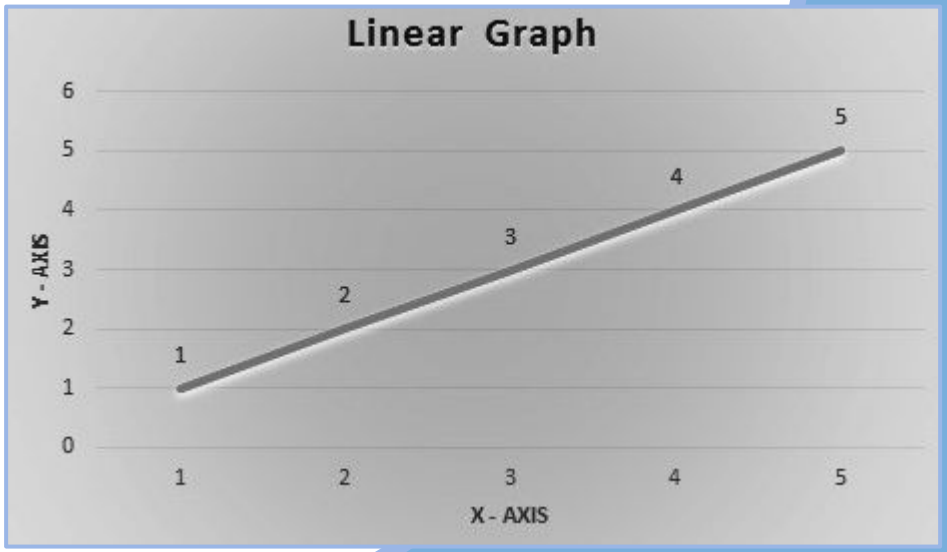
Mathematical View of ML Algos

Linear Regression

Linear Regression is a predictive algorithm which provides a Linear relationship between Prediction *(Call it 'Y') and Input (Call is 'X').*

Input (X) = 1,2,3,4,5

Prediction (Y) = 1,2,3,4,5



Linear Regression : Equation

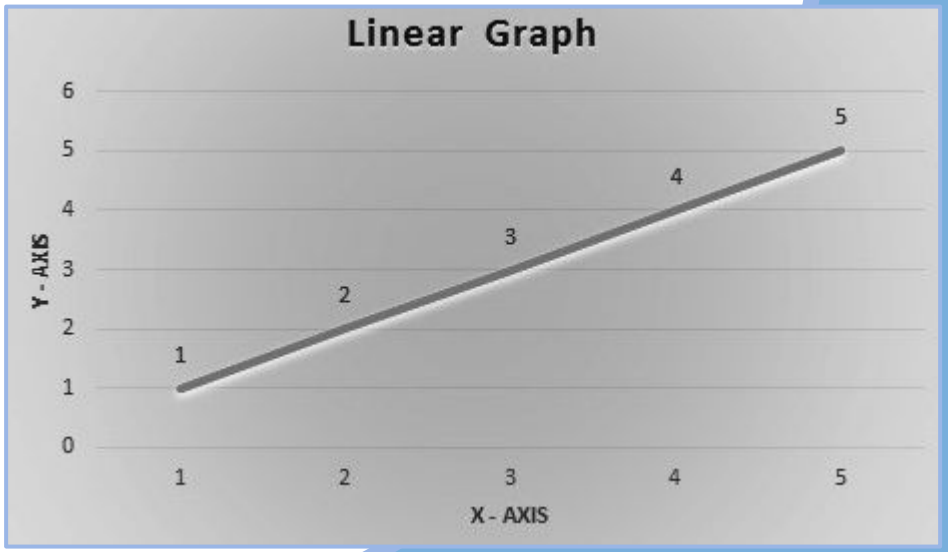
Equation of Straight Line : –

$$y = mx+b$$

m – Slope

b – Intercept

Linear regression is a way to predict the 'Y' values for unknown values of Input 'X'

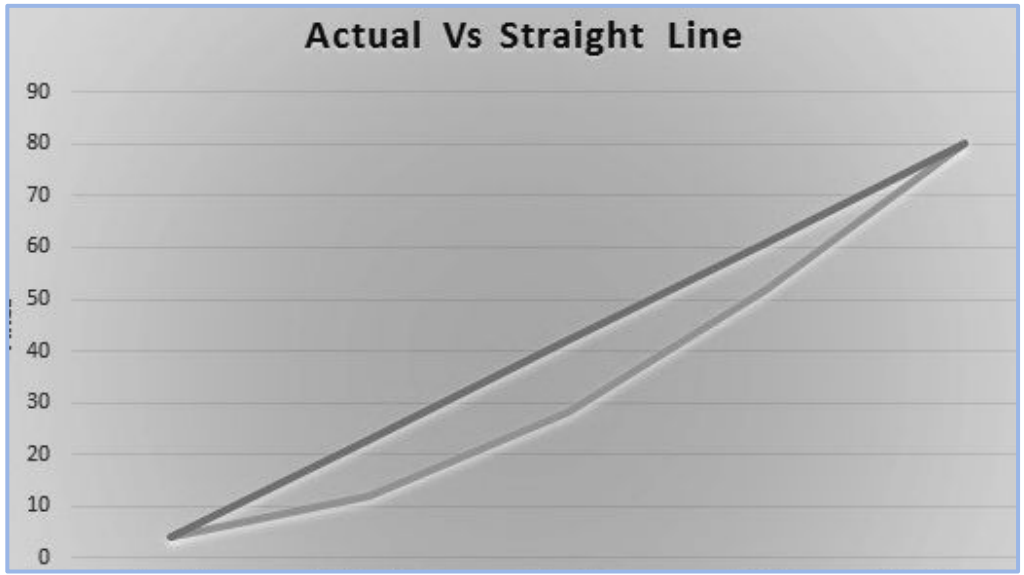


Linear Regression : Procedure

1. The first step is to come up with a formula in the form of y = mx + b where x is a known value and y is the predicted value.
2. To calculate the Prediction y for any Input value x we have two unknowns, the m = slope(Gradient) and b = y-intercept(also called bias)
3. Slope (m = Change in y/ Change in x)

   $m = (y2-y1)/(x2-x1)$

Projected Graph Against Original Graph



Minimizing the Error

The error is defined as the difference of values between actual points and the points on the straight line).

Ideally, we'd like to have a straight line where the error is minimized across all points.

Mathematically, we do this by Least Square Method

Least Square Regression is a method which minimizes the error in such a way that the sum of all square error is minimized.

Calculating Slope (m) for Least Square

$$m = \frac{\text{Sum of ALL } (x - xmean)*(y - yMean)}{\text{Sum of } (x - xmean)**2}$$

“

Calculating xmean and x-xmean for all x values

Mean Value of all 'x' => (1 + 2 + 3 + 4 + 5) / 5 = 3
At x = 1 : (x - xmean) => (1 - 3) = -2
At x = 2 : (x - xmean) => (2 - 3) = -1
At x = 3 : (x - xmean) => (3 - 3) = 0
At x = 4 : (x - xmean) => (4 - 3) = 1
At x = 5 : (x - xmean) => (5 - 3) = 2

Calculating y (mean) and y-y (mean) for all y Values

Mean Values of all 'y' => (4 + 12 + 28 + 52 + 80) / 5 = 35.2

At y = 1 : (y - ymean) => (4 - 35.2) = -31.2
At y = 2 : (y - ymean) => (12 - 35.2) = -23.2
At y = 3 : (y - ymean) => (28 - 35.2) = -7.2
At y = 4 : (y - ymean) => (52 - 35.2) = 16.8
At y = 5 : (y - ymean) => (80 - 35.2) = 44.8

Calculating SUM Of ALL { (x — xmean) * (y — ymean)}

(x,y) => (1,4) : (x — xmean)*(y — ymean) => { -2 * -31.2 = 62.4 }
(x,y) => (2,12) : (x — xmean)*(y — ymean) => { -1 * -23.2 = 23.2 }
(x,y) => (3,28) : (x — xmean)*(y — ymean) => { 0 * -7.2 = 0 }
(x,y) => (4,52) : (x — xmean)*(y — ymean) => { 1 * 16.8 = 16.8 }
(x,y) => (4,52) : (x — xmean)*(y — ymean) => { 2 * 44.8 = 89.6 }

Sum of All => { 62.4 + 23.2 + 0 + 16.8 + 89.6 } = 192

Calculating `Sum of (x — xmean)**2`

`{ -2**2 , -1**2 , 0**2 , 1**2 , 2**2 } =` `{ 4 , 1 , 0 , 1 , 4 } =` `{ 10 }`

Calculating The Slope

$$m = \frac{\text{Sum of ALL } (x - \text{xmean}) * (y - \text{ymean})}{\text{Sum of } (x - \text{xmean}) ** 2}$$

$$m = 192 / 10 = 19.2$$

Calculating bias (y-intercept)

```
b = ymean — m * xmean
```

```
b = 35.2 - 19.2 * 3
b = 35.2 - 57.6
b = -22.4
```

Calculating the Equation

$$y = mx + b \Rightarrow \{ y = 19.2x + (-22.4) \}$$

Predicting y for all x using Least Square

```
x => 1    { y = 19.2 * 1 - 22.4 } = - 3.2
x => 2    { y = 19.2 * 2 - 22.4 } = 16
x => 3    { y = 19.2 * 3 - 22.4 } = 35.2
x => 4    { y = 19.2 * 4 - 22.4 } = 54.4
x => 5    { y = 19.2 * 5 - 22.4 } = 73.6
```

“

Comparing Actual Vs Least Square

This method is intended to reduce the sum square of all error values. The lower the error, lesser the overall deviation from the original point.

Error in 'Y' With Least Square Equation

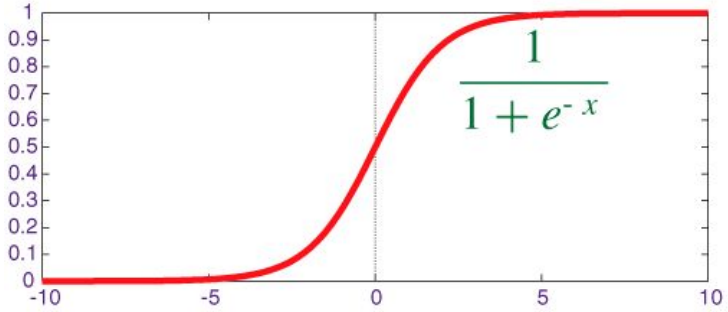
At X-Value	Y-Value	Actual-Value	Error	Square-Error
1	4	-3.2	7.2	51.84
2	12	16	-4	16
3	28	35.2	-7.2	51.84
4	52	54.4	-2.4	5.76
5	80	73.6	6.4	40.96

Sum of Square Error = { 51.84 + 16 + 51.84 + 5.76 + 40.96 } = 166.4

Logistic Regression

- A classification algorithm used when the dependent variable is categorical, and the independent variables can be discrete or continuous.

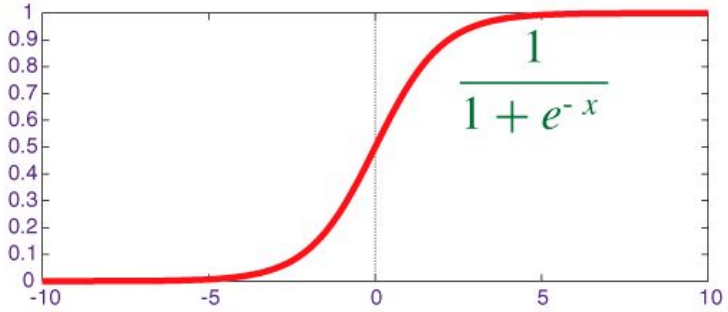
- Logistic Regression returns the ***probability*** of occurrence of an event by fitting the data to a mathematical function called **'logit function'.**



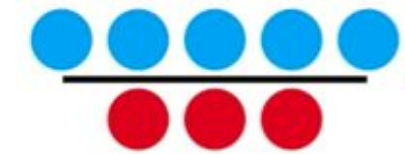
Logistic Regression

- It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1

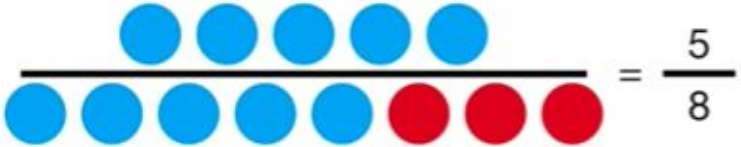
- e is the base of the natural logarithms
- x is the actual numerical value that you want to transform.



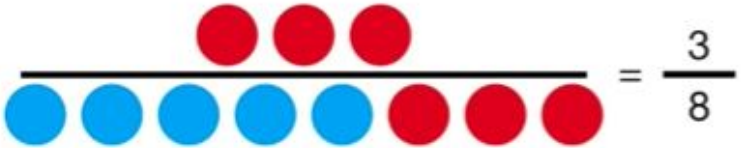
Odds and Log Odds



$= \frac{5}{3} = 1.7$



$= \frac{5}{8}$



$= \frac{3}{8}$

$$\frac{\text{The ratio of the probability of winning...}}{\text{...to (1 - the probability of winning)}} = \frac{5/8}{3/8} = \frac{5}{3} = 1.7$$

Odds can be calculated from probabilities

Odds and Log Odds

$$\log(\text{odds}) = \log\left(\frac{5}{3}\right) = \log\left(\frac{p}{(1-p)}\right) = \log(1.7) = 0.53$$

Log of Odds can be calculated either from counts or probabilities

"logit" = "log odds"

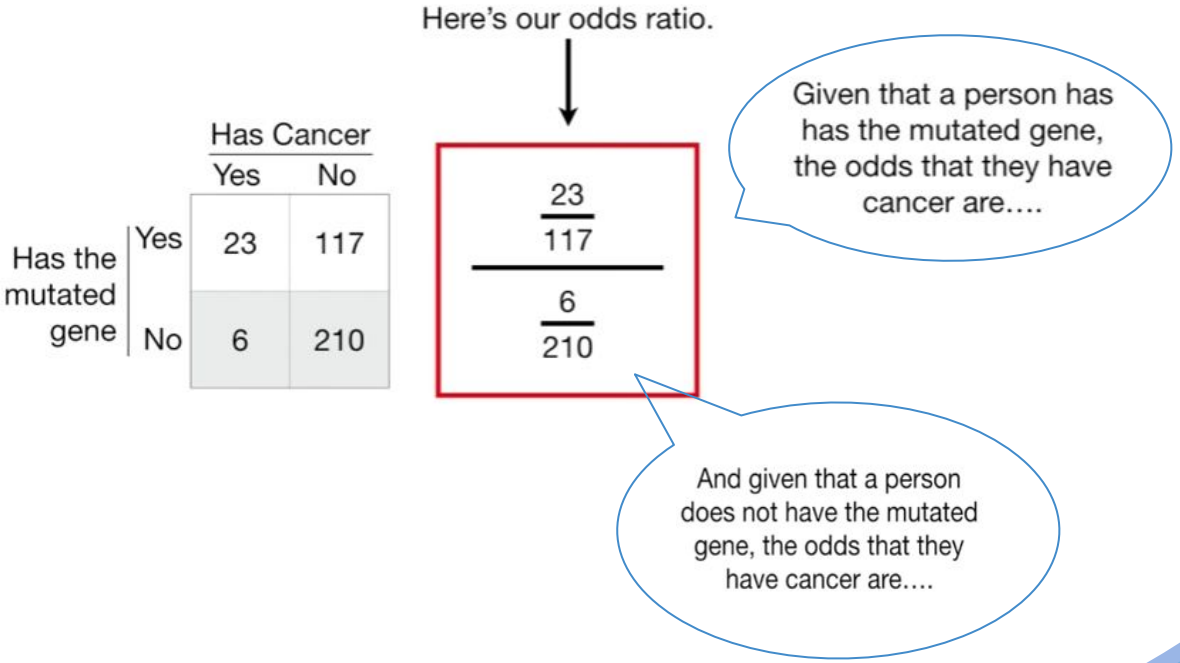
$$odds = \frac{P(event)}{1 - P(event)}$$

$$\log\left(\frac{p(x)}{1 - p(x)}\right) = \beta X$$

The log of the ratio of the probabilities is called the logit function.

Odds Ratio and Log Odds Ratio

It is the ratio of odds of something over odds of something else



Odds Ratio and Log Odds Ratio

It is the ratio of odds of something over odds of something else

Here's our odds ratio.

Has the mutated gene	Has Cancer: Yes	Has Cancer: No
Yes	23	117
No	6	210

$$\frac{\frac{23}{117}}{\frac{6}{210}} = \frac{0.2}{0.03} = 6.88$$

The odds ratio tells us that the odds are 6.88 times greater that someone with the mutated gene will also have cancer

Odds Ratio and Log Odds Ratio

It is the ratio of odds of something over odds of something else

Here's our odds ratio.

Has the mutated gene	Has Cancer: Yes	Has Cancer: No
Yes	23	117
No	6	210

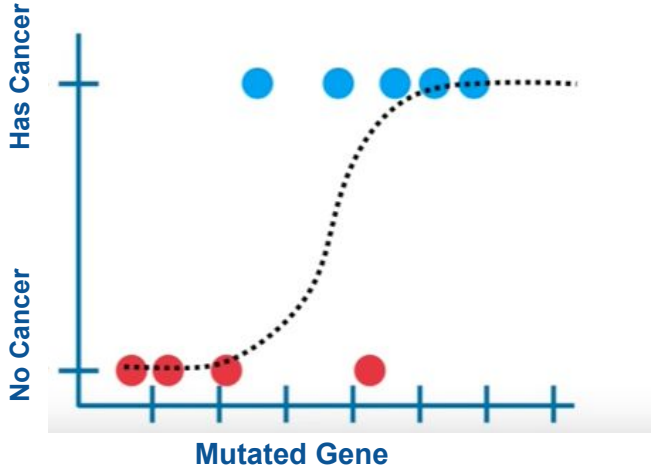
$$\frac{\frac{23}{117}}{\frac{6}{210}} = \frac{0.2}{0.03} = 6.88$$

$$\log(6.88) = 1.93$$

Odds Ratio and Log Odds Ratio

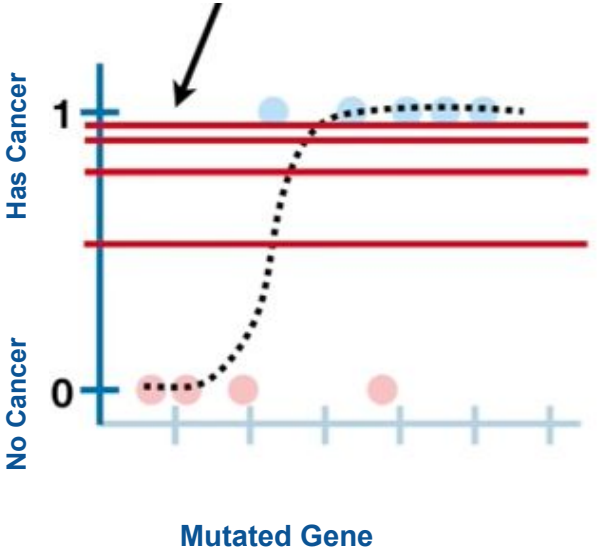
Log Odds Ratio talks about the relationship between - having cancer and having a mutated gene. A smaller value suggests a weaker relationship.

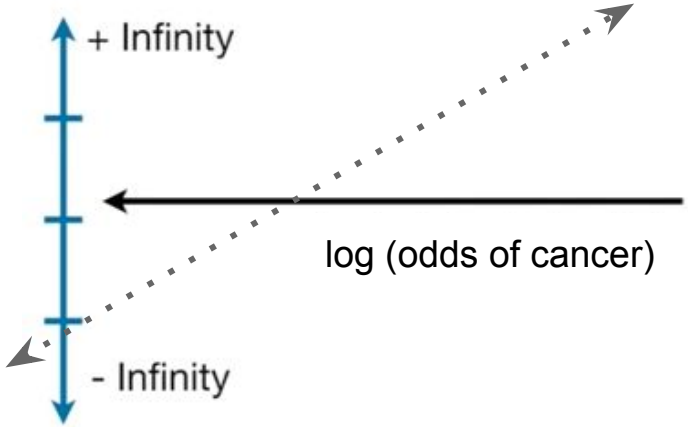
$$\log(\text{odds ratio}) = \log\left(\frac{\frac{23}{117}}{\frac{6}{210}}\right) = \log(6.88) = 1.93$$



Probability [0,1] that a person has cancer given the weight of him having mutated gene

Odds Ratio and Log Odds Ratio





We plug different values of p between 0 and 1 to log odds

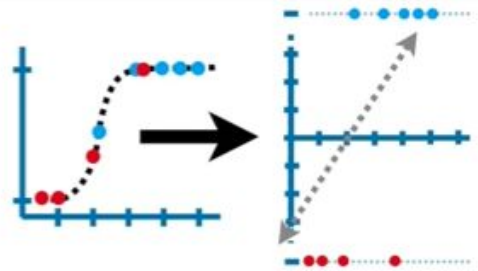
E.g.,
log (odds) =log (p/1-p)
say, p=0.5
log(0.5/1-0.5) = log(1) = 0

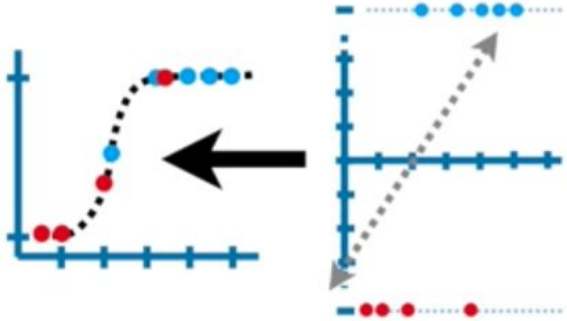
Odds Ratio and Log Odds Ratio

??

How to Convert an Equation that takes probability as input and outputs log (odds)
Into
An Equation that takes log(odds) as input and outputs probability

$$\log\left(\frac{p}{1-p}\right) = \log(\text{odds})$$





$$p = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

Logit function

$$\log\left(\frac{p}{1-p}\right) = \log(\text{odds})$$

Exponentiate both sides…

$$\frac{p}{1-p} = e^{\log(\text{odds})}$$

Multiply both sides by $(1 - p)$…

$$p = (1-p)e^{\log(\text{odds})}$$

Multiply $(1 - p)$ and $e^{\log(\text{odds})}$…

$$p = e^{\log(\text{odds})} - pe^{\log(\text{odds})}$$

Add $pe^{\log(\text{odds})}$ to both sides…

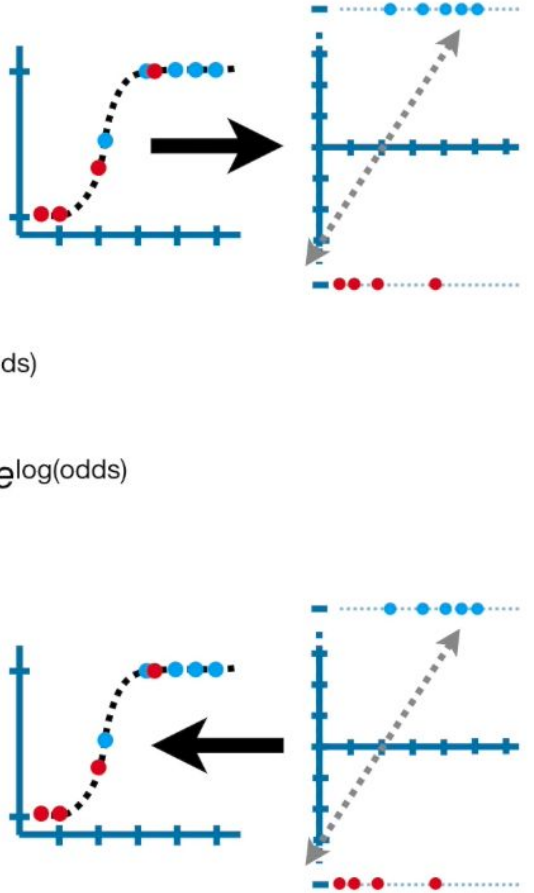
$$p + pe^{\log(\text{odds})} = e^{\log(\text{odds})}$$

Pull p out…

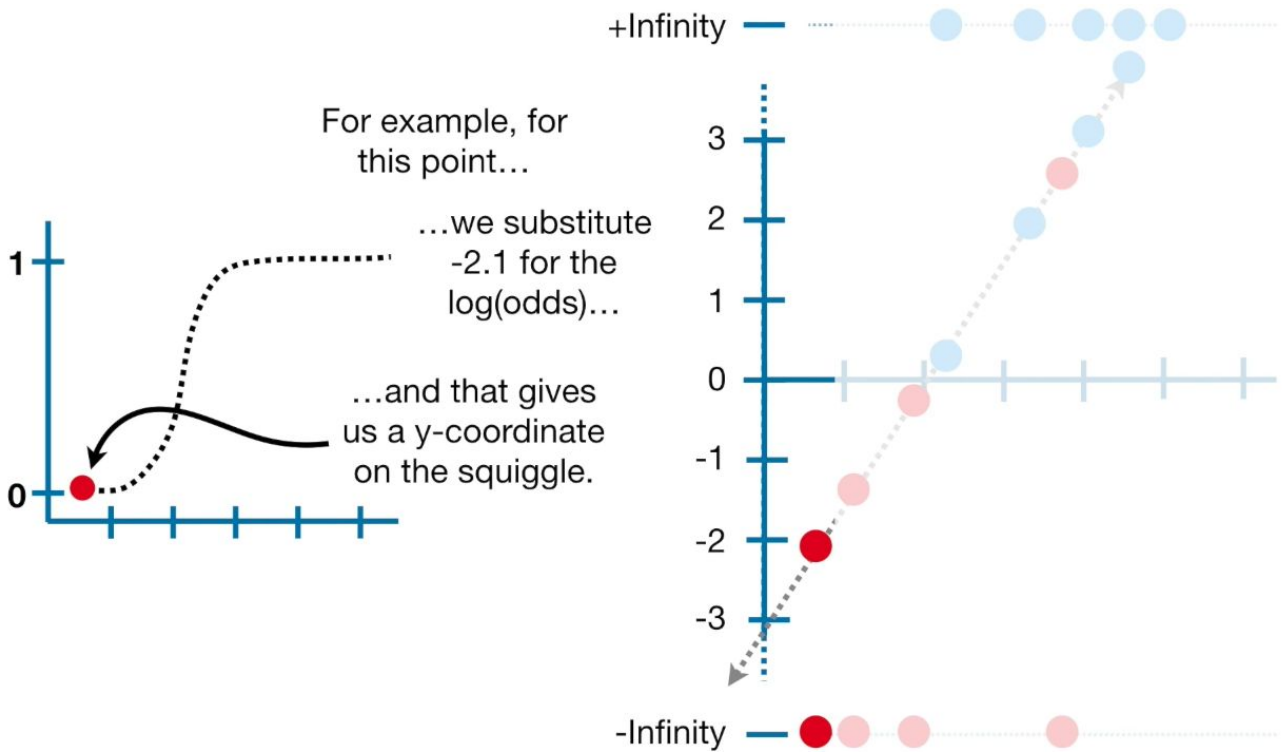
$$p(1 + e^{\log(\text{odds})}) = e^{\log(\text{odds})}$$

Divide both sides by $(1 + e^{\log(\text{odds})})$…

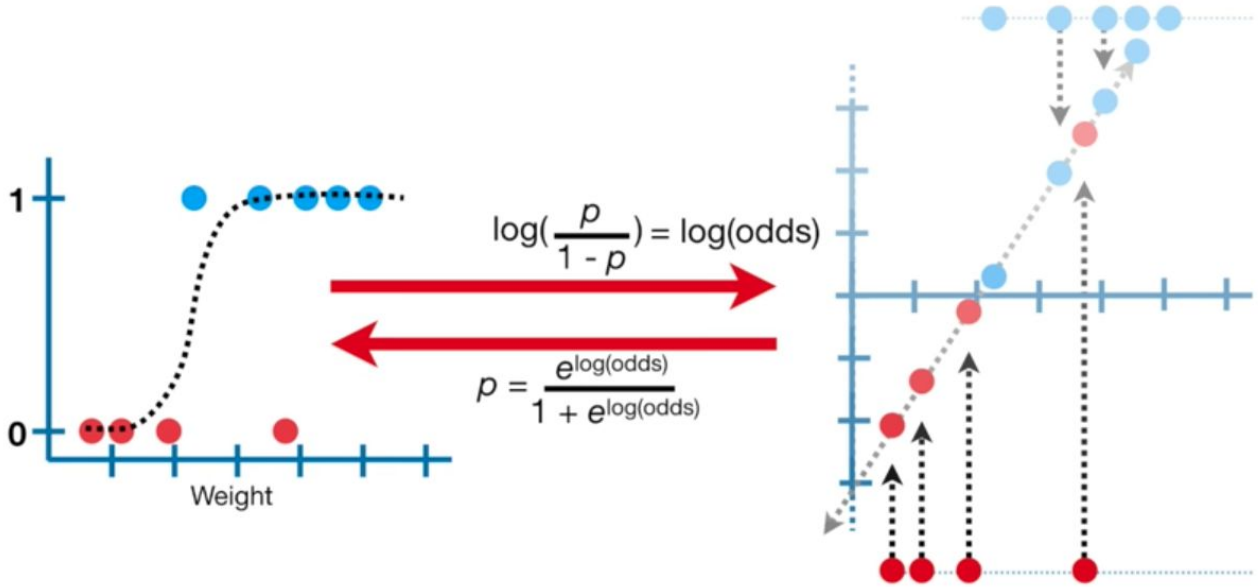
$$p = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$



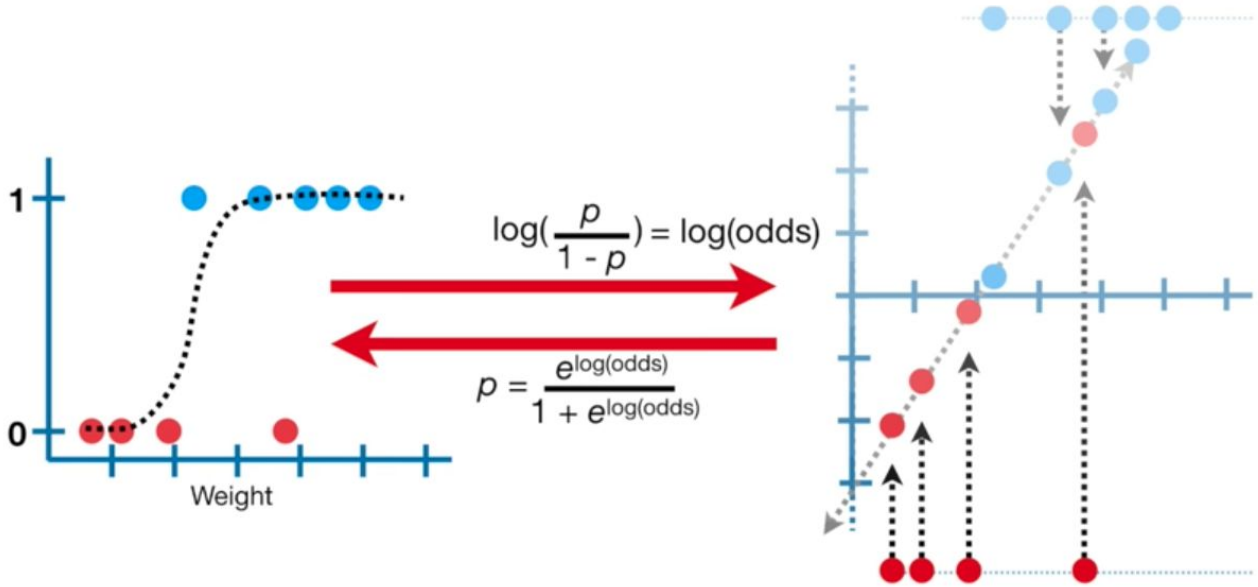
Logit function



Logistic Regression



Logistic Regression



Metrics

- confusion_matrix
- accuracy_score
- recall_score
- precision_score
- f1_score
- roc_curve
- roc_auc_score

Metrics

- confusion_matrix

Confusion Matrix		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Metrics

- accuracy_score

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

Fraction predicted correctly

Metrics

- recall_score

$$\text{Recall (Sensitivity)} = \frac{TP}{TP + FN}$$

Fraction of positives predicted correctly

Metrics

- **Precision_score**

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

Fraction of predicted positives that are actually positive

Metrics

- **F1_score**

a. The f1 score is the harmonic mean of recall and precision, with a higher score as a better model.

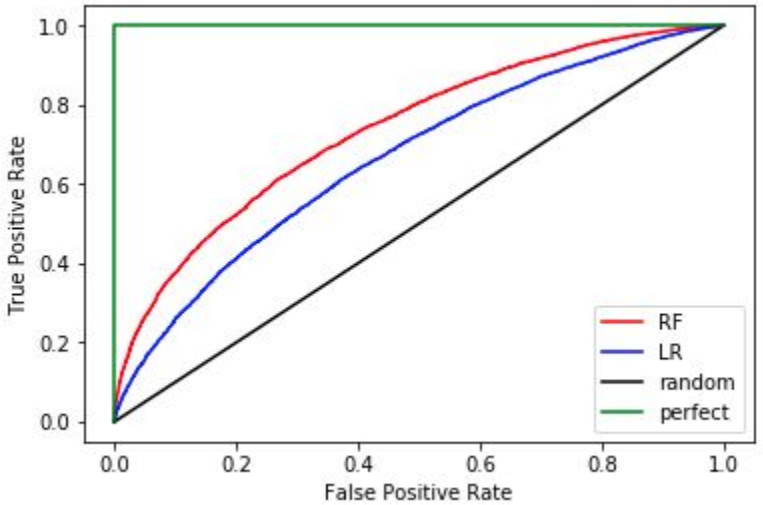
$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 * (precision * recall)}{precision + recall}$$

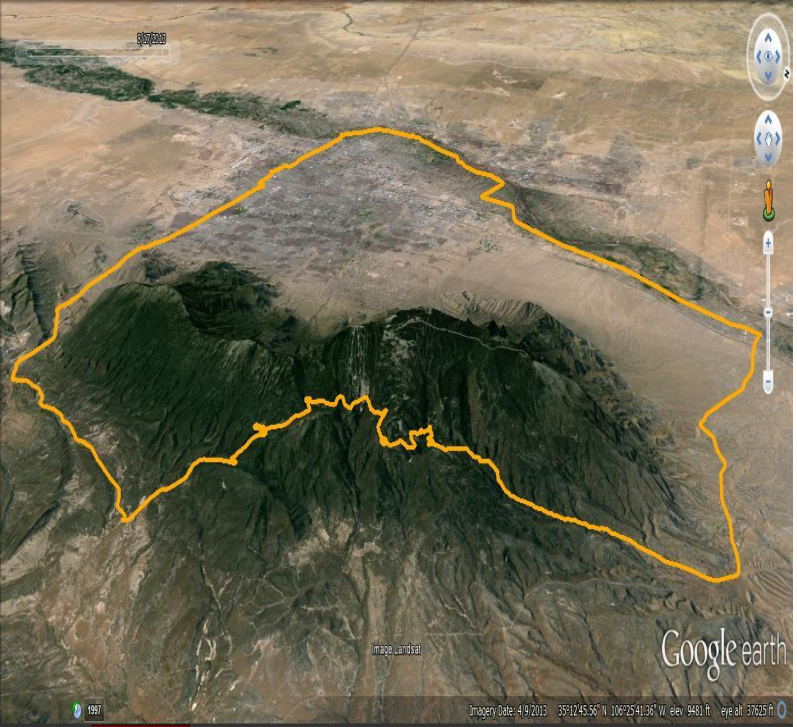
Metrics

- ## Roc curve

- A model that randomly guesses the label → **black line**
- We want to have a model that has a curve above this black line
- An ROC that is farther away from the black line is better,

  so RF (red) looks better than LR (blue)





Gradient Descent

Mathematical Explanation

What is a Derivative

Derivatives are used to show rate of change

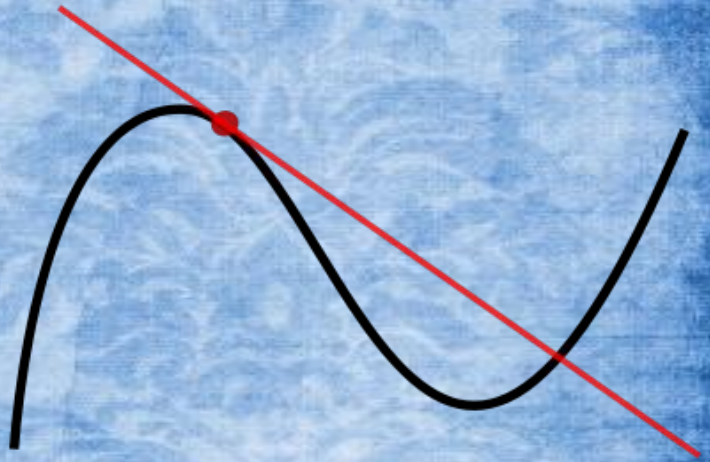
i.e., the amount by which a function is changing at one given point.

dy/dx = diff in y/diff. In x

y= mx+b

= d(mx+b)/dx

= d(mx)/dx + d(b)/dx = m (slope)



Rules of Derivations

Linear Function

$$\frac{d}{dx}(x) = 1$$

Power Function

$$\frac{d}{dx}x^a = ax^{a-1} \quad \Rightarrow \quad \frac{d}{dx}x^6 = 6x^5$$

Rules of Derivations

Exponential Function

$$\frac{d}{dx}\left(ab^{f(x)}\right) = ab^{f(x)} \cdot f'(x) \cdot \ln(b)$$

$$\Rightarrow \quad \frac{d}{dx}\left(3 \cdot 2^{3x^2}\right) = 3 \cdot 2^{3x^2} \cdot 6x \cdot \ln(2) = \ln(2) \cdot 18x \cdot 2^{3x^2}$$

Rules of Derivations

Logarithmic Function

$$\frac{d}{dx}\left(ab^{f(x)}\right) = ab^{f(x)} \cdot f'(x) \cdot \ln(b)$$

$$\Rightarrow \quad \frac{d}{dx}\left(3 \cdot 2^{3x^2}\right) = 3 \cdot 2^{3x^2} \cdot 6x \cdot \ln(2) = \ln(2) \cdot 18x \cdot 2^{3x^2}$$

Rules of Derivations

Logarithmic Function $\frac{d}{dx}\ln(x) = \frac{1}{x}.$

$\Rightarrow \frac{d}{dx}\ln\left(\frac{5}{x}\right) \Rightarrow \frac{d}{dx}(\ln(5)) - \frac{d}{dx}(\ln(x))$

$\Rightarrow 0 - \frac{d}{dx}\ln(x) = -\frac{1}{x}$

Rules of Derivations

Trigonometric Function

$$\frac{d}{dx}\sin(x) = \cos(x)$$

$$\frac{d}{dx}\cos(x) = -\sin(x)$$

$$\frac{d}{dx}\sec(x) = \sec(x)\tan(x).$$

Rules of Partial Derivations

The partial derivative is taken in a multivariable function

A Partial variable of a function is its derivative with respect to one of those variables with the others held constant.

Rules of Partial Derivations

$$f(x,y) = x^2 + y$$

$$\frac{\partial}{\partial y}[f(x,y)] = 1$$

$$\frac{\partial}{\partial x}[f(x,y)] = 2x$$

Gradient Descent

Gradient Descent is an Iterative Optimization Algorithm that runs many times.

It is used to find the weights that minimize the cost.

Gradient Descent

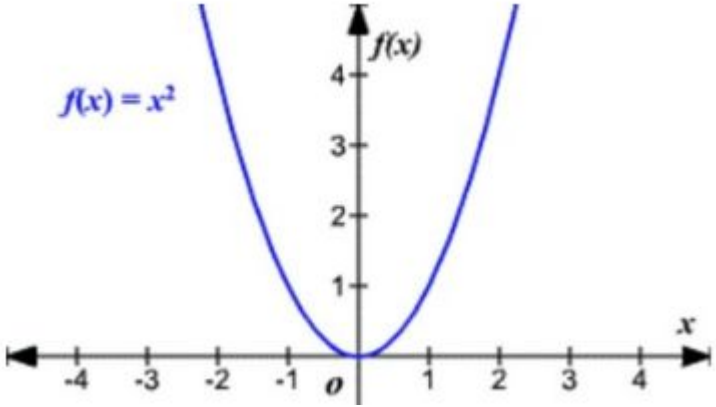
- Gradient Descent is an Iterative Optimization Algorithm that runs many times.
- It is used to find the weights that minimize the cost , thru $\textit{Cost Function } J(\theta)$
- The objective is to find the θ which minimizes the cost

Gradient Descent

$$y = \theta 0 + \theta 1 * X$$

```
error = (h(x)-y)² --> (Predicted - Actual)²
```

For $(h(x)-y)^2$ function we get always positive values.
Plotting the error graph.....



How do we Find the Gradient?

Gradient Descent

1. It draws the line (Tangent) from the point.
2. It finds the slope of that line.
3. It identifies how much change is required by taking the partial derivative of the function with respective to θ
4. The change value will be multiplied with a variable called alpha(learning rate) *we provide the value for alpha usually 0.01*
5. It subtracts this change value from the earlier θ value to get new θ value .

Gradient Descent

On each iteration, we apply the following “update rule” --

$$\theta := \theta - \alpha \frac{d}{d\theta} J(\theta)$$

(the := symbol means replace theta with the value computed on the right)

For a Simple Cost Function of a single variable, J(Θ) = Θ^2,
we get --

$$\alpha = 0.1, \qquad \frac{d}{d\theta} J(\theta) = 2\theta$$

Why GD uses Derivatives of the Cost Function

1

To find the direction to move theta or coeff.(s) in.

A positive slope means function goes upward as you move right, so we should move left in order to find the minimum.

A negative slope means the function goes downward towards the right, so we want to move right to find the minimum.

Why GD uses Derivatives of the Cost Function

2

To find how big of a step to take

If the slope is small, we want to take smaller steps.

If the slope is large, we want to take a large step because we are far from the minima.

Learning Rate

Learning Rate controls how large steps we make.

- If the learning rate is too high, you can overstep the minima and even diverge and bounce around the optimum.

- If the learning rate is too small you will need to run more iterations of gradient descent before you reach the minima increasing the training potentially never reaching the optima.

Gradient Descent – Multiple Variables

- When there are multiple variables in the minimization objective, gradient descent defines a separate update rule for each variable.
- The update rule for uses the partial derivative of J w.r.t θ_1 treating θ_2 as constant.

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Gradient Descent – Multiple Variables

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

Gradient Descent – MSE

Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

Gradient Descent – Multiple Variables

Scalar multiple rule: $\frac{d}{dx}(\alpha u) = \alpha \frac{du}{dx}$

Sum rule: $\frac{d}{dx}\sum u = \sum \frac{du}{dx}$

Power rule: $\frac{d}{dx}u^n = nu^{n-1}\frac{du}{dx}$

Chain rule: $\frac{d}{dx}f(g(x)) = f'\big(g(x)\big)g'(x)$

Gradient Descent – Multiple Variables

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$\frac{d}{d\theta_0}J(\theta_0, \theta_1) = \frac{d}{d\theta_0}\left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right)$$

$$= \frac{1}{m}\sum_{i=1}^{m}\frac{d}{d\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}2\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\frac{d}{d\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$= \frac{2}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

Gradient Descent – Multiple Variables

Partial derivative of the MSE function with respect to Θ0

- *Θ1, x and y are treated as constant*

$$\frac{d}{d\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) = \frac{d}{d\theta_0}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right) = 1$$

Gradient Descent – Multiple Variables

Partial derivative of the MSE function with respect to Θ1

- *Θ0, x and y are treated as constant*

$$\frac{d}{d\theta_1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) = \frac{d}{d\theta_1}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right) = x^{(i)}$$

Gradient Descent – Multiple Variables

Derivatives:

$$\frac{d}{d\theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)$$

$$\frac{d}{d\theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) \cdot x^{(i)}$$

NOTE : We multiply our MSE cost function by 1/2 so that when we take the derivative, the 2s cancel out. Multiplying the cost function by a scalar does not affect the location of its minimum

RANDOM FORESTS AND DECISION TREES

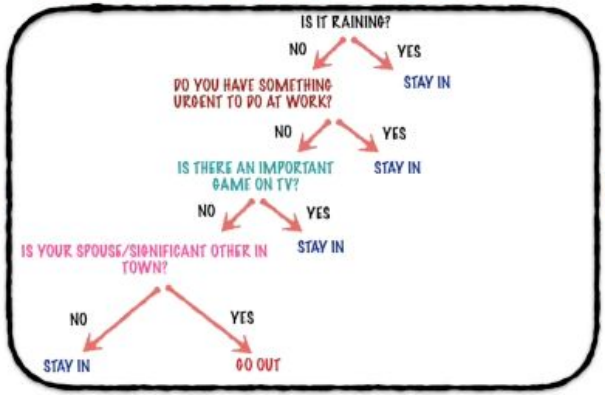
DECISION TREE LEARNING

IS THE PROCESS OF CREATING/
LEARNING A DECISION TREE
FROM TRAINING DATA

START WITH TRAINING DATA
IN THE FORM OF
FEATURE VECTOR, LABEL/OUTCOME

DECISION TREE

INPUT VARIABLES/ PREDICTORS



OUTCOME/OUTPUT VARIABLES

THE INPUT VARIABLES COULD BE CATEGORICAL OR CONTINUOUS

THE WEATHER

IS IT RAINING?

YES

NO

CATEGORICAL

CONTINUOUS

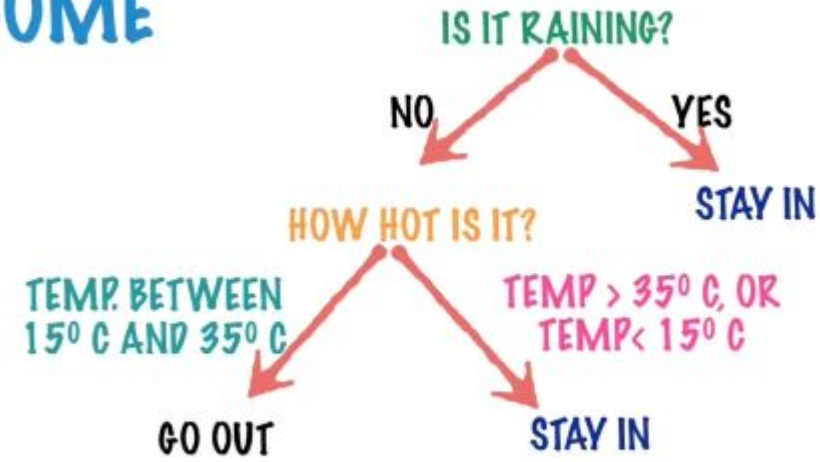
HOW HOT IS IT?

TEMP. BETWEEN 15^0 C AND 35^0 C

TEMP > 35^0 C, OR TEMP< 15^0 C

THE TEMPERATURE VALUE IS SPLIT INTO THE RANGES - 15-35; <15; >35

EACH RANGE LEADS TO A DIFFERENT OUTCOME



DECISION TREES CAN BE USED TO SOLVE CLASSIFICATION OR REGRESSION PROBLEMS

THE OUTPUT VARIABLES COULD BE CATEGORICAL OR CONTINUOUS

CLASSIFICATION TREES

CATEGORICAL

WILL YOU BE STAYING IN OR GOING OUT?

STAYING IN GOING OUT

IS THIS EMAIL SPAM OR HAM?

SPAM HAM

IS THIS CUSTOMER GOING TO DEFAULT ON THEIR CARD PAYMENT?

YES NO

THE OUTPUT VARIABLES COULD BE CATEGORICAL OR CONTINUOUS

CONTINUOUS

WHAT WILL THE SALE PRICE OF A HOUSE BE?

REGRESSION TREES

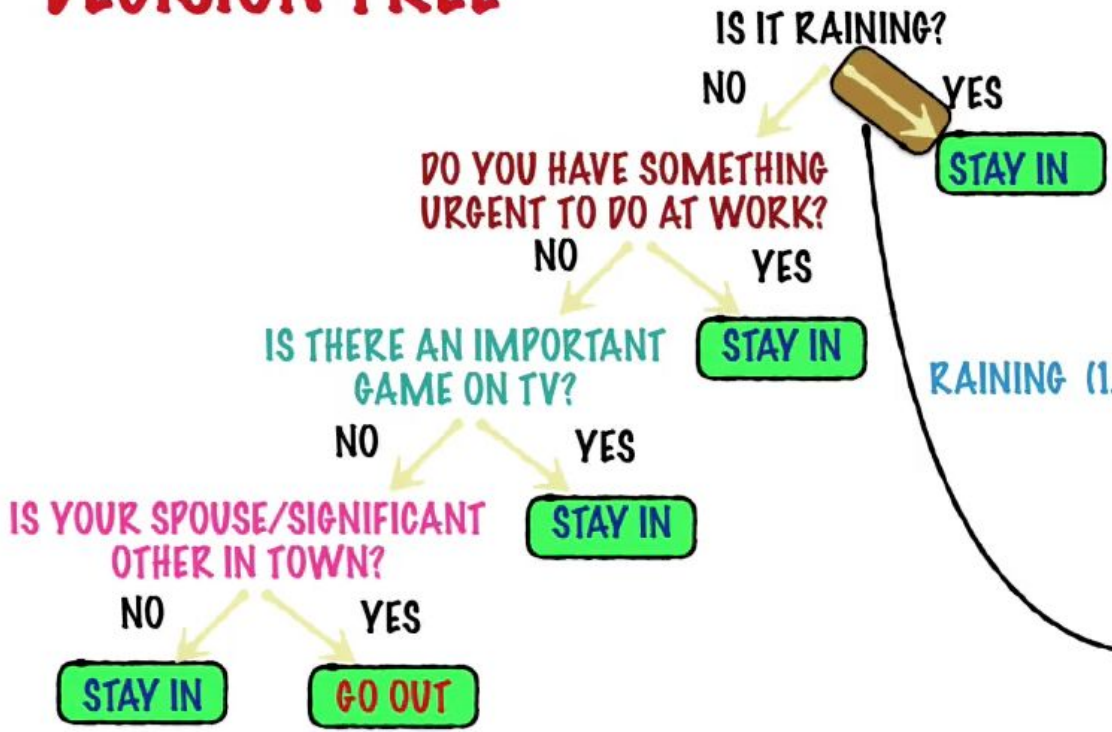
HOW MANY DAYS WILL A PATIENT STAY IN THE HOSPITAL?

DECISION TREE

THE LEAVES OF THE TREE ARE
THE OUTCOMES

THE BRANCHES ARE
THE COMBINATIONS OF
THE PREDICTOR VALUES
THAT LEAD TO A
PARTICULAR OUTCOME

DECISION TREE



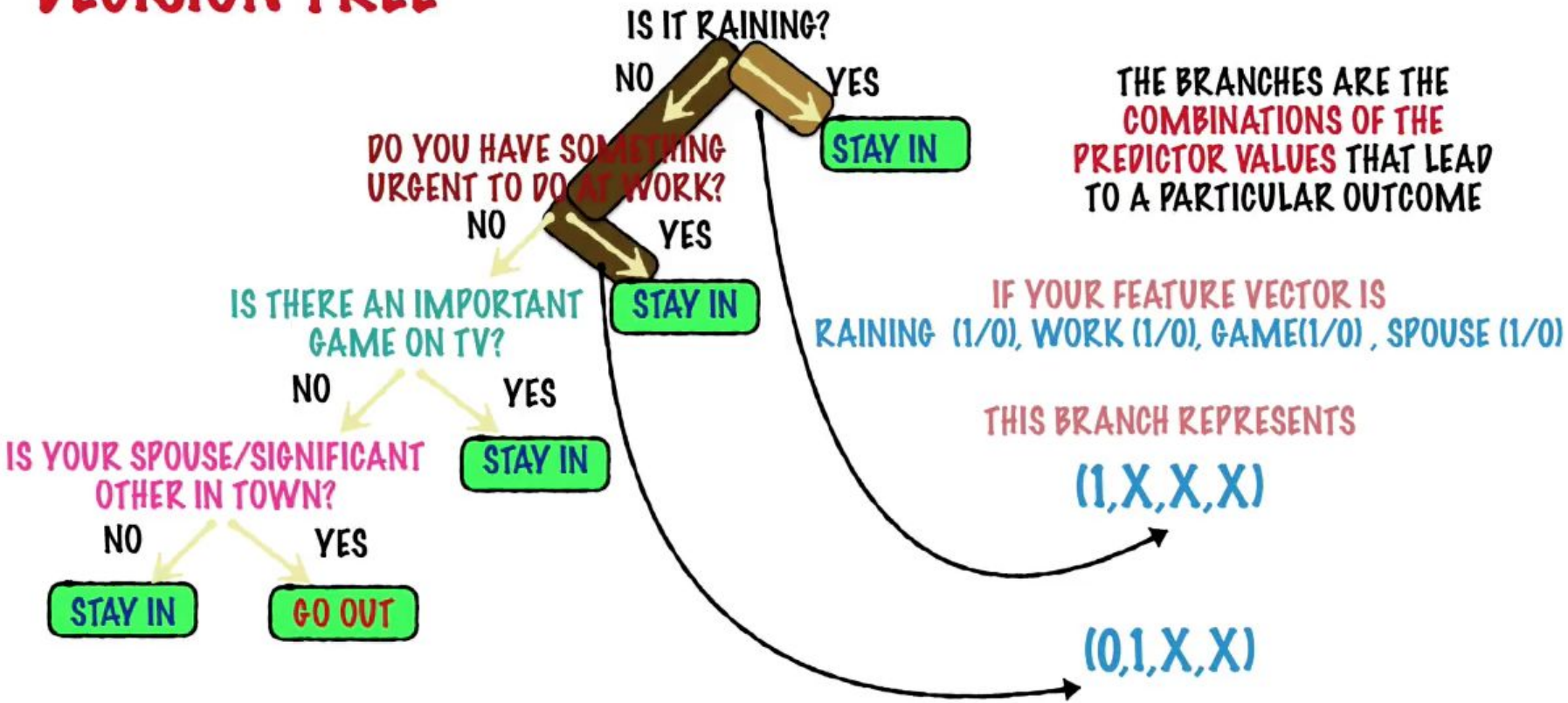
THE BRANCHES ARE THE COMBINATIONS OF THE PREDICTOR VALUES THAT LEAD TO A PARTICULAR OUTCOME

IF YOUR FEATURE VECTOR IS
RAINING (1/0), WORK (1/0), GAME(1/0) , SPOUSE (1/0)

THIS BRANCH REPRESENTS

(1,X,X,X)

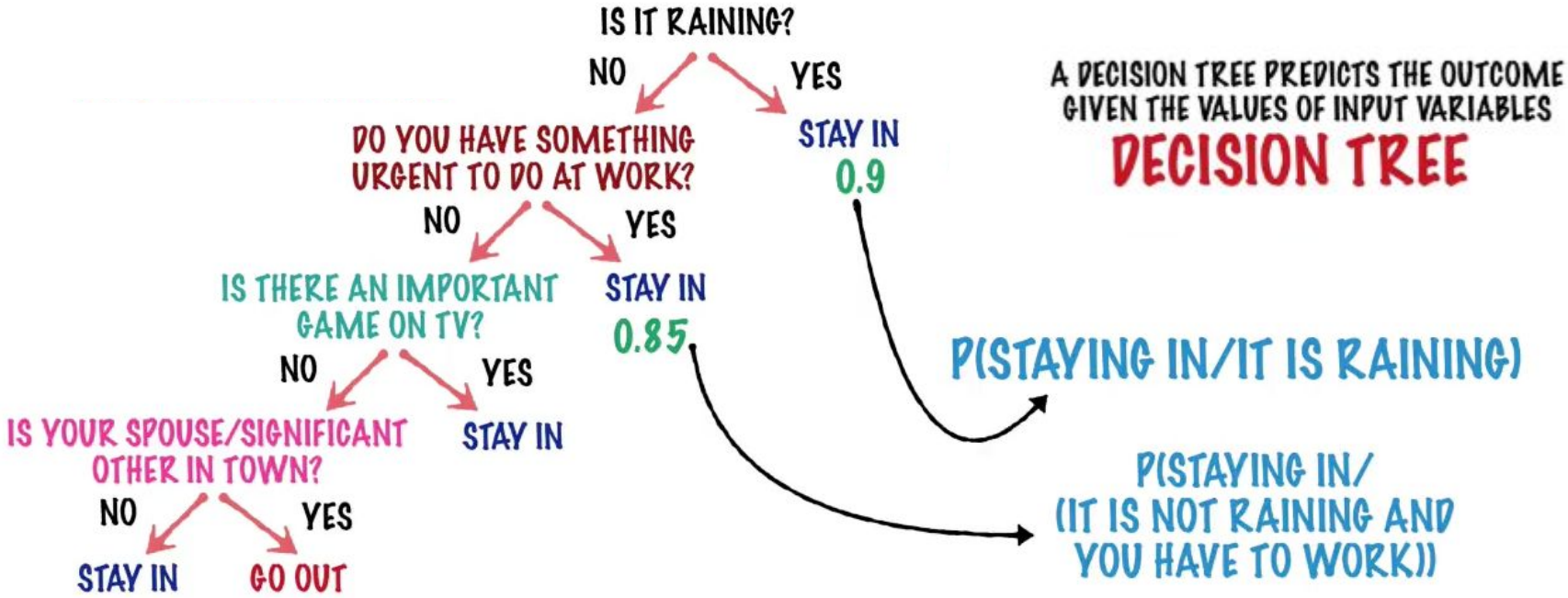
DECISION TREE



A DECISION TREE USUALLY ALSO GIVES US THE CONDITIONAL PROBABILITY OF THE OUTCOME GIVEN THE VALUES OF THE INPUT VARIABLES

GIVEN THE INPUT VALUES, THE OUTCOMES ARE NOT DETERMINISTIC

THE LEAVES OF THE DECISION TREE REPRESENT THE MOST LIKELY OUTCOME - THE ONE WITH THE HIGHEST CONDITIONAL PROBABILITY GIVEN THE VALUE OF THE VARIABLE


IS IT RAINING?
NO
YES
STAY IN
0.9
DO YOU HAVE SOMETHING URGENT TO DO AT WORK?
NO
YES
STAY IN
0.85
IS THERE AN IMPORTANT GAME ON TV?
NO
YES
STAY IN
IS YOUR SPOUSE/SIGNIFICANT OTHER IN TOWN?
NO
YES
STAY IN
GO OUT
A DECISION TREE PREDICTS THE OUTCOME GIVEN THE VALUES OF INPUT VARIABLES
DECISION TREE
P(STAYING IN/IT IS RAINING)
P(STAYING IN/ (IT IS NOT RAINING AND YOU HAVE TO WORK))

RECURSIVE PARTITIONING

IS THE MOST COMMON STRATEGY FOR DECISION TREE LEARNING

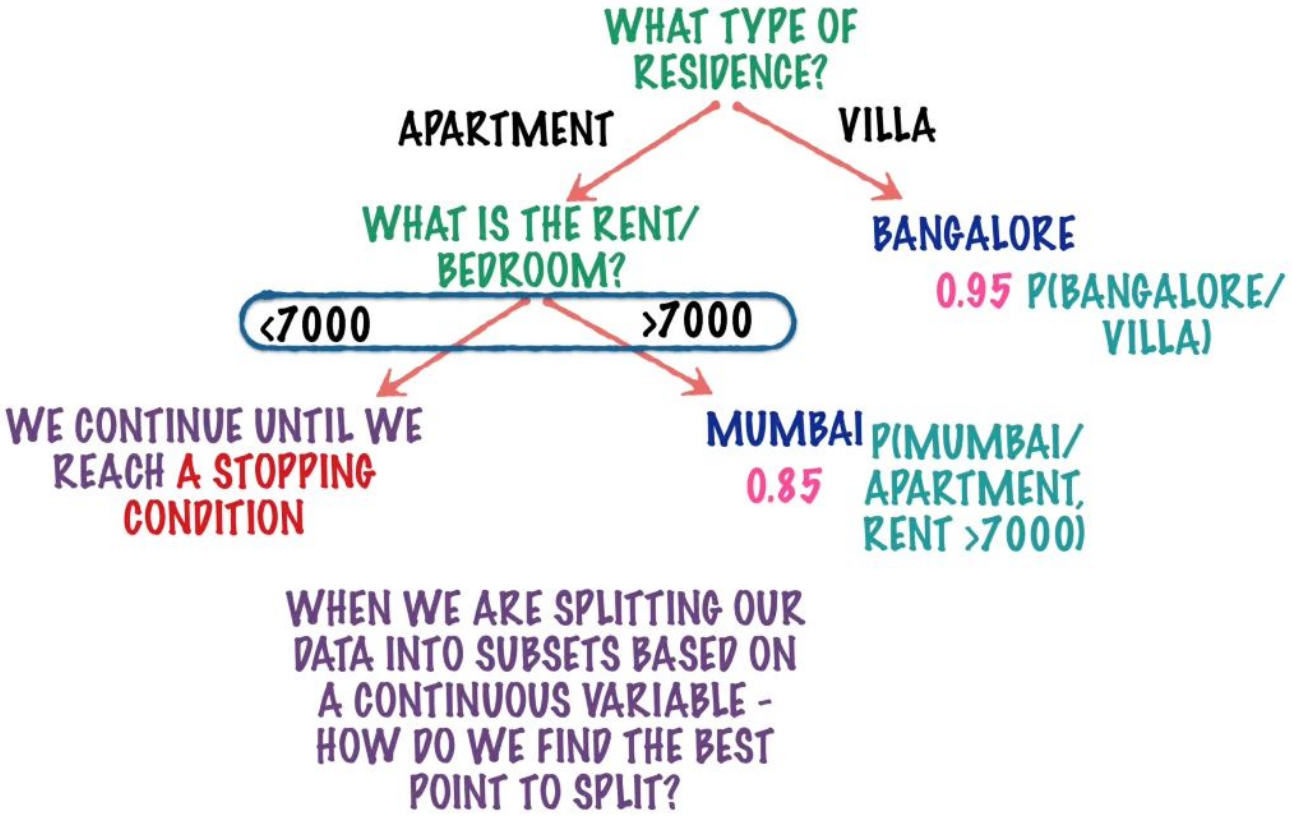
IT INVOLVES SPLITTING THE TRAINING DATA INTO SUBSETS BASIS THE INPUT VARIABLES/ATTRIBUTES

FOR EACH SPLIT, ONE ATTRIBUTE IS CHOSEN TO BE THE BASIS OF THE SPLIT

GIVEN THE
TYPE OF HOUSING,
RENT/BEDROOM AND
THE YEAR IT WAS BUILT

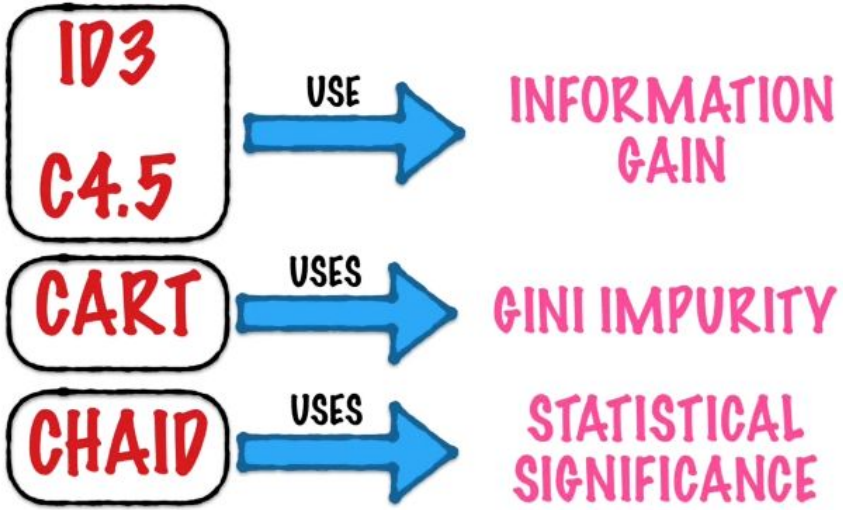
PREDICT THE CITY TO
WHICH A RESIDENCE
BELONGS

MUMBAI (OR)
BANGALORE


WHAT TYPE OF RESIDENCE?
APARTMENT
VILLA
WHAT IS THE RENT/ BEDROOM?
<7000
>7000
BANGALORE
0.95 P(BANGALORE/ VILLA)
WE CONTINUE UNTIL WE REACH A STOPPING CONDITION
MUMBAI
0.85 P(MUMBAI/ APARTMENT, RENT >7000)
WHEN WE ARE SPLITTING OUR DATA INTO SUBSETS BASED ON A CONTINUOUS VARIABLE - HOW DO WE FIND THE BEST POINT TO SPLIT?

DECISION TREE LEARNING ALGORITHMS BASED ON RECURSIVE PARTITIONING

EACH HAS A SLIGHTLY DIFFERENT WAY OF ARRIVING AT THE BEST ATTRIBUTE (OR) MEASURING THE HOMOGENEITY OF A SUBSET



INFORMATION GAIN

BOTH ID3 AND C4.5 WORK BY MAXIMIZING INFORMATION GAIN AT EACH STEP

THE IDEA OF INFORMATION GAIN IS TO REDUCE ENTROPY AND MAXIMIZE INFORMATION

C4.5 IMPROVES UPON ID3 - IT SUPPORTS CONTINUOUS VARIABLES AS WELL AS CATEGORICAL VARIABLES

ANY STATEMENT , NEWS OR MESSAGE CONTAINS INFORMATION

SOME HAVE MORE INFORMATION AND SOME LESS

LET'S SAY YOU HAVE TO CLASSIFY AN ANIMAL AS A GIRAFFE OR A HIPPO

LET'S SAY YOU HAVE TO CLASSIFY AN ANIMAL AS A GIRAFFE OR A HIPPO

IF YOU WERE TOLD, THIS ANIMAL HAS 4 LEGS

THIS IS BASICALLY USELESS! BOTH GIRAFFES AND HIPPOS HAVE 4 LEGS, SO THIS STATEMENT GIVES US NO INFORMATION

BUT IF YOU WERE TOLD, THIS ANIMAL IS 10 FEET TALL

THIS IS USEFUL INFORMATION!

IT TELLS YOU THAT THE ANIMAL
IS VERY LIKELY A GIRAFFE

SO, CLEARLY - THE VALUES OF SOME ATTRIBUTES GIVE US MORE INFORMATION THAN OTHERS

AND THERE IS A MATHEMATICAL WAY TO MEASURE THIS INFORMATION

COMPUTE THE INFORMATION GAIN FOR EACH ATTRIBUTE AND CHOOSE THE ATTRIBUTE WITH MAX INFORMATION GAIN AS THE DECISION TREE NODE

SO, CLEARLY - THE VALUES OF SOME ATTRIBUTES GIVE US MORE INFORMATION THAN OTHERS

AND THERE IS A MATHEMATICAL WAY TO MEASURE THIS INFORMATION

At each node, there is a certain amount of uncertainty or entropy.

As we know the value of each attribute, entropy is reduced.

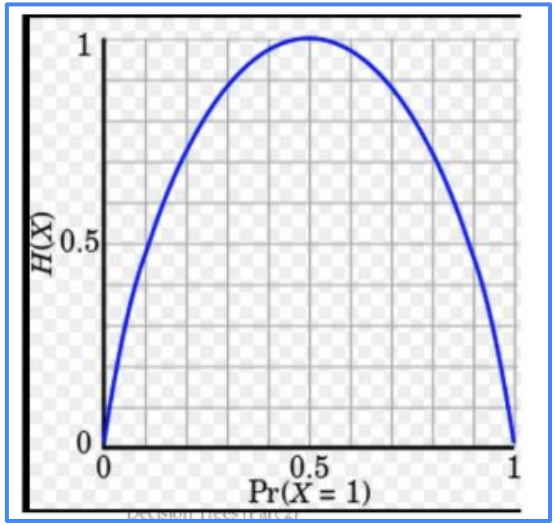
HOW TO COMPUTE

We compute the information gain or reduction in entropy for each attribute.

Then we choose the attribute with maximum information gain or most reduction in entropy.

$$\text{Entropy} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

ENTROPY



Formulas for computing entropy:

$$\text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n$$

$$H(X) = -\sum_x p(x) \log p(x)$$

Information Gain Methods are biased to choose attributes with more levels.

More levels inherently implies more entropy and hence knowing their value gives us more information.

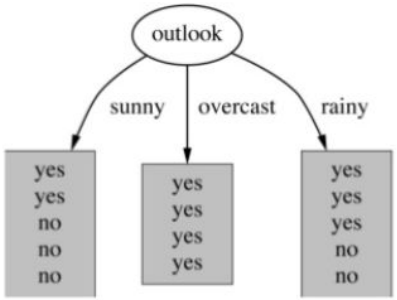
THE WEATHER DATASET

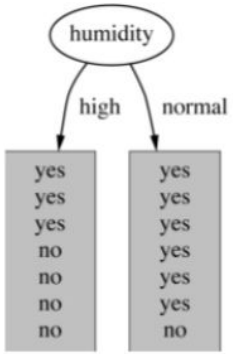
OUTLOOK	TEMP	HUMIDITY	WINDY	PLAY
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No%%

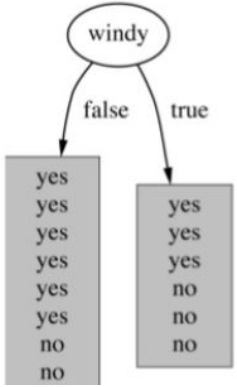
Entropy is a probabilistic measure of uncertainty. Information Gain is a measure of reduction of uncertainty.

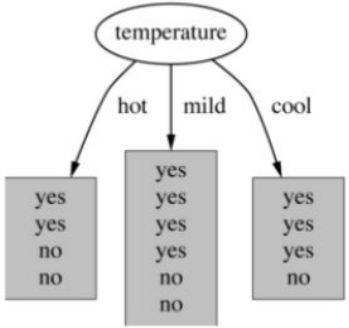
We use entropy to measure the purity of the node !

Which attribute to select?



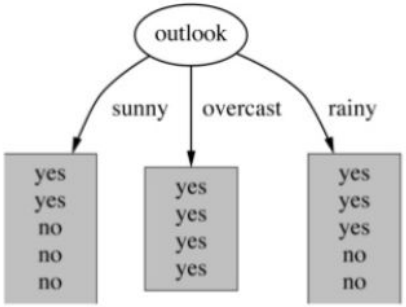


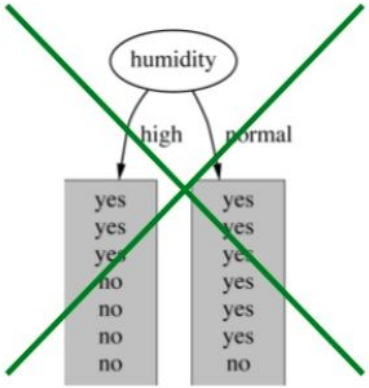


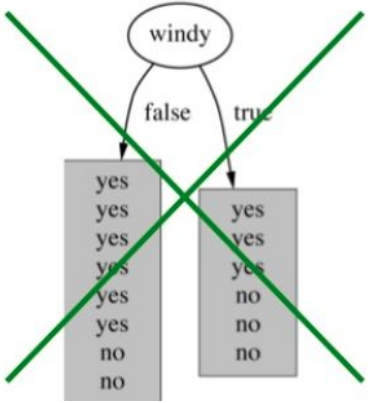


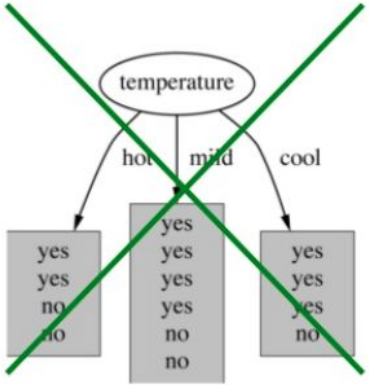
Which is the best attribute?

- Want to get the smallest tree
- Heuristic – Choose the attribute that produces the purest nodes.









Which is the best attribute?

- **Strategy** - Information gain increases with the average purity of the subsets.

- Choose attribute that gives the greatest information gain.

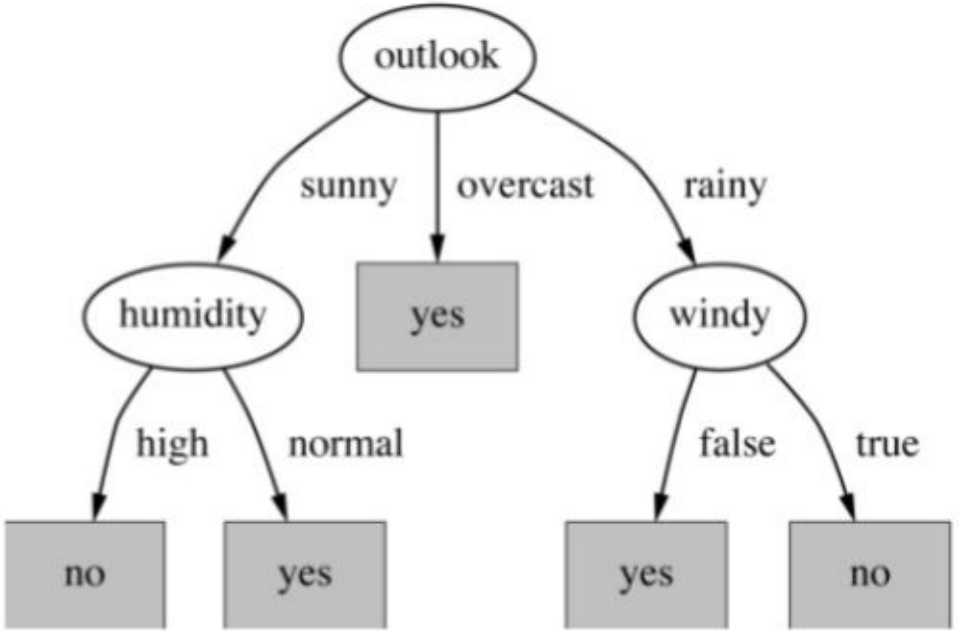
- Information gain: information before splitting – information after splitting

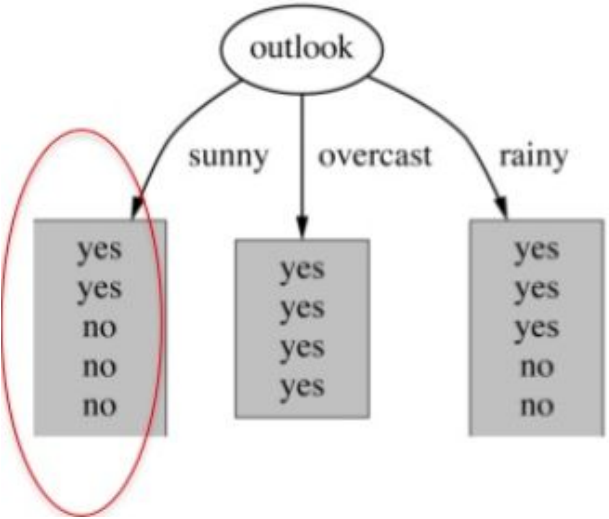
$$\begin{aligned}\text{gain}(\textit{Outlook}) &= \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) \\ &= 0.940 - 0.693 \\ &= 0.247 \text{ bits}\end{aligned}$$

- Information gain for attributes from weather data:

$$\begin{aligned}\text{gain}(\textit{Outlook}) &= 0.247 \text{ bits} \\ \text{gain}(\textit{Temperature}) &= 0.029 \text{ bits} \\ \text{gain}(\textit{Humidity}) &= 0.152 \text{ bits} \\ \text{gain}(\textit{Windy}) &= 0.048 \text{ bits}\end{aligned}$$

Final decision tree





$$\text{info}([2,3]) = -2/5 \times \log 2/5 - 3/5 \times \log 3/5 = 0.971 \text{ bits},$$

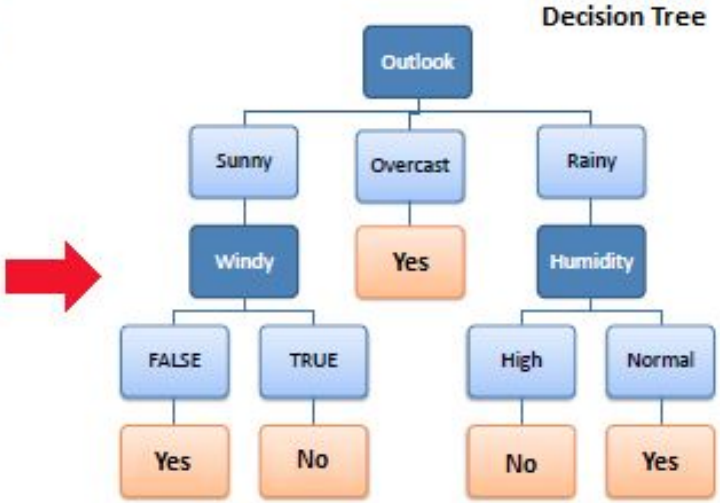
= (((-2) / 5) log2(2 / 5)) + (((-3) / 5) x log2(3 / 5)) = 0.97095059445

- *Outlook = Sunny* :
  info([2,3])=entropy(2/5,3/5)=−2/5log(2/5)−3/5log(3/5)=0.971bits
- *Outlook = Overcast* :
  info([4,0])=entropy(1,0)=−1log(1)−0log(0)=0bits
  Note: this is normally undefined.
- *Outlook = Rainy* :
  info([2,3])=entropy(3/5,2/5)=−3/5log(3/5)−2/5log(2/5)=0.971bits

EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

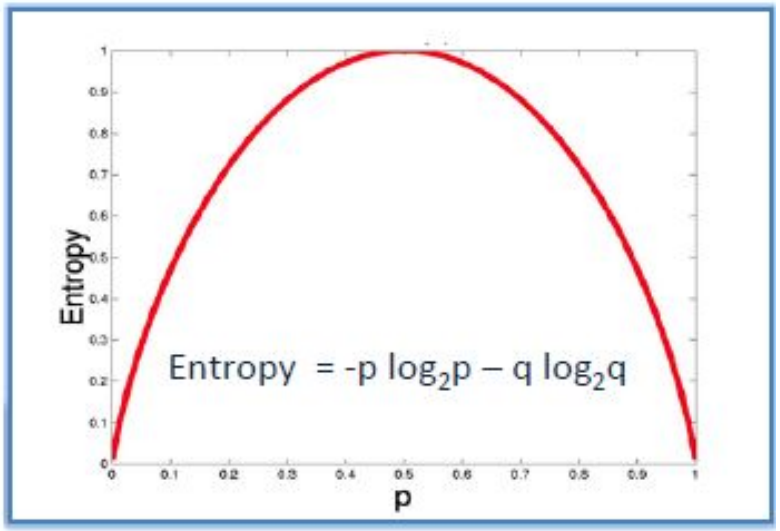
Remember : - ID3 algorithm uses entropy to calculate the homogeneity of a sample.
- If the sample is completely homogeneous the entropy is zero and if the sample is equally divided it has entropy of one.

Predictors				Target
Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

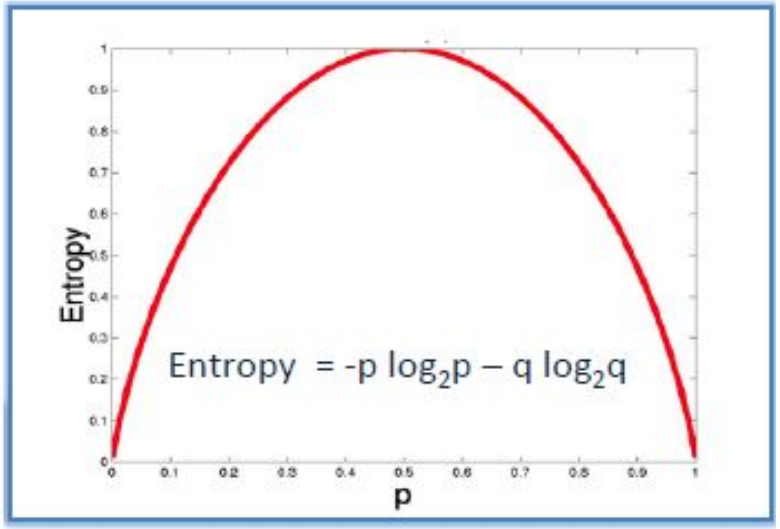
Remember : - ID3 algorithm uses entropy to calculate the homogeneity of a sample.
- If the sample is completely homogeneous the entropy is zero and if the sample is equally divided it has entropy of one.



Entropy = $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

Entropy is the negative of the sum of the probabilities of each outcome multiplied by the logarithm of probabilities for each outcome

What purpose does the logarithm serve in this equation?



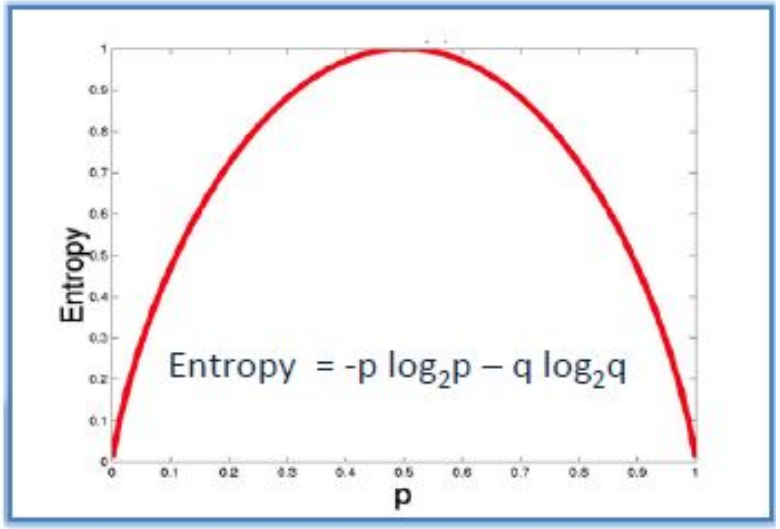
Entropy = -0.5 $\log_2 0.5$ – 0.5 $\log_2 0.5$ = 1

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

The binary logarithm ($\log_2 n$) is the power to which the number 2 must be raised to obtain the value n.

The reason for the base 2 in the logarithm is that here we're measuring the information in *bits*.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$



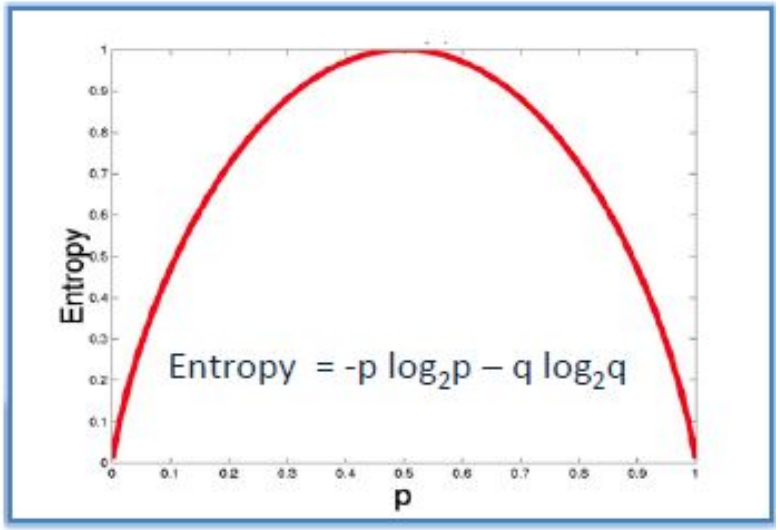
Entropy = -0.5 $\log_2$0.5 – 0.5 $\log_2$0.5 = 1

If all events happen with probability p, it means that there are 1/p events.

To tell which event have happened, we need to use log(1/p) bits

For example, if N=8 events, you need 3 bits to contain all the information content.

The binary logarithm function is the inverse function of the power of two function $\log_2$.



Entropy = -0.5 $\log_2$0.5 – 0.5 $\log_2$0.5 = 1

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

Entropy using the frequency table of one attribute:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Step 1 : Calculate the entropy of the target

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

Entropy using the frequency table of two attributes:

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

Step 2 : The dataset is then split on the different attributes. The entropy for each branch is calculated.

Then it is added proportionally, to get total entropy for the split.

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

Step 3 : The resulting entropy is subtracted from the entropy before the split.

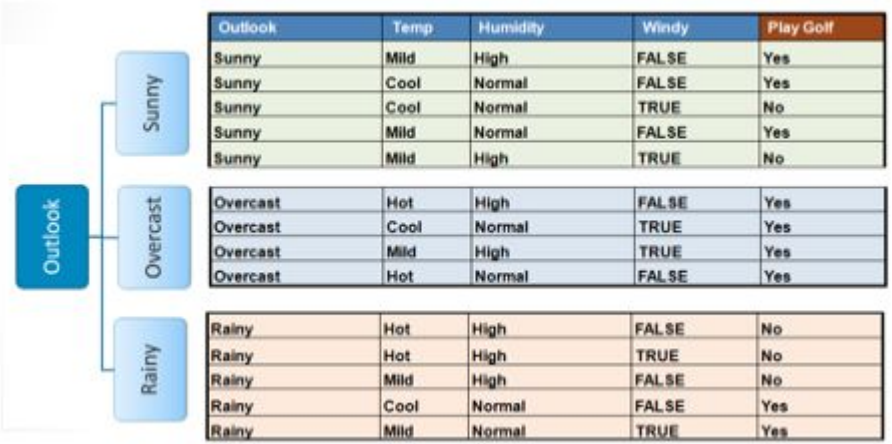
The result is the Information Gain, or decrease in entropy.

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			



Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Overcast	Hot	High	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes

Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes

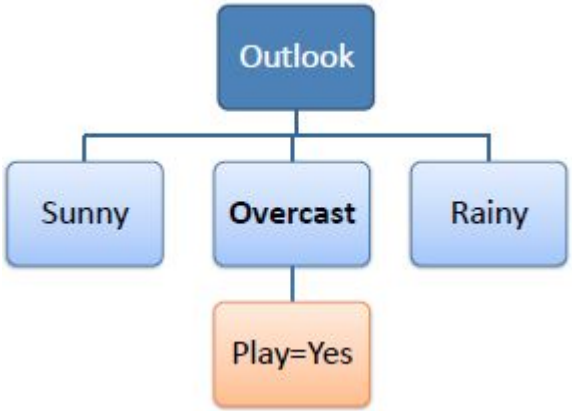
Step 4 : Choose attribute with the largest information gain as the decision node.

Divide the dataset by its branches and repeat the same process on every branch.

EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

Step 4a : A branch with entropy of 0 is a leaf node.

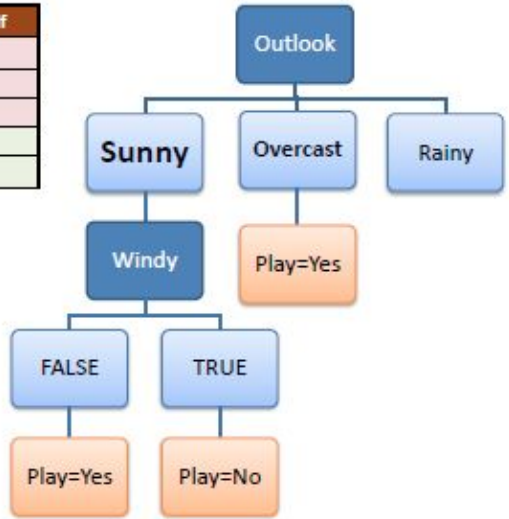
Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes



EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

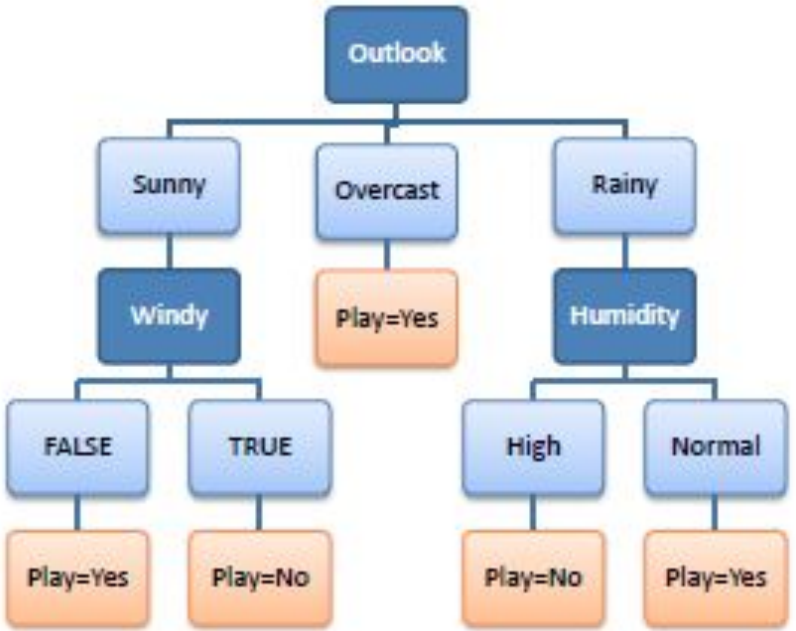
Step 4b : A branch with entropy more than 0 needs further splitting.

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



EXAMPLE : Using Entropy and Information Gain to build a Decision Tree

Step 5 : The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.



CART IS ANOTHER DECISION TREE LEARNING METHOD

(CLASSIFICATION AND REGRESSION TREES)

IT USES A DIFFERENT WAY TO CHOOSE AN ATTRIBUTE

MINIMIZING GINI IMPURITY

CHOOSE THE ATTRIBUTE SUCH THAT - IF YOU STOP THE DECISION TREE WITH THAT ATTRIBUTE AND GO NO FURTHER

THE PROBABILITY OF A FALSE LABEL IS MINIMIZED

Gini impurity is a measure of how impure a set is.

If you have a set of items, such as [A, A, B, B, B, C], then Gini impurity tells you the probability that you would be wrong if you picked one item and randomly guessed its label.

If the set were all As, you would always guess A and never be wrong, so the set would be totally pure.

$$G(k) = \sum_{i=1}^{J} P(i) * (1 - P(i))$$

Where P(i) is the probability of a certain classification i per the training data set.

"Will I Go Running" Data Set

Day	Weather	Just Ate	Late at Work	Will I go Running?
1	'Sunny'	'yes'	'no'	'yes'
2	'Rainy'	'yes'	'yes'	'no'
3	'Sunny'	'no'	'yes'	'yes'
4	'Rainy'	'no'	'no'	'no'
5	'Rainy'	'no'	'no'	'yes'
6	'Sunny'	'yes'	'no'	'yes'
7	'Rainy'	'no'	'yes'	'no'

In the data set above, there are two classes in which data can be classified:
"yes" (I will go running)
"no" (I will not go running)

```
G(will I go running) = P("yes") * 1 - P("yes") +
P("no") * 1 - P("no")

G(will I go running) = 4 / 7 * (1 - 4/7) + 3 / 7 *
1 - P(3/7)

G(will I go running) = 0.489796
```

This means there is a **48.97%** chance of a new data point being incorrectly classified, based on the observed training data we have at our disposal.

GINI GAIN

"Gini Gain is calculated when building a decision tree to help determine which attribute gives us the most information about which class a new data point belongs to."

$$Gini\_Gain(attribute) = total\_impurity - impurity\_remainder(attribute)$$

$$remainder(attribute) = \sum_{branch\_n} P(attribute\_branch\_n) * G(branch)_{branch}$$

(CHI-SQUARED AUTOMATIC INTERACTION DETECTOR) **CHAID** IS SLIGHTLY DIFFERENT FROM THE OTHER METHODS WE'VE SEEN

CHAID CHECKS WHETHER THE ATTRIBUTES/VARIABLES WE ARE USING ARE CORRELATED

IF THEY ARE CORRELATED, IT MERGES THEM TO CREATE ONE VARIABLE

IT ALSO PERFORMS STATISTICAL SIGNIFICANCE TESTS BEFORE SPLITTING THE DATA INTO SUBSETS

Example

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30

Decision Tree

Standard Deviation for Response Variable (one attribute)

Hours Played
25
30
46
45
52
23
43
35
38
46
48
52
44
30

$$S = \sqrt{\frac{\sum (x - \mu)^2}{n}}$$

Standard Deviation

S = 9.32

Decision Tree

Standard deviation (Hours Played) = 9.32

		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

		Hours Played (StDev)
Temp.	Cool	10.51
	Hot	8.95
	Mild	7.65
SDR=0.17		

$$SDR(T,X) = S(T) - S(T,X)$$

SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

= 9.32 – 7.66 = 1.66

		Hours Played (StDev)
Humidity	High	9.36
	Normal	8.37
SDR=0.28		

		Hours Played (StDev)
Windy	False	7.87
	True	10.59
SDR=0.29		

Decision Tree: *Algorithm*

Standard deviation (Hours Played) = 9.32

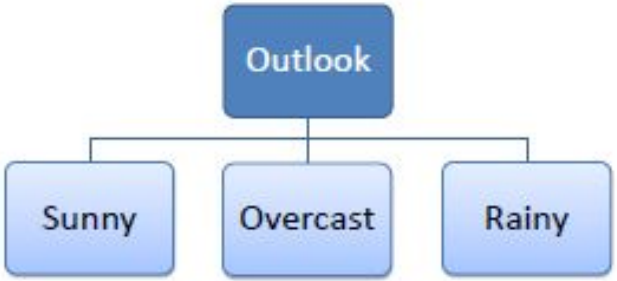
		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

$$SDR(T,X) = S(T) - S(T,X)$$

SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

= 9.32 – 7.66 = 1.66

Decision Tree: *Algorithm*



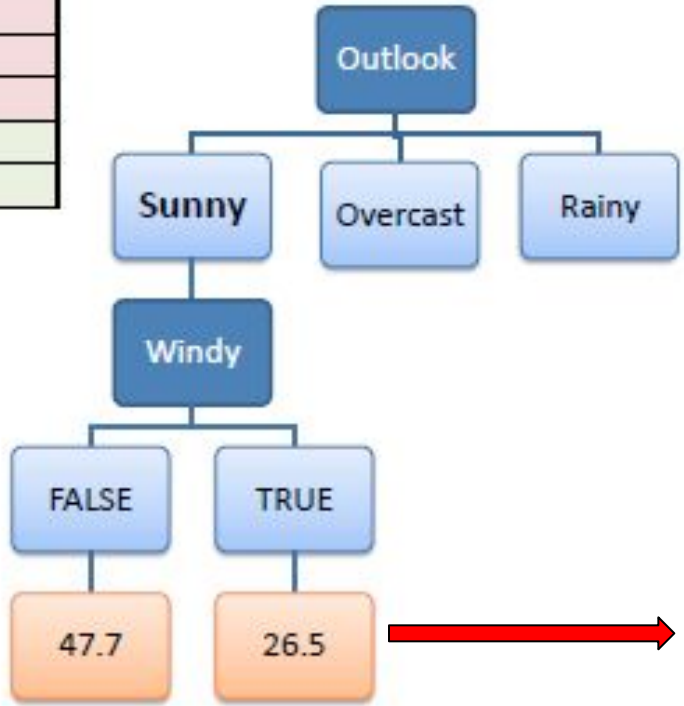
Outlook	Temp	Humidity	Windy	Hours Played
Sunny	Mild	High	FALSE	45
Sunny	Cool	Normal	FALSE	52
Sunny	Cool	Normal	TRUE	23
Sunny	Mild	Normal	FALSE	46
Sunny	Mild	High	TRUE	30
Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48
Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44

Decision Tree: *Algorithm*

Temp	Humidity	Windy	Hours Played
Mild	High	FALSE	45
Cool	Normal	FALSE	52
Mild	Normal	FALSE	46
Cool	Normal	TRUE	23
Mild	High	TRUE	30

★		Hours Played (StDev)
Windy	False	3.09
	True	3.50
SDR= 7.62		

SDR = 10.87-((3/5)*3.09 + (2/5)*3.5)

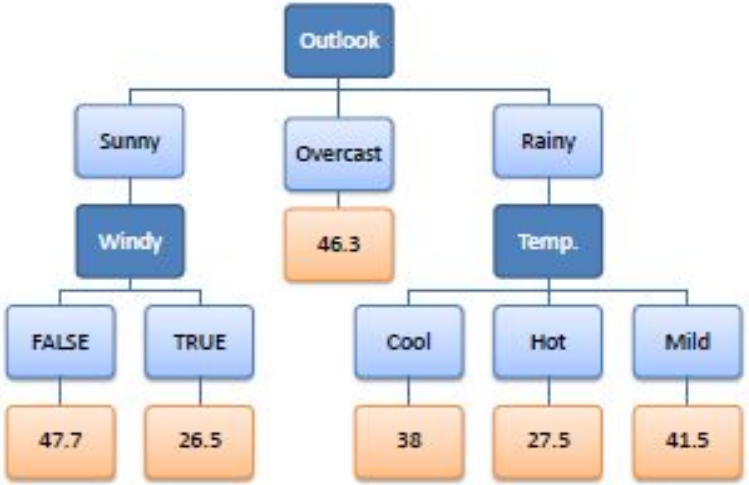


When the number of instances is more than one at a leaf node we calculate the *average* as the final value for the target.

Decision Tree: *Algorithm*

Predictors | Target

Outlook	Temp	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



The process is run recursively on the non-leaf branches, until all data is processed.

OVERFITTING IS THE BUGBEAR OF MACHINE LEARNING

SO WHAT IS OVERFITTING? AND WHY IS IT SUCH A PROBLEM?

CROSS VALIDATION

REGULARIZATION

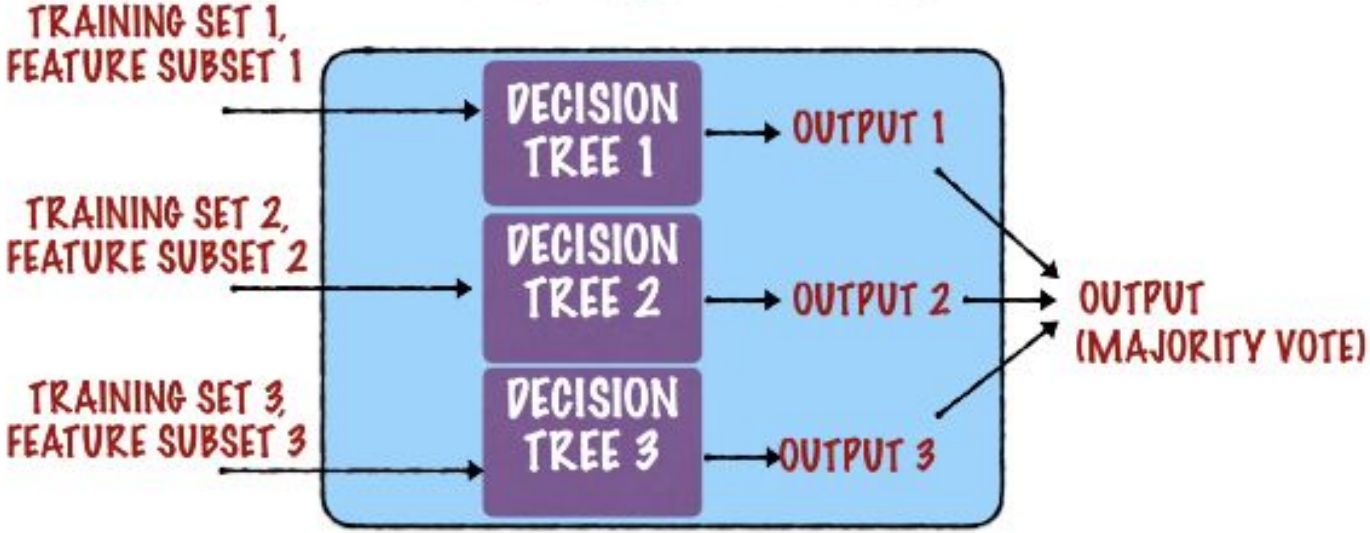
SOME OF THE WAYS TO MITIGATE THIS PROBLEM

ENSEMBLE LEARNING

ENSEMBLE LEARNING

INVOLVES THE USE OF MULTIPLE LEARNERS
AND COMBINING THEIR RESULTS

RANDOM FOREST



ENSEMBLE LEARNING

IN 2006, NETFLIX HELD AN OPEN COMPETITION FOR A MACHINE LEARNING ALGORITHM TO PREDICT A USER'S RATING OF A MOVIE

THE COMPETITION WENT ON FOR 3 YEARS, BEFORE A GRAND PRIZE WINNER WAS DECLARED

THE CONTESTANTS FOUND THAT, INSTEAD OF USING 1 SINGLE MODEL, COMBINING MULTIPLE MODELS WORKED BETTER

ENSEMBLE LEARNING

TEAMS STARTED MERGING INTO LARGER TEAMS, THEY WOULD COMBINE THEIR MODELS TO DO BETTER

IN THE END, THE GRAND PRIZE WINNER (AND A VERY CLOSE RUNNER UP) WERE BOTH ENSEMBLES OF MORE THAN A 100 LEARNERS EACH..

AND COMBINING THEM IMPROVED THE RESULTS EVEN FURTHER!

THE IDEA OF ENSEMBLE LEARNING

MODELS TEND TO OVERFIT

IF YOU TRAIN MULTIPLE MODELS

THE OVERFITTING COMPONENTS OF EACH OF THE MODELS WOULD BE DIFFERENT

WHEN YOU COMBINE THESE MODELS

THE OVERFITTING COMPONENTS OF THE MODELS WOULD CANCEL EACH OTHER OUT

AND YOU ARE LEFT WITH THE COMPONENTS THAT REALLY DESCRIBE YOUR DATA

LET'S TAKE AN EXAMPLE

CLASSIFY A TWEET AS POSITIVE OR NEGATIVE SENTIMENT
(THIS IS A CLASSIFICATION PROBLEM)

METHOD 1. CHOOSE 1 TECHNIQUE

NAIVE BAYES (OR) SUPPORT VECTOR MACHINES (OR) NEURAL NETWORKS

METHOD 2. USE AN ENSEMBLE

NAIVE BAYES (AND) SUPPORT VECTOR MACHINES (AND) NEURAL NETWORKS

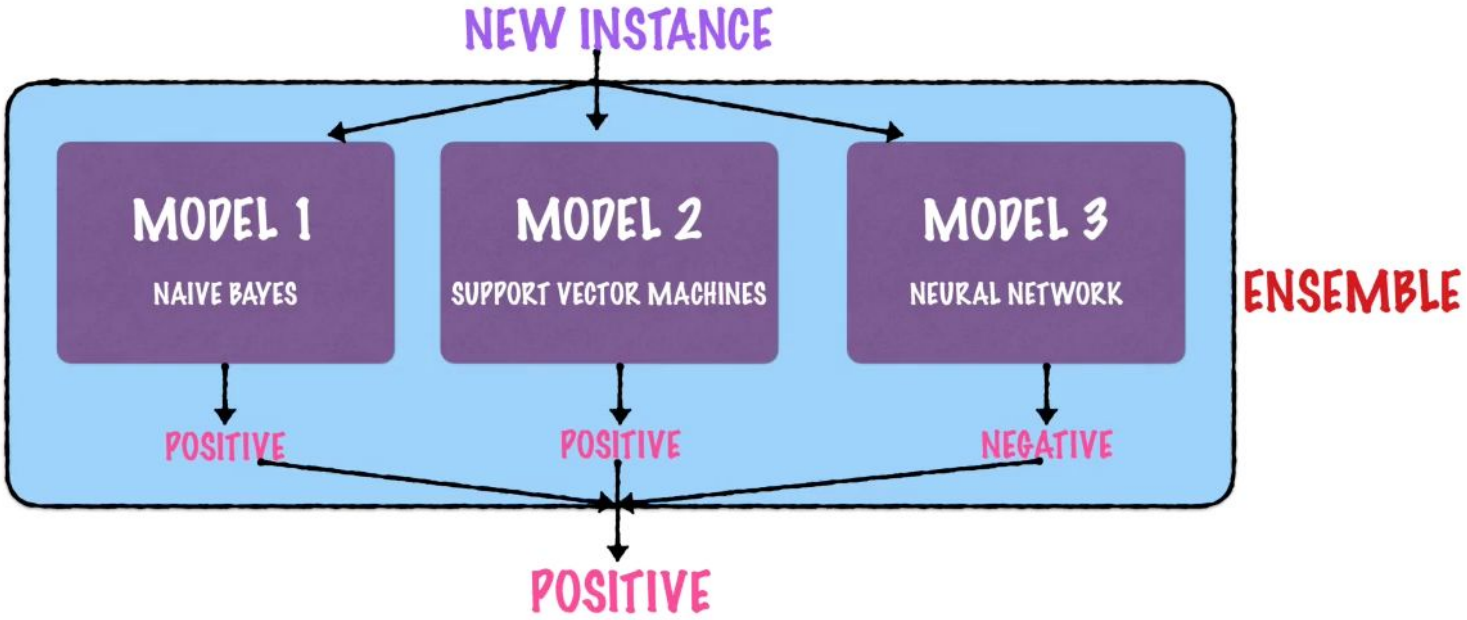
METHOD 2. USE AN ENSEMBLE

NAIVE BAYES (AND) SUPPORT VECTOR MACHINES (AND) NEURAL NETWORKS

1. TAKE THE TRAINING SET AND TRAIN EACH OF THE ABOVE CLASSIFIERS ON IT

2. WHEN A NEW INSTANCE (TWEET) COMES IN, GET THE PREDICTIONS FROM EACH OF THE MODELS

3. TAKE THE MAJORITY VOTE OF THE MODELS AND THAT WILL BE THE FINAL PREDICTION



A MACHINE LEARNING ENSEMBLE IS A COLLECTION OF MODELS

THE MODELS IN THE ENSEMBLE CAN BE

- BASED ON DIFFERENT TECHNIQUES
- TRAINED ON DIFFERENT TRAINING SETS
- USING DIFFERENT FEATURES
- USING DIFFERENT VALUES OF PARAMETERS

THE MODELS IN THE ENSEMBLE CAN BE

BASED ON DIFFERENT TECHNIQUES

A COLLECTION WITH 1 SVM, 1 DECISION TREE, 1 NAIVE BAYES, 1 KNN

TRAINED ON DIFFERENT TRAINING SETS

A COLLECTION OF SVMS, EACH TRAINED ON A DIFFERENT TRAINING SET

USING DIFFERENT FEATURES

A COLLECTION OF DECISION TREES, EACH GIVEN A DIFFERENT SET OF FEATURES

USING DIFFERENT VALUES OF PARAMETERS

A COLLECTION OF K-NEAREST NEIGHBOURS, EACH WITH A DIFFERENT VALUE OF K

SAY YOU HAVE 7 POSSIBLE FEATURES TO USE TO PREDICT SURVIVAL ON THE TITANIC

GENDER

AGE

PORT OF EMBARKATION

SIBLINGS

PARENTS

PASSENGER CLASS

FARE

EACH DECISION TREE IN THE RANDOM FOREST IS GIVEN A DIFFERENT SUBSET OF THESE 7 FEATURES TO LEARN FROM

THIS SUBSET IS RANDOMLY CHOSEN

SAY YOU HAVE 7 POSSIBLE FEATURES TO USE TO PREDICT SURVIVAL ON THE TITANIC

GENDER

AGE

PORT OF EMBARKATION

SIBLINGS

PARENTS

PASSENGER CLASS

FARE

EACH DECISION TREE IN THE RANDOM FOREST IS GIVEN A DIFFERENT SUBSET OF THESE 7 FEATURES TO LEARN FROM

THIS SUBSET IS RANDOMLY CHOSEN

AN ENSEMBLE LEARNER COMBINES THE RESULTS FROM INDIVIDUAL MODELS

AVERAGE OF THE RESULT FROM INDIVIDUAL MODELS

A WEIGHTED FUNCTION OF THE RESULT FROM INDIVIDUAL MODELS

A WEIGHTED FUNCTION OF THE RESULT FROM INDIVIDUAL MODELS

ONE OPTION IS TO WEIGHT EACH MODEL WITH IT'S ACCURACY

THIS MIGHT WORK, BUT IT ALSO RISKS OVERFITTING

THIS WEIGHTED FUNCTION CAN ALSO BE "LEARNED"

STACKING

INVOLVES TRAINING A LEARNER TO COMBINE THE RESULT FROM INDIVIDUAL MODELS

BAGGING

BOOSTING

ARE SPECIAL ENSEMBLE LEARNING TECHNIQUES

THEY INVOLVE CREATING MULTIPLE TRAINING SETS FROM THE MAIN TRAINING SET

BAGGING

(BOOTSTRAP-AGGREGATING)

EACH MODEL IN THE ENSEMBLE IS TRAINED
ON A DIFFERENT TRAINING SET

THESE TRAINING SETS ARE RANDOMLY
GENERATED FROM THE ORIGINAL TRAINING SET

FOR THE FINAL RESULT, EACH MODEL IS GIVEN AN
EQUAL WEIGHT AND A MAJORITY VOTE IS TAKEN

THESE TRAINING SETS ARE RANDOMLY GENERATED FROM THE ORIGINAL TRAINING SET

THE TRAINING SETS ARE GENERATED USING A STATISTICAL TECHNIQUE KNOWN AS

BOOTSTRAP SAMPLING

UNIFORM SAMPLING WITH REPLACEMENT

IS KNOWN AS **BOOTSTRAP SAMPLING**

BAGGING

(BOOTSTRAP-AGGREGATING)

IS AN ENSEMBLE LEARNING TECHNIQUE THAT USES BOOTSTRAP SAMPLING TO CREATE MULTIPLE TRAINING SETS

BOOSTING

IS AN ALGORITHM FOR ITERATIVELY ADDING LEARNERS TO THE ENSEMBLE

EACH LEARNER IN THE ENSEMBLE WILL PERFORM VERY POORLY BY ITSELF

WEAK LEARNERS

THE THEORY OF BOOSTING IS THAT AN ENSEMBLE OF WEAK LEARNERS CAN TOGETHER BE VERY STRONG

ADABOOST

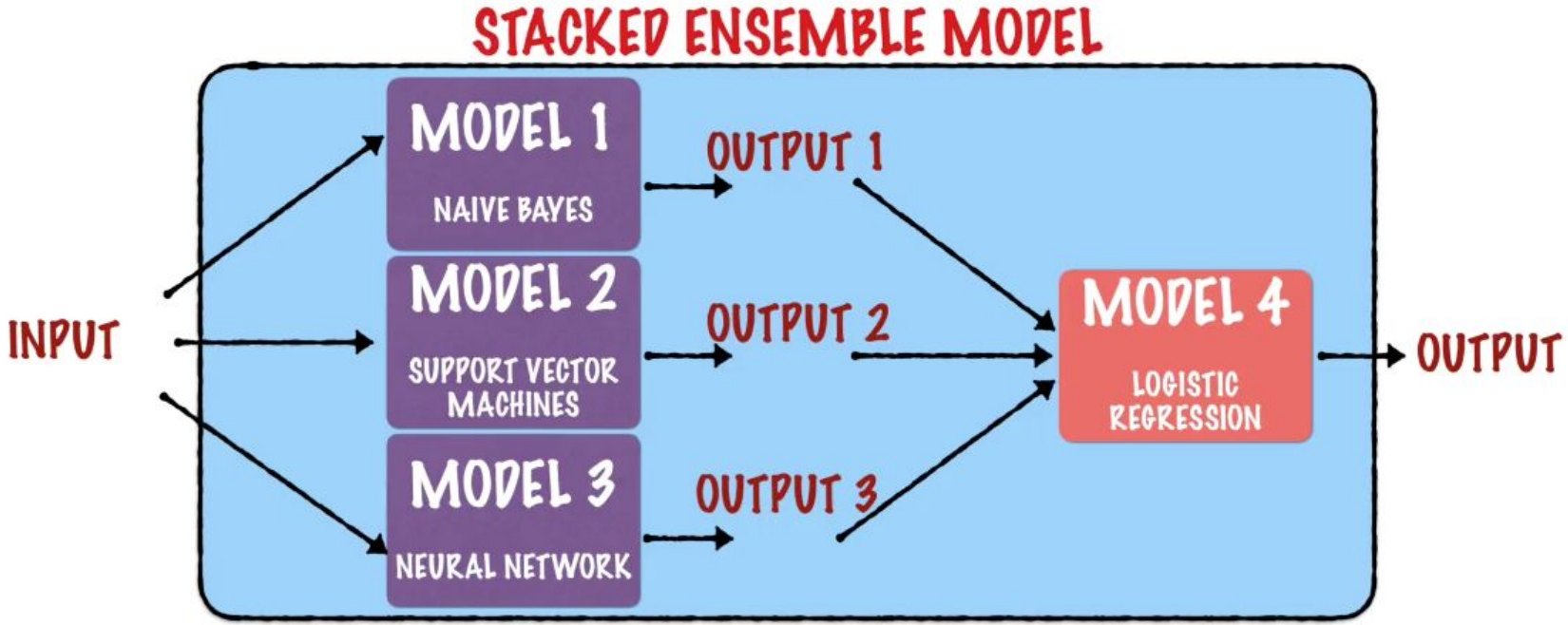
IS THE MOST WELL KNOWN VARIANT OF A BOOSTING ALGORITHM

WHEN PREDICTING FOR A NEW INSTANCE, ADABOOST USES AN OPTIMALLY WEIGHTED VOTE OF THE ENSEMBLE LEARNERS

STACKING

(AKA BLENDING AKA STACKED GENERALIZATION)

INVOLVES USING A MACHINE LEARNING APPROACH TO COMBINE THE RESULTS OF THE ENSEMBLE MEMBERS

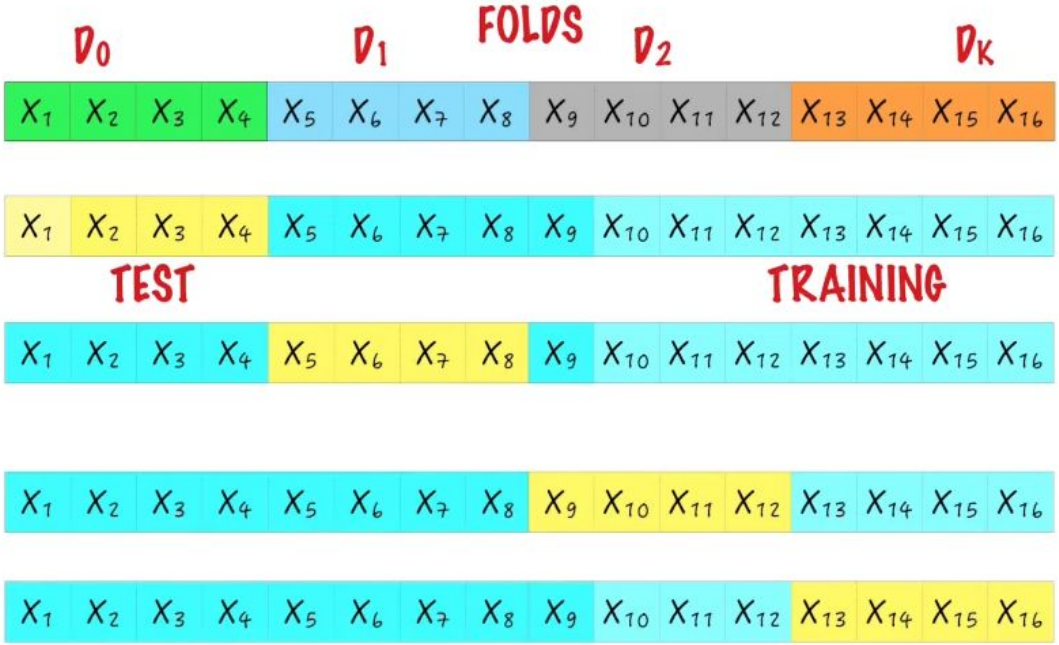


2-FOLD CROSS VALIDATION

1. DIVIDE THE TRAINING SET RANDOMLY INTO TWO EQUAL PARTS - D_0 AND D_1

K-FOLD CROSS VALIDATION

1. DIVIDE THE TRAINING SET RANDOMLY INTO K EQUAL PARTS - $D_0, D_1, D_2, D_3, \ldots D_K$



2. USE D_0 TO TEST THE MODEL PERFORMANCE AND THE REST TO TRAIN THE DATA

3. USE D_1 TO TEST THE MODEL PERFORMANCE AND THE REST TO TRAIN THE DATA

4. CONTINUE UNTIL EACH OF THE PARTS HAS BEEN USED FOR TESTING EXACTLY ONCE

PRUNING

IS THE MOST POPULAR WAY TO AVOID THIS PROBLEM

THIS INVOLVES REMOVING SOME OF THE NODES OF YOUR DECISION TREE AND REPLACING THEM WITH A LEAF

PRUNING

IS THE MOST POPULAR WAY TO AVOID THIS PROBLEM

THIS INVOLVES REMOVING SOME OF THE NODES OF YOUR DECISION TREE AND REPLACING THEM WITH A LEAF

IN GENERAL PRUNING IS PERFORMED BY REMOVING A NODE/SUB-TREE AND CHECKING WHETHER THE ACCURACY OF PREDICTION IS AFFECTED

IF THE ACCURACY IF NOT AFFECTED, THAT NODE/SUB-TREE IS PRUNED

REGULARIZATION

PENALIZES MODELS WHICH ARE TOO COMPLEX

FINDING A MODEL USUALLY INVOLVES MINIMIZING AN ERROR FUNCTION

FOR EXAMPLE, THE ERROR FUNCTION COULD BE THE SUM OF SQUARES OF DISTANCES BETWEEN THE PREDICTED POINTS AND THE ACTUAL POINTS IN THE TRAINING SET

LET THE ERROR FUNCTION BE E(f) FOR A MODEL f

LET THE ERROR FUNCTION BE E(f) FOR A MODEL f

A REGULARIZATION TERM IS ADDED TO THIS FUNCTION

$$E'(f) = E(f) + \lambda R(f)$$

NEW ERROR FUNCTION THAT NEEDS TO BE MINIMIZED

A PARAMETER THAT CONTROLS THE IMPORTANCE OF THE REGULARIZATION TERM

REGULARIZATION TERM THAT INCREASES WITH COMPLEXITY

WE GET A MODEL THAT GIVES LOW ERROR ON THE TRAINING SET, WHILE KEEPING THE COMPLEXITY LOW AS WELL

RECAP

ENSEMBLE LEARNING

INVOLVES THE USE OF MULTIPLE LEARNERS AND COMBINING THEIR RESULTS

THE IDEA OF ENSEMBLE LEARNING IS SIMPLE..

MODELS TEND TO OVERFIT

IF YOU TRAIN MULTIPLE MODELS

THE OVERFITTING COMPONENTS OF EACH OF THE MODELS WOULD BE DIFFERENT

WHEN YOU COMBINE THESE MODELS
THE OVERFITTING COMPONENTS OF THE MODELS WOULD CANCEL EACH OTHER OUT

AND YOU ARE LEFT WITH THE COMPONENTS THAT REALLY DESCRIBE YOUR DATA

Naive Bayes

1. Used for probabilistic binary and multiclass classification
2. Predicts probability of each class
3. Often Applied to Text Classification
4. Commonly used for Text Classification Tasks

1. Naïve Bayes classifier is based on Bayesian inference.
2. Bayesian Inference rule relates to conditional and marginal probabilities.
3. It shows how the conditional (posterior) probability of an event can be calculated based on its marginal (prior) probability and the inverse conditional probability.

1. For two events A and B, the rule can be written as

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

P(A) ⇒ is the prior probability of A
P(A|B) ⇒ is the conditional (posterior) probability of A given B
P(B|A) ⇒ is the conditional probability of B given A
P(B) ⇒ is the probability of B.

$$P(A_i \cap A_j) = 0 \quad \text{for } i \neq j \qquad \sum_{i=1}^{k} P(A_i) = 1$$

Mutually Exclusive Events

$$A_1, A_2, \ldots, A_k$$

$$P(A_i \cap A_j) = 0 \quad \text{for } i \neq j$$

$$\sum_{i=1}^{k} P(A_i) = 1$$

Typically, A_i represent a set of alternative *hypotheses*, and B represents some available *evidence* that may affect their probability.

B

$$P(A_i \cap A_j) = 0 \quad \text{for } i \neq j$$

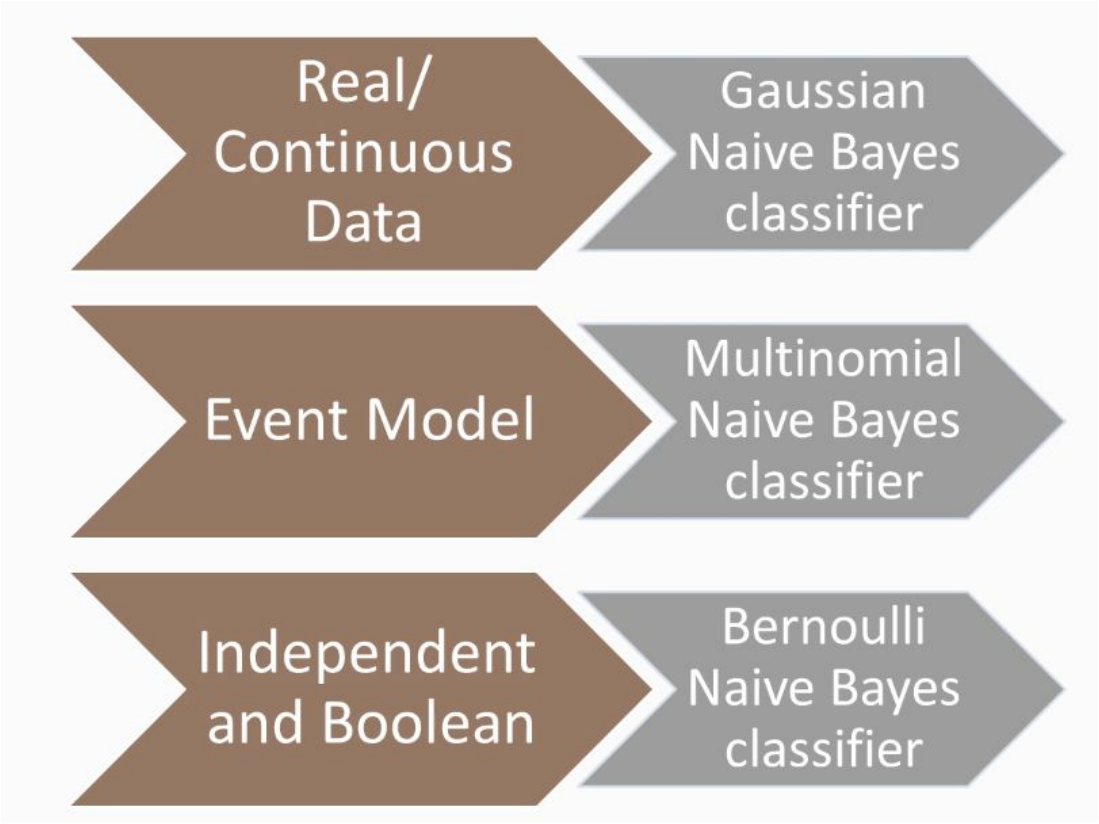
$$\sum_{i=1}^{k} P(A_i) = 1$$

From the law of total probability,

$$P(B) = \sum_{j=1}^{k} P(A_j)P(B|A_j)$$

which allows one to rewrite the Bayes rule as

$$P(A_i|B) = \frac{P(A_i)P(B|A_i)}{\sum_{j=1}^{k} P(A_j)P(B|A_j)}$$


Real/ Continuous Data
Gaussian Naive Bayes classifier
Event Model
Multinomial Naive Bayes classifier
Independent and Boolean
Bernoulli Naive Bayes classifier

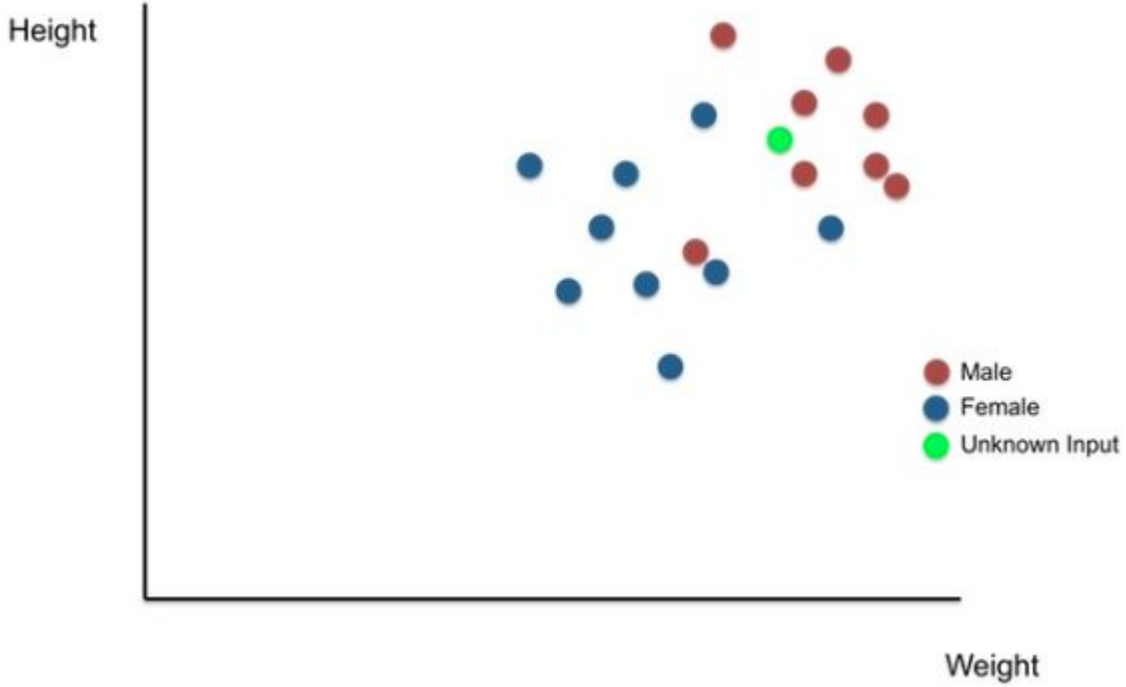
In [16]:
```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)
from sklearn.metrics import classification_report
print (classification_report(Y_test, Y_pred))
```

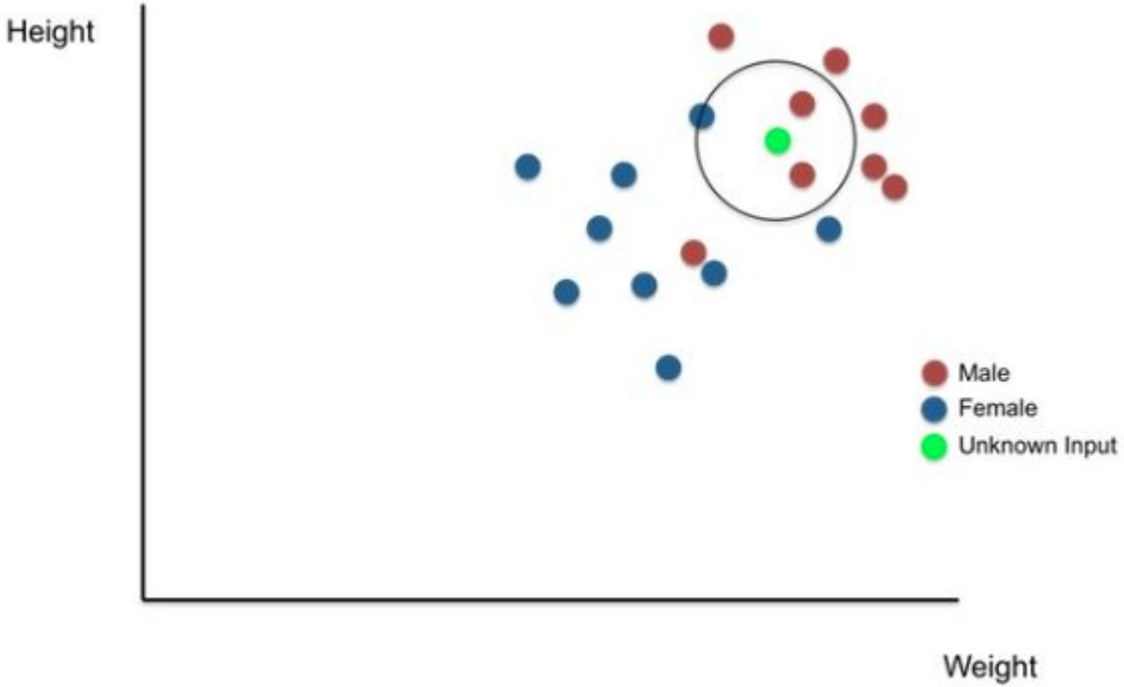
```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        11
          1       0.93      1.00      0.96        13
          2       1.00      0.83      0.91         6

avg / total       0.97      0.97      0.97        30
```

K-Nearest Neighbour


Height
Weight
Male
Female
Unknown Input


Height
Weight
Male
Female
Unknown Input

Classification by just looking at K closest examples

Popular Target – classification Label

Two Parameters

- Neighbourhood cardinality (K)
- Some measure to Evaluate the Similarity

Does not work well with large data sets

PROPERTIES OF K-NN

- k-NN is non-parametric – it does not make any assumptions about the probability distribution of the input.
- Useful for applications with input properties that are unknown
- However, Parametric algorithms tend to produce fewer errors than non-parametric ones, since taking input probabilities into account can influence decision making.

Lazy Learning and Eager Learning

- k-NN is a type of lazy learning, which is a learning method that generalizes data in the testing phase, rather than during the training phase.
- Also known as instance based learning.
- Eager learning, generalizes data in the training phase rather than the testing phase.
- Benefit of lazy learning is that it can quickly adapt to changes, since it is not expecting a certain generalized dataset.
- Major downside is that a huge amount of computation occurs during testing (actual use) rather than pre-computation during training.

Classification and Regression

- K-nearest neighbors can be used in classification or regression machine learning tasks.
- Classification involves placing input points into appropriate categories whereas regression involves establishing a relationship between input points and the rest of the data.
- Determining a neighbor can be performed using many different notions of distance.
- The most common being **Euclidean** and **Hamming** distance.

Euclidean and Hamming Distance

- Euclidean distance represents the length of a straight line between two points.

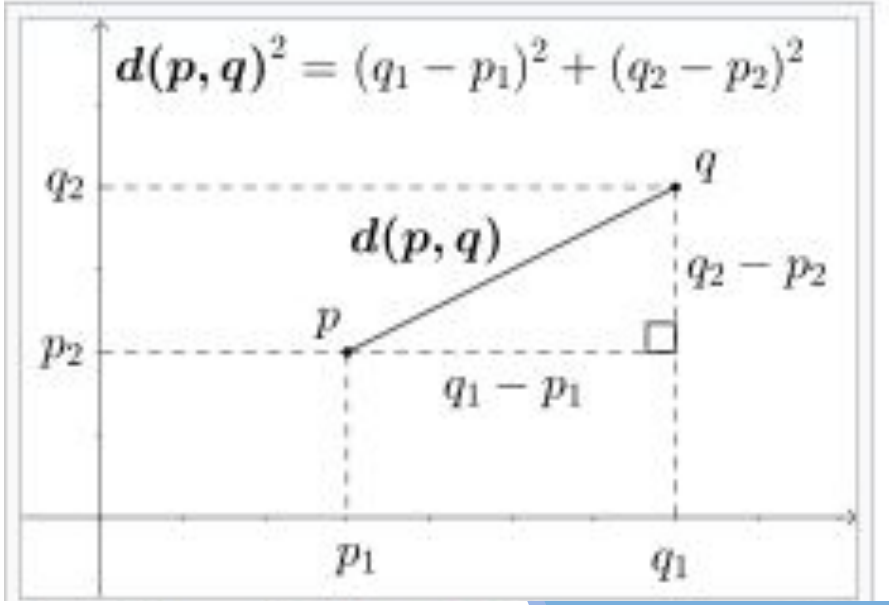
- Hamming distance is calculated by the minimum number of changes required to make to two strings to make them equal.

- Technically, It measures the minimum number of substitutions required to change one string into the other, or the minimum number of errors that could have transformed one string into the other.

Euclidean Distance

One Dimensional

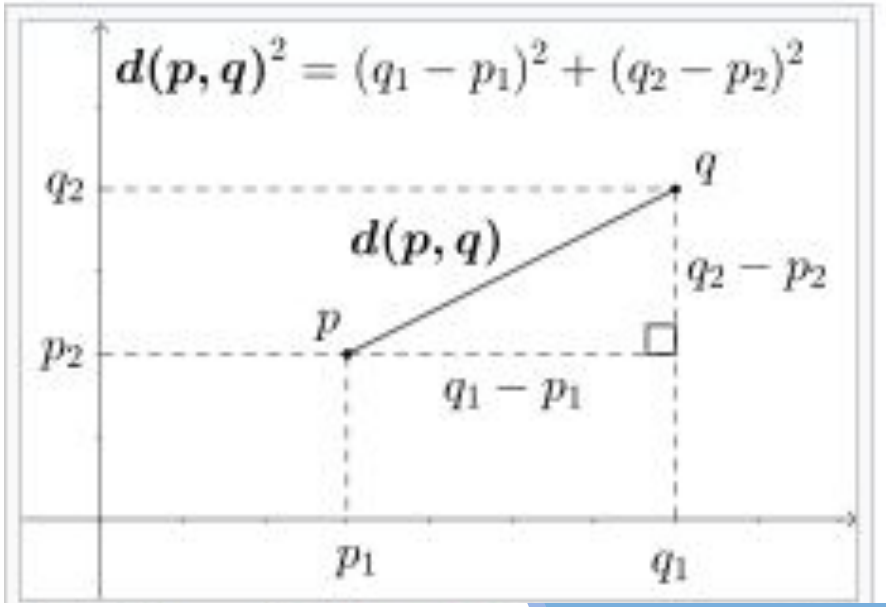
$$\sqrt{(q-p)^2} = |q-p|.$$



Euclidean Distance

Two Dimensional

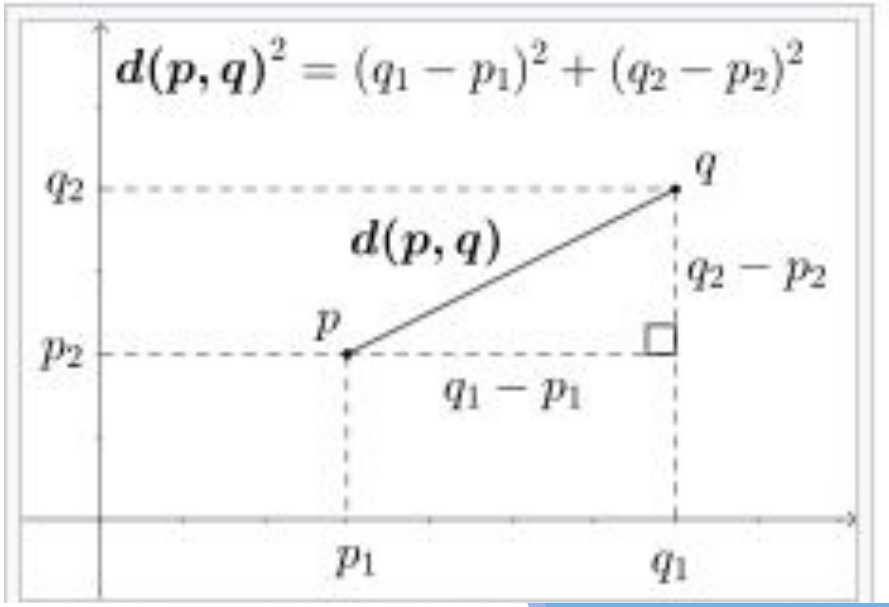
$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$



Euclidean Distance

Three Dimensional

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}.$$



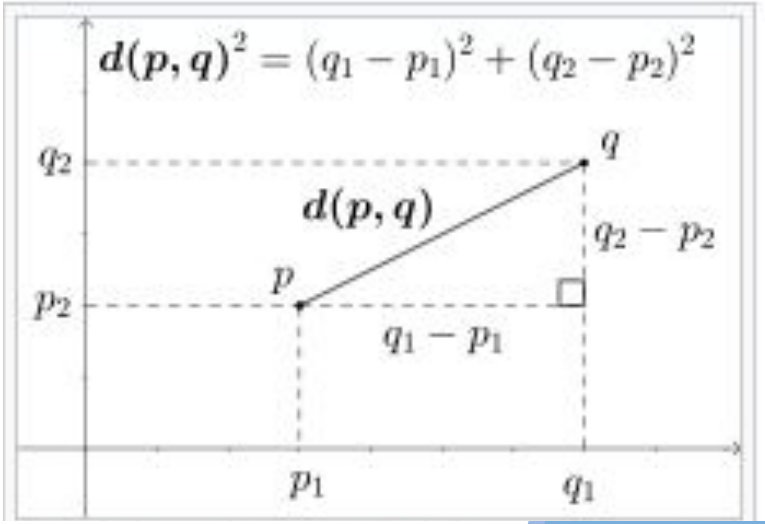
Euclidean Distance

n Dimensional

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}.$$

Squared Euclidean Distance

It is frequently used in optimization problems in which distances only have to be compared.



$$d^2(\mathbf{p}, \mathbf{q}) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2.$$

Example : KNN CLASSIFICATION

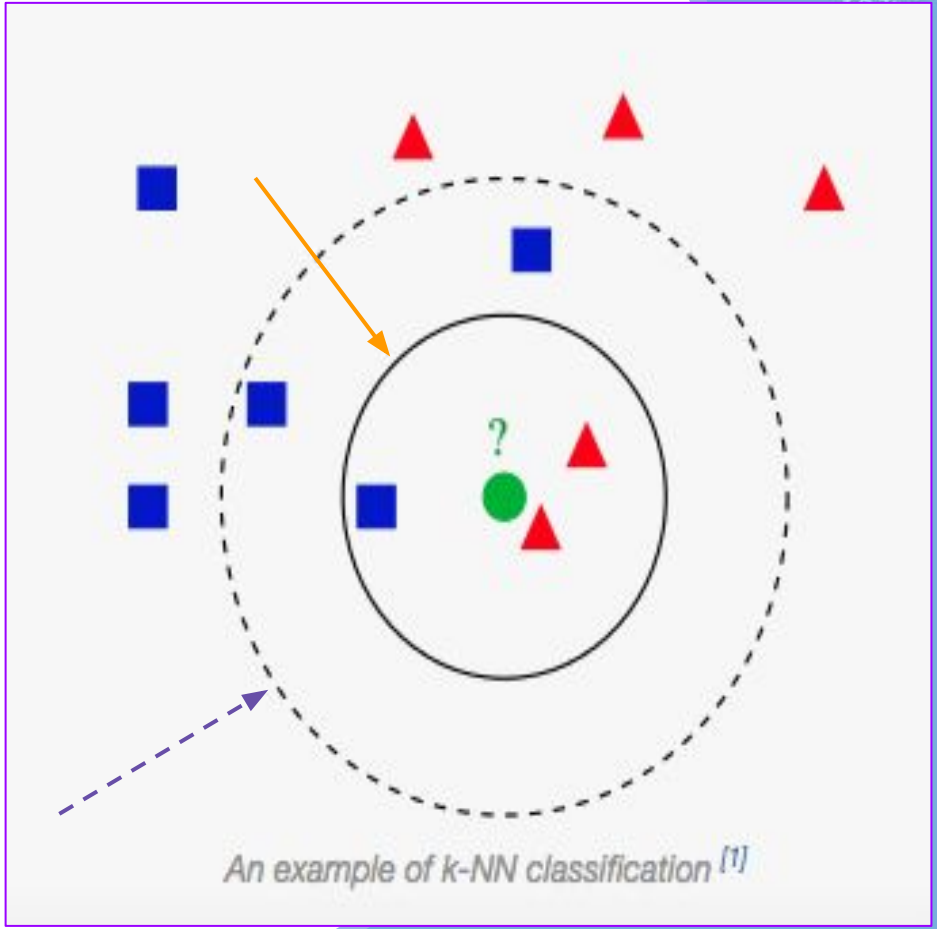
- For k-NN classification, an input is classified by a majority vote of its neighbors.
- The algorithm obtains the class membership of its k neighbors and outputs the class that represents a majority of the k neighbors.

Example

Suppose we are trying to classify the green circle.

Let us begin with (the solid line). In this case, the algorithm would return a red triangle, since it constitutes a majority of the 3 neighbors.

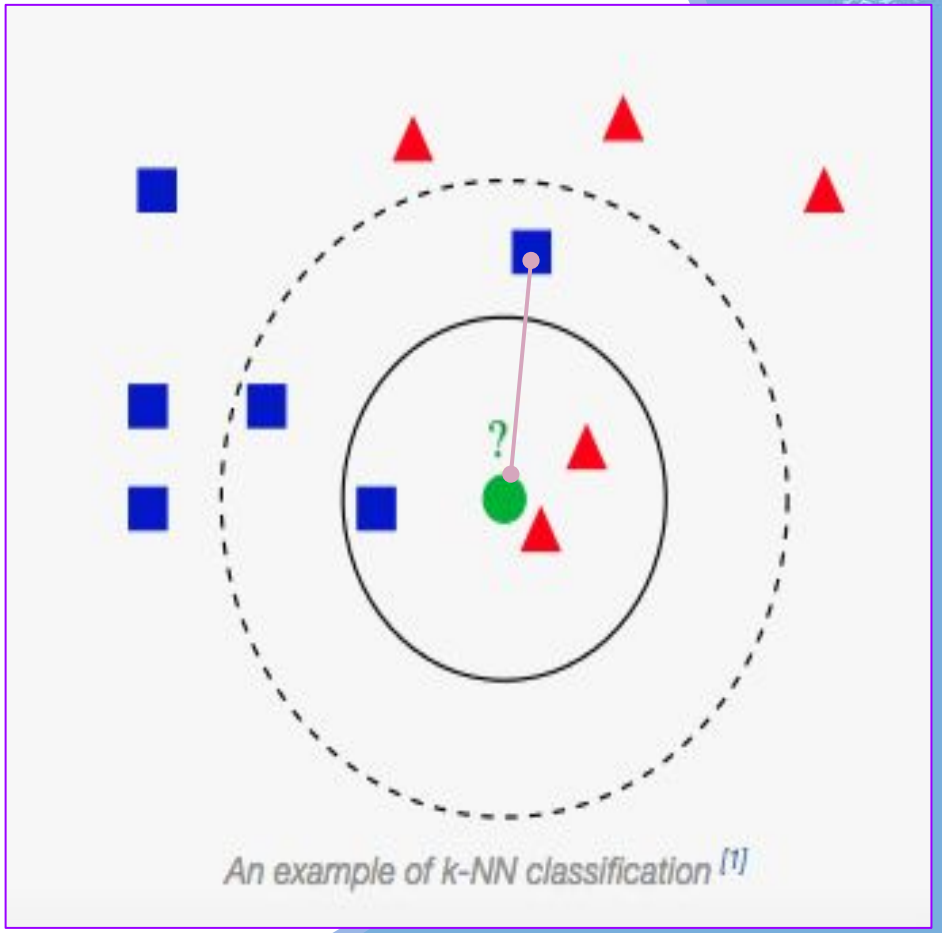
Likewise, with (the dotted line), the algorithm would return a blue square.



An example of k-NN classification [1]

Example

- If no majority is reached with the k neighbors, many courses of action can be taken.
- For example, one could use a plurality system or even use a different algorithm to determine the membership of that data point.
- A majority is more than half. A plurality is the greatest number for one but less than half.

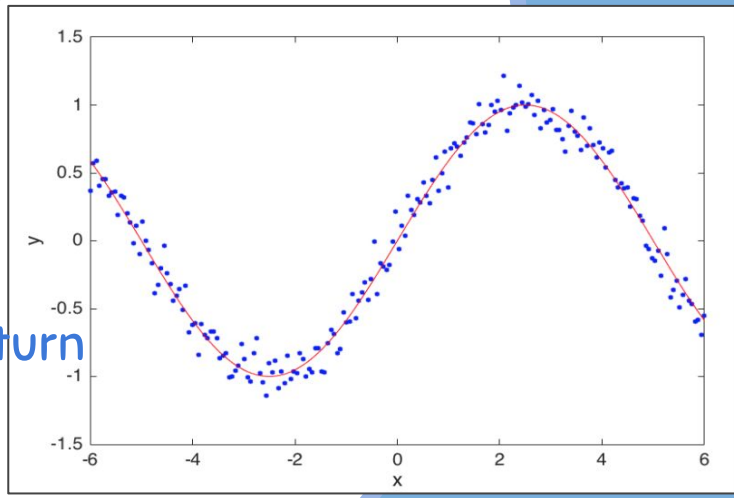


An example of k-NN classification [1]

Example : KNN REGRESSION

- The value returned is the average value of the input's k neighbors.

- If k-NN were asked to return the corresponding y value for x=0, the algorithm would find the k nearest points to x=0 and return the average y value corresponding to these k points.



In [ ]:
```
from sklearn.utils import shuffle
from sklearn.cross_validation import train_test_split

#imports the mnist object previously pickled and dumped to disk
import pickle
mnist = pickle.load(open( "mnist.pickle", "rb" ))

mnist.data, mnist.target = shuffle(mnist.data, mnist.target)
```

In [19]:
```
# We reduce the dataset size, otherwise it'll take too much time to run
mnist.data = mnist.data[:1000]
mnist.target = mnist.target[:1000]
X_train, X_test, Y_train, Y_test = train_test_split(mnist.data,
                    mnist.target, test_size=0.8, random_state=0)
```

In [20]:
```
from sklearn.neighbors import KNeighborsClassifier
# KNN: K=10, default measure of distance (euclidean)
clf = KNeighborsClassifier(3)
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)

from sklearn.metrics import classification_report
print (classification_report(Y_test, Y_pred))
```

	precision	recall	f1-score	support
0.0	0.80	0.92	0.85	72
1.0	0.66	1.00	0.79	111
2.0	0.90	0.74	0.82	86
3.0	0.66	0.75	0.70	68
4.0	0.58	0.84	0.69	76
5.0	0.89	0.44	0.59	55
6.0	0.92	0.92	0.92	74
7.0	0.79	0.79	0.79	97
8.0	0.87	0.57	0.69	80
9.0	0.64	0.31	0.42	81
avg / total	0.77	0.74	0.73	800

```
In [21]: %timeit clf.fit(X_train, Y_train)
         100 loops, best of 3: 2.98 ms per loop

In [22]: %timeit clf.predict(X_test)
         1 loop, best of 3: 245 ms per loop
```

Applications

- Computer vision
- Gene Expression Analysis

Pros

- Very easy to understand and implement.
- Does not assume any probability distributions on the input data.
- Can quickly respond to changes in input.

Cons

- Sensitive to localized data.
  a. Since k-NN gets all of its information from the input's neighbors, localized anomalies affect outcomes significantly, rather than for an algorithm that uses a generalized view of the data.
- Computation time.
  a. Lazy learning requires that most of kNN computation be done during testing, rather than during training. This can be an issue for large datasets.
- Normalization
  a. if one type of category occurs much more than another, classifying an input will be more biased towards that one category (since it is more likely to be neighbors with the input).
- Dimensions.
  a. In the case of many dimensions, inputs can commonly be "close" to many data points. This reduces the effectiveness of k-NN, since the algorithm relies on a correlation between closeness and similarity

Dimensionality Reduction

- Dimensionality reduction includes a set of techniques to help deal with the problem of the curse of dimensionality.

- These techniques are aimed at reducing the number of variables to be considered by the models we build.

- This can be achieved using feature extraction and feature selection

- The term feature extraction is associated with techniques that seek to build a dataset derived and transformed from the original data.

- The most used techniques to reduce the dimensionality is Principal Components Analysis or PCA.

Principal Component Analysis

Principal Component Analysis

This technique finds the causes that explain the variability of a dataset and lists them in the order of their importance.

Principal Component Analysis

The PCA searches for a projection where the data is best represented.

Principal Component Analysis

PCA performs a linear transformation to obtain a new coordinate system in which the largest variance is considered as the first principal component, the second largest variance as the second principal component, and so on...

An advantage of principal component analysis as a dimensionality reduction method is that it preserves the characteristics of the dataset that contribute most to its variance.

Another advantage connected with reduced dimensions is that the behavior of the dataset is easier to observe in 2D and 3D systems, which makes it more interpretable by humans.

Computing Principal Components

Principal Component Analysis

Ways in which you can calculate PCA :

Using a correlation matrix: For data which is not dimensionally uniform about the magnitude of the variables.

Using a covariance matrix: For data that is dimensionally uniform about the magnitude of the variables and have similar mean values.

Principal Component Analysis

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

vs.

$$var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

Principal Component Analysis

Below is a data about the performance of 10 students in five classes: Maths, Science, English, History, and Sports. Each of these subjects is as a dimension, that is, we will work with a dataset of five dimensions

5 Dimensions

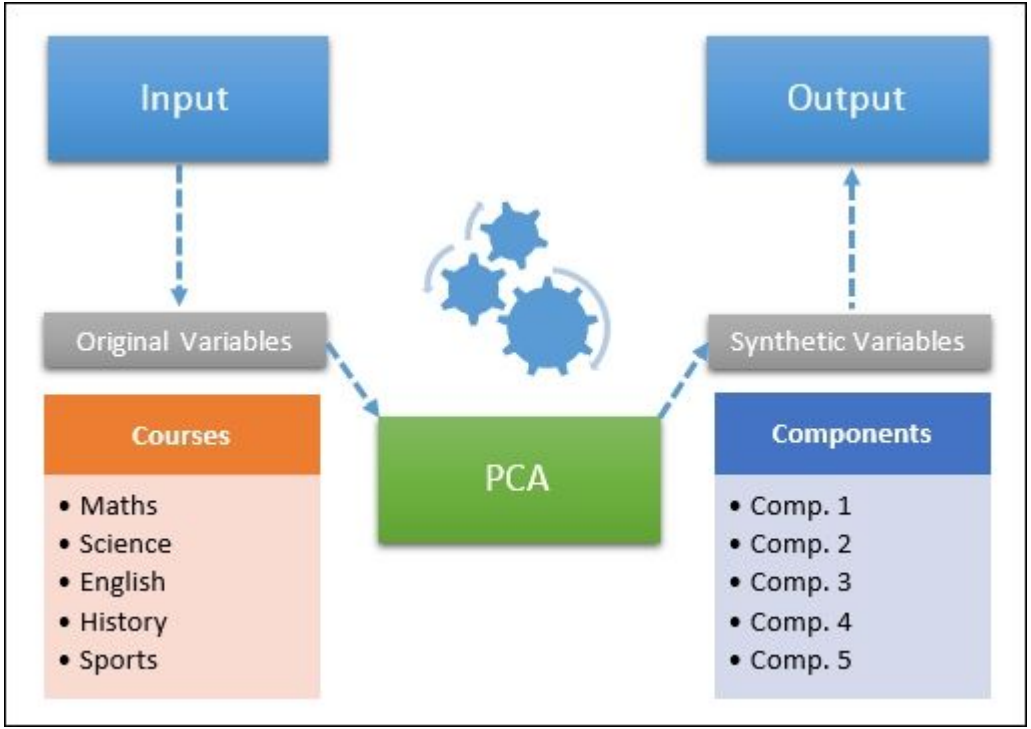
10 Observations

Student	Maths	Science	English	History	Sports
Rosa	7	6,5	9,2	8,6	8
Denis	7,5	9,4	7,3	7	7
Edgar	7,6	9,2	8	8	7,5
Yeison	5	6,5	6,5	7	9
Silvia	6	6	7,8	8,9	7,3
Arturo	7,8	9,6	7,7	8	6,5
Elizabeth	6,3	6,4	8,2	9	7,2
Erik	7,9	9,7	7,5	8	6
Dante	6	6	6,5	5,5	8,7
Sasha	6,8	7,2	8,7	9	7

Principal Component Analysis

	Maths	Science	English	History	Sports
Rosa	7.0	6.5	9.2	8.6	8.0
Denis	7.5	9.4	7.3	7.0	7.0
Edgar	7.6	9.2	8.0	8.0	7.5
Yeison	5.0	6.5	6.5	7.0	9.0
Silvia	6.0	6.0	7.8	8.9	7.3
Arturo	7.8	9.6	7.7	8.0	6.5
Elizabeth	6.3	6.4	8.2	9.0	7.2
Erik	7.9	9.7	7.5	8.0	6.0
Dante	6.0	6.0	6.5	5.5	8.7
Sasha	6.8	7.2	8.7	9.0	7.0

Principal Component Analysis

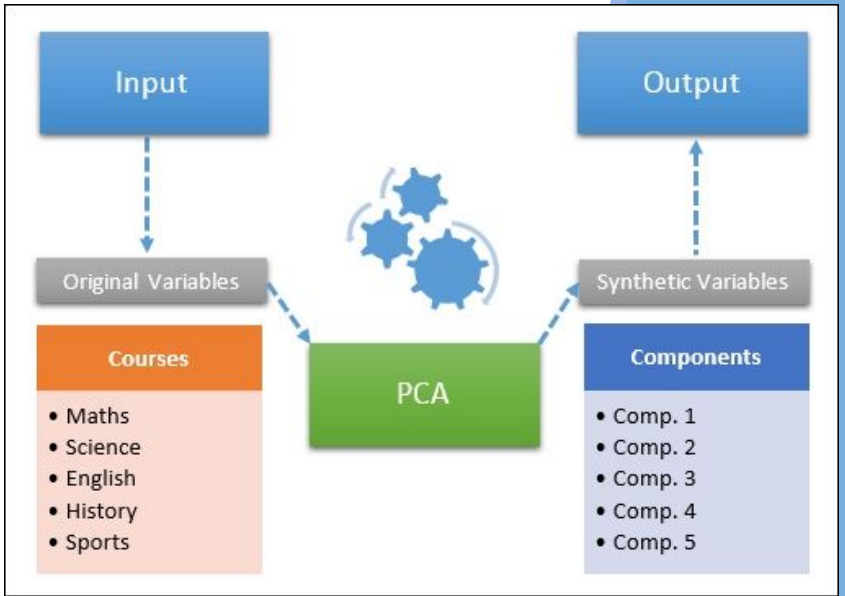


Principal Component Analysis

The input variables are the results of students in each of the subjects mentioned above, that is the original variables

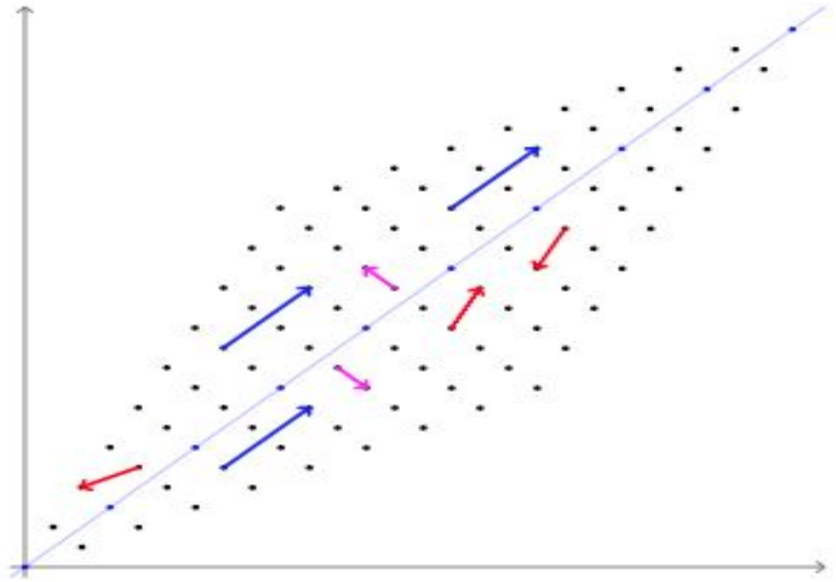
When you run the PCA, changes will take place in the input data and synthetic variables are created.

These new variables are the components that will be the model output.



Principal Component Analysis

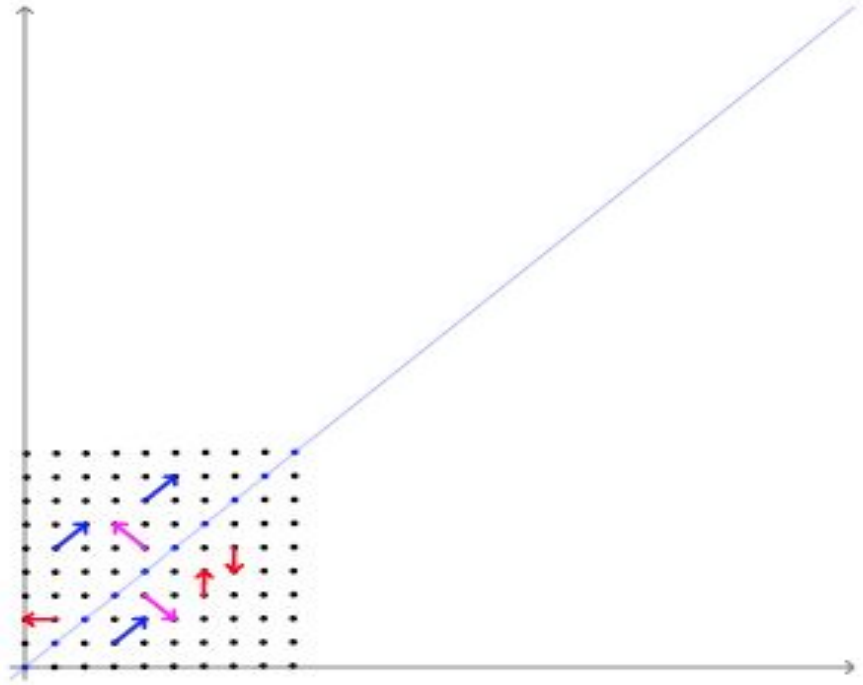
The eigenvectors of a square matrix are those special vectors that are simply scaled when the matrix is applied to them, and the eigenvalues are the scaling constants. A matrix of order k has k eigenvalues,



Principal Component Analysis

Eigenvalues are simply the coefficients attached to eigenvectors, which give the axes magnitude.

They are the measure of the data's covariance. By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.



Principal Component Analysis

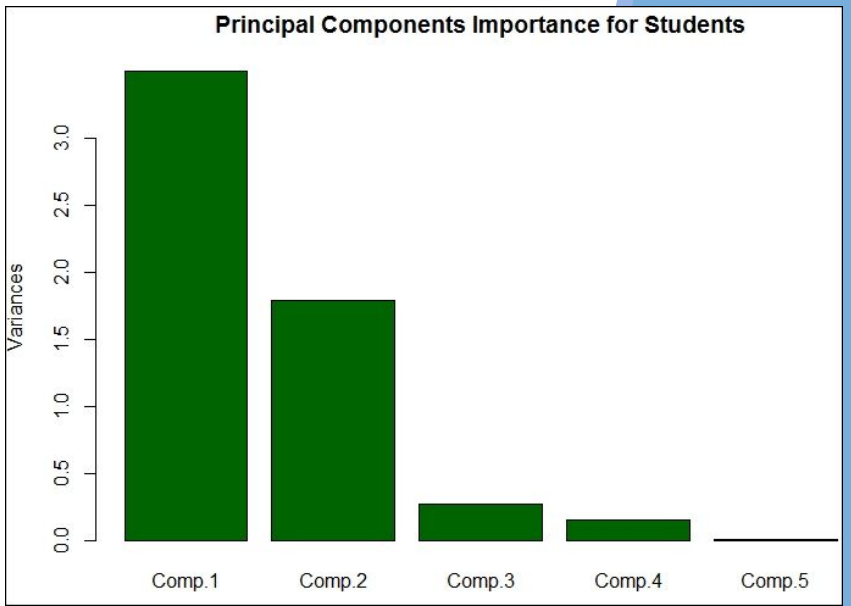
A component with an eigenvalue greater than 1 indicates that the component accounts for more variance than accounted for by one of the original variables in the standardized data.

This can be used as a cutoff point to determine the number of principal components to retain.

	eigenvalue
comp 1	2.893249673
comp 2	1.628650425
comp 3	0.346596049
comp 4	0.122612460
comp 5	0.008891393

Principal Component Analysis

The eigenvalues correspond to the amount of the variation explained by each principal component. They are large for the first component and small for the subsequent components.



Principal Component Analysis

Steps

Subtract the mean from each of the data points

The data now becomes centered around mean

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

Principal Component Analysis

Steps

Dividing by standard deviation transforms the data across unit variances.

$$s = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{(n-1)}}$$

Principal Component Analysis

Steps

Calculate the covariance matrix. Covariance is a measure of how two variables move together.

C_{nxn} is a matrix whose each value gives a covariance between two separate dimensions.

So, if there are two dimensions, there are a total of 4 covariance values, viz. C00, c01, c10, c11, i.e covariance between the same dimensions and covariance between first and second dimensions.

Principal Component Analysis

Steps

Calculate the eigenvector and eigenvalue of the covariance matrix.

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

Eigen vectors indicate the direction of the new axis (PCs) or the direction of rotation.

$$Ax = \lambda x$$

Each eigen vector corresponds to an eigen value whose magnitude indicates how much of the data's variability is explained by the vector.

Principal Component Analysis

Steps

$[C_{nxn}] \times [\text{Eigen Vector}] = [\text{Eigen vector}] \times [\text{Eigen value}]$

Select principal components from the feature vector once eigenvectors are formed, by ordering them from highest to lowest.

$$\det(A - \lambda I)$$

The number of eigenvectors you choose is the number of dimensions of the new dataset. The first PC is the axis that spans the most variation.
PCs are mutually orthogonal
and independent of each other to avoid repetition.

Applications

PCA technique is most commonly used in problems dealing with high dimensional data.

- Stock Trading
- Market Risks
- Images of faces
- Text from Articles

K-Means Clustering

Clustering is an example of unsupervised learning, in which we work with completely unlabeled data

Example

A data set showing where millionaires live probably has clusters in places like Beverly Hills and Manhattan.

Example

A data set showing how many hours people work each week probably has a cluster around 40.

- Each input can be represented as a vector in d-dimensional space.

- Our goal is to identify clusters of similar inputs and (sometimes) to find a representative value for each cluster.

- k-means is a simple clustering method in which the number of clusters, k is chosen in advance.

- The goal is to partition the inputs into sets in a way that minimizes the total sum of squared distances from each point to the mean of its assigned cluster.

Finding an optimal clustering is done by an iterative algorithm which assign n points to (1...k) clusters and finds the mean distortion for each cluster.

$$\min_{C_1 \cup C_2 \cup \cdots \cup C_k = \mathcal{P}} \sum_{i=1}^{k} \sum_{x \in C_i} \left\| x - \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j \right\|^2$$

STEPS

- Start with a set of k-means.These are points in d-dimensional space.
- Assign each point to the mean to which it is closest.
- If no point's assignment has changed, stop and keep the clusters.
- If some point's assignment has changed, recompute the means and return to step 2.

STEPS

- Start with a set of k-means.These are points in d-dimensional space.
- Assign each point to the mean to which it is closest.
- If no point's assignment has changed, stop and keep the clusters.
- If some point's assignment has changed, recompute the means and return to step 2.

Bottom-up Hierarchical Clustering

1. Make each input its own cluster of one.
2. As long as there are multiple clusters remaining, find the two closest clusters and merge them.
3. At the end, we'll have one giant cluster containing all the inputs.
4. If we keep track of the merge order, we can recreate any number of clusters by unmerging.
5. For example, if we want three clusters, we can just undo the last two merges.

S- Support

V- Vector

M- Machines

What is SVM

- SVM is a classifier that works on the principle of separating hyperplanes.
- Given a training dataset, the algorithms find a hyperplane that maximizes the separation of the classes and uses these partitions for the prediction of a new dataset.
- The hyperplane for a two-dimensional dataset is a line. These hyperplanes are also known as linear decision boundaries

Identifying a Separating Plane

For any data point x from one class

- $wx + b > 0$

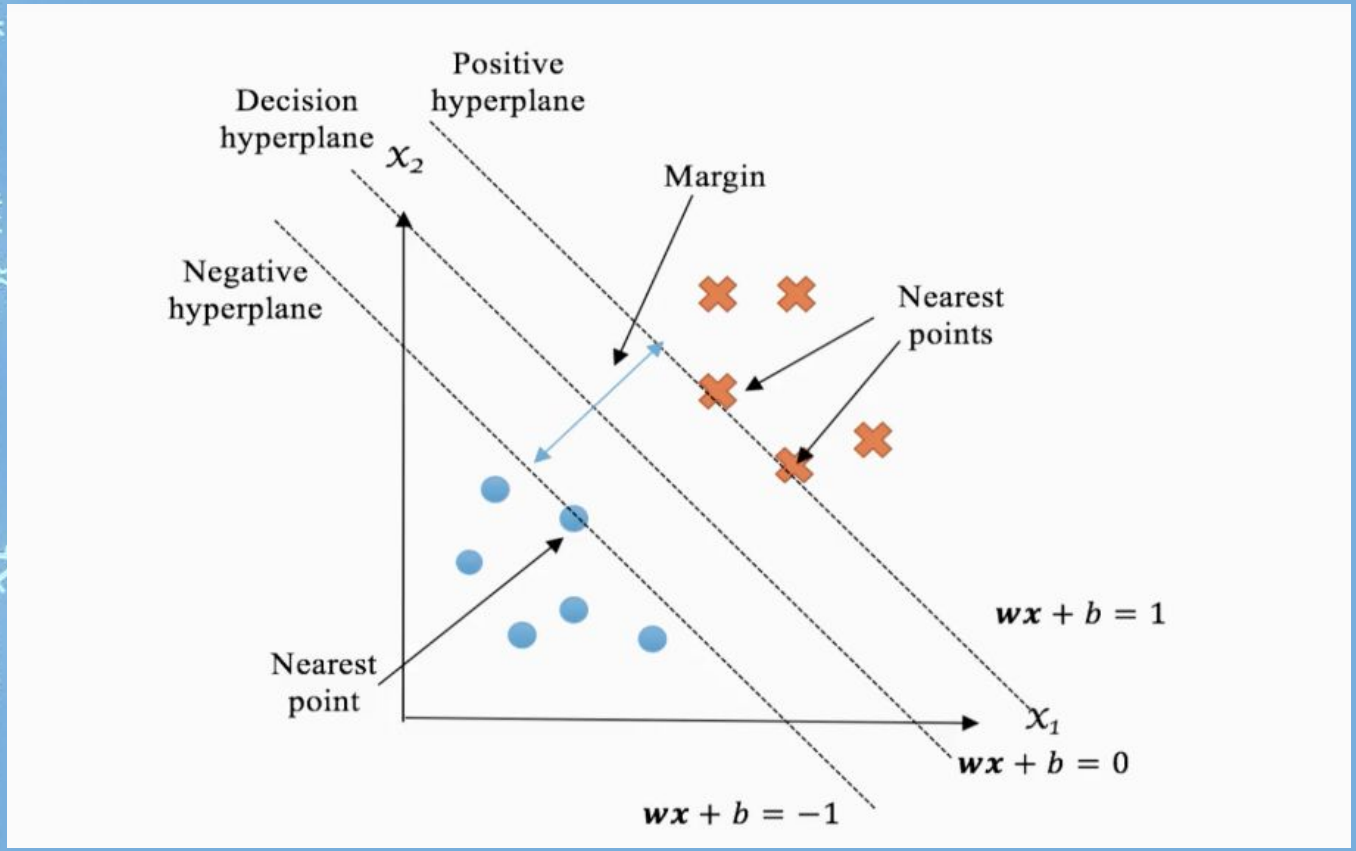
For any data point x from another class

- $wx + b < 0$

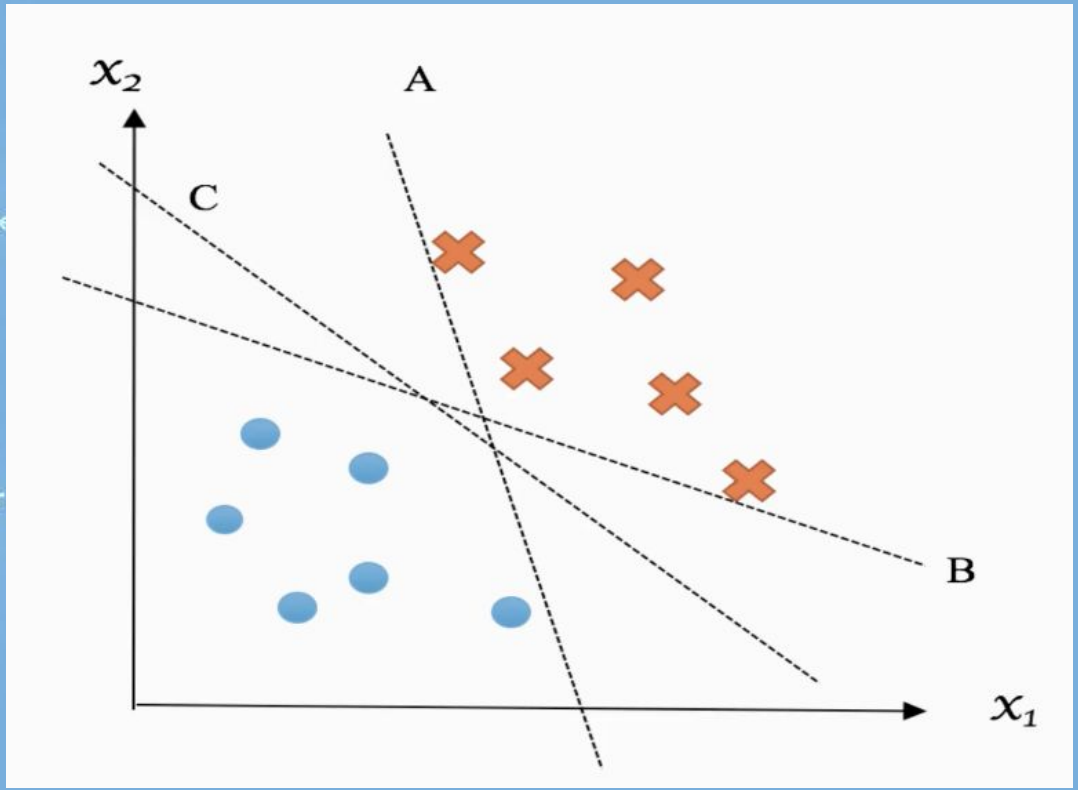
Where is SVM used?

SVM has similar use cases as that of other classifiers, but SVM is suited well for cases when the number of features/attributes are high compared to the number of data points/records.

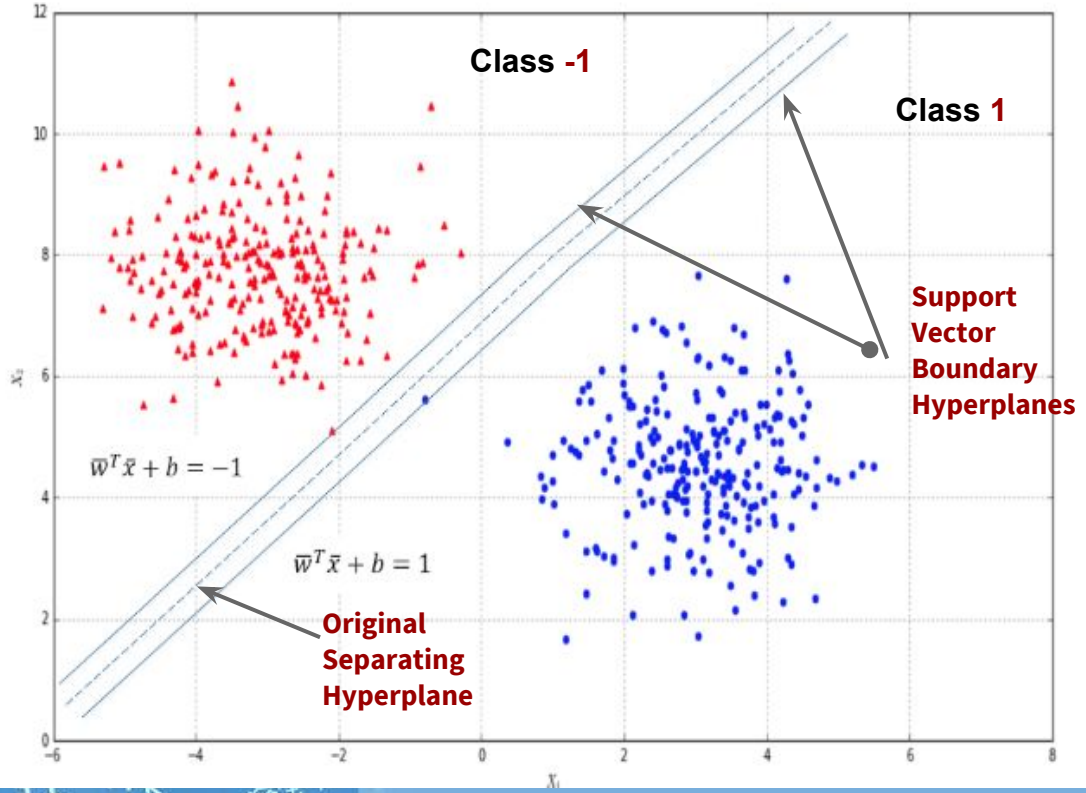
Support Vector Machines



Determining the Optimal Hyperplane



Intuition Behind SVM



Our goal is to maximize the distance between the two boundary hyperplanes to reduce the probability of misclassification

Linear SVM

- Let's consider a dataset of feature vectors we want to classify –

$$X = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\} \;\; where \;\; \bar{x}_i \in \mathbb{R}^m$$

- We set our class labels as –1 and 1:

$$Y = \{y_1, y_2, \ldots, y_n\} \;\; where \;\; y_i \in \{-1, 1\}$$

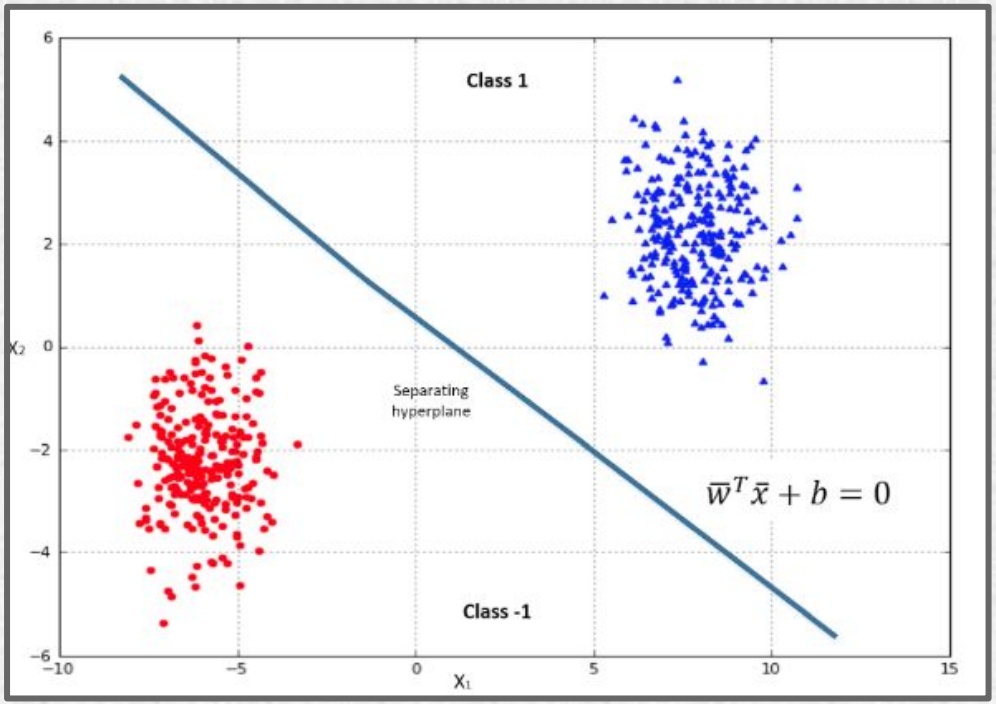
Linear SVM

- Our goal is to find the best separating hyperplane, for which the equation is as follows:-

$$\bar{\omega}^T \cdot \bar{x} + b = 0 \;\; where \;\; \bar{\omega} = (\omega_1, \omega_2, \ldots, \omega_m)^T$$

Linear SVM

- Structure of a Linear SVM Bipolar Problem –



Linear SVM

- In this way, our classifier can be written as follows –

$$\tilde{y} = f(\bar{x}) = sign\left(\bar{\omega}^T \cdot \bar{x} + b\right)$$

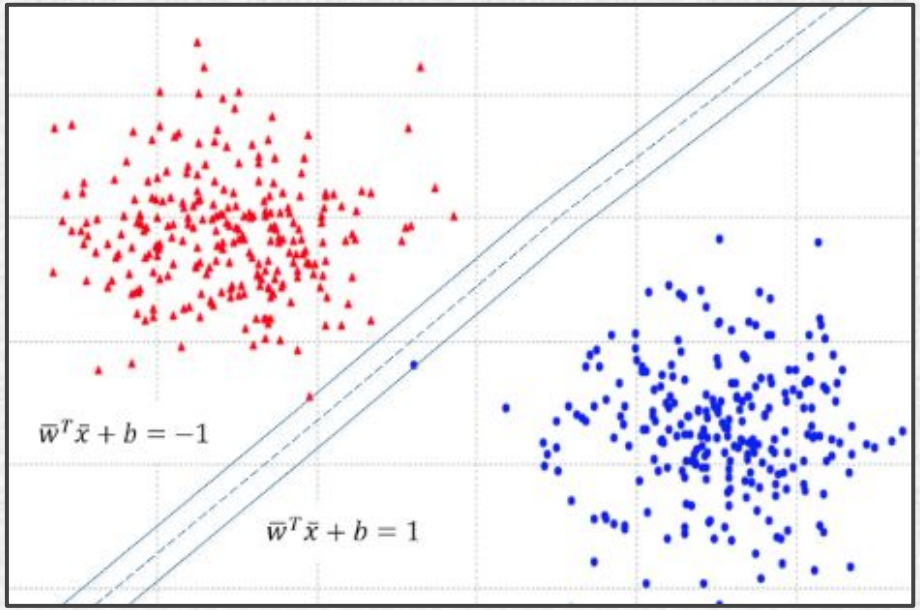
Linear SVM

- For a more generic mathematical expression, it's preferable to renormalize our dataset so that the support vectors will lie on two hyperplanes with the following equations: -

$$\begin{cases} \bar{\omega}^T \cdot \bar{x} + b = -1 \\ \bar{\omega}^T \cdot \bar{x} + b = 1 \end{cases}$$

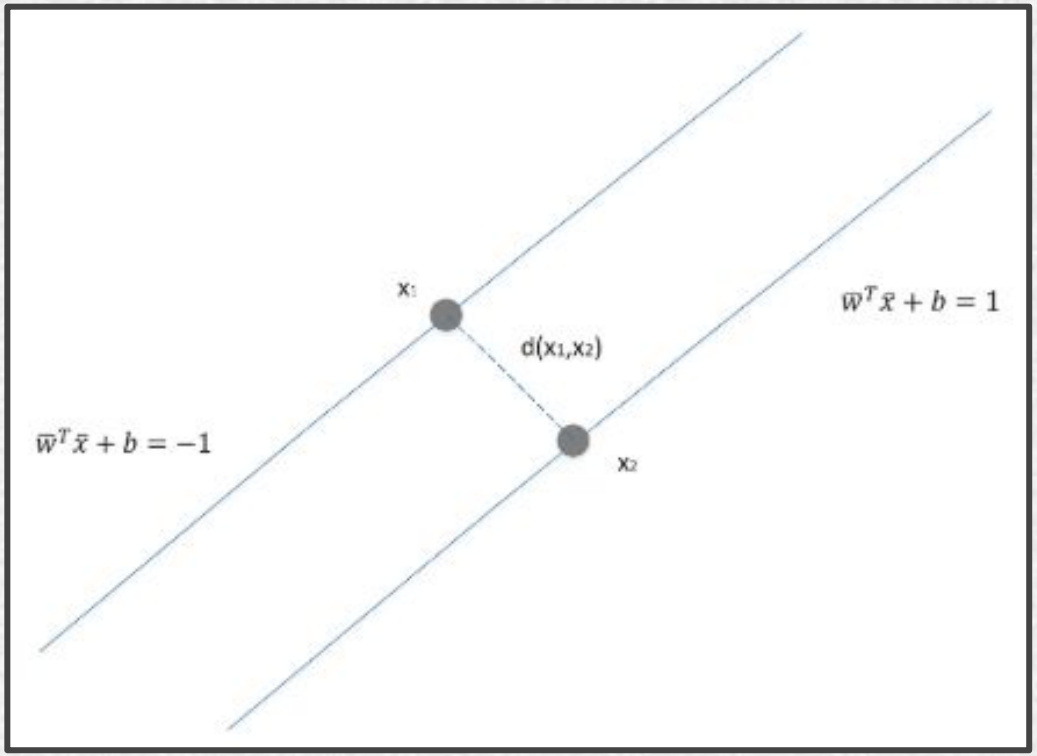
Linear SVM

- Separating hyperplane (dashed line) and support vector hyperplanes (solid lines)



Linear SVM

- Considering that the boundaries are parallel, the distance between them is defined by the length of the segment perpendicular to both and connecting two points.



Linear SVM

- Distance between the support vector hyperplanes

- Considering the points as vectors, therefore, we have the following:

$$\bar{x}_2 - \bar{x}_1 = t\bar{\omega}$$

Linear SVM

Manipulating the boundary hyperplane equations, we get this:

$$\bar{\omega}^T \cdot \bar{x}_2 + b = \bar{\omega}^T \cdot (\bar{x}_1 + t\bar{\omega}) = 1$$

$$\Rightarrow \ (\bar{\omega}^T \cdot \bar{x}_1 + b) + t\|\bar{\omega}\|^2 = 1$$

Linear SVM

The first term of the last part is equal to -1, so we solve for t, and we obtain this equation:

$$t = \frac{2}{\|\bar{\omega}\|^2}$$

Linear SVM

The distance between x1 and x2 is the length 1 of segment t; substituting the previous expression, we can derive another equation:

$$d(\bar{x}_1, \bar{x}_2) = t \, \|\bar{\omega}\| = \frac{2}{\|\bar{\omega}\|}$$

Linear SVM

Now, considering all the points of our dataset, we can impose the following constraint:

$$(\bar{\omega}^T \cdot \bar{x}_i + b) \geqslant 1 \ \forall \ (\bar{x}_i, y_i)$$

Linear SVM

- This is guaranteed by using -1, 1 as class labels and boundary margins.
- The equality is true only for the support vectors, while for all the other points it will be greater than 1.
- It's important to consider that the model doesn't take into account vectors beyond this margin.

Implementation

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Implementation

```
hr_data = pd.read_csv('data/hr.csv', header=0)
hr_data.head()
hr_data = hr_data.dropna()
print(" Data Set Shape ", hr_data.shape)
print(list(hr_data.columns))
print(" Sample Data ", hr_data.head())
```

Implementation

```
data_trnsf = pd.get_dummies(hr_data, columns =['salary', 'sales'])
data_trnsf.columns
X = data_trnsf.drop('left', axis=1)
X.columns
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, data_trnsf.left, test_size=0.3, random_state=42)
print(X_train)
```

Implementation

Next, we create an SVM model instance. We set the kernel to be linear, as we want a line to separate the two classes.

```
attrition_svm = svm.SVC(kernel='linear')
attrition_svm.fit(X_train, Y_train)
```

Implementation

After fitting the SVM model instance to the train data, we predict the Y values for the test set and create a confusion matrix to evaluate the model performance.

```
Y_pred = attrition_svm.predict(X_test)
from sklearn.metrics import confusion_matrix
confusionmatrix = confusion_matrix(Y_test, Y_pred)
print(confusionmatrix)
```

Implementation

After fitting the SVM model instance to the train data, we predict the Y values for the test set and create a confusion matrix to evaluate the model performance.

```
Y_pred = attrition_svm.predict(X_test)
from sklearn.metrics import confusion_matrix
confusionmatrix = confusion_matrix(Y_test, Y_pred)
print(confusionmatrix)
```

```
[[3212  216]
 [ 811  261]]
```

Implementation

Values for model accuracy and other metrics as calculated --

```
print('Accuracy of SVM classifier on test set: {:.2f}'.format(attrition_svm.score(X_test, Y_test)))
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_pred))
```

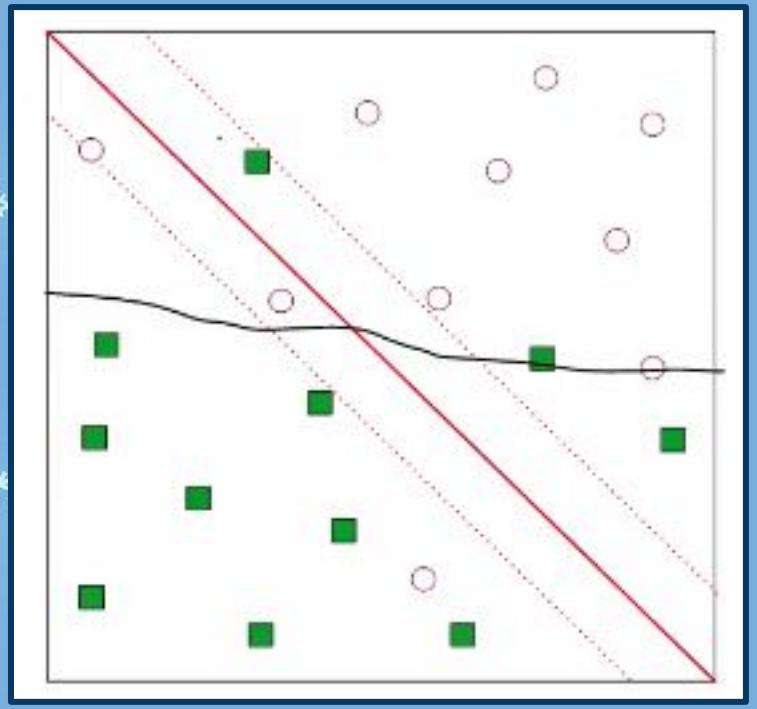
Implementation

Values for model accuracy and other metrics as calculated –-

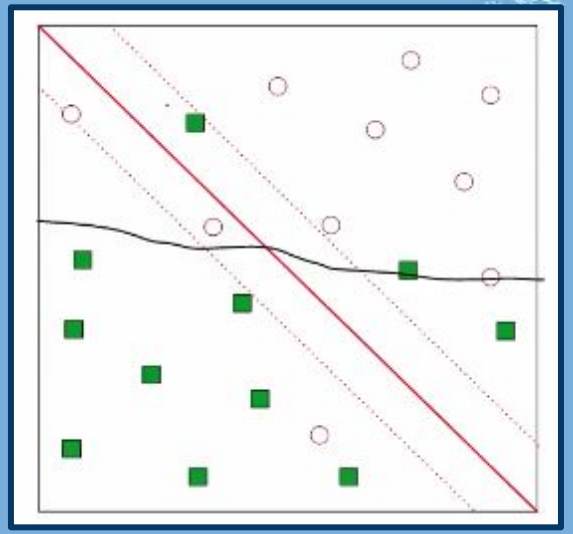
Accuracy of SVM classifier on test set: 0.77

	precision	recall	f1-score	support
0	0.80	0.94	0.86	3428
1	0.55	0.24	0.34	1072
avg / total	0.74	0.77	0.74	4500

What if the problem is not linearly separable??



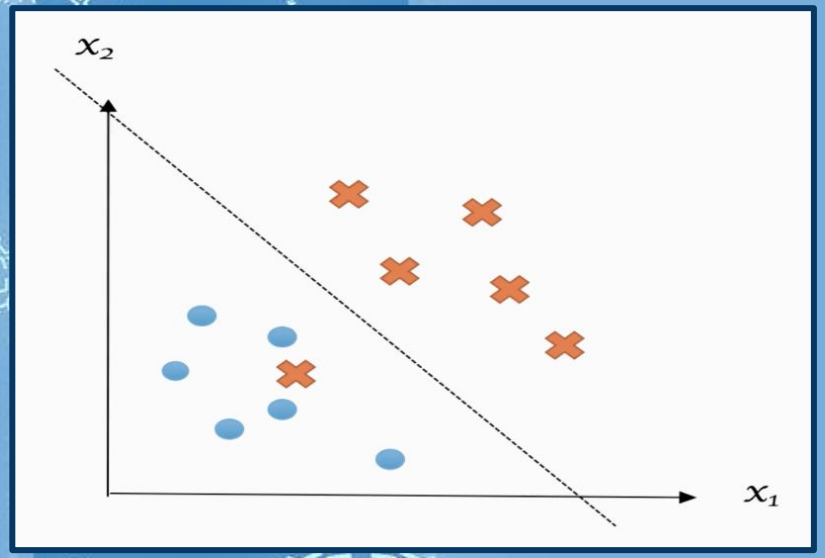
Possible Solutions



1) Still classify with a hyperplane by allowing for some misclassification.

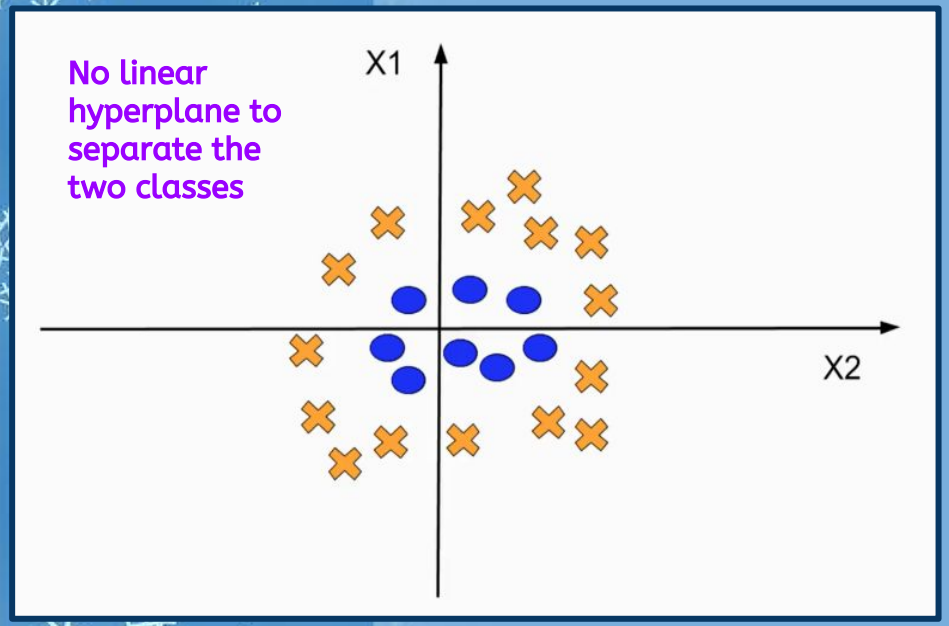
2) Map to a higher dimension
   a) Kernel Trick

Purpose of kernels in SVM



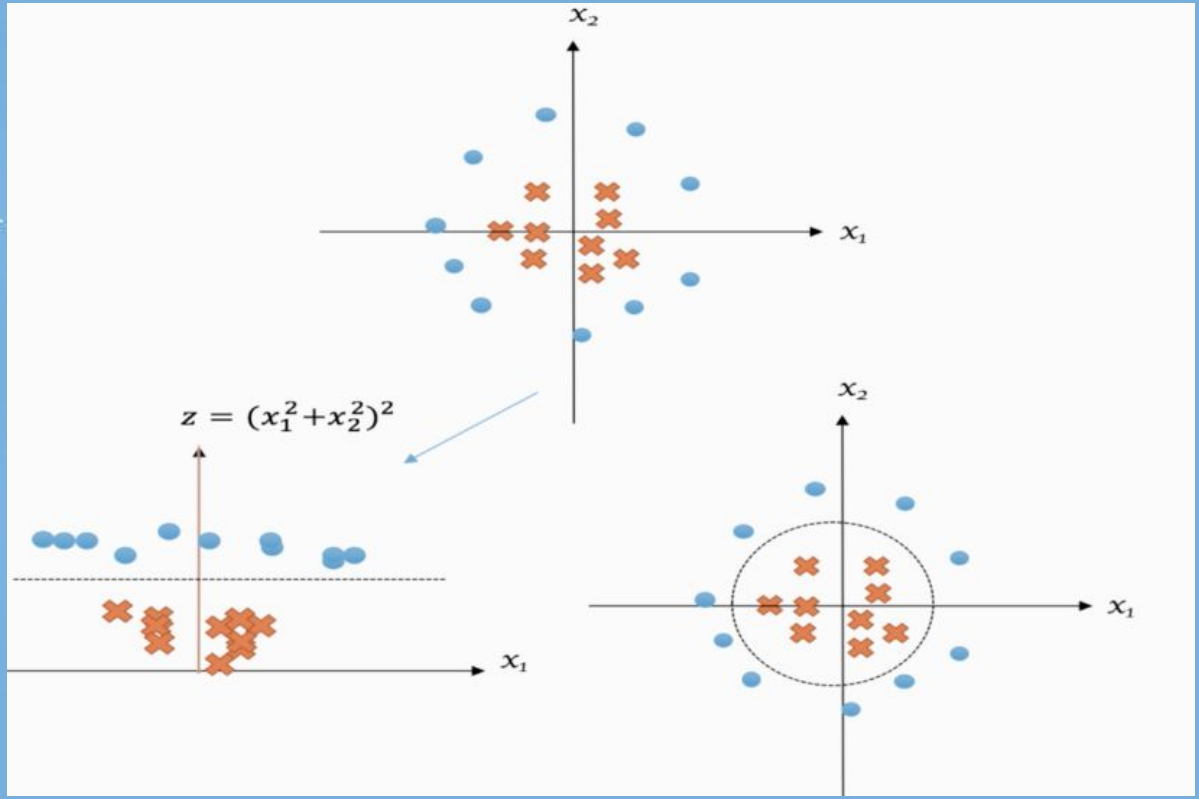
A kernel K(x, x') is a symmetric, non-negative real function that takes two real arguments (values of two features).

Kernels in SVM



Kernel Trick is used in SVM to solve nonlinear problems when there is no line that can separate the two classes in X1–X2 plane.

Choosing between the Linear and the RBF Kernel



Common Kernel Functions

The linear kernel (dot product)

This is useful in the case of very high-dimensional data, where problems can be expressed as a linear combination of the original features.

The polynomial kernel

This extends the linear kernel for a combination of features that are not completely linear.

The Radial Basis Function (RBF)

This is the most commonly applied kernel. It is appropriate where the labeled or target data is noisy and requires some level of regularization.

The sigmoid kernel

This is used in conjunction with neural networks.

The Laplacian kernel

This is a variant of RBF with a higher regularization impact on training data.

The log kernel

This is used in image processing.

Probabilistic kernels

These are kernels derived from generative models. Probabilistic models such as Gaussian processes can be used as a kernel function.

Smoothing kernels

This is the nonparametric formulation, averaging density with the nearest neighbor observations

Reproducible kernel Hilbert spaces

This is the dot product of finite or infinite basis functions

Common sklearn Modules and Methods

Models most used in scikit learn

1. Logistic Regression
2. Linear Regression
3. Decision Trees
4. Stochastic Gradient Descent
5. Random Forests
6. K-means
7. Hierarchical Clustering
8. Naive Bayes
9. Principal Component Analysis
10. K Nearest Neighbour

1. Logistic Regression

sklearn.linear_model.LogisticRegression

class `sklearn.linear_model.` **`LogisticRegression`** (*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None*)

2. Linear Regression

sklearn.linear_model.LinearRegression

class `sklearn.linear_model.` **`LinearRegression`** (*fit_intercept=True, normalize=False, copy_X=True, n_jobs=None*)

3. Decision Trees

sklearn.tree.DecisionTreeClassifier

class `sklearn.tree.` **`DecisionTreeClassifier`** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*) ¶

4. Stochastic Gradient Descent

sklearn.linear_model.SGDClassifier

class `sklearn.linear_model.` **`SGDClassifier`** (*loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False, n_iter=None*) [source]

5. Random Forest

sklearn.ensemble.RandomForestClassifier

class `sklearn.ensemble.` **`RandomForestClassifier`** (*n_estimators='warn', criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None*) [source]

6. K-Means

sklearn.cluster.KMeans

class `sklearn.cluster.` **`KMeans`** (*n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=None, algorithm='auto'*)

7. Hierarchical Clustering

sklearn.cluster.AgglomerativeClustering

class `sklearn.cluster.` **`AgglomerativeClustering`** (*n_clusters=2, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', pooling_func='deprecated'*)

8. Naive Bayes

sklearn.naive_bayes.GaussianNB
sklearn.naive_bayes.MultinomialNB
sklearn.naive_bayes.BernoulliNB

class `sklearn.naive_bayes.` **`GaussianNB`** (*priors=None, var_smoothing=1e-09*)

class `sklearn.naive_bayes.` **`MultinomialNB`** (*alpha=1.0, fit_prior=True, class_prior=None*)

class `sklearn.naive_bayes.` **`BernoulliNB`** (*alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None*)

9. Principal Component Analysis

sklearn.decomposition.PCA

class `sklearn.decomposition.` **PCA** (*n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', random_state=None*) [source]

10. K- Nearest Neighbour

sklearn.neighbors.KNeighborsClassifier

class `sklearn.neighbors.` **`KNeighborsClassifier`** (*n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*)

sklearn.preprocessing

The sklearn.preprocessing module includes scaling, centering, normalization, binarization

sklearn.preprocessing

`preprocessing.Binarizer` ([threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold
`preprocessing.FunctionTransformer` ([func, …])	Constructs a transformer from an arbitrary callable.
`preprocessing.KBinsDiscretizer` ([n_bins, …])	Bin continuous data into intervals.
`preprocessing.KernelCenterer` ()	Center a kernel matrix
`preprocessing.LabelBinarizer` ([neg_label, …])	Binarize labels in a one-vs-all fashion
`preprocessing.LabelEncoder`	Encode labels with value between 0 and n_classes-1.
`preprocessing.MultiLabelBinarizer` ([classes, …])	Transform between iterable of iterables and a multilabel format
`preprocessing.MaxAbsScaler` ([copy])	Scale each feature by its maximum absolute value.
`preprocessing.MinMaxScaler` ([feature_range, copy])	Transforms features by scaling each feature to a given range.
`preprocessing.Normalizer` ([norm, copy])	Normalize samples individually to unit norm.
`preprocessing.OneHotEncoder` ([n_values, …])	Encode categorical integer features as a one-hot numeric array.
`preprocessing.OrdinalEncoder` ([categories, dtype])	Encode categorical features as an integer array.
`preprocessing.PolynomialFeatures` ([degree, …])	Generate polynomial and interaction features.
`preprocessing.PowerTransformer` ([method, …])	Apply a power transform featurewise to make data more Gaussian-like.

sklearn.preprocessing

`preprocessing.QuantileTransformer` ([…])	Transform features using quantiles information.
`preprocessing.RobustScaler` ([with_centering, …])	Scale features using statistics that are robust to outliers.
`preprocessing.StandardScaler` ([copy, …])	Standardize features by removing the mean and scaling to unit variance
`preprocessing.add_dummy_feature` (X[, value])	Augment dataset with an additional dummy feature.
`preprocessing.binarize` (X[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix
`preprocessing.label_binarize` (y, classes[, …])	Binarize labels in a one-vs-all fashion
`preprocessing.maxabs_scale` (X[, axis, copy])	Scale each feature to the [-1, 1] range without breaking the sparsity.
`preprocessing.minmax_scale` (X[, …])	Transforms features by scaling each feature to a given range.
`preprocessing.normalize` (X[, norm, axis, …])	Scale input vectors individually to unit norm (vector length).
`preprocessing.quantile_transform` (X[, axis, …])	Transform features using quantiles information.
`preprocessing.robust_scale` (X[, axis, …])	Standardize a dataset along any axis
`preprocessing.scale` (X[, axis, with_mean, …])	Standardize a dataset along any axis
`preprocessing.power_transform` (X[, method, …])	Apply a power transform featurewise to make data more Gaussian-like.

sklearn.metrics

`metrics.accuracy_score` (y_true, y_pred[, …])	Accuracy classification score.
`metrics.auc` (x, y[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
`metrics.average_precision_score` (y_true, y_score)	Compute average precision (AP) from prediction scores
`metrics.balanced_accuracy_score` (y_true, y_pred)	Compute the balanced accuracy
`metrics.brier_score_loss` (y_true, y_prob[, …])	Compute the Brier score.
`metrics.classification_report` (y_true, y_pred)	Build a text report showing the main classification metrics
`metrics.cohen_kappa_score` (y1, y2[, labels, …])	Cohen's kappa: a statistic that measures inter-annotator agreement.
`metrics.confusion_matrix` (y_true, y_pred[, …])	Compute confusion matrix to evaluate the accuracy of a classification
`metrics.f1_score` (y_true, y_pred[, labels, …])	Compute the F1 score, also known as balanced F-score or F-measure
`metrics.fbeta_score` (y_true, y_pred, beta[, …])	Compute the F-beta score
`metrics.hamming_loss` (y_true, y_pred[, …])	Compute the average Hamming loss.
`metrics.hinge_loss` (y_true, pred_decision[, …])	Average hinge loss (non-regularized)
`metrics.jaccard_similarity_score` (y_true, y_pred)	Jaccard similarity coefficient score
`metrics.log_loss` (y_true, y_pred[, eps, …])	Log loss, aka logistic loss or cross-entropy loss.
`metrics.matthews_corrcoef` (y_true, y_pred[, …])	Compute the Matthews correlation coefficient (MCC)

sklearn.metrics

`metrics.precision_recall_curve` (y_true, …)	Compute precision-recall pairs for different probability thresholds
`metrics.precision_recall_fscore_support` (…)	Compute precision, recall, F-measure and support for each class
`metrics.precision_score` (y_true, y_pred[, …])	Compute the precision
`metrics.recall_score` (y_true, y_pred[, …])	Compute the recall
`metrics.roc_auc_score` (y_true, y_score[, …])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
`metrics.roc_curve` (y_true, y_score[, …])	Compute Receiver operating characteristic (ROC)
`metrics.zero_one_loss` (y_true, y_pred[, …])	Zero-one classification loss.

sklearn.metrics

`metrics.explained_variance_score` (y_true, y_pred)	Explained variance regression score function
`metrics.mean_absolute_error` (y_true, y_pred)	Mean absolute error regression loss
`metrics.mean_squared_error` (y_true, y_pred[, …])	Mean squared error regression loss
`metrics.mean_squared_log_error` (y_true, y_pred)	Mean squared logarithmic error regression loss
`metrics.median_absolute_error` (y_true, y_pred)	Median absolute error regression loss
`metrics.r2_score` (y_true, y_pred[, …])	R^2 (coefficient of determination) regression score function.

sklearn.metrics

`metrics.pairwise.cosine_similarity` (X[, Y, …])	Compute cosine similarity between samples in X and Y.
`metrics.pairwise.cosine_distances` (X[, Y])	Compute cosine distance between samples in X and Y.
`metrics.pairwise.distance_metrics` ()	Valid metrics for pairwise_distances.
`metrics.pairwise.euclidean_distances` (X[, Y, …])	Considering the rows of X (and Y=X) as vectors, compute the distance matrix between each pair of vectors.
`metrics.pairwise.kernel_metrics` ()	Valid metrics for pairwise_kernels
`metrics.pairwise.laplacian_kernel` (X[, Y, gamma])	Compute the laplacian kernel between X and Y.
`metrics.pairwise.linear_kernel` (X[, Y, …])	Compute the linear kernel between X and Y.
`metrics.pairwise.manhattan_distances` (X[, Y, …])	Compute the L1 distances between the vectors in X and Y.
`metrics.pairwise.pairwise_kernels` (X[, Y, …])	Compute the kernel between arrays X and optional array Y.
`metrics.pairwise.polynomial_kernel` (X[, Y, …])	Compute the polynomial kernel between X and Y:
`metrics.pairwise.rbf_kernel` (X[, Y, gamma])	Compute the rbf (gaussian) kernel between X and Y:
`metrics.pairwise.sigmoid_kernel` (X[, Y, …])	Compute the sigmoid kernel between X and Y:
`metrics.pairwise.paired_euclidean_distances` (X, Y)	Computes the paired euclidean distances between X and Y
`metrics.pairwise.paired_manhattan_distances` (X, Y)	Compute the L1 distances between the vectors in X and Y.
`metrics.pairwise.paired_cosine_distances` (X, Y)	Computes the paired cosine distances between X and Y
`metrics.pairwise.paired_distances` (X, Y[, metric])	Computes the paired distances between X and Y.
`metrics.pairwise_distances` (X[, Y, metric, …])	Compute the distance matrix from a vector array X and optional Y.
`metrics.pairwise_distances_argmin` (X, Y[, …])	Compute minimum distances between one point and a set of points.

Sklearn.model_selection

Model validation function

`model_selection.cross_validate` (estimator, X)	Evaluate metric(s) by cross-validation and also record fit/score times.
`model_selection.cross_val_predict` (estimator, X)	Generate cross-validated estimates for each input data point
`model_selection.cross_val_score` (estimator, X)	Evaluate a score by cross-validation
`model_selection.learning_curve` (estimator, X, y)	Learning curve.
`model_selection.permutation_test_score` (…)	Evaluate the significance of a cross-validated score with permutations
`model_selection.validation_curve` (estimator, …)	Validation curve.

sklearn.model_selection

Splitter function

model_selection.check_cv ([cv, y, classifier])	Input checker utility for building a cross-validator
model_selection.train_test_split (*arrays, …)	Split arrays or matrices into random train and test subsets

Thanks

Any questions?

You can find me at

- @ankita90
- ankita.sinha8118@gmail.com