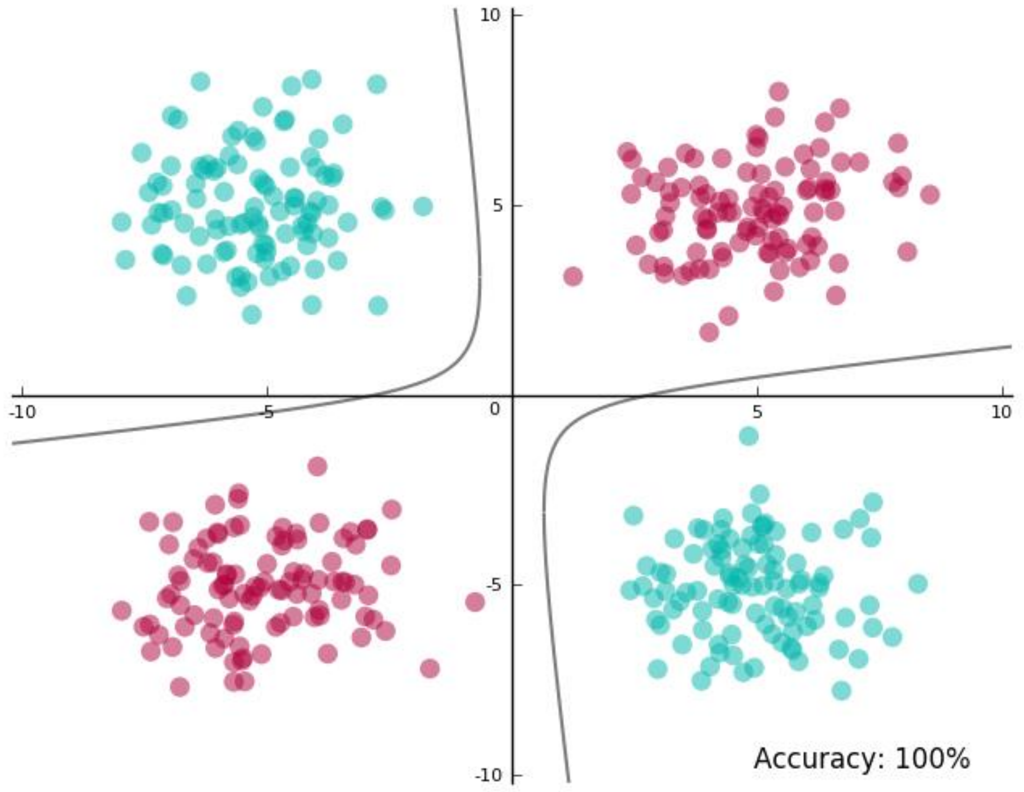
A tuning parameter C defines the amount of violation of the margin allowed. When C is large that means no violation and we are back to the inflexible **hard margin classifier**.



6. Support Vector Machines (SVM)

What about non linearly separable data?
SVM will make "projection" the data into a space where it is linearly separable and find a hyperplane in this new space.
(See: https://bit.ly/2vzndA8)





6. Support Vector Machines (SVM)

The secret of SVM that make projection of data is called Kernel function.
For **Linear Kernel SVM**, the equation to make prediction is expressed as follow, which is basically dot product between input (x) and each support vector (x_i):

$$f(x) = B0 + \sum_{i=1}^{n} B_i \times (x \times xi)$$

Where B0 and Bi coefficients must be estimated by learning algorithm.
The dot product in the above equation is called **Kernel function, K**. Other Kernel function can be used such as **Polynomial** and **RBF** (radial basis function) Kernel.

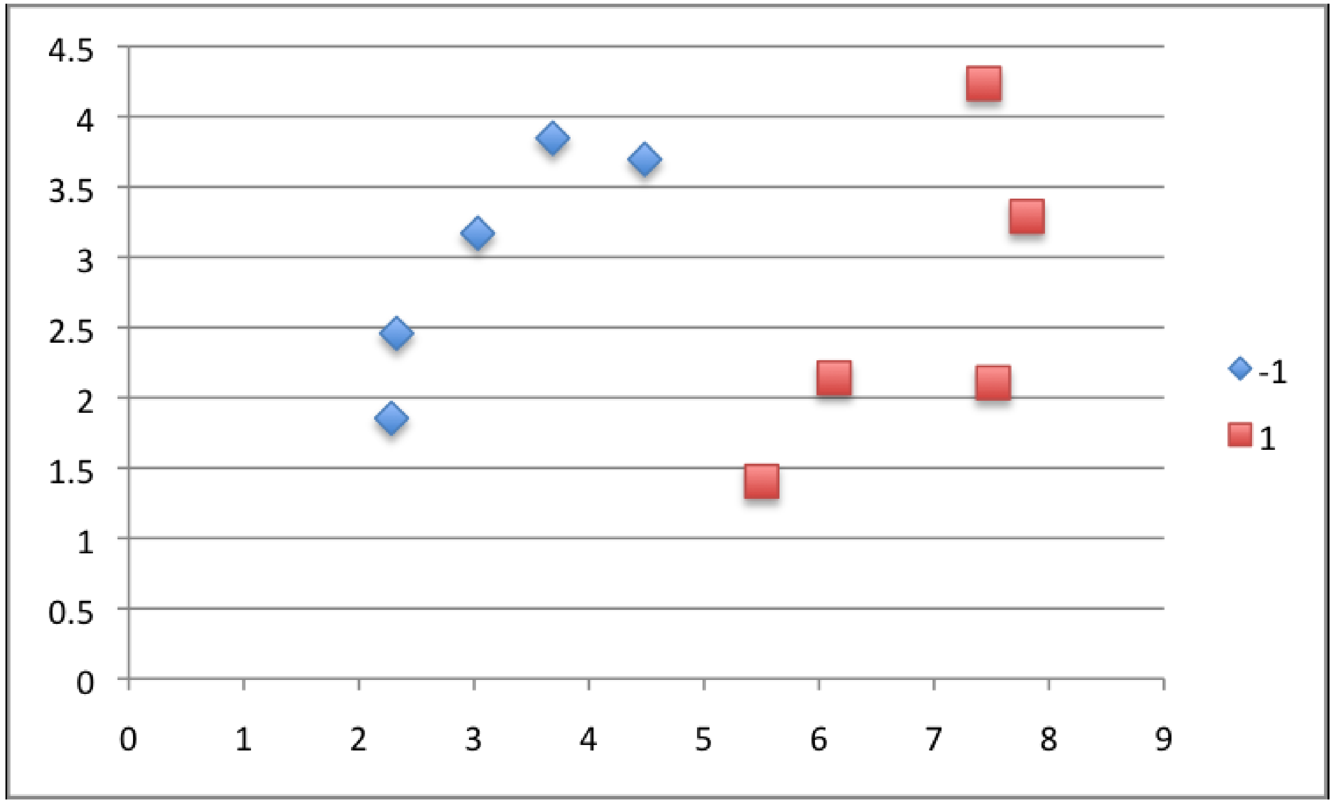
Respectively here, Linear Kernel, Polynomial Kernel and RBF Kernel:

$$K(x, x_i) = \sum (x \times x_i) \qquad K(x, x_i) = 1 + \sum (x \times x_i)^d \qquad K(x, x_i) = e^{-gamma \times \sum ((x - x_i)^2)}$$

6. Support Vector Machines - Tutorial

The dataset is purposely made so that the classes are linearly separable. This means that a straight line can be drawn to separate the classes.

X1	X2	Y
2.327868056	2.458016525	-1
3.032830419	3.170770366	-1
4.485465382	3.696728111	-1
3.684815246	3.846846973	-1
2.283558563	1.853215997	-1
7.807521179	3.290132136	1
6.132998136	2.140563087	1
7.514829366	2.107056961	1
5.502385039	1.404002608	1
7.432932365	4.236232628	1



6. Support Vector Machines - Tutorial

This part describes the form of the Linear Kernel SVM model and how it can be learned using a variant of the gradient descent optimization procedure.
Let’s form a Linear SVM that look like a line equation passing through origin (for the sake of tutorial simplicity):

$$(B1 \times X1) + (B2 \times X2) = 0$$

The coefficients (B1 and B2) update works as follows. First the output value is calculated as:

$$output = Y \times ((B1 \times X1) + (B2 \times X2))$$

Two different update procedures are used depending on the output value. If the output value is greater than 1 it suggests that the training pattern was not a support vector.

$$b = (1 - \frac{1}{t}) \times b$$

If the output is less than 1 then it is assumed that the training instance is a support vector and must be updated as:

$$b = (1 - \frac{1}{t}) \times b + \frac{1}{lambda \times t} \times (y \times x)$$

6. Support Vector Machines - Tutorial

Iteration#1

We set: B1=0 and B2=0
and iteration number t=1

First data instance:
*X*1 = 2.327868056, *X*2 = 2.458016525, *Y* = −1.

$$output = Y \times ((B1 \times X1) + (B2 \times X2))$$
$$output = -1 \times ((0.0 \times 2.327868056) + (0.0 \times 2.458016525))$$
$$output = 0.0$$

The output is less than 1.0, therefore:

$$b = (1 - \frac{1}{t}) \times b + \frac{1}{lambda \times t} \times (y \times x)$$
$$B1 = (1 - \frac{1}{1}) \times 0.0 + \frac{1}{0.45 \times 1} \times (-1 \times 2.327868056)$$
$$B1 = -5.173040124$$

$$B2 = (1 - \frac{1}{1}) \times 0.0 + \frac{1}{0.45 \times 1} \times (-1 \times 2.458016525)$$
$$B2 = -5.462258944$$

6. Support Vector Machines - Tutorial

Repeat this process for the rest of the dataset. One pass through the dataset is called an epoch. After 15 epochs that we achieve an accuracy of 100% on the training data. You should arrive at final values for the coefficients that :

$$B1 = 0.552391765$$
$$B2 = -0.724533592$$

The form of the learned hyperplane (a straight line in this case) is therefore:

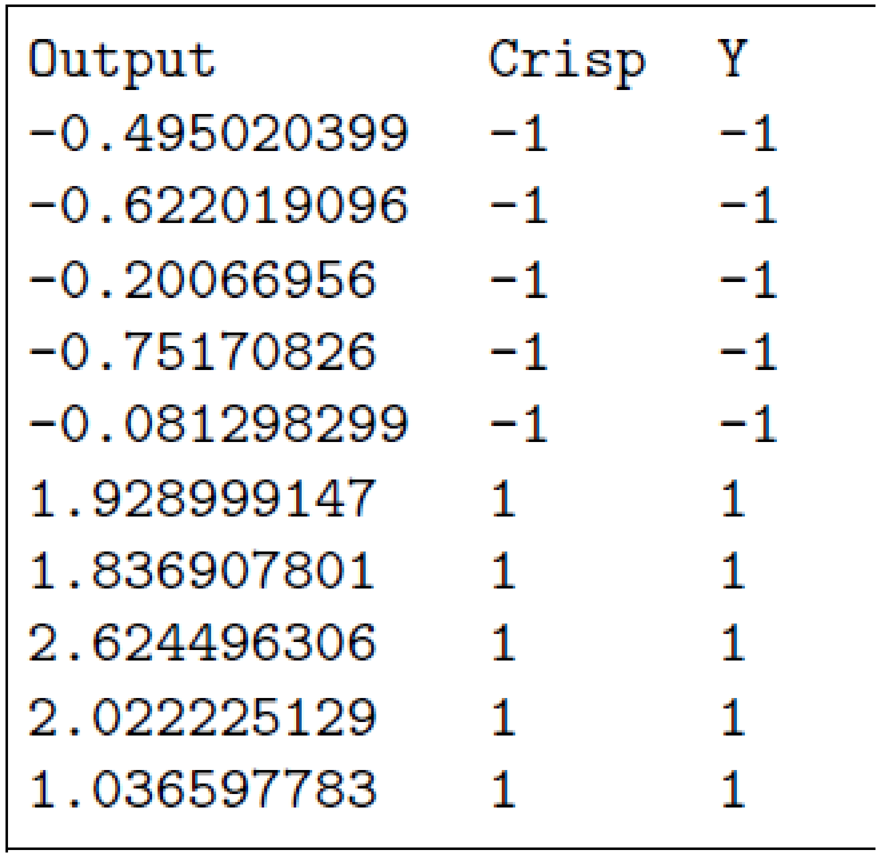
$$(0.552391765 \times X1) + (-0.724533592 \times X2) = 0$$

Then, prediction can be made using:

$$output = (B1 \times X1) + (B2 \times X2)$$
$$Y = -1 \text{ IF } output < 0$$
$$Y = +1 \text{ IF } output > 0$$

6. Support Vector Machines - Tutorial

Using the above coefficients and these prediction rules, we can make a prediction for each instance in the training dataset:

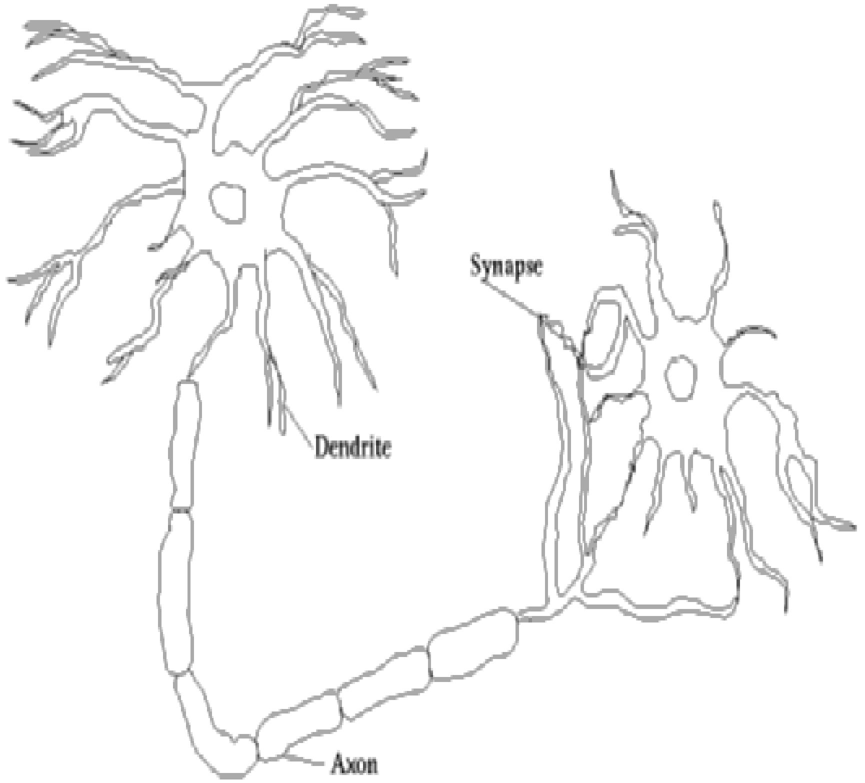


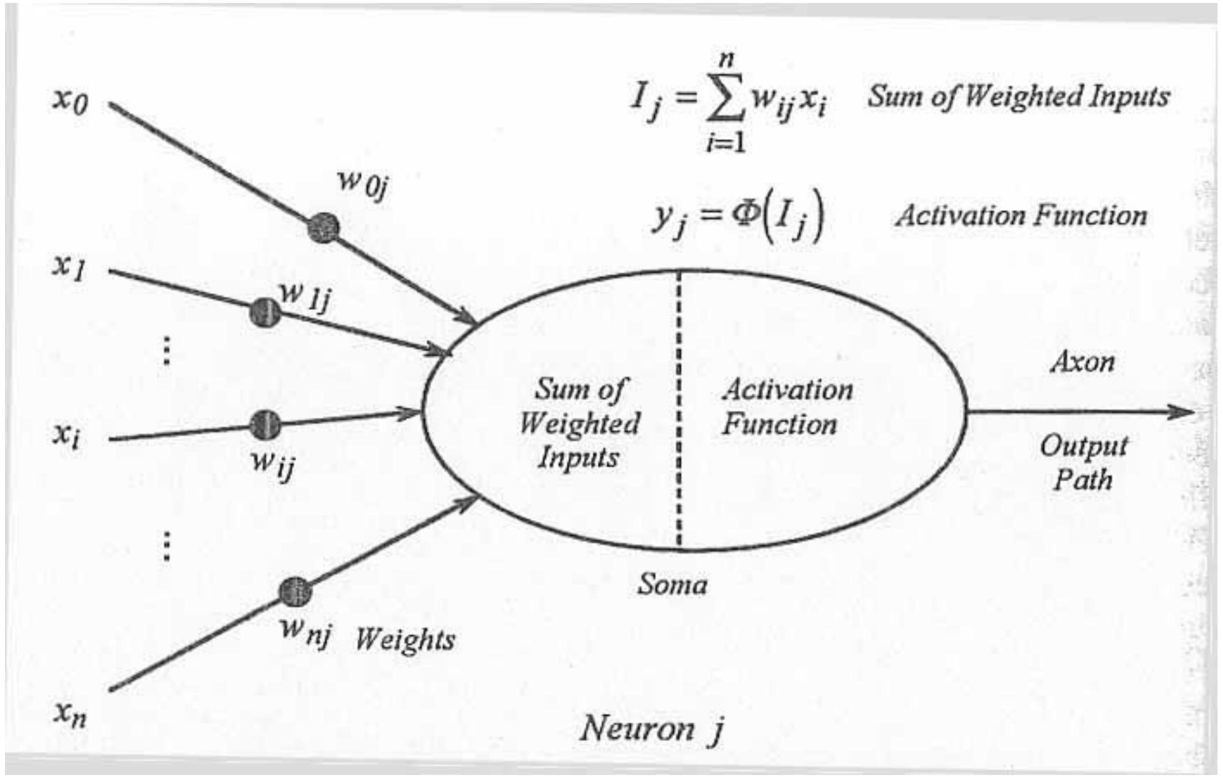
Output	Crisp	Y
-0.495020399	-1	-1
-0.622019096	-1	-1
-0.20066956	-1	-1
-0.75170826	-1	-1
-0.081298299	-1	-1
1.928999147	1	1
1.836907801	1	1
2.624496306	1	1
2.022225129	1	1
1.036597783	1	1

Comparing the prediction (Crisp value) to the actual output (Y), we can see that our model has achieved 100% accuracy.

7. Artificial Neural Networks (ANN)

Artificial neuron is a model whose components have direct analogy to component of an actual neuron.

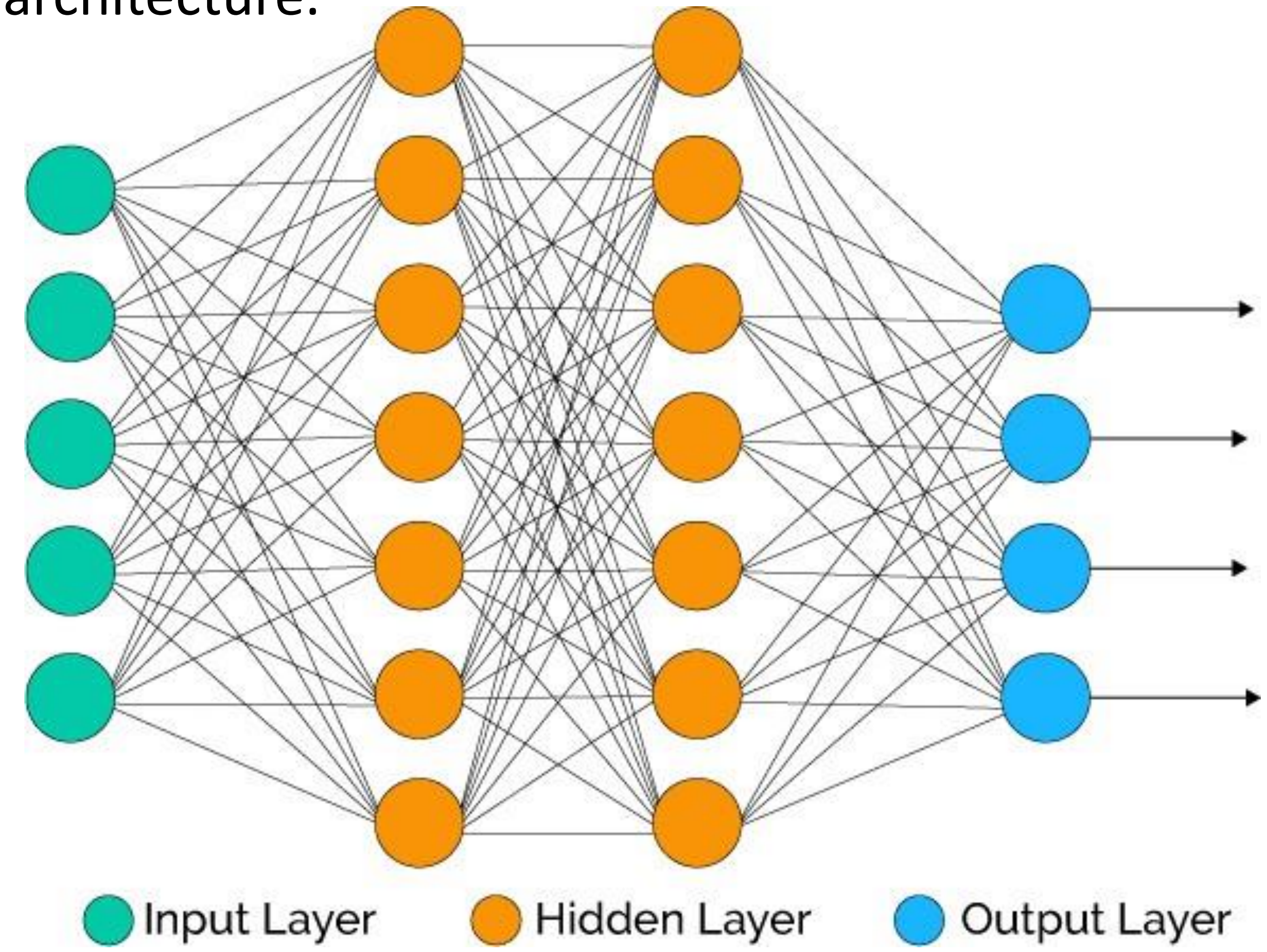




The human brain contains about 10 billion nerve cells (neurons).

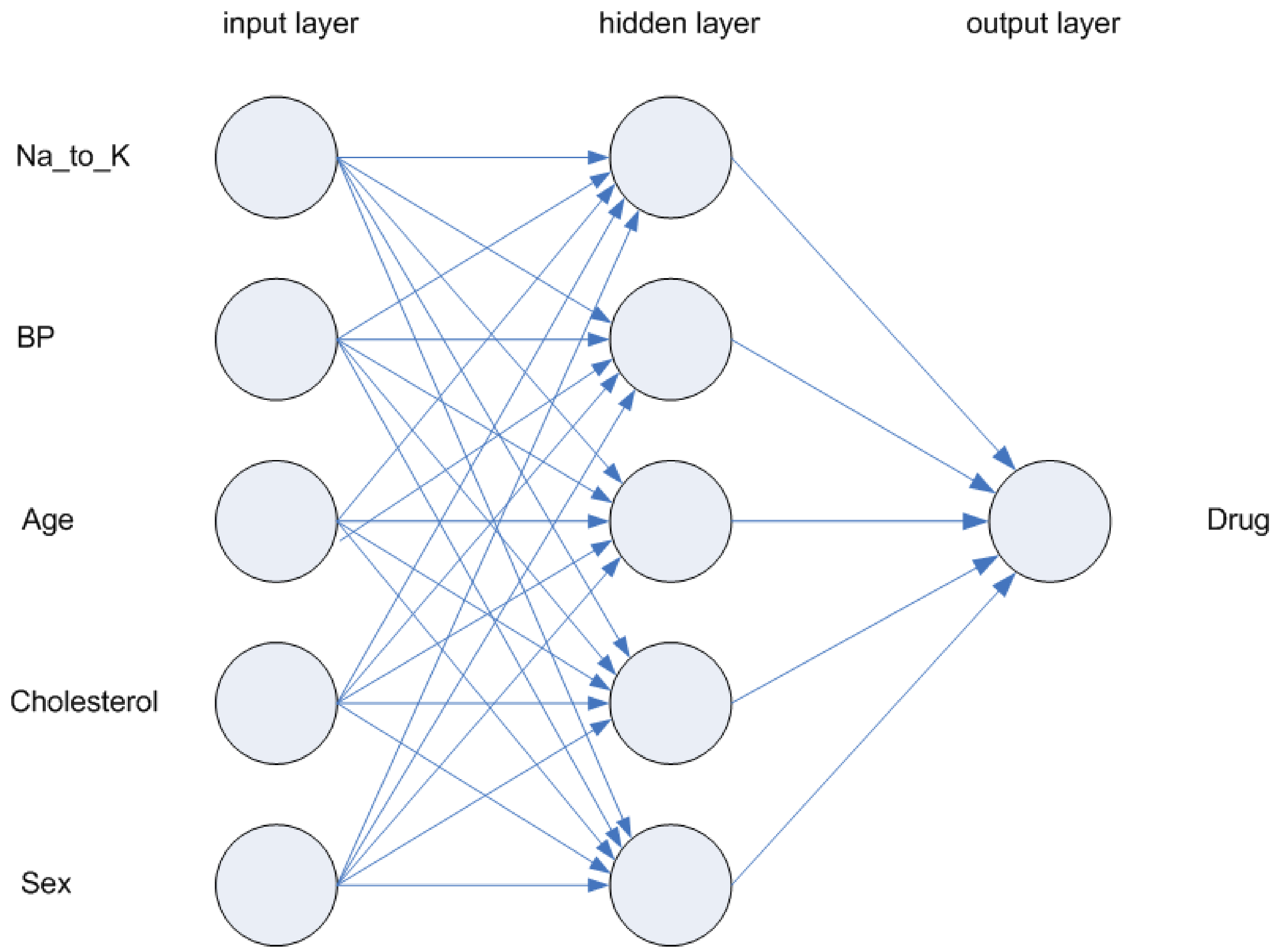
7. Artificial Neural Networks (ANN)

An ANN consists of many artificial neurons that are linked together according to a specific network architecture.



7. Artificial Neural Networks (ANN)

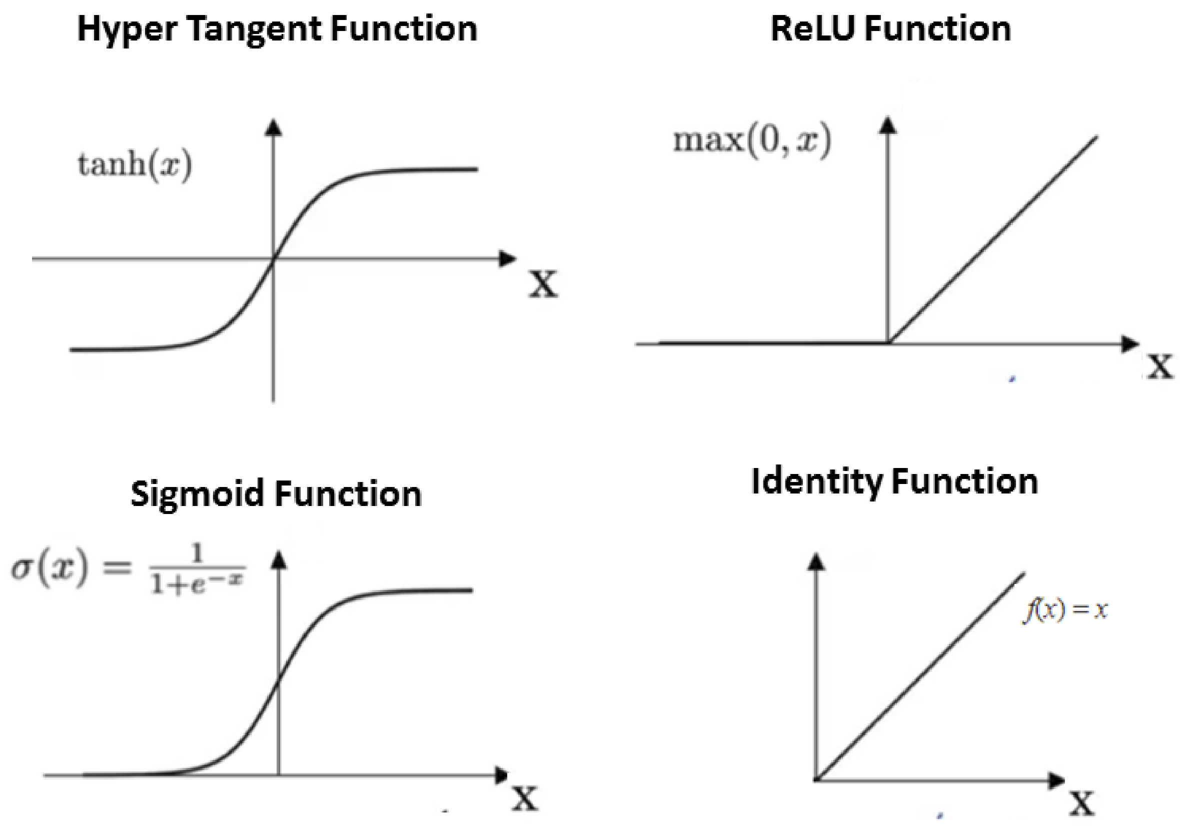
Example of ANN model is here. The task of ANN training/learning is to update the weights iteratively using a certain training algorithm.



7. Artificial Neural Networks (ANN)

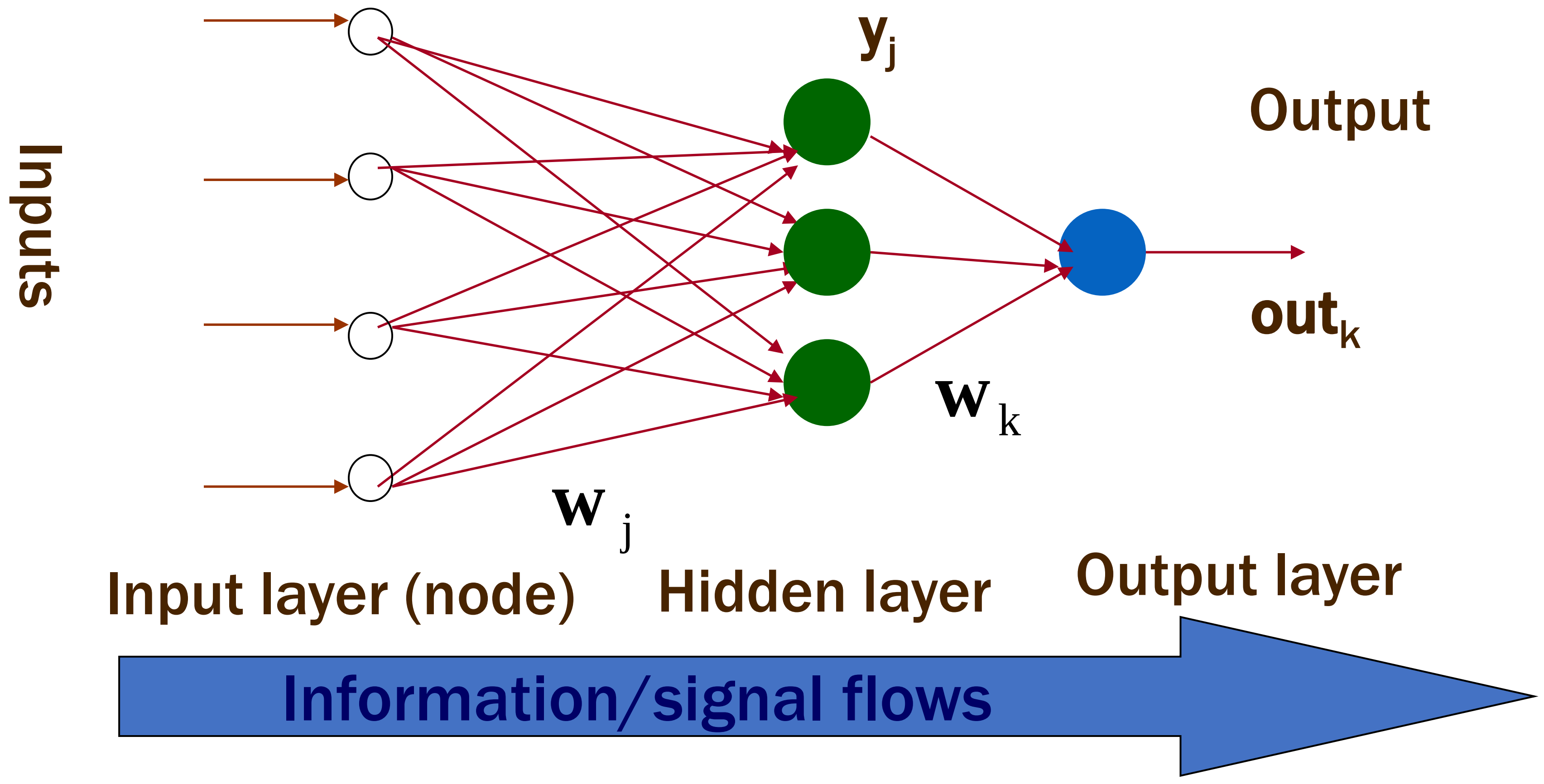
There are several possible activation functions:

- Linear function (Identity)
- Signum function
- Step function
- Ramp function
- Unipolar sigmoid function
- Bipolar sigmoid function
- ReLu function
- Etc.



7. Artificial Neural Networks -Tutorial

Multi layer Feed-forward Network (MFN)



There may be several layers in hidden layer

7. Artificial Neural Networks -Tutorial

Mathematical expression of MFN

- Output of the hidden layer

$$\mathbf{y}_j = \mathbf{\Phi}\left[\mathbf{w}_j^T \mathbf{x}\right]$$

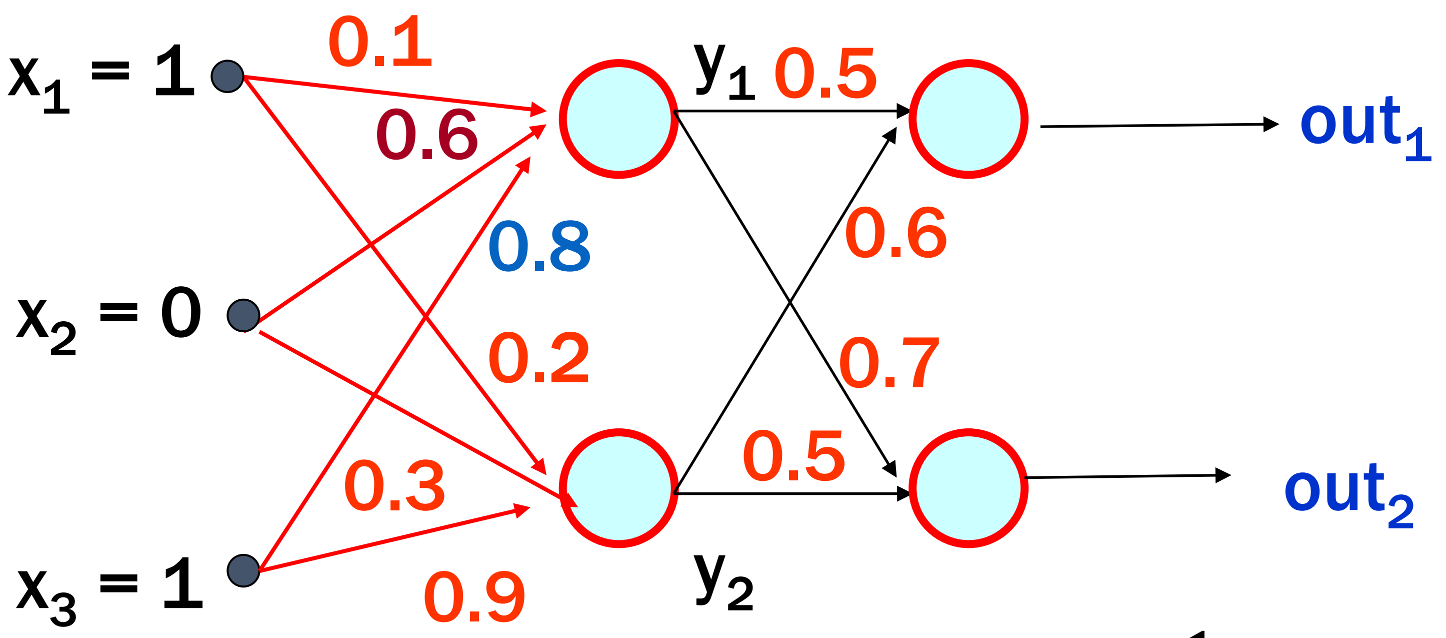
- Output of the output layer (output of the NN)

$$out_k = \mathbf{\Phi}\left[\mathbf{w}_k^T \mathbf{y}_j\right]$$

Outputs of the hidden layer become the input of the next hidden layer or the output layer

7. Artificial Neural Networks -Tutorial

Example: Calculation of output in MFN



Sigmoid ctivation function for all neurons:

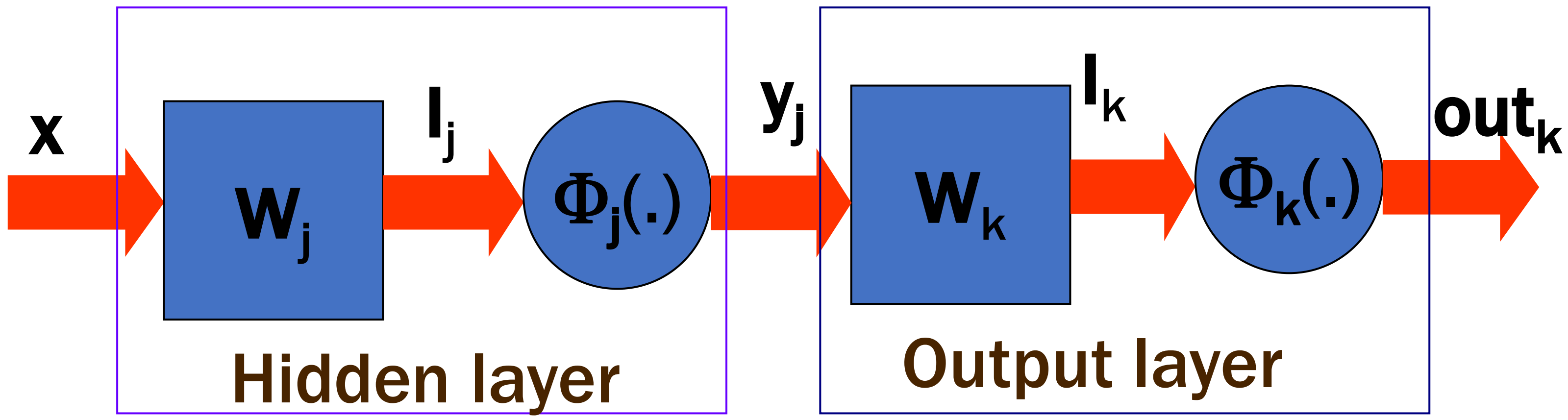
$$\Phi(I) = \frac{1}{1+e^{-\lambda I}}, \lambda = 1$$

What is out_k ?

CCP-ML DrMIS

7. Artificial Neural Networks -Tutorial

Block Diagram Representation



From the NN diagram

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Input data

$$\mathbf{w}_j = \begin{bmatrix} 0.1 & 0.2 \\ 0.6 & 0.3 \\ 0.8 & 0.9 \end{bmatrix}$$

Weight at hidden layer

$$\mathbf{w}_k = \begin{bmatrix} 0.5 & 0.7 \\ 0.6 & 0.5 \end{bmatrix}$$

Weight at output layer

7. Artificial Neural Networks -Tutorial

For hidden layer,

$$\mathbf{I}_j = \mathbf{w}_j^T \mathbf{x} = \begin{bmatrix} 0.1 & 0.6 & 0.8 \\ 0.2 & 0.3 & 0.9 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 1.1 \end{bmatrix}$$

$$\mathbf{y}_j = \begin{bmatrix} \Phi(0.9) \\ \Phi(1.1) \end{bmatrix} = \begin{bmatrix} 0.711 \\ 0.750 \end{bmatrix}$$

For output layer,

$$\mathbf{I}_k = \mathbf{w}_k^T \mathbf{y}_j = \begin{bmatrix} 0.806 \\ 0.873 \end{bmatrix} \qquad out_k = \begin{bmatrix} \Phi(0.806) \\ \Phi(0.873) \end{bmatrix} = \begin{bmatrix} 0.691 \\ 0.705 \end{bmatrix}$$

7. Artificial Neural Networks -Tutorial

Back-propagation Training Algorithm

-Weight update of output layer:

$$\mathbf{w}_k^{T,new} = \mathbf{w}_k^{T,old} + \eta \boldsymbol{\delta}_o \mathbf{y}_j^T$$

$$\boldsymbol{\delta}_o = \left(\mathbf{d} - out\right) \cdot \Phi_k{}'(\mathbf{I}_k)$$

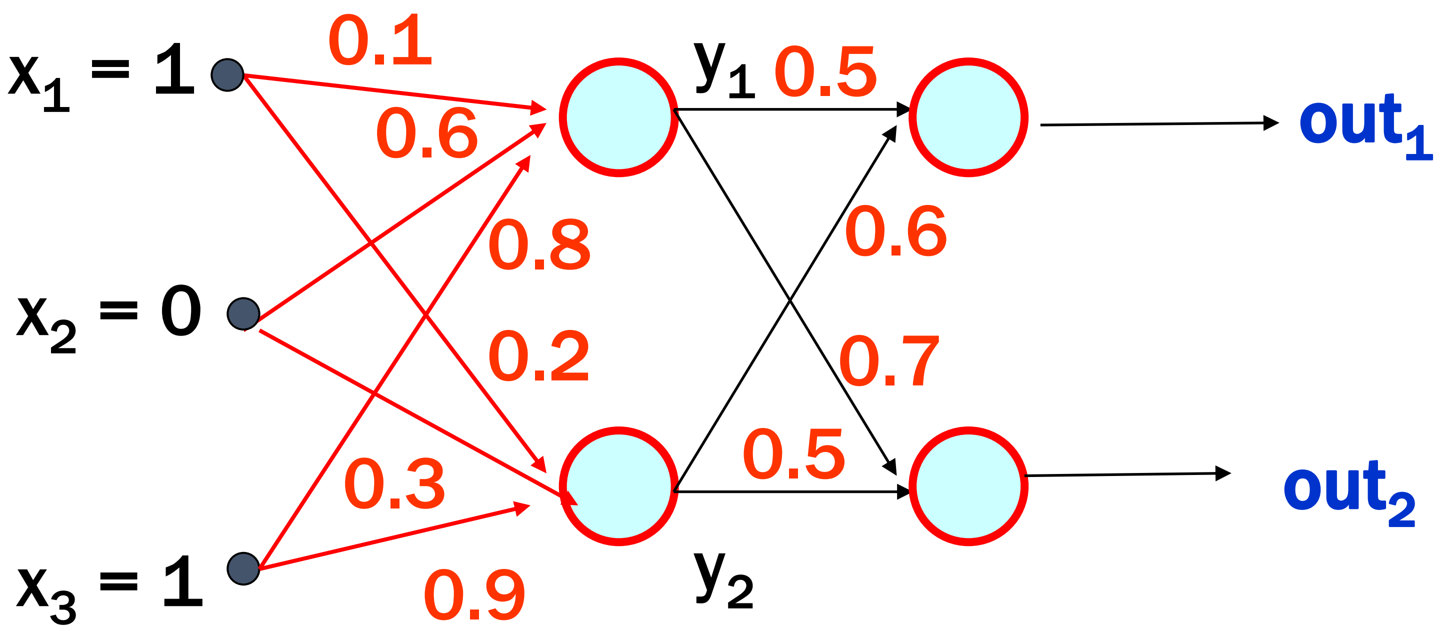
-Weight update of hidden layer:

$$\mathbf{w}_j^{T,new} = \mathbf{w}_j^{T,old} + \eta \boldsymbol{\delta}_y \mathbf{x}^T$$

$$\boldsymbol{\delta}_y = \left(\mathbf{w}_k \boldsymbol{\delta}_o\right) \cdot \Phi'_j(I_j)$$

7. Artificial Neural Networks -Tutorial

Example: Multi layer NN trained with Back-propagation Algorithm



$\eta = 0.5$ $$\Phi(I) = \frac{1}{1+e^{-\lambda I}}; \lambda = 1$$ (for all neurons)

The desired output d.

Training the NN until for 1 iteration.

$$d = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

7. Artificial Neural Networks -Tutorial

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{w}_j = \begin{bmatrix} 0.1 & 0.2 \\ 0.6 & 0.3 \\ 0.8 & 0.9 \end{bmatrix} \quad \mathbf{w}_{\mathrm{k}} = \begin{bmatrix} 0.5 & 0.7 \\ 0.6 & 0.5 \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Iteration#1

Forward propagation

$$\mathbf{I}_j = \mathbf{w}_j^T \mathbf{x} = \begin{bmatrix} 0.1 & 0.6 & 0.8 \\ 0.2 & 0.3 & 0.9 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 1.1 \end{bmatrix}$$

$$\mathbf{y}_j = \begin{bmatrix} \Phi_j(0.9) \\ \Phi_j(1.1) \end{bmatrix} = \begin{bmatrix} 0.711 \\ 0.750 \end{bmatrix}$$

7. Artificial Neural Networks -Tutorial

$$\mathbf{I}_k = \mathbf{w}_k^T \mathbf{y}_j = \begin{bmatrix} 0.806 \\ 0.873 \end{bmatrix} \qquad out_k = \begin{bmatrix} \Phi(0.806) \\ \Phi(0.873) \end{bmatrix} = \begin{bmatrix} 0.691 \\ 0.705 \end{bmatrix}$$

Back propagation

$$\boldsymbol{\delta}_{out} = (\mathbf{d} - out) \cdot \boldsymbol{\Phi}_k{}'(\mathbf{I}_k) = (\mathbf{d} - out) \cdot out \cdot (1 - out) = \begin{bmatrix} 0.066 \\ 0.061 \end{bmatrix}$$

$$\boldsymbol{\delta}_y = (\mathbf{w}_k \boldsymbol{\delta}_{out}) \cdot \Phi_j{}'(\mathbf{I}_j) = (\mathbf{w}_k \boldsymbol{\delta}_{out}) \cdot \mathbf{y}_j \cdot (1 - \mathbf{y}_j) = \begin{bmatrix} 0.016 \\ 0.013 \end{bmatrix}$$

7. Artificial Neural Networks -Tutorial

Weight adjustment of the output layer

$$\mathbf{w}^T_{k,new} = \mathbf{w}^T_{k,old} + \eta \boldsymbol{\delta}_{out} \mathbf{y}^T_j$$

$$\mathbf{w}^T_{k,new} = \begin{bmatrix} 0.5 & 0.6 \\ 0.7 & 0.5 \end{bmatrix} + 0.5 \begin{bmatrix} 0.066 \\ 0.061 \end{bmatrix} \begin{bmatrix} 0.711 & 0.750 \end{bmatrix}$$

$$\mathbf{w}^T_{k,new} = \begin{bmatrix} 0.523 & 0.625 \\ 0.722 & 0.523 \end{bmatrix}$$

7. Artificial Neural Networks -Tutorial

Weight adjustment of the hidden layer

$$\mathbf{w}^T_{j,new} = \mathbf{w}^T_{j,old} + \eta \boldsymbol{\delta}_y \mathbf{x}^T$$

$$\mathbf{w}^T_{j,new} = \begin{bmatrix} 0.1 & 0.6 & 0.8 \\ 0.2 & 0.3 & 0.9 \end{bmatrix} + 0.5 \begin{bmatrix} 0.016 \\ 0.013 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{w}^T_{j,new} = \begin{bmatrix} 0.108 & 0.600 & 0.808 \\ 0.207 & 0.300 & 0.907 \end{bmatrix}$$

7. Artificial Neural Networks -Tutorial

Iteration #1 has been completed.

The new output after iteration#1 →

$$out = \begin{bmatrix} 0.699 \\ 0.713 \end{bmatrix} \Longrightarrow \mathrm{d} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Iteration #2

?

7. Artificial Neural Networks -Tutorial

Iteration#2

$$w^T_{k,new} = \begin{bmatrix} 0.546 & 0.648 \\ 0.743 & 0.545 \end{bmatrix}$$

$$w^T_{j,new} = \begin{bmatrix} 0.114 & 0.600 & 0.814 \\ 0.214 & 0.300 & 0.914 \end{bmatrix}$$

$$out = \begin{bmatrix} 0.707 \\ 0.720 \end{bmatrix} \Longrightarrow \mathrm{d} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

7. Artificial Neural Networks -Tutorial

Iteration#2000.

$$w_{k,new}^{T} = \begin{bmatrix} 2.149 & 2.295 \\ 2.302 & 2.1454 \end{bmatrix}$$

$$w_{j,new}^{T} = \begin{bmatrix} 0.806 & 0.600 & 1.506 \\ 0.852 & 0.300 & 1.552 \end{bmatrix}$$

$$out = \begin{bmatrix} 0.983 \\ 0.983 \end{bmatrix} \Longrightarrow d = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

8. Ensemble Algorithms

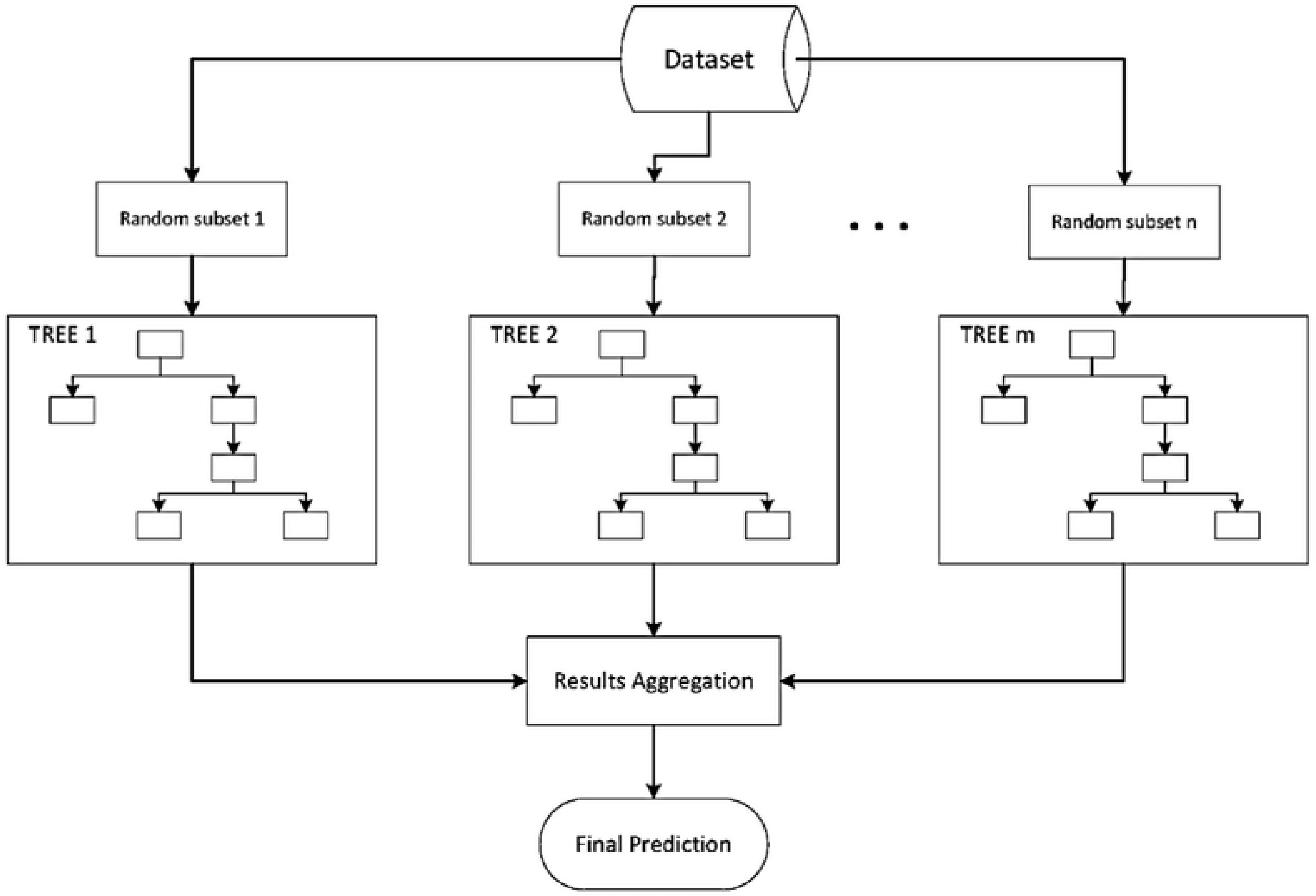
A powerful and more advanced type of machine learning algorithm are ensemble algorithms. These are techniques that combine the predictions from multiple models in order to provide more accurate predictions.

Among the available ensemble algorithms are: bagging (or bootstrap aggregation) random forest and boosting ensemble such as AdaBoost.

We will see Bagged Decision Tree Tutorial in the next pages.

8. Ensemble Algorithms – Tutorial Bagged Decision Tree

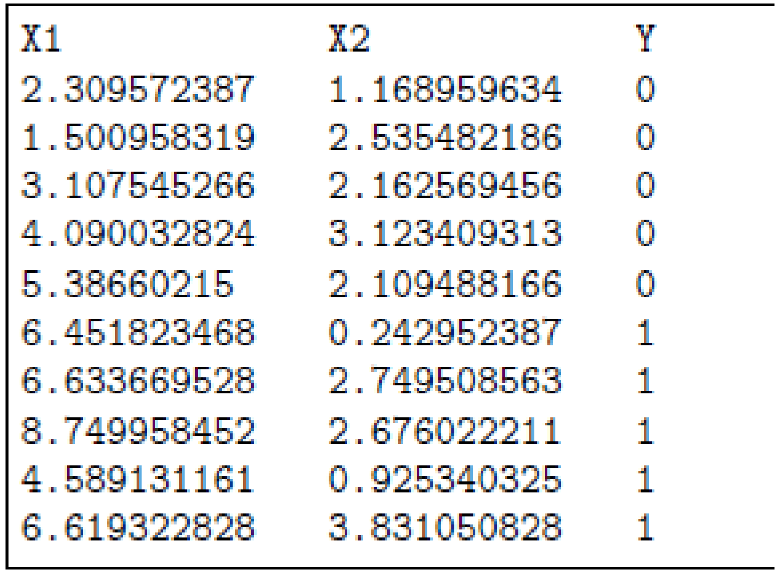
Bagging is a simple ensemble method that almost always results in a lift in performance over the baseline models.



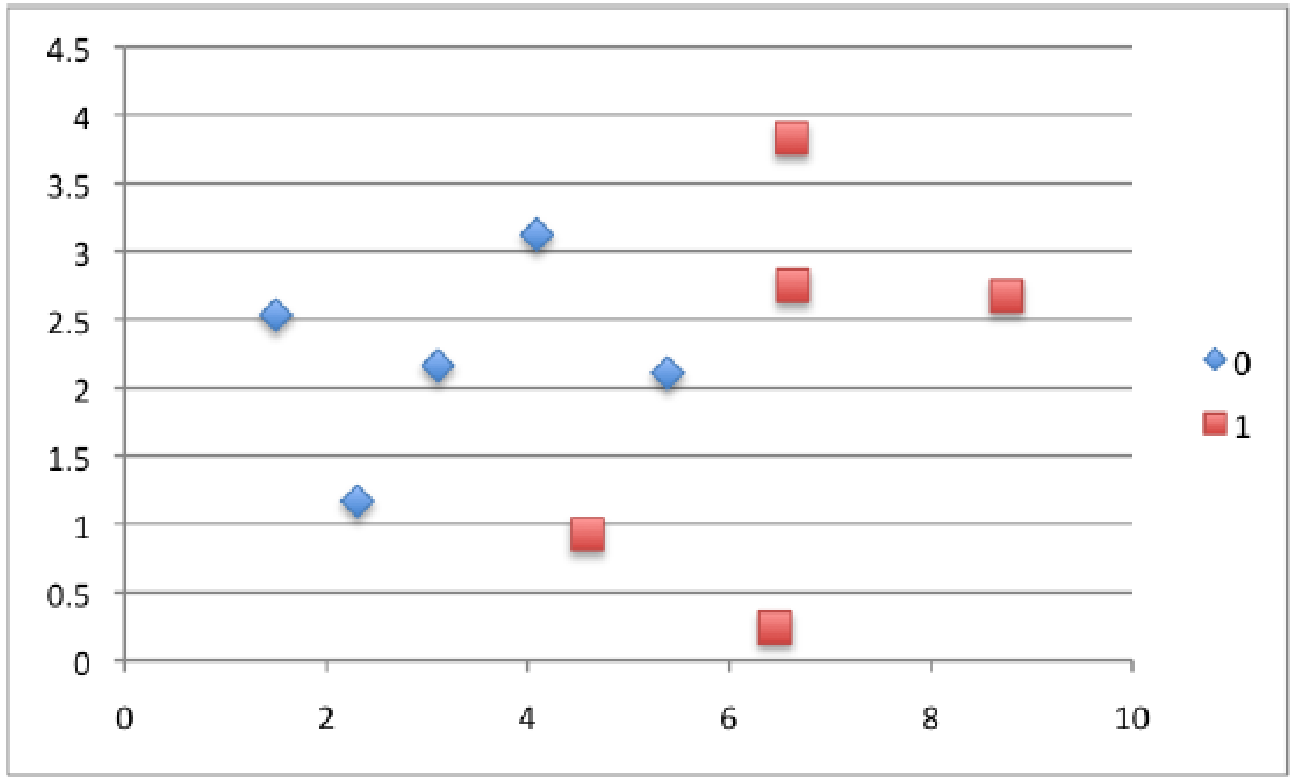
Given a Dataset, bootstrapped subsamples are pulled. A Decision Tree is formed on each bootstrapped sample. The results of each tree are aggregated to yield the strongest, most accurate predictor.

8. Ensemble Algorithms – Tutorial Bagged Decision Tree

In this tutorial we will use dataset with two input variables (X1 and X2) and one output variable (Y). The input variables are real-valued random numbers drawn from a Gaussian distribution. The output variable has two values, making the problem a binary classification problem.



X1	X2	Y
2.309572387	1.168959634	0
1.500958319	2.535482186	0
3.107545266	2.162569456	0
4.090032824	3.123409313	0
5.38660215	2.109488166	0
6.451823468	0.242952387	1
6.633669528	2.749508563	1
8.749958452	2.676022211	1
4.589131161	0.925340325	1
6.619322828	3.831050828	1



8. Ensemble Algorithms – Tutorial Bagged Decision Tree

Bagging (or Bootstrap Aggregation) works by taking a subsample of your training dataset (with replacement) and creating a model, often a decision tree because they have high variance. The process is repeated creating as many trees as you desire. Later when making predictions for new data, the outputs from each tree are combined by taking the average.

The interesting part of bagged decision trees is how they are combined to make ensemble predictions. With this in mind, we will arrange 3 decision trees from the training data. A decision tree with one split point is called a decision stump.

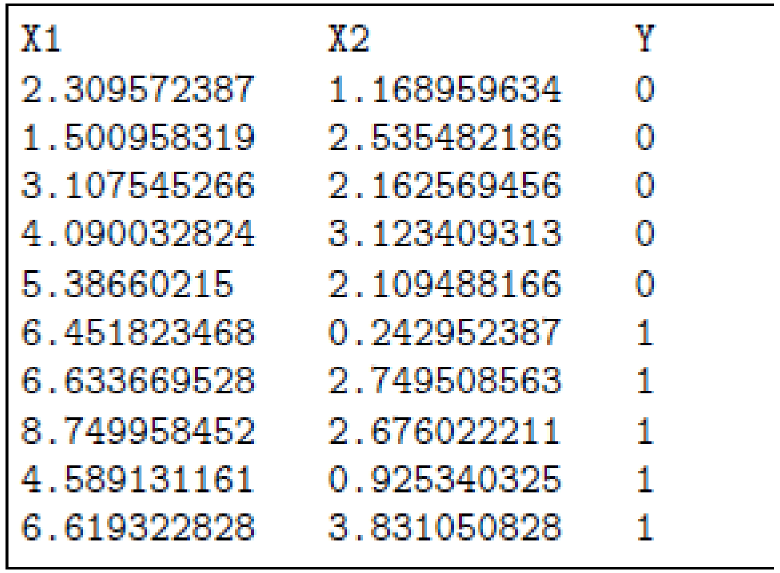
In this example we are only using 3 models, often you will create tens, hundreds or even thousands. For demonstration purpose, the split points is chosen manually for each are as follows:

$$Model1 : X1 \leq 5.38660215$$
$$Model2 : X1 \leq 4.090032824$$
$$Model3 : X2 \leq 0.925340325$$

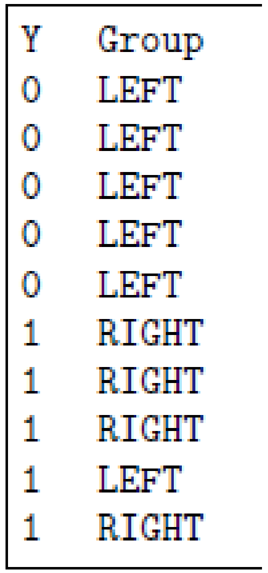
8. Ensemble Algorithms – Tutorial Bagged Decision Tree

Decision Stump Model 1

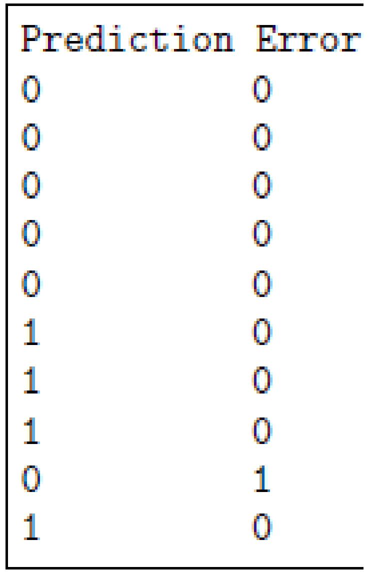
The split point for the first model is X1 ≤ 5.38660215. Using this split point we can separate the training data into two groups:



X1	X2	Y
2.309572387	1.168959634	0
1.500958319	2.535482186	0
3.107545266	2.162569456	0
4.090032824	3.123409313	0
5.38660215	2.109488166	0
6.451823468	0.242952387	1
6.633669528	2.749508563	1
8.749958452	2.676022211	1
4.589131161	0.925340325	1
6.619322828	3.831050828	1



Y	Group
0	LEFT
0	LEFT
0	LEFT
0	LEFT
0	LEFT
1	RIGHT
1	RIGHT
1	RIGHT
1	LEFT
1	RIGHT



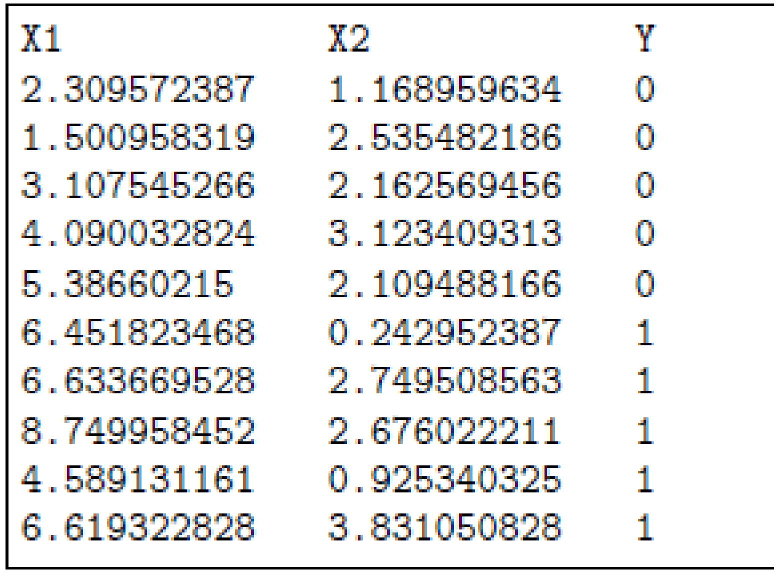
Prediction	Error
0	0
0	0
0	0
0	0
0	0
1	0
1	0
1	0
0	1
1	0

Suppose that LEFT group is for class 0 and RIGHT group is for class 1, therefore the accuracy of this MODEL 1 is 90% (got 1 error of 10).

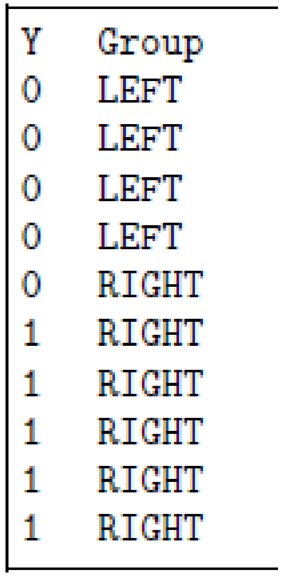
8. Ensemble Algorithms – Tutorial Bagged Decision Tree

Decision Stump Model 2

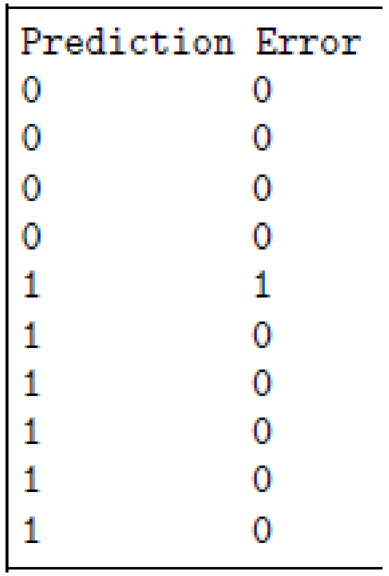
The split point for the first model is X1 ≤ 4.0900328. Using this split point we can separate the training data into two groups:



X1	X2	Y
2.309572387	1.168959634	0
1.500958319	2.535482186	0
3.107545266	2.162569456	0
4.090032824	3.123409313	0
5.38660215	2.109488166	0
6.451823468	0.242952387	1
6.633669528	2.749508563	1
8.749958452	2.676022211	1
4.589131161	0.925340325	1
6.619322828	3.831050828	1



Y	Group
0	LEFT
0	LEFT
0	LEFT
0	LEFT
0	RIGHT
1	RIGHT
1	RIGHT
1	RIGHT
1	RIGHT
1	RIGHT



Prediction	Error
0	0
0	0
0	0
0	0
1	1
1	0
1	0
1	0
1	0
1	0

Again suppose that LEFT group is for class 0 and RIGHT group is for class 1, therefore the accuracy of this MODEL 1 is 90% (got 1 error of 10).

8. Ensemble Algorithms – Tutorial
Bagged Decision Tree

Decision Stump Model 3

The split point for the first model is X2 ≤ 0.925340325. Using this split point we can separate the training data into two groups:

X1	X2	Y
2.309572387	1.168959634	0
1.500958319	2.535482186	0
3.107545266	2.162569456	0
4.090032824	3.123409313	0
5.38660215	2.109488166	0
6.451823468	0.242952387	1
6.633669528	2.749508563	1
8.749958452	2.676022211	1
4.589131161	0.925340325	1
6.619322828	3.831050828	1

Y	Group
0	RIGHT
0	RIGHT
0	RIGHT
0	RIGHT
0	RIGHT
1	LEFT
1	RIGHT
1	RIGHT
1	LEFT
1	RIGHT

Prediction	Error
0	0
0	0
0	0
0	0
0	0
1	0
0	1
0	1
1	0
0	1

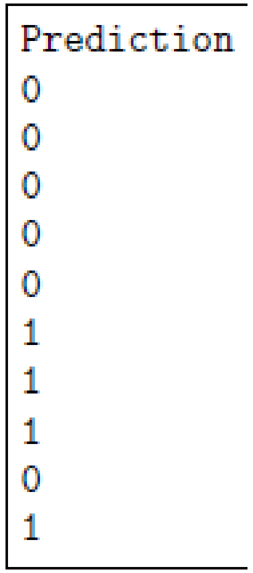
Suppose that RIGHT group is for class 0 and LEFT group is for class 1, therefore the accuracy of this MODEL 1 is 70% (got 3 error of 10).

8. Ensemble Algorithms – Tutorial Bagged Decision Tree

Final Prediction

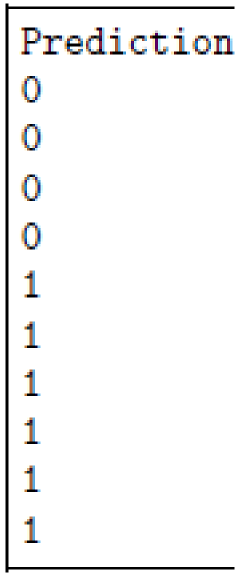
Now that we have predictions from the bootstrap models we can aggregate the predictions into an ensemble prediction. We can do that by taking the mode of each models prediction for each training instance.
In this case, the most common class value predicted. Using this simple procedure we get the following predictions:

Model 1



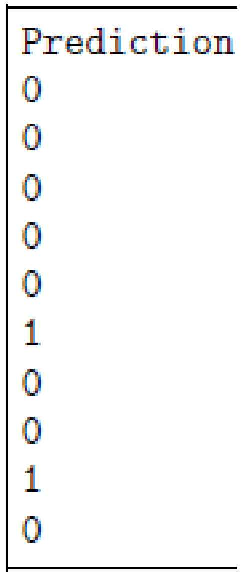
Prediction
0
0
0
0
0
1
1
1
0
1

Model 2



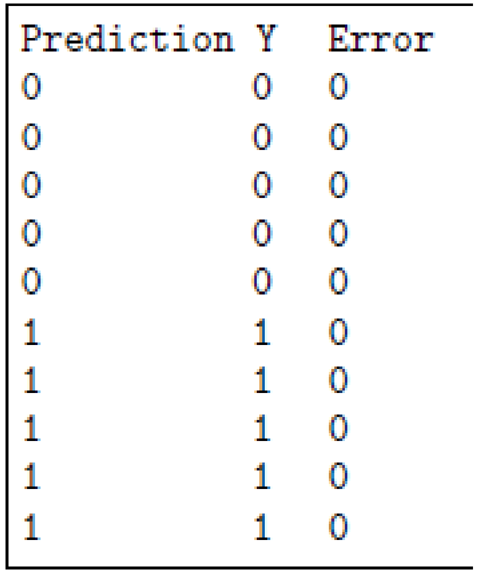
Prediction
0
0
0
0
1
1
1
1
1
1

Model 3



Prediction
0
0
0
0
0
1
0
0
1
0

FINAL



Prediction	Y	Error
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0

You can see that all 10 training instances were classified correctly at 100% accuracy.

9. Principal Component Analysis (PCA)

- A data exploratory technique mainly used to reduce the dimensionality of the dataset
- PCA can be used to:
  - Reduce number of dimensions in data
  - Find patterns in high-dimensional data
  - Visualize data of high dimensionality
- Example of applications:
  - Face recognition
  - Image compression
  - Gene expression analysis
  - Etc.

9. Principal Component Analysis (PCA)

Example: Data Visualization

- Given 53 blood and urine samples (features) from 65 people.
- How can we visualize the measurements?

9. Principal Component Analysis (PCA)

- Matrix format (65x53)

Instances

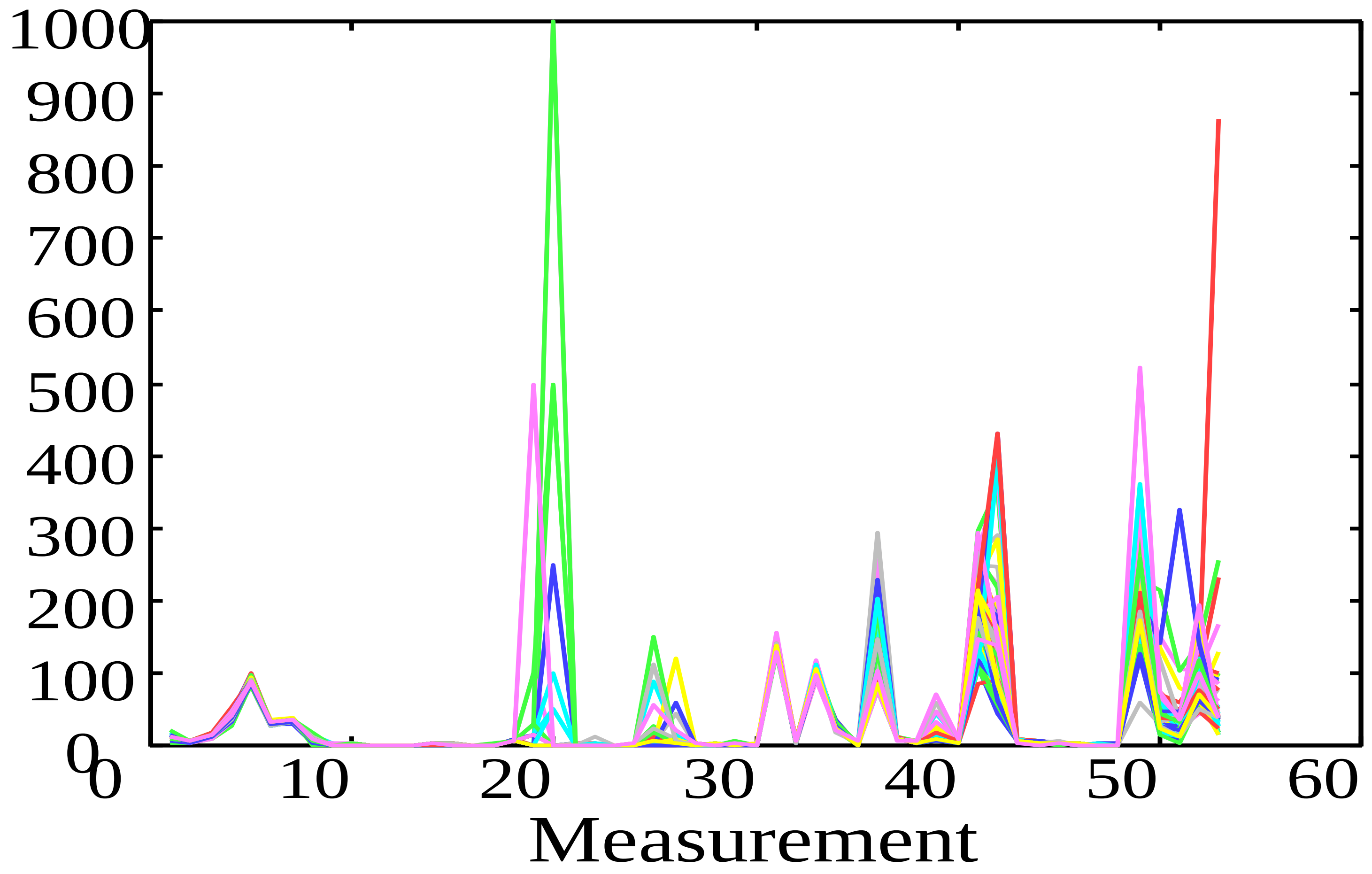
	H-WBC	H-RBC	H-Hgb	H-Hct	H-MCV	H-MCH	H-MCHC
A1	8.0000	4.8200	14.1000	41.0000	85.0000	29.0000	34.0000
A2	7.3000	5.0200	14.7000	43.0000	86.0000	29.0000	34.0000
A3	4.3000	4.4800	14.1000	41.0000	91.0000	32.0000	35.0000
A4	7.5000	4.4700	14.9000	45.0000	101.0000	33.0000	33.0000
A5	7.3000	5.5200	15.4000	46.0000	84.0000	28.0000	33.0000
A6	6.9000	4.8600	16.0000	47.0000	97.0000	33.0000	34.0000
A7	7.8000	4.6800	14.7000	43.0000	92.0000	31.0000	34.0000
A8	8.6000	4.8200	15.8000	42.0000	88.0000	33.0000	37.0000
A9	5.1000	4.7100	14.0000	43.0000	92.0000	30.0000	32.0000

Features

Difficult to see the correlations between the features (high dimensionality).

9. Principal Component Analysis (PCA)

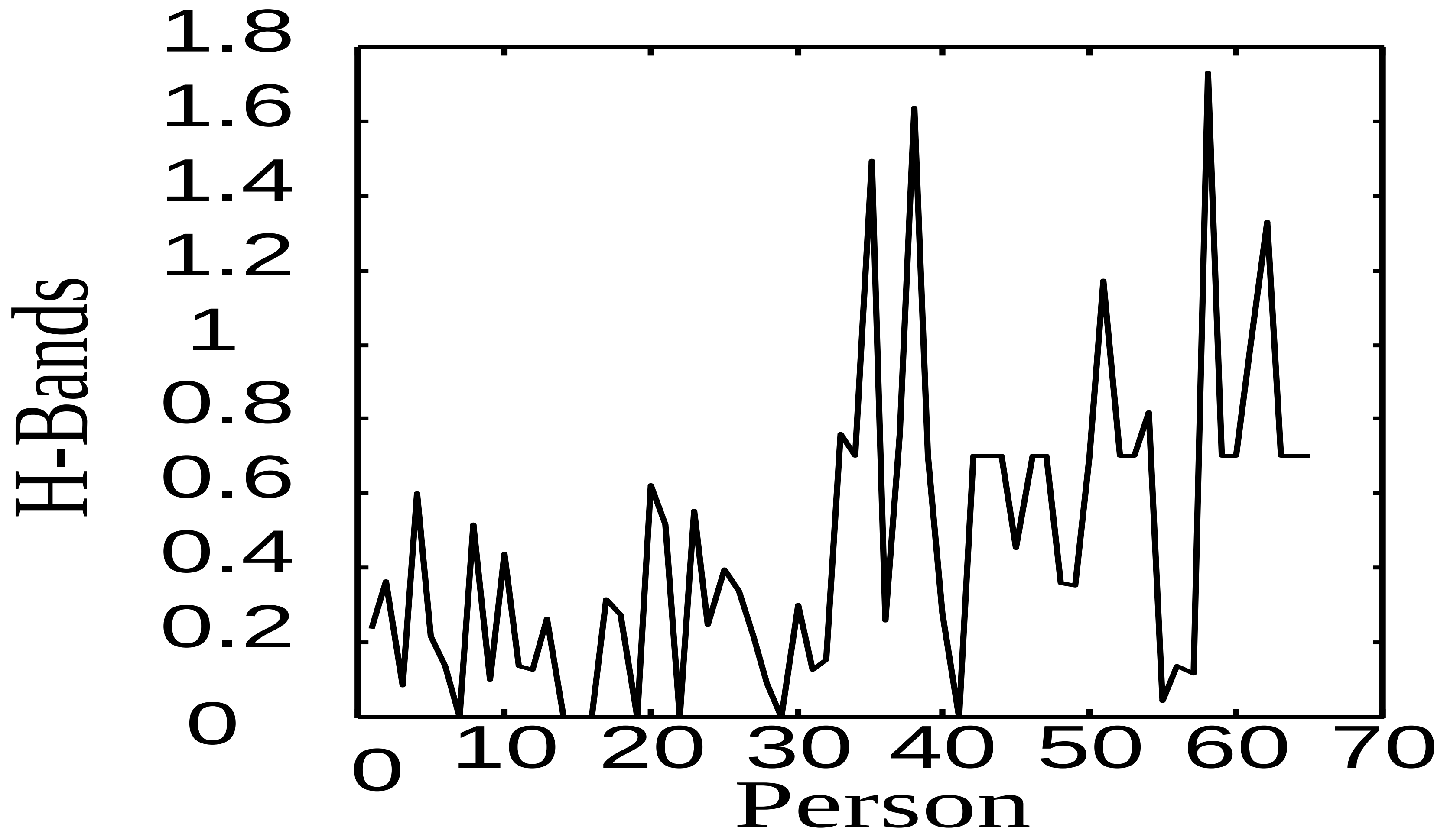
- Spectra format (65 pictures, one for each person)



Difficult to compare from different patients.

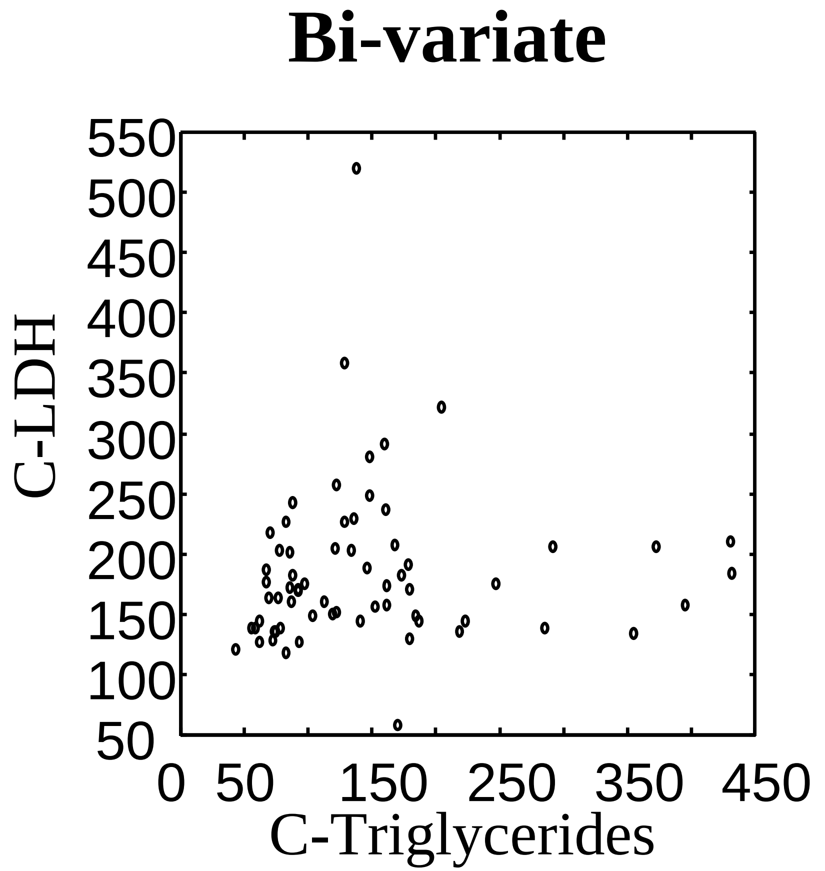
9. Principal Component Analysis (PCA)

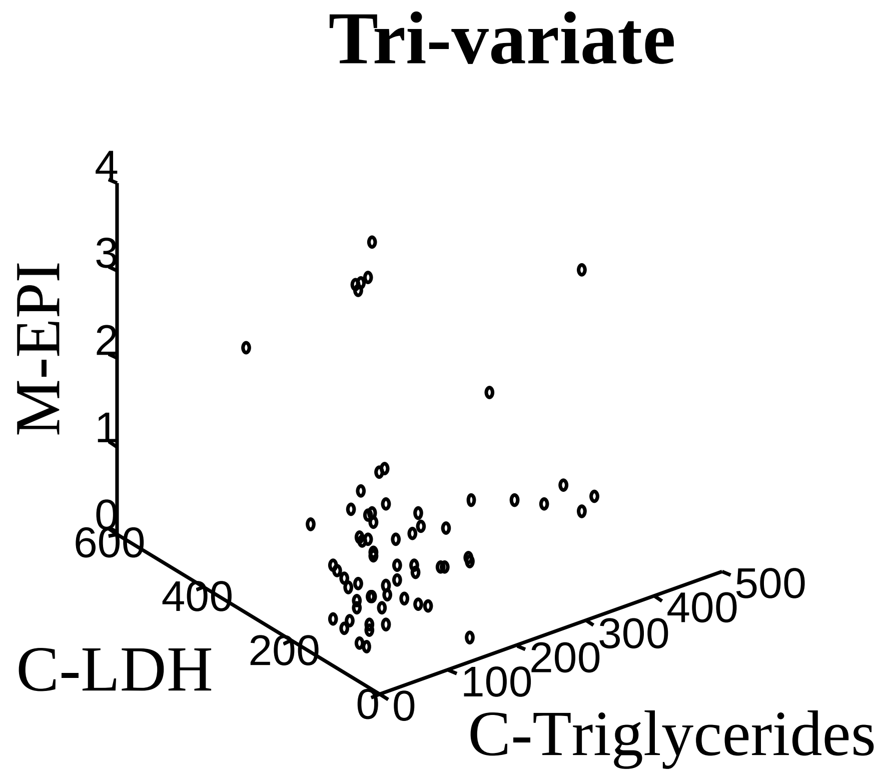
- Spectra format (53 pictures, one for each feature)



Difficult to see the correlations between the features.

9. Principal Component Analysis (PCA)





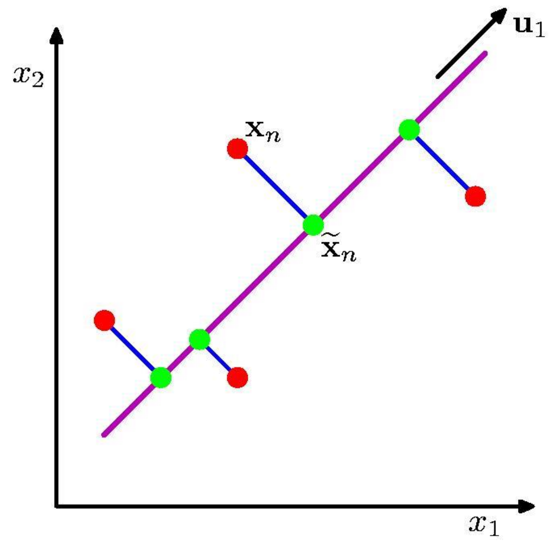
How can we visualize the other variables??

Difficult to see in 4 or higher dimensional spaces.

9. Principal Component Analysis (PCA)

- Is there a representation better than the coordinate axes?
- Is it really necessary to show all the 53 dimensions?
  - … what if there are strong correlations between the features?
- How could we find
  the *smallest* subspace of the 53-D space that
  keeps the *most information* about the original data?
- A solution: **Principal Component Analysis**

9. Principal Component Analysis (PCA)



PCA:

Orthogonal projection of data onto lower-dimension linear space that...

- maximizes variance of projected data (purple line)
- minimizes mean squared distance between
  - data point and the projections (sum of blue lines)

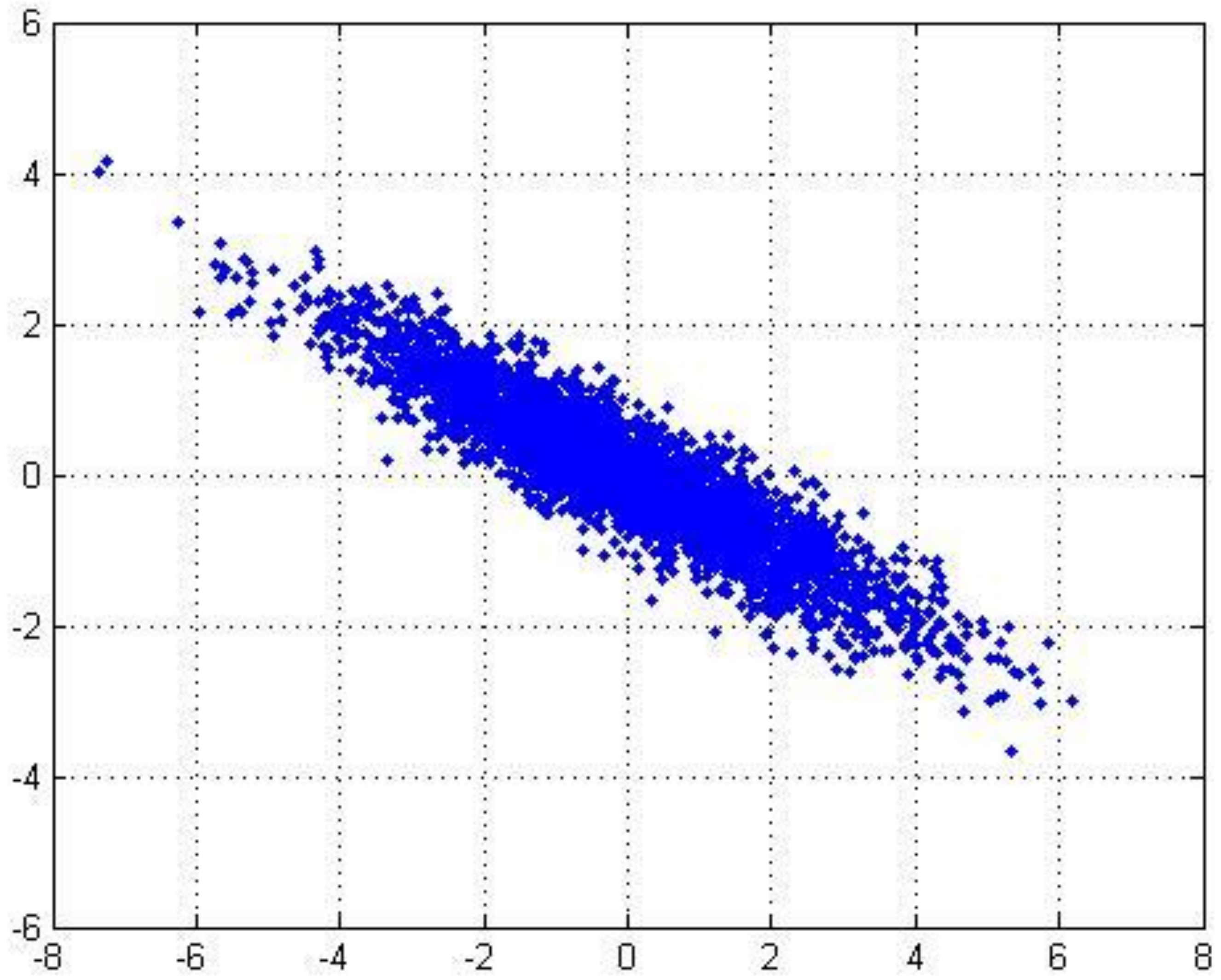
9. Principal Component Analysis (PCA)

Principal Components (PCA axis)

- PC#1 indicates the direction of the **largest variance**.

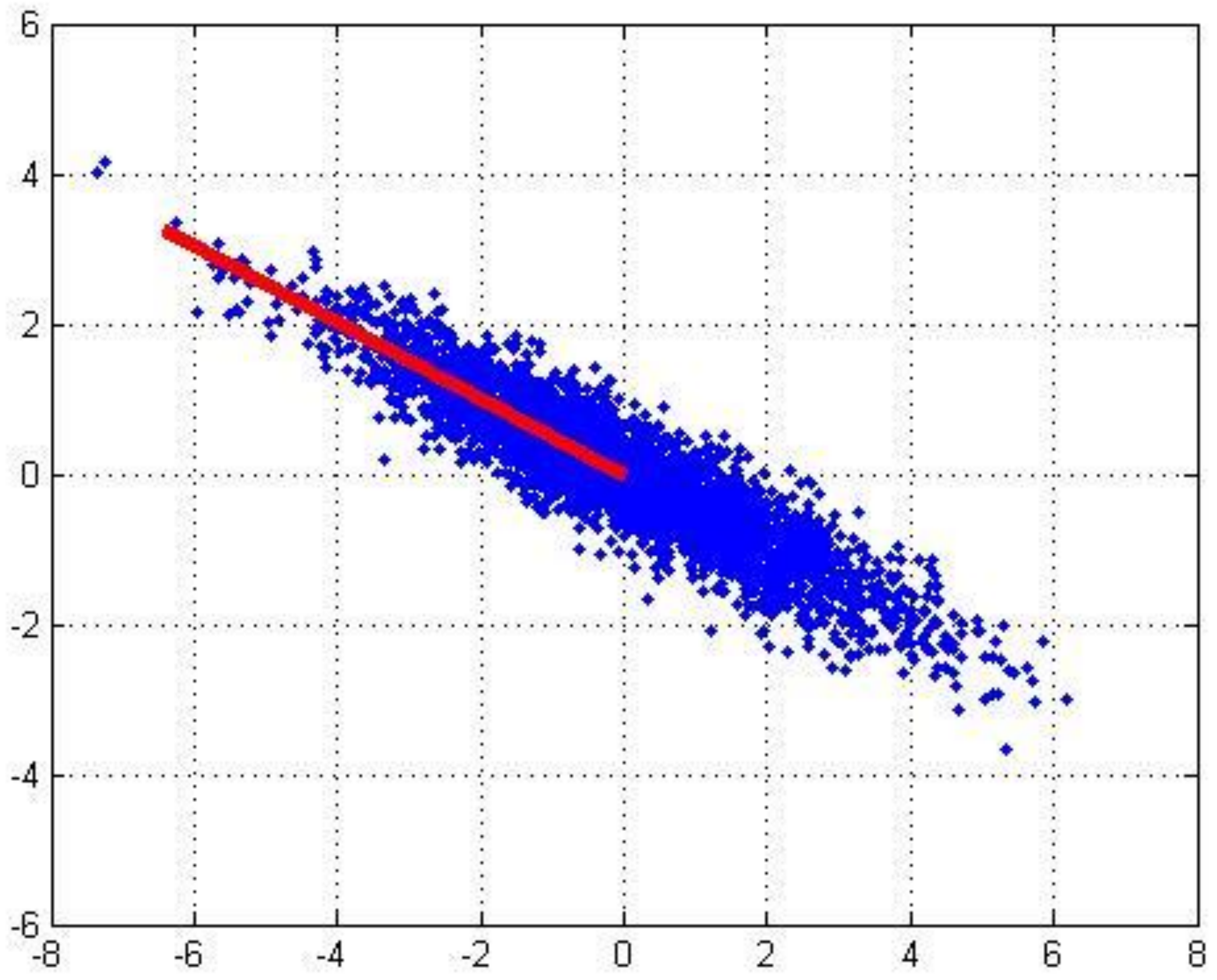
- Each subsequent principal component (PC#2 and so on)
  - is **orthogonal (perpendicular)** to the previous ones, and
  - Indicates the directions of the **next largest variance of the residual subspace**

9. Principal Component Analysis (PCA)



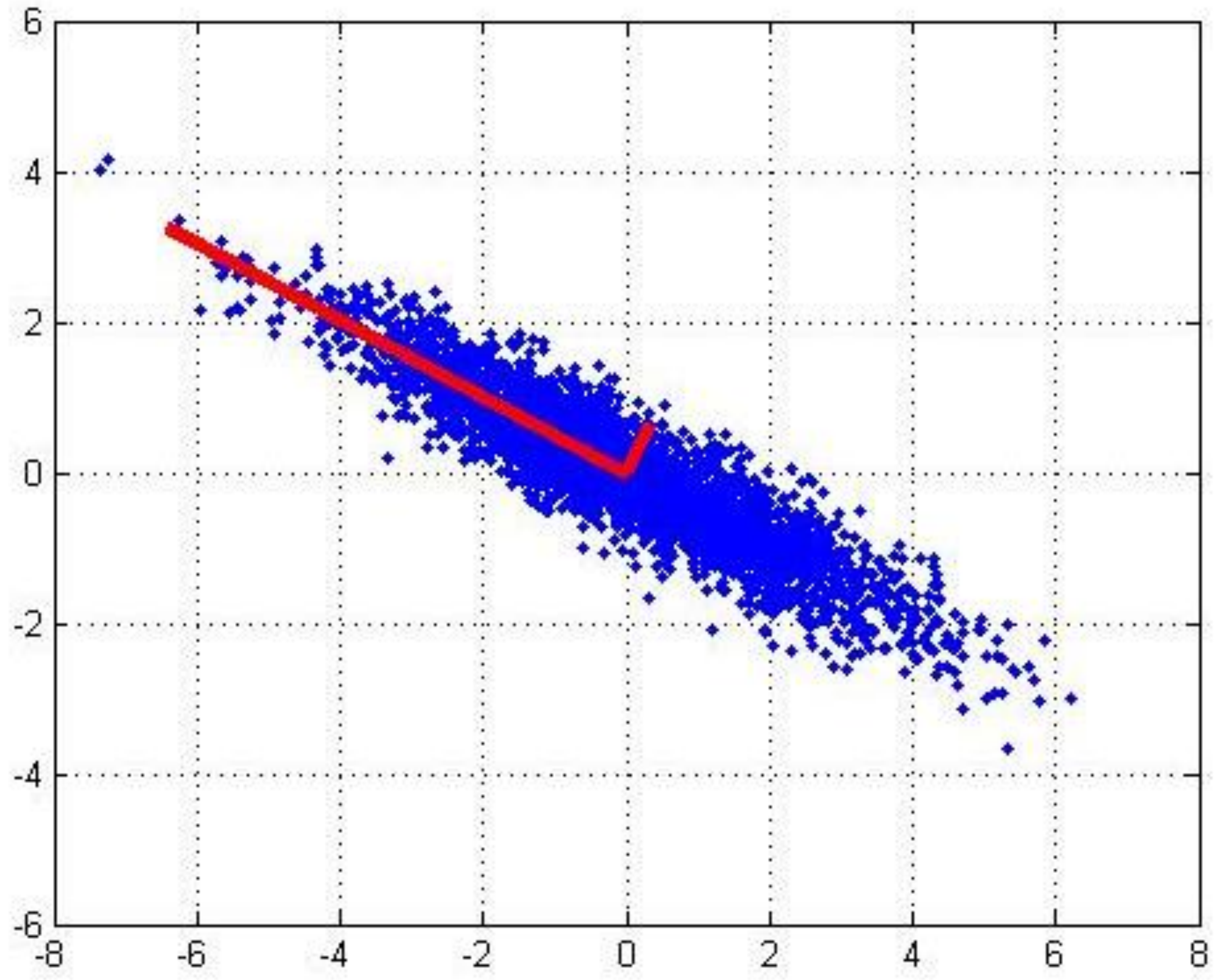
9. Principal Component Analysis (PCA)

1st PCA axis (PC#1)

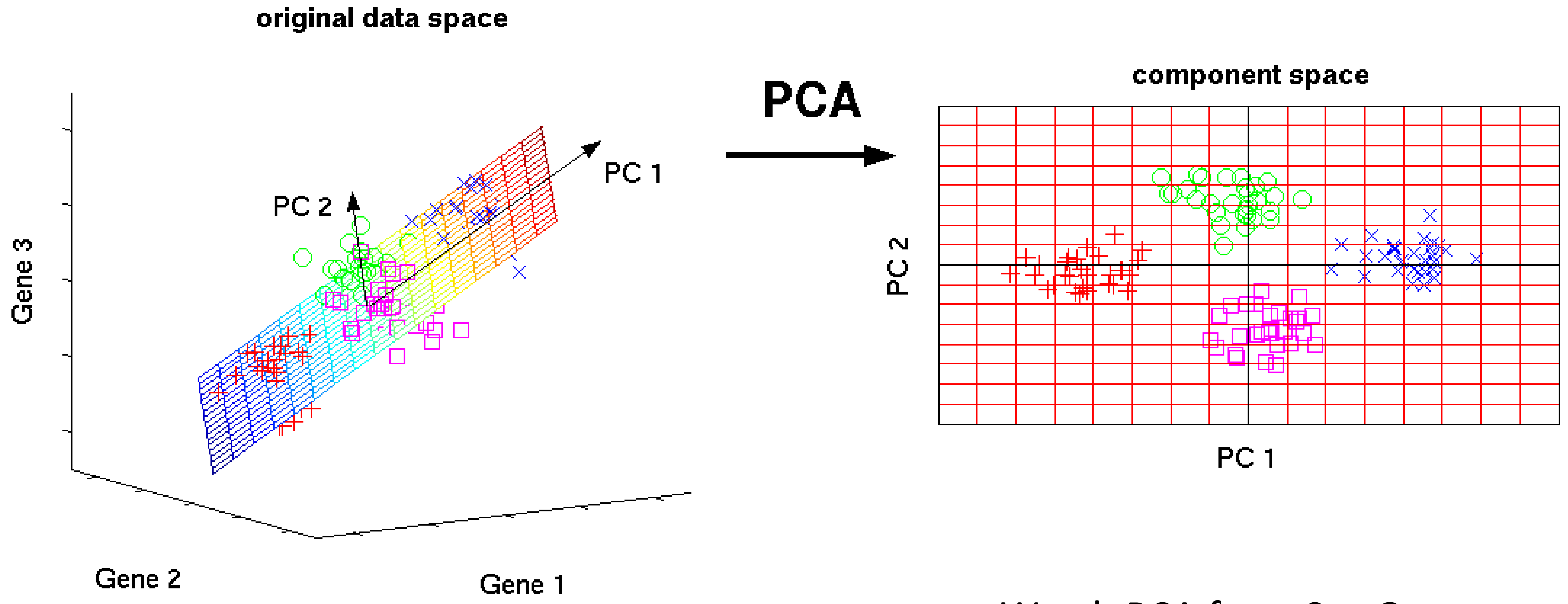


9. Principal Component Analysis (PCA)

2nd PCA axis (PC#2)

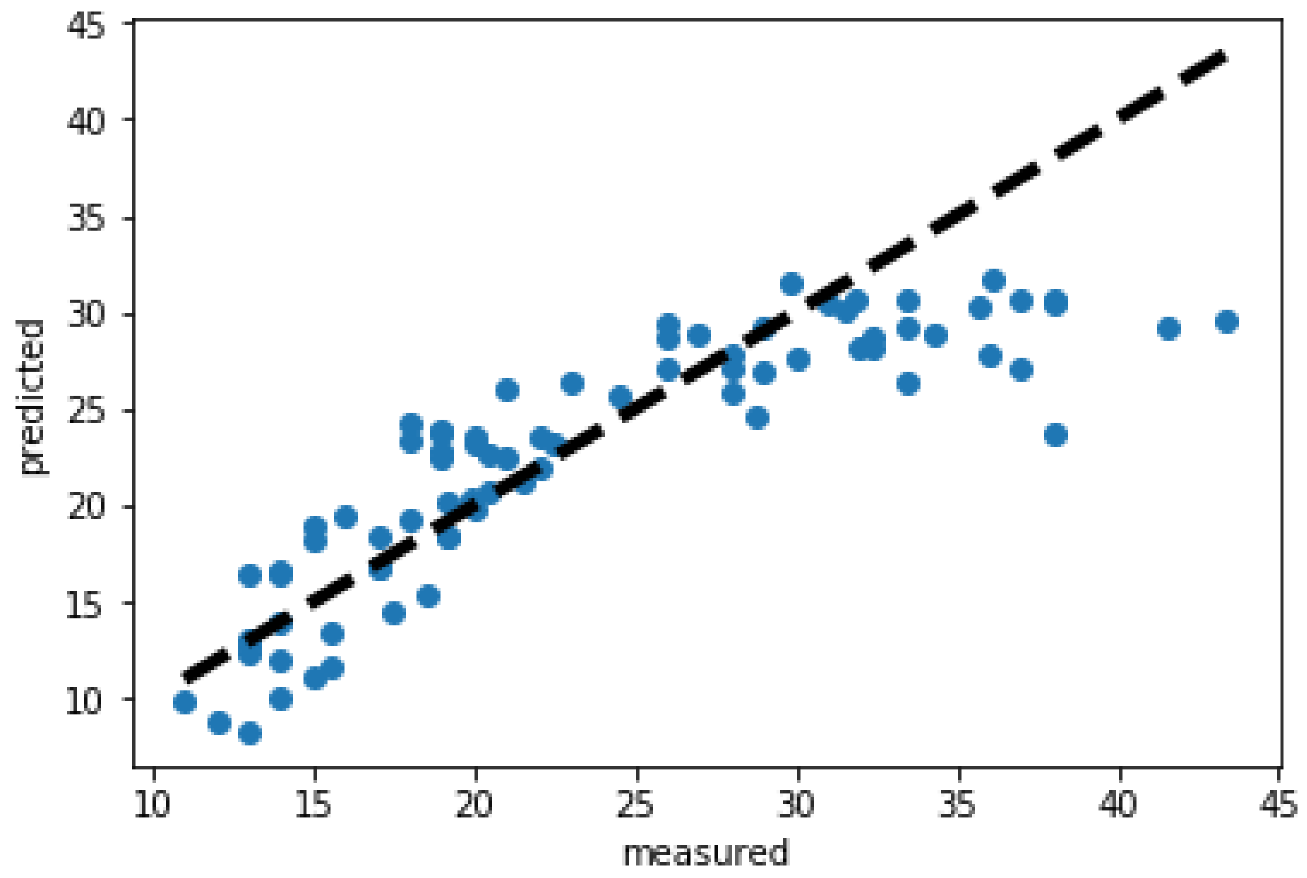


9. Principal Component Analysis (PCA)



Watch PCA from StatQuest:
https://bit.ly/2MZcn0Y

Model Assessments for Regression

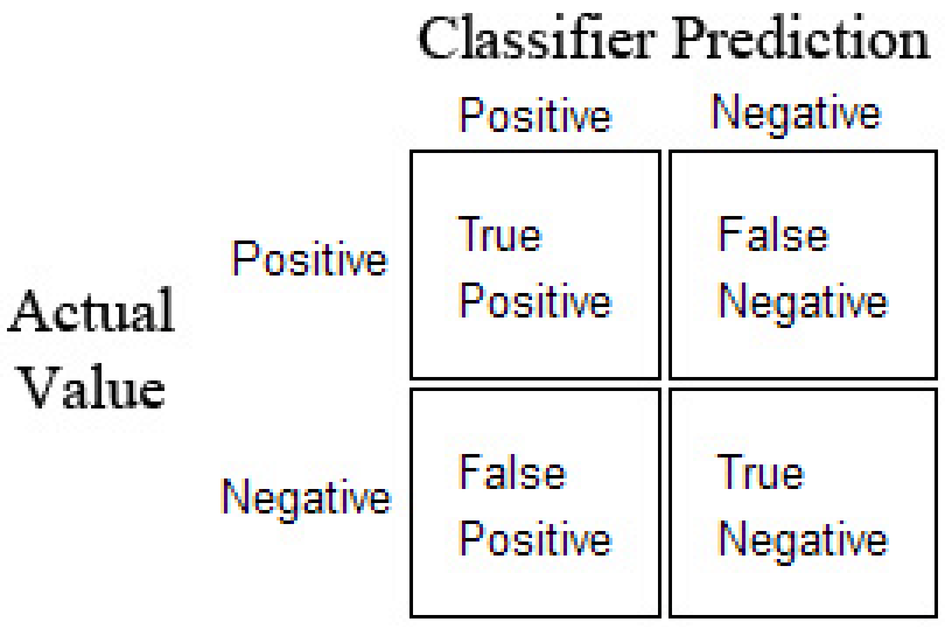


$$R^2 = \frac{\sum(p_i - \bar{y_i})^2}{\sum(y_i - \bar{y_i})^2}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - y_i)^2}{n}}$$

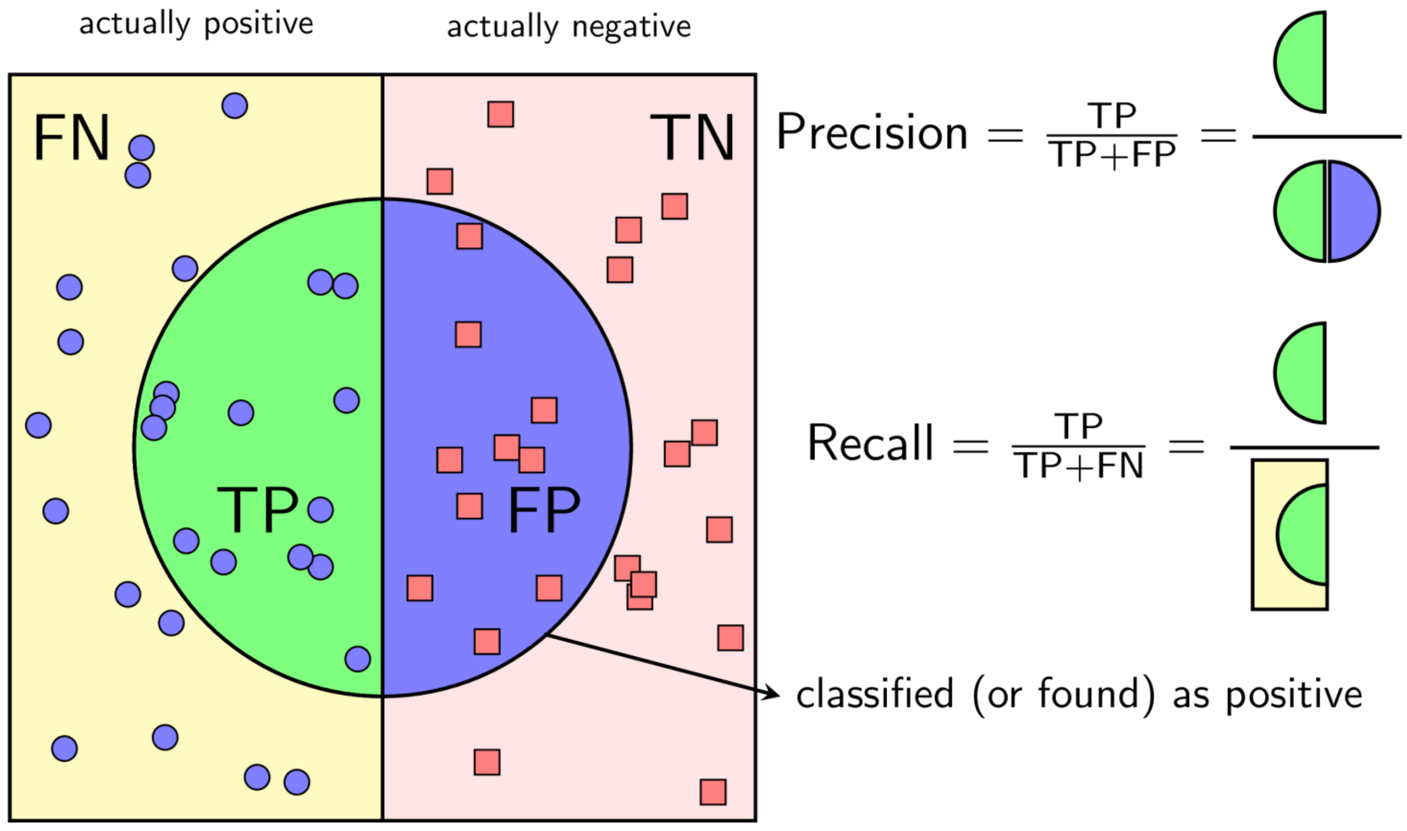
A value of R^2=1.0 indicates a perfect fit, and it is thus a perfect model for future forecasts. A value of 0, on the other hand, would indicate that the model fails to accurately model the data at all.
A good model normally gives R^2 value above 0.7.

Model Assessments for Classification: Confusion Matrix



		Classifier Prediction	
		Positive	Negative
Actual Value	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

Confusion Matrix



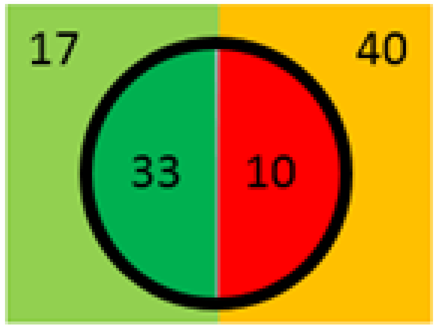
$$\text{Precision} = \frac{TP}{TP+FP} =$$

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

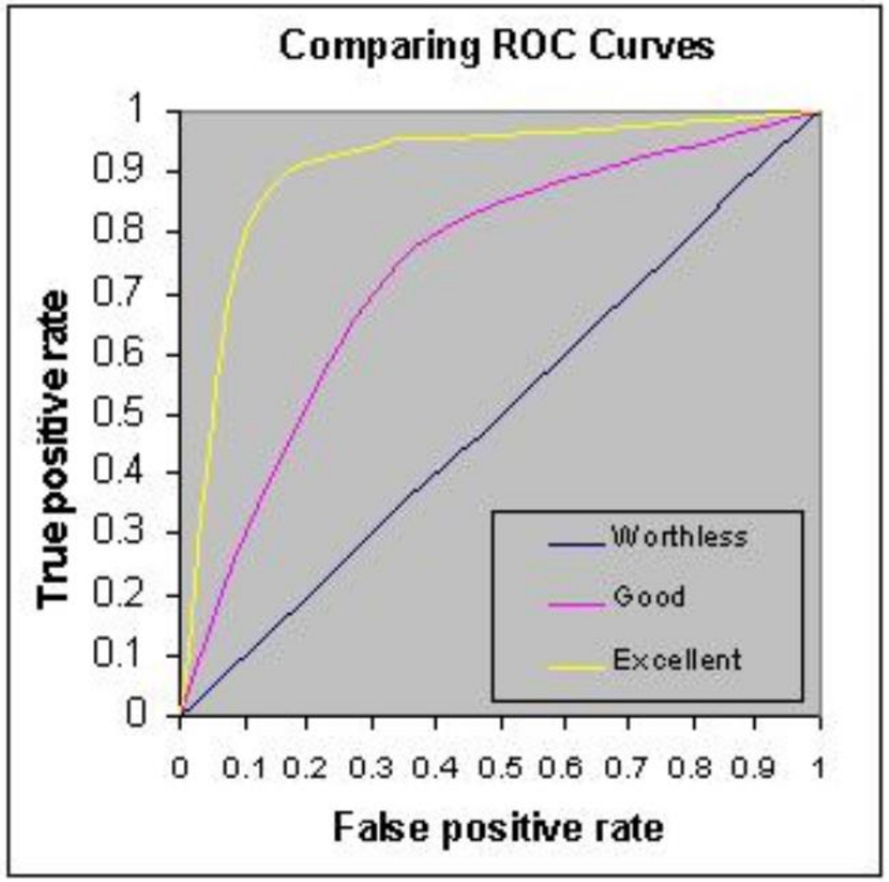
Model Assessments for Classification: Confusion Matrix

Confusion Matrix		Actual	
		Hospitalized	Not Hospitalized
Predicted	Hospitalized	33	10
	Not Hospitalized	17	40

Recall (Sensitivity)	33/50 = 0.66
Specificity	40/50 = 0.8
Accuracy	(33+40)/100 = .73
Precision	33/(33+10) = 0.77



Model Assessments for Classification: ROC Curve



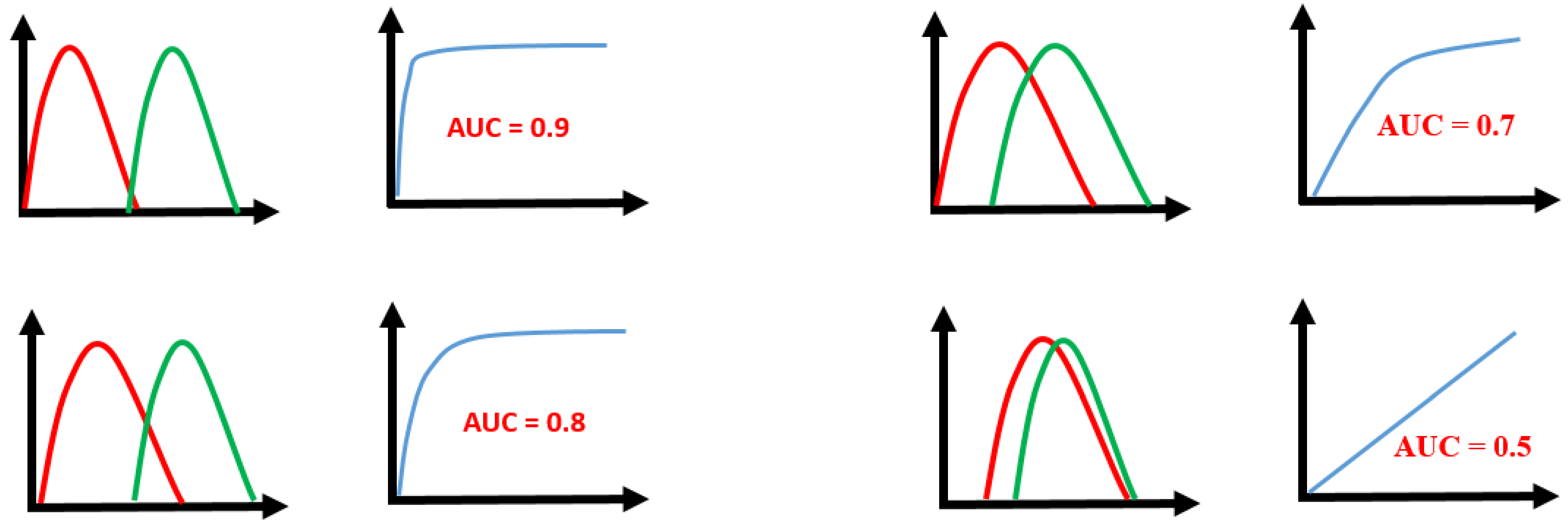
ROC curve is used to compare different models/classifiers. The yellow one is nearly ideal (the best classifier).

The area under the curve is called AUC.

AUC (**Area Under The Curve**)
ROC (**Receiver Operating Characteristics**)

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Model Assessments for Classification: ROC Curve



The first model does a good job of distinguishing the positive and the negative values. Therefore, there the AUC score is 0.9 as the area under the ROC curve is large.

Whereas, if we see the last model, predictions are completely overlapping each other and we get the AUC score of 0.5.