Module - 1

What is Machine Learning?

Machine learning can be defined as a branch of Artificial Intelligence that has the ability to learn on its own without having all information with respect to a domain in the program itself.

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicity programmed. Machine learning focuses on the development of Computer Programs that can teach themselves to grow and change when exposed to new data.

The process of machine learning is similar to that of data mining. Both system search through data to look for pattern. However, instead of extracting data for human Comprehension as is the case in data mining applications, machine learning uses that data to improve the program's own understanding. Machine learning programs detect patterns in data and adjust program action accordingly.

Key Points

- ML turns Data into information (Meaningful Data)
- ML is a combination of Computer Science, Engineering and statistics
- ML uses statistical concept to process data and to solve problems.
- ML interprets data and acts on it.
- ML programs a computer to optimize performance criteria using past experience.

Key Terminologies.

Expert System: The expert system are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence, expertise and decision-making ability of a human expert.

Characteristics of Expert Systems

- High performance
- Understandable
- Reliable.
- Highly responsive

The expert System are capable of

- Advising.
- Instructing and assisting human in decision making.
- Demonstrating.
- Deriving a solution.
- Diagnosing.
- Explaining
- Interpreting input
- Predicting results.
- Justifying the conclusion.
- Suggesting alternative options to a problem

The Components of Expert System includes -

- **Knowledge Base**: Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.
- **Factual Knowledge**: Factual Knowledge is that knowledge of the task domain that is widely shared, typically found in textbook or journals, and commonly agreed upon by those knowledgeable in the particular field.

- **Heuristic Knowledge:** Heuristic knowledge is the less rigorous, more experiential, more judgmental knowledge of performance. In contrast to factual knowledge, heuristic knowledge is rarely discussed, and is largely individualistic.

- **Knowledge Representation:** Knowledge representation is the method to organize and formalize the knowledge in the knowledge base. It is the form of IF-THEN-ELSE rules. The IF part of the rule is satisfied; consequently, the THEN part can be concluded.

- **Knowledge Acquisition:** The success of any expert system majorly depends on the quality, completeness, and accuracy of the information stored in the knowledge base. The knowledge base is formed by reading from various experts, Scholars, and the knwoledge Engineers.

- **Interface Engine** is essential in deducting a correct, flawless solution. It acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution.

- **User Interface** provides interaction between user of the ES and the ES itself. It explain how the ES has arrived at a particular recommendation.

Training Set : A set of example used for learning to fit the parameters of the classifier. The training set can be selected by applying a Random filter to the data, e.g. select 20% of the points as Random to generate the model and test against the remaining 80%.

Validation Set : A set of example used to tune the parameters of a classifier. It is usually used to adjust the classification parameters in order to avoid overfitting.

Test Set : Test set is the data, whose outcome is already known and is used to determine the accuracy of the machine learning algorithm, based on the training set

Target Variable : It is the output to be predicted from a machine learning algorithm. It could be binary 0 or 1 if you are classifying or it could be a continuous variable if you are doing a Regression.

Classification : Classification is the problem of identifying to which of a set of categories a new observation belong, on the basis of a training set of data containing observation whose category membership is known.

Regression : Regression analysis is a statistical process for estimating the relationship among variables.

Features : A feature is an individual measurable property of a phenomenon being observed. Features and attributes are individual measurements, when combined with other features makes training example.

Types of Machine Learning :

Supervised learning :- Learning process with the help of a teacher is called supervised Learning. In supervised learning the algorithm generates a function that maps inputs to desired outputs. One standard formulation of the supervised learning task is the classification problem : the learner is required to learn a function which maps a vector into one of several classes by looking at several input-output example of the function.

Unsupervised Learning :- Learning process without Teacher is called unsupervised learning. During training the machine learning algorithm receives the input pattern and organizes these patterns to form clusters. When a new input pattern is applied, the gives an output response indicating the cluster to which the input pattern belongs.

Reinforcement Learning : Similar to supervised learning. In this learning process only critic information is available not the exact information. The learning process is done based on the critic information and a feedback signal called reinforcement signal is sent back from output to the input.

Issues in Machine Learning

1. What algorithms exist for learning general target from specific training example?
2. In what settings will particular algorithms converge to the desired function, given sufficient training data?
3. Which algorithms peyorm best for which types of problems and representations?
4. How much training data is sufficient?
5. What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
6. When and how can prior knowledge held by the learner guide the process of generalizing from example?
7. Can prior knowledge be helpful even when it is only approximetely correct?
8. What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?

What is the best way to reduce the learning
to one or more function approximation problem?

What specific functions should the system attempt
learn? Can this process itself be automated?

How can the learner automatically alter its
representation to improve its ability to represent
and learn the target function?

Steps in developing a machine learning applications

Collect data: This includes collecting the samples by scraping a website and extracting data, from an RSS feed or an API. Use necessary sensors or make use of publicly available data.

Prepare Input data: Once data is collected make sure it's in a useable format. The benefit of having this standard format is that you can mix and match algorithms and data sources.

Analyse the Input data: is looking at the data from the previous task. It involves recognizing patterns, identifying outliers and detection of novelty.

Train the algorithm. The actual machine learning tasks. place here. This step and the next step are where the "core" algorithms lie, depending on the algorithm. Feed the algorithm good clean data from the first who steps and extract knowledge or information. The knowledge extracted is stored in a format that's readily usable by a machine for the next two steps.

Test the algorithm: The information learned in the previous steps is put to use during testing.

Use it: Make use of real program to do some task, and once again you see if all the previous steps worked as you expected.

Application of Machine Learning

1. **Learning Associations**: is a method for discovering interesting relations between variables in large databases. It is used to identify regularities among large scale databases. The concept of learning association is mainly used in Market analysis which is finding association between products bought by customers: If people who buy x typically also buy y, and if there is a customer who buys x and does not buy y, he or she is a potential Y customer. Once such customers are identified, they can be targeted for cross-selling

2. **Classification**: Classification is the problem of identifying to which of a set of categories a new observation to which of a set of categories a new observation belong, on the basis of a training set of data containing observation whose category membership is known

3. **Pattern Recognition**: is a branch of machine learning that focuses on the regn recognition of similarities and regularities in data.

Optical character recognition: is the recognition of character codes from their images. In this case there are multiple classes, as many as there are characters that is to beg recognized.

Handwriting recognition: People have different handwriting styles; charaters may be written small or large, slanted with a pen or pencil, and there are many possibl images corresponding to the same character.

Face recognition : The input is an image and the classes are people to be recognized. The learning program should learn to associate the face images to identities.

Medical diagnosis : The inputs are the relevant information about the patient and the classes are the illnesses.

Speech recognition : The input is acoustic and the classes are words that can be uttered. The association to be learned is from an acoustic signal to a word of some languages.

4. Natural language processing : is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages.

5. Biometrics : is recognition or authentications of people using their physiological and/or ~~rule is a simple model that explains the data, using which an explanation about the process~~ behavioral characteristics that requires an integration of inputs from different modalities.

6. Knowledge Extraction: Learning a rule from data is known as knowledge extraction. The rule is a simple model that explains the data, using which an explanation about the process underlying the data is generated.

7. Compression : By fitting a rule to the data an explanation that is simpler than the data, requiring less memory to store and less computation to process can be generated. This process is known as Compression.

8. Outlier Detection : Outlier detection is the process of finding the instances that do not obey the rule and are exceptions to the standard case.

9. Regression : Regression analysis is a statistical process for estimating the relationships among variables.

10. Density Estimation : There is a structure to the input space such that certain patterns occur more often than others, and based on the frequency of occurance, it has to be analysed that what happens and what does not. One type of density estimation is Clustering.

Real World Applications of Machine learning.

- Web search.
- Computational Biology.
- Finance.
- E- Commerce.

- Space Exploration
- Robotics.
- Information Extraction.
- Text-based sentiment analysis in social Networking
- Network intrusion detection and fraud detection.
- Email spam filtering.
- Prediction of equipment failures.
- Linguistic rule creation.

Module 04 - Learning with Regression and Trees

Contents

Linear Regression

Supervised Learning and Regression

- Regression analysis falls under supervised machine learning.
- The system tries to predict a value for an input based on previous information.
- Characteristics of regression:
  - The responses obtained from the model are always quantitative in nature.
  - The model can be constructed only if past data is available.

Regression Analysis: Estimating Relationships

Regression Analysis is a study of relationship between a set of independent variables and the dependent variable.

Independent variables are characteristics that can be measured directly (example the area of a house). These variables are also called predictor variables (used to predict the dependent variable) or explanatory variables (used to explain the behavior of the dependent variable).

Dependent variable is a characteristic whose value depends on the values of independent variables.

Regression Analysis: Estimating Relationships

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \pm \varepsilon$$

Purpose of Regression Analysis

Explanation: Use regression analysis to develop a mathematical model to explain the variance in the dependent variable based on values of independent variables.

Prediction: If the regression model adequately explains the dependent variable, use the model to predict values of the dependent variable.

Purpose of Regression Analysis

Explain Selling Price of a house (dependent) based on its characteristics (independents) . If the model is valid, use it for prediction.

Develop Regression Model using known data (sample)

Selling Price = 40,000 + 100(Sq. ft.)*Area + 20,000*(#Baths)

If the above model is reliable and valid, use this model to predict the Selling Price of any house based on its area (Sq. ft.) and the number of bathrooms (#Baths).

General Nature of Linear Regression

Most types of nonlinear relationships may be reduced to linear cases by means of appropriate preliminary transformations to the original observations.

General Nature of Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2$$

General Nature of Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2$$

Let $x_1^2 = x_2$

Therefore,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

General Nature of Linear Regression

$$y = e^{\beta_0 + \beta_1 x}$$

General Nature of Linear Regression

$$y = e^{\beta_0 + \beta_1 x}$$

Let $y' = \ln y$

Therefore,

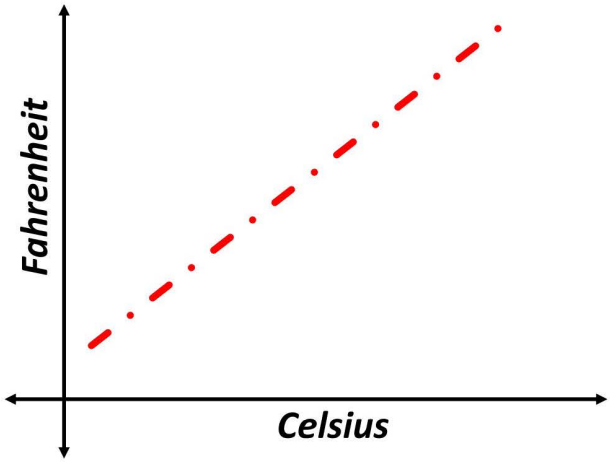
$$y' = \beta_0 + \beta_1 x$$

Simple Linear Regression

- Is a statistical method that allows us to summarize and study the relationship between two continuous (quantitative) variables.
- Between predictor, explanatory, or independent variable and response, outcome, or dependent variable.
- Statistical relationships are different between deterministic relationships

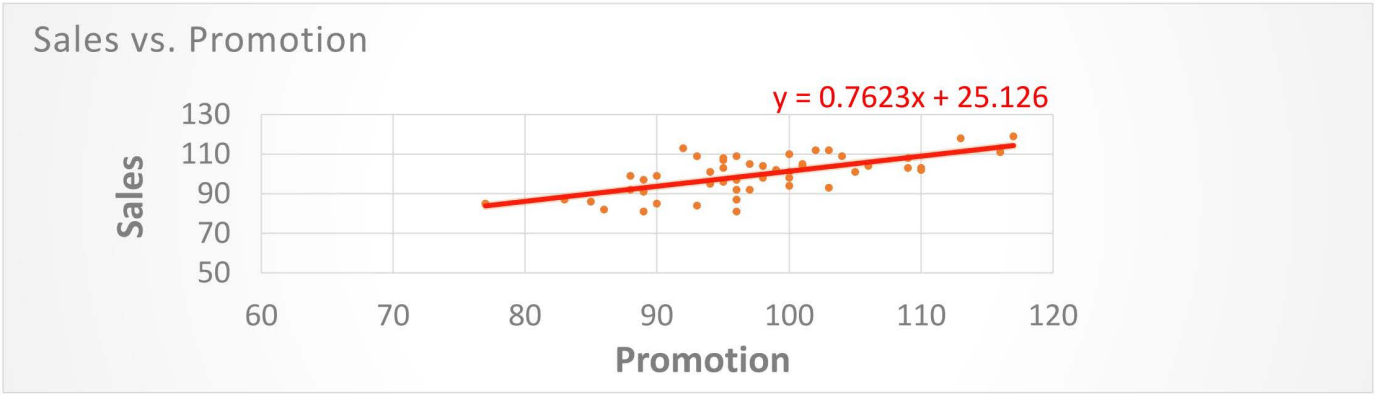
Simple Linear Regression

$$y = \beta_0 + \beta_1 x \pm \varepsilon$$



Selecting Independent Variables: Scatter Plots

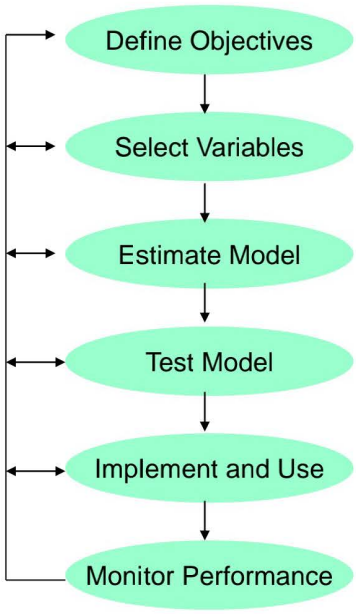
Scatter Plots are used to visualize the relationship between any two variables. For regression analysis, we are looking for *strong linear relationship* between the *independent* and *dependent* variable.



Selecting Independent Variables: Scatter Plots

- In statistical relationships, the plot is called a *scatter diagram* or *scatter plot* because of the scattering of points in the relation.
- The purpose of regression models is to identify a functional relationship between the target variable and a subset of the remaining attributes contained in the dataset.
- It enables to
  - Highlight and interpret the dependency of the target variable on other variables.
  - Predict the future value of the target variable

Procedure for Building Regression Models



Define/Clarify Purpose. Identify and describe the measurement of the dependent variable

Identify possible independent variables (predictors – should make sense). Use scatter plots and correlations for selection.

Estimate Regression Coefficients (using least squares method).

Test to see if all coefficients are significant (reliability). Establish validity (are relationships as expected, do predictions match actuals).

Implement the model in Decision Support System. Incorporate error in predictions. Outline limitations/constraints of the model.

Compare predictions with actual values. Modify/Refine/Expand model if necessary. It is about continuous improvement.

Least Square Method
(Simple Linear Regression)

$$\min \Sigma(y_i - \hat{y}_i)^2$$

where:

y_i = observed value of the dependent variable for the ith observation

$\hat{y}_i$ = estimated value of the dependent variable for the ith observation

Least Square Method
(Simple Linear Regression)

Slope for the estimated regression line (hypothesis function)

$$b_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

Least Square Method
(Simple Linear Regression)

y-intercept for the estimated regression line (hypothesis function)

$$b_0 = \bar{y} - b_1\bar{x}$$

where:

x_i = value of independent variable for ith observation

y_i = value of dependent variable for ith observation

$\bar{x}$ = mean value for independent variable

$\bar{y}$ = mean value for dependent variable

n = total number of observations

Simple Linear Regression

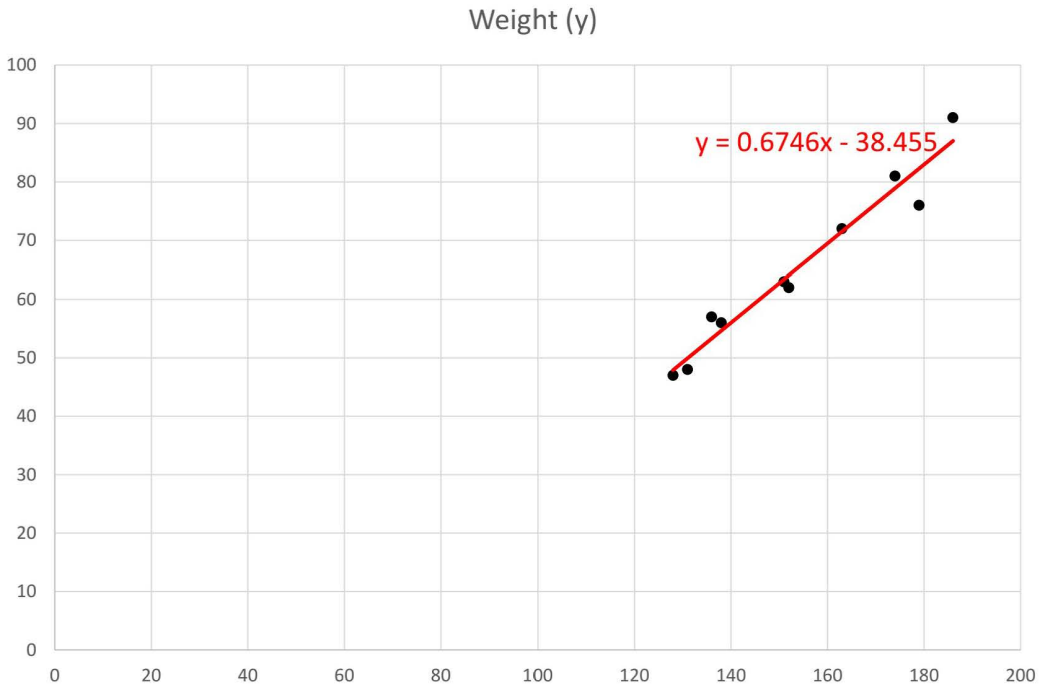
Estimate the regression line (hypothesis function) for the given data.

Height (*x*)	**Weight (*y*)**
151	63
174	81
138	56
186	91
128	47
136	57
179	76
163	72
152	62
131	48

Simple Linear Regression

S. No.	Height (X) cms	Weight (Y) kgs	$(X_i - \bar{X})$	$(Y_i - \bar{Y})$	$(X_i - \bar{X})(Y_i - \bar{Y})$	$(X_i - \bar{X})^2$
1	151	63	–2.8	–2.3	6.44	7.84
2	174	81	20.2	15.7	317.14	408.04
3	138	56	–15.8	–9.3	146.94	249.64
4	186	91	32.2	25.7	827.54	1036.8
5	128	47	–25.8	–18.3	472.14	665.64
6	136	57	–17.8	–8.3	147.74	316.84
7	179	76	25.2	10.7	269.64	635.04
8	163	72	9.2	6.7	61.64	84.64
9	152	62	–1.8	–3.3	5.94	3.24
10	131	48	–22.8	–17.3	394.44	519.84
	$\bar{X} =$ 153.8	$\bar{Y} =$ 65.3			$\Sigma = 2649.6$	$\Sigma = 3927.6$

Simple Linear Regression

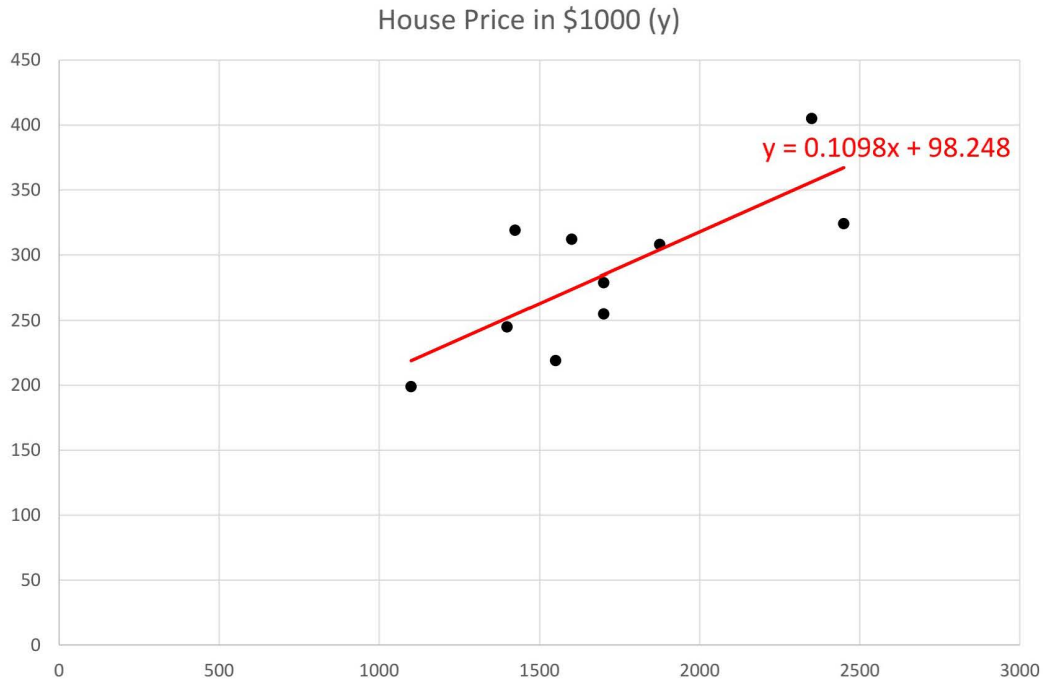


Simple Linear Regression

Estimate the regression line (hypothesis function) for the given data.

Area in Square feet (*x*)	**House Price in $1000 (*y*)**
1400	245
1600	312
1700	279
1875	308
1100	199
1550	219
2350	405
2450	324
1425	319
1700	255

Simple Linear Regression

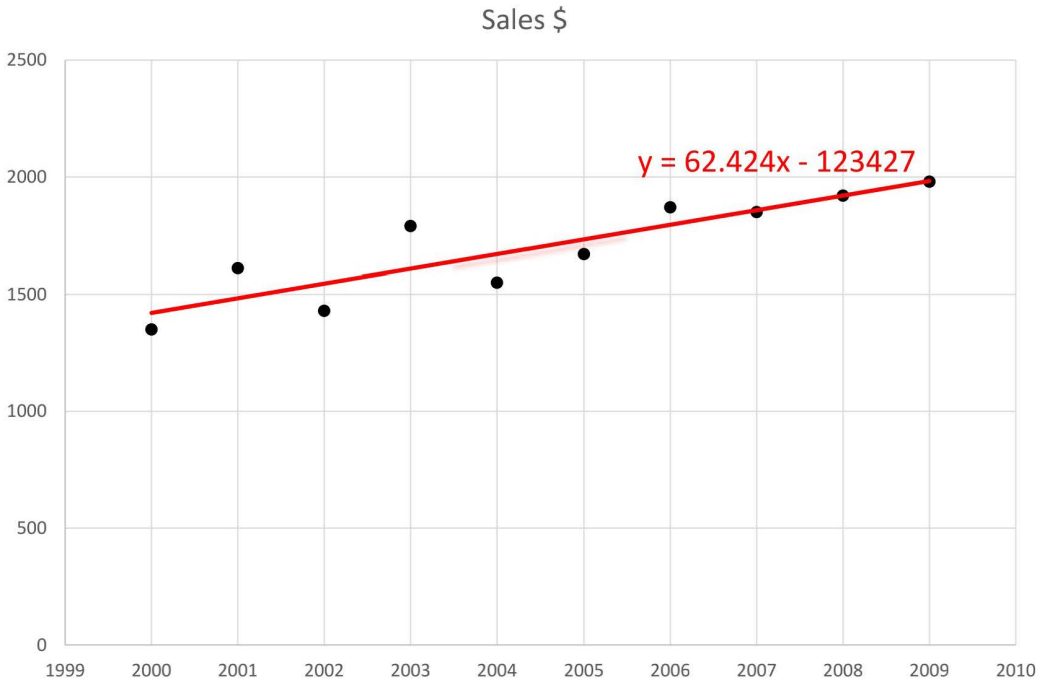


Simple Linear Regression

Estimate the regression line (hypothesis function) for the given data.

Year (*x*)	**Sales $ (*y*)**
2000	1350
2001	1610
2002	1430
2003	1790
2004	1550
2005	1670
2006	1870
2007	1850
2008	1920
2009	1980

Simple Linear Regression

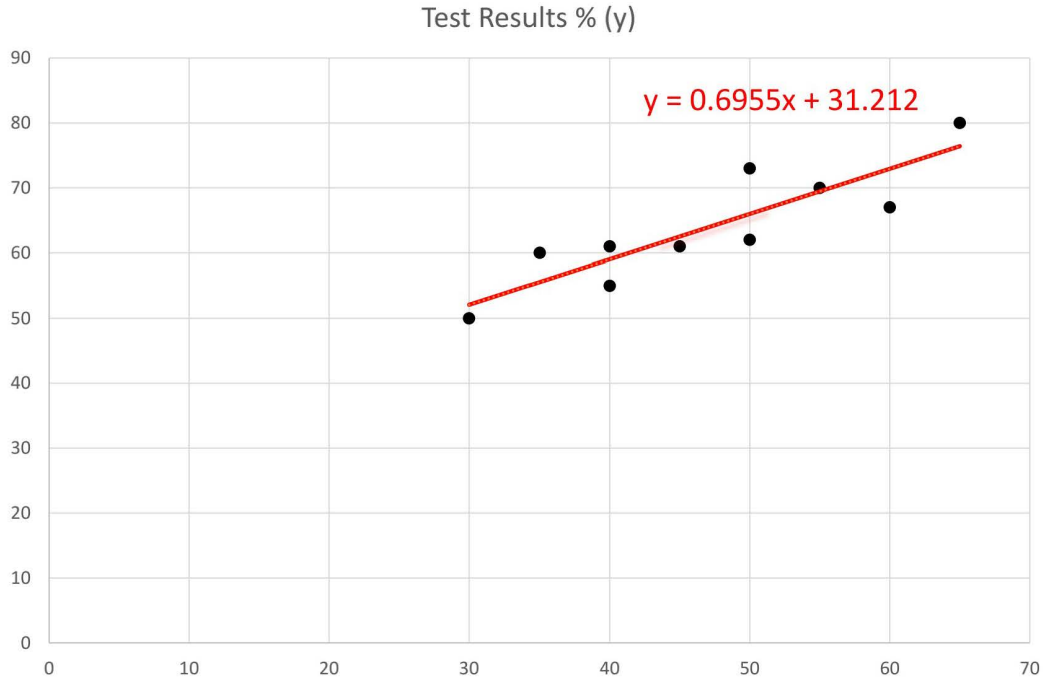


Simple Linear Regression

Estimate the regression line (hypothesis function) for the given data.

Study Hours (*x*)	**Test Results % (*y*)**
60	67
40	61
50	73
65	80
35	60
40	55
50	62
30	50
45	61
55	70

Simple Linear Regression



Linear Regression

Find out linear Regression Eqn for Given data

Student	x_i	y_i	A $x_i - \bar{x}$	B $y_i - \bar{y}$	AB	A^2
1	95	85	17	8	136	289
2	85	95	7	18	126	49
3	80	70	2	-7	-14	4
4	70	65	-8	-12	96	64
5	60	70	-18	-7	126	324
Total	390	385			470	730

$$\bar{x} = \frac{95+85+80+70+60}{5} = 78$$

$$\bar{y} = \frac{85+95+70+65+70}{5} = 77$$

Now

$$\beta_1 = \frac{\Sigma AB}{\Sigma A^2} = \frac{470}{730} = 0.64$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

$$= 77 - (0.64)78$$

$$= 27.08$$

Then,

when $x = 80$ then $y = ?$

for this we use regression equation

$$Y = \beta_0 + \beta_1 X \Rightarrow Y = 27.08 + (0.64)80$$

$$Y = 78.28$$

Evaluation of Model Estimators

- Once the model is established, you need to confirm whether the model is good enough to make predictions
- Various metrics are used
  - Karl Pearson's Coefficient of Correlation
  - R-Square
  - Multiple-R
  - Standard Error of Estimate

Karl Pearson's Coefficient of Correlation

Karl Pearson's coefficient of correlation is a helpful statistical formula that quantifies the strength between two variables.

This coefficient value helps in determining how strong that relationship is between the two variables.

$$r = \frac{N\sum xy - \sum x \sum y}{\sqrt{\left[N\sum x^2 - \left(\sum x\right)^2\right]\left[N\sum y^2 - \left(\sum y\right)^2\right]}}$$

where x and y are variables and N is the number of instances.

Karl Pearson's Coefficient of Correlation

- It has a value between +1 and −1
- 1 is total positive linear correlation
- 0 is no linear correlation
- −1 is total negative linear correlation
- Limitations of Pearson coefficients
  - It always assumes linear relationship.
  - Interpreting the value of *r* is difficult.
  - Value of the correlation coefficient is affected by extreme values.
  - It is time consuming.

Karl Pearson's Coefficient of Correlation

Estimate the Karl Pearson's correlation coefficient for the given data.

Height (x)	**Weight (y)**
151	63
174	81
138	56
186	91
128	47
136	57
179	76
163	72
152	62
131	48

Karl Pearson's Coefficient of Correlation

Height (x)	Weight (y)	xy	x^2	y^2
151	63			
174	81			
138	56			
186	91			
128	47			
136	57			
179	76			
163	72			
152	62			
131	48			
$\sum x$	$\sum y$	$\sum xy$	$\sum x^2$	$\sum y^2$

Karl Pearson's Coefficient of Correlation

Height (x)	Weight (y)	xy	x^2	y^2
151	63	9513	22801	3969
174	81	14094	30276	6561
138	56	7728	19044	3136
186	91	16926	34596	8282
128	47	6016	16384	2209
136	57	7752	18496	3249
179	76	13604	32041	5776
163	72	11736	26569	5184
152	62	9424	23104	3844
131	48	6288	17161	2304
$\sum x = 1538$	$\sum y = 653$	$\sum xy = 103081$	$\sum x^2 = 240472$	$\sum y^2 = 44513$

Karl Pearson's Coefficient of Correlation

$$r = \frac{N * \sum xy - \sum x \sum y}{\sqrt{[N * \sum x^2 - (\sum x)^2] * [N * \sum y^2 - (\sum y)^2]}}$$

$$r = \frac{10 * 103081 - 1538 * 653}{\sqrt{[10 * 240272 - 1538^2] * [10 * 44513 - 653^2]}}$$

$r = 0.977$

R-Square

- R-square gives information about the goodness-of-fit measure for linear regression models
- It indicates percentage of variance in the dependent–independent variable pair.
- It measures the strength of the relationship in a 0 to 100% scale.

R-Square

For the regression line,

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

where

y_i is the observed output for the input x_i
$\overline{y}$ is the mean of the observed outputs
$\hat{y}_i$ is the estimated output for the input x_i

$\sum(y_i - \overline{y})^2 =$ SST is the total sum of squared deviations in y from its mean.
$\sum(\hat{y}_i - \overline{y})^2 =$ SSR is the sum of squares due to regression.
$\sum(y_i - \hat{y}_i)^2 =$ SSE is the sum of squared residuals (errors).

R-Square

Relationship Among SST, SSR, SSE

SST = SSR + SSE

$$\sum (y_i - \bar{y})^2 = \sum (\hat{y}_i - \bar{y})^2 + \sum (y_i - \hat{y}_i)^2$$

where:

SST = total sum of squares
SSR = sum of squares due to regression
SSE = sum of squares due to error

R-Square

The coefficient of determination is:

$$r^2 = SSR/SST$$

where:

SSR = sum of squares due to regression

SST = total sum of squares

R-Square

Estimate the R-Square for the given data.

Height (x_i)	Weight (y_i)
151	63
174	81
138	56
186	91
128	47
136	57
179	76
163	72
152	62
131	48

R-Square

Height (x_i)	Weight (y_i)	$\hat{y}_i$	$(y_i - \overline{y})^2$	$(\hat{y}_i - \overline{y})^2$	$(y_i - \hat{y}_i)^2$
151	63				
174	81				
138	56				
186	91				
128	47				
136	57				
179	76				
163	72				
152	62				
131	48				
	$\overline{y}$		***SST***	***SSR***	***SSE***

R-Square

Height (x_i)	Weight (y_i)	$\hat{y}_i$	$(y_i - \overline{y})^2$	$(\hat{y}_i - \overline{y})^2$	$(y_i - \hat{y}_i)^2$
151	63	63.410	5.29	3.574	0.168
174	81	78.925	246.49	185.652	4.304
138	56	54.640	86.49	113.640	1.850
186	91	87.021	660.49	471.784	15.836
128	47	47.894	334.89	302.976	0.799
136	57	53.291	68.89	144.226	13.760
179	76	82.298	114.49	288.946	39.670
163	72	71.505	44.89	38.500	0.245
152	62	64.084	10.89	1.478	4.344
131	48	49.918	299.29	236.618	3.677
	$\overline{y} = 65.3$		***SST* = 1872.1**	***SSR* = 1787.392**	***SSE* = 84.652**

R-Square

$$R^2 = \frac{SSR}{SST} = \frac{178.392}{1872.1} = 0.955$$

Multiple R

- Multiple R is a correlation coefficient.
- Gives the strength of a linear relationship
- It is the square root of R-Square.

$$Multiple\ R = \sqrt{R\text{–}Square} = \sqrt{\frac{SSR}{SST}} = \sqrt{0.955} = 0.977$$

Standard Error of Estimate

The standard error of the estimate is a measure of the accuracy of predictions. It is the measure of variation of an observation made around the computed regression line.

$$\sigma_{est} = \sqrt{\frac{\sum(\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^2}{N}} = \sqrt{\frac{SSE}{N}}$$

The denominator is the sample size (n) reduced by the number of model parameters (p) estimated from the same data.

$N = (n - p)$ for p regressors

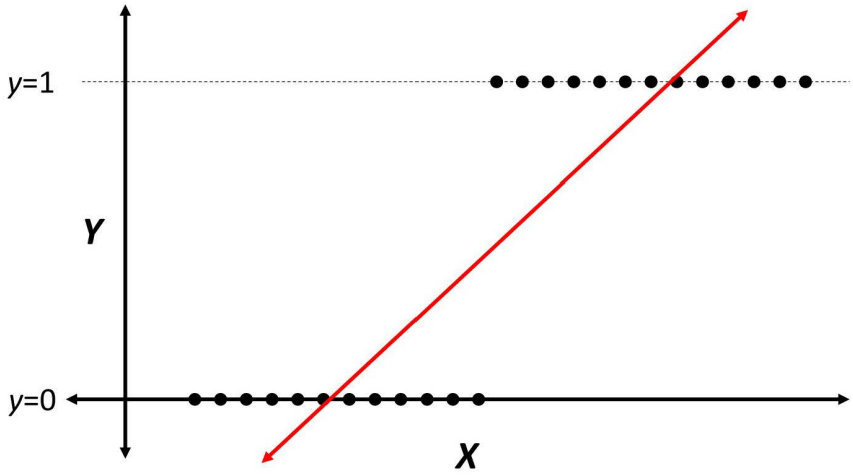
$N = (n - p - 1)$ if an intercept is used

Standard Error of Estimate

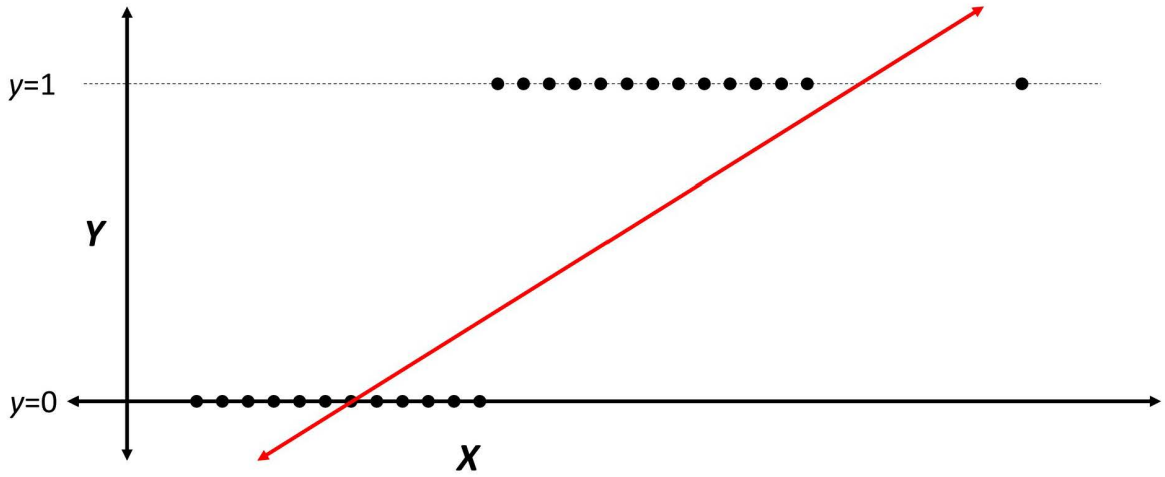
$$\sigma_{est} = \sqrt{\frac{\sum(\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^2}{N}} = \sqrt{\frac{SSE}{N}} = \sqrt{\frac{84.652}{8}} = 3.253$$

Logistic Regression

Why not linear regression?



Why not linear regression?



Examples of Binary Classification Model

- Spam Detection: Predicting if an email is Spam or not.
- Credit Card Fraud: Predicting if a given credit card transaction is fraud or not.
- Health: Predicting if a given mass of tissue is benign or malignant.
- Marketing: Predicting if a given user will buy an insurance product or not.
- Banking: Predicting if a customer will default on a loan.

Logistic Regression

- In linear regression, the response Y is *continuous*.
- If output is *discrete*, it is a classification problem.

- Logistic Regression is a binary classification algorithm used when the response variable is dichotomous (1 or 0).
- Output: A random variable Y_i that take values (1 and 0) with probabilities p_i and $1 - p_i$, respectively.

Why not linear regression?

- Let p denote probability that $Y = 1$ when $X = x$.
- For linear model to describe p, the model for the probability would be

$$p = \text{Probability}(Y = 1|X = x) = \beta_0 + \beta_1 x$$

- Since, p is probability it must lie between 0 and 1.
- The linear function is unbounded, and hence cannot be used to model probability.

Odds

- The odds of an event is the ratio of the expected number of times that an event will occur to the expected number of times it will not occur.
- If p is the probability of an event and O is the odds of the event, then

$$O = \frac{p}{1-p} = \frac{probability\ of\ event}{probability\ of\ no\ event}$$

- Transforming the probability to odds removes the upper bound.

Why Odds?

- Unlike probabilities, there is no upper bound on the odds.
- However, Odds do have lower limit.

Logit Function

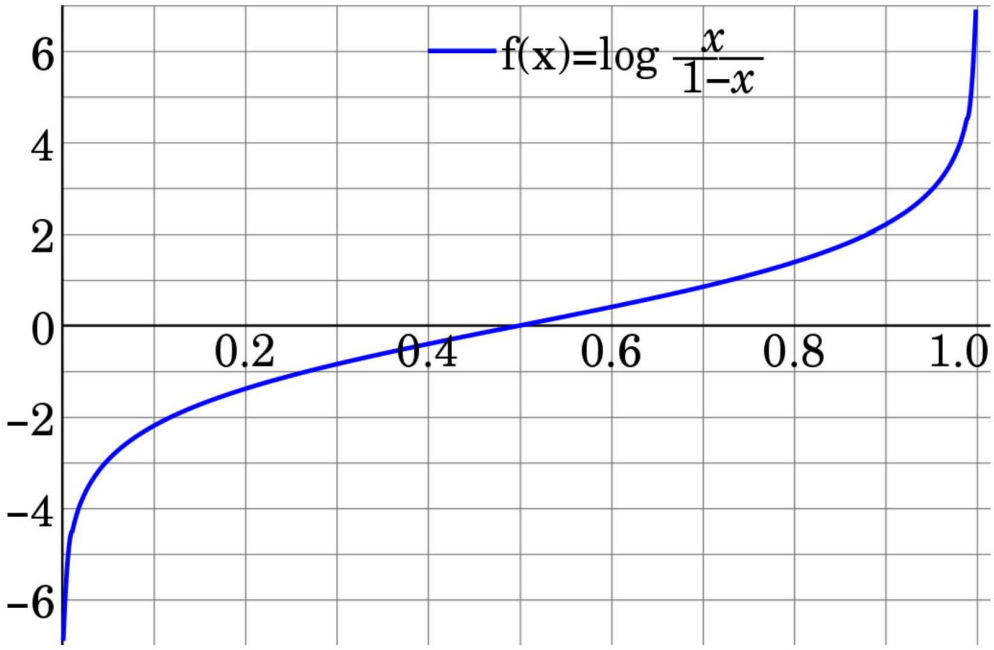
- If we take the logarithm of the odds, we also remove the lower bound.
- Thus, we get the mathematical model as,

$$\log\left[\frac{p_i}{1-p_i}\right] = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik}$$

- The expression $\log\left[\frac{p_i}{1-p_i}\right]$ is called logit function.

Note: In the above model, we have taken the natural log.

Logit Function



Logistic Regression

What is p_i?

Logistic Regression

- We can solve the logit equations for p_i to obtain

$$\frac{p_i}{1-p_i} = e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_k x_{ik})}$$

$$\frac{1-p_i}{p_i} = \frac{1}{e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_k x_{ik})}}$$

$$\frac{1-p_i}{p_i} + 1 = \frac{1}{e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_k x_{ik})}} + 1$$

Logistic Regression

$$\frac{1}{p_i} = \frac{1 + e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_k x_{ik})}}{e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_k x_{ik})}}$$

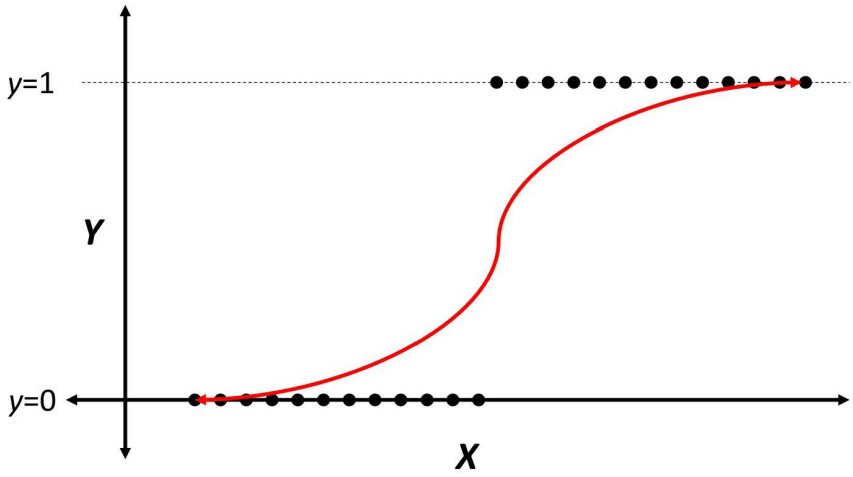
$$\frac{p_i}{1} = \frac{e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_k x_{ik})}}{1 + e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_k x_{ik})}}$$

- We can simplify it by dividing both numerator and denominator by numerator to obtain:

$$p_i = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_k x_{ik})}}$$

Logistic Regression

This function is the Logistic Function.



Maximum Likelihood Estimation

- In linear regression, we used the *method of least squares* to estimate regression coefficients.
- In logistic regression, we use another approach called *maximum likelihood estimation*.
- The maximum likelihood estimate of a parameter is that value that maximizes the probability of the observed data.

Maximum Likelihood Estimation

- Likelihood function is probability that the observed values of the dependent variable may be predicted from the observed values of the independent variables.
- The likelihood varies from 0 to 1
- It is easier to work with the logarithm of the likelihood function. This function is known as the log-likelihood.

Maximum Likelihood Estimation

- Suppose in a population, each individual has the same probability p that an event occurs.
- For sample of size n, $Y_i = 1$ indicates that an event occurs for the i^{th} subject, otherwise, $Y_i = 0$.
- The observed data are $Y_1, Y_2, \ldots Y_n$ and $X_1, X_2, \ldots X_n$.

Maximum Likelihood Estimation

- The joint probability of the data (the likelihood) is given by

$$L=\prod_{i=1}^{n} p\left(y/x\right)^{Y_i}\left(1-p\left(y/x\right)\right)^{1-Y_i}=p\left(y/x\right)^{\sum_{i=1}^{n}Y_i}\left(1-p\left(y/x\right)\right)^{n-\sum_{i=1}^{n}Y_i}$$

- Natural logarithm of the likelihood is

$$l=\log\left(L\right)=\sum_{i=1}^{n} Y_i \log\left[p\left(y/x\right)\right]+\left(n-\sum_{i=1}^{n} Y_i\right)\log\left[1-p\left(y/x\right)\right]$$

- In which

$$p\left(y/x\right)=\frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

Maximum Likelihood Estimation

- Estimating the parameters α and β is done using the first derivatives of log-likelihood, and solving them for α and β.
- For this, iterative computing is used.
- An arbitrary value for the coefficients (usually 0) is first chosen.
- Then log-likelihood is computed and variation of coefficients values observed.
- Reiteration is then performed until maximization of L.
- The results are the maximum likelihood estimates of α and β.

Estimating the Parameters

$$h(x_i) = g(\beta^T x_i) = \frac{1}{1 + e^{-\beta^T x_i}}$$

$$P(y_i = 1 | x_i; \beta) = h(x_i)$$

$$P(y_i = 0 | x_i; \beta) = 1 - h(x_i)$$

$$P(y_i | x_i; \beta) = (h(x_i))^{y_i} (1 - h(x_i))^{1-y_i}$$

Estimating the Parameters

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_i'=0} \left(1 - p\left(x_i'\right)\right)$$

$$L(\beta) = \prod_{i=1}^{n} \left(h(x_i)\right)^{y_i} \left(1 - h(x_i)\right)^{1-y_i}$$

$$\ell(\beta) = \log\left(L(\beta)\right)$$

Estimating the Parameters

$$\ell(\beta) = \sum_{i=1}^{n} y_i \log\left(h(x_i)\right) + (1 - y_i) \log\left(1 - h(x_i)\right)$$

Using Gradient Descent Algorithm

Cost Function:

$$J(\beta) = -\frac{1}{n}\left[\sum_{i=1}^{n} y_i \log\big(h(x_i)\big) + (1 - y_i)\log\big(1 - h(x_i)\big)\right]$$

Objective:

$$\min_{\beta} J(\beta)$$

Using Gradient Descent Algorithm

$Repeat\ \{$

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$ (simultaneously update all β_j)

$\}$

$Repeat\ \{$

$$\beta_j := \beta_j - \alpha \sum_{i=1}^{n} (h(x_i) - y_i) x_{ij}$$ (simultaneously update all β_j)

$\}$

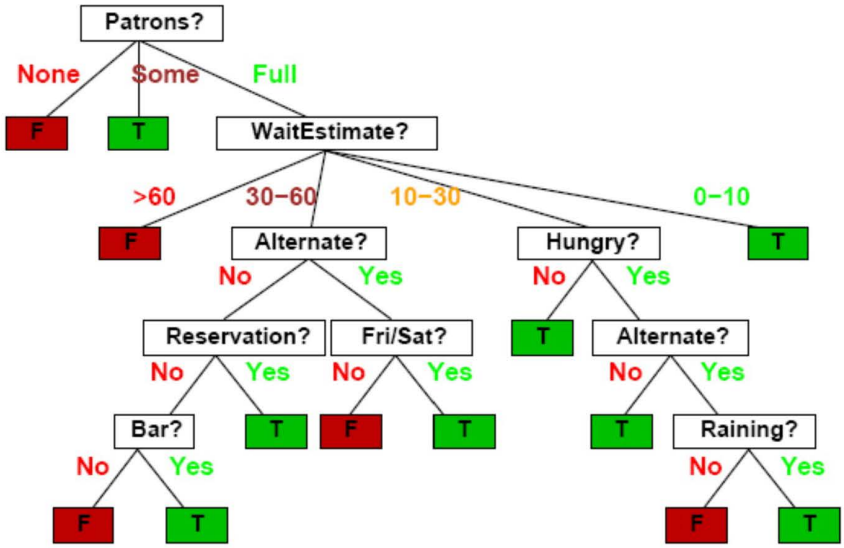
Decision Trees

Classification and Decision Tree

- A tree is build in which the leaf nodes contain the output category.
- The class of the output is predicted based on the rules generated from the tree structure.
- Learned trees can be represented as a set of IF–THEN rules as well.

Decision Trees

Should I wait at this restaurant?



Classification and Decision Tree

- A decision tree is a classifier that partitions data recursively into to form groups or classes.
- This is a supervised learning algorithm which can be used in discrete or continuous data for classification or regression.
- The Algorithm used in the decision trees are ID3 , C4.5, CART, C5.0, CHAID, QUEST, CRUISE, etc.
- The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called “root” that has no incoming edges.
- All other nodes have exactly one incoming edge.
- A node with outgoing edges is called an internal or test node.
- All other nodes are called leaves.

Problem Solving Using Decision Trees

Decision trees can be used to solve problems that have the following features:

- Instances or tuples are represented as attribute value pairs, where the attributes take a small number of disjoint possible values.
- The target function has discrete output values such as yes or no.
- Decision trees require disjunctive descriptions which implies that the output of the decision tree can be represented using a rule-based classifier.
- Decision tree can be used when training data contains errors and when it contains missing attribute values.

NOTES FROM Date

Last Moment tuitions

Decision tree notes

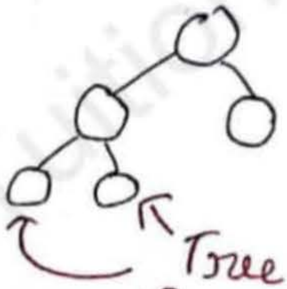
Age	Competition	Type	Profit
old	Yes	Software	Down
old	NO	Software	Down
old	NO	Hardware	Down
mid	Yes	Software	Down
mid	yes	Hardware	Down
mid	NO	Hardware	UP
mid	NO	Software	UP
new	Yes	Software	UP
new	NO	Hardware	UP
new	NO	Software	UP

Draw a decision tree for given dataset ?
yeh question us university exam me 10 marks ke liye Pucha gaya tha

Let see how to solve this

- Dekh bhai koi bhi dataset ya Table diya ho uska jo last column hota hai na voh hota hai class ATTRIBUTE

- jaise apne table me PROFIT hai aur uski jo value rahengi voh honge mere leaf node



Tree ke jo last wale node ko leaf node kehta hai

Given qushon me Decision tree isliye Pucha hai meko aise decision tree banake jo jisse Pata chale mera Profit UP and down kaise honge.

$$I(P_i, N_i) = \frac{-P}{P+N} \log_2\left(\frac{P}{P+N}\right) - \frac{N}{P+N} \log_2\left(\frac{N}{P+N}\right)$$

$P = 0, \; N = 3.$

$$I(P_i, N_i)_{old} = \frac{-0}{0+3} \log_2\left(\frac{0}{0+3}\right) - \frac{3}{0+3} \log_2\left(\frac{3}{0+3}\right)$$

(Dekh bhai jab P ya N dono me se ek bhi valu 0 hai to hr answer bhi 0 hi hoga)

$I(P_i, N_i)_{old} = 0$

similarly we find for

$P = 2, \; N = 2$

$$I(P_i, N_i)_{mid} = \frac{-2}{4} \log_2\left(\frac{2}{4}\right) - \frac{2}{4} \log_2\left(\frac{2}{4}\right)$$

(when P=N then entrophy will be 1)

$I(P_i, N_i)_{mid} = 1$

$P=3, N=0$

$I(P_i, N_i) = 0$

New

∴ Entropy of Age

$$= \sum \frac{P_i + N_i}{P+N} I(P_i, N_i)$$

↖ yeh P and N class attribute ki value hai where P=5, N=5

$$= \frac{0+3}{10}(0) + \frac{2+2}{10}(1) + \frac{3+0}{10}(0)$$

$$= \frac{4}{10} = 0.4$$

∴ Gain = Entropy(class) − Entropy of Age

↖ (yeh jo humne starting Pe Profit ki entrophy nikali thi voh hai)

$= 1 - 0.4$

$= 0.6$

Step 1:-

In Profit

Ut
Class P = Profit (up) = 5
Class N = Profit (down) = 5

(Toh hame class attribute hai Profit
usko entrophy nikalna hai)

Entrophy (P, N) =

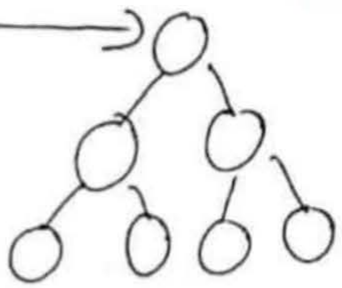
$$\frac{-P}{P+N}\log_2\left(\frac{P}{P+N}\right) - \frac{N}{P+N}\log_2\left(\frac{N}{P+N}\right)$$

after putting value of P=5, N=6 ae
get

$$= \frac{-5}{10}\log_2\left(\frac{5}{10}\right) - \frac{5}{10}\log_2\left(\frac{5}{10}\right)$$

Entrophy (Profit) = 1

chalo itna Pata chal gaya ki up and dow
mere leaf node honge so questions
comes root node kon honga



To find this - hum saare column ka
gain nikalenge aur compare karenge
jiska gain sabse zyaada bad voh root node

For Age	Pi	Ni	I(Pi, Ni)
old	0	3	0
mid	2	2	1
new	3	0	0

Now for Competition

	Pi	Ni	I(Pi, Ni)
Yes	1	3	0.81127
NO	4	2	

$$I(P_i, N_i)_{Yes} = \frac{-1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right)$$

$$= 0.81127$$

$$I(P_i, N_i)_{NO} = \frac{-4}{6}\log_2\left(\frac{4}{6}\right) - \frac{2}{6}\log_2\left(\frac{2}{6}\right)$$

$$= 0.918295$$

Entrophy (competition)

$$= \sum \frac{P_i + N_i}{P+N} (I(P_i, N_i))$$

$$= \frac{(1+3)}{10}(0.81127) + \frac{(4+2)}{5+5}(0.918295)$$

$$= 0.8754$$

Gain = Class Entrophy − Entrophy (competition)

= 1 − 0.8754

= 0.124515.

for Type

	P_i	N_i	$I(P_i, N_i)$
software	3	3	1
hardware	2	2	1

$I(P_i, N_i)$ = 1 ∵because $P = N$
software

$I(P_i, N_i)$ = 1
Hardware

Entrophy (Type)

$$= \frac{3+3}{5+5}(1) + \frac{(2+2)}{5+5}(1)$$

$$= \frac{6}{10} + \frac{4}{10} = \frac{10}{10} = 1$$

Gain = Class entrophy − Entrophy (Type)

$$= 1 - 1$$

$$= 0$$

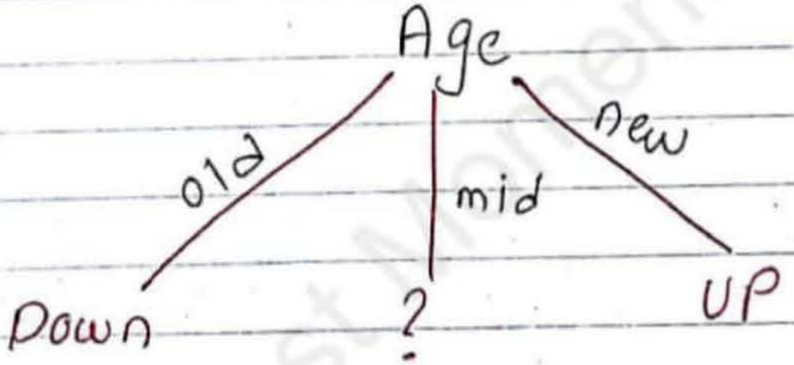
Information Gain

Age — 0.6 ← root node

Competition — 0.124515

Type — 0

- Toh gain sabse zyaada age ka hai root node will be age
- Ab AGE me jitni bhi value hai us utne root node se branches niklenge



Agar app Age old ki values and Profit ki values ko compare kare toh sab value down hai direct down likh do same for new

Age	Profit	Age	Profit
old →	Down	new →	UP
old →	Down	new →	UP
old →	Down	new →	UP

But MID me Up and down dono value ...
isliye Hum direct kuch Put nahi kar sakte
so hame firse Sirf mid ka table banana
Padenga

Age	Competition	Type	Profit
mid	Yes	software	Down
mid	Yes	Hardware	Down
mid	NO	Hardware	UP
mid	NO	Software	UP

Ab same Process firse karna hai
class antribute ki entrophy nikalni hai
and Competition and Type ka gain nikalke
jo bada honga usko mid ka node
banana hai

class P = Profit(UP) = 2
class N = Profit (down) = 2.

Entrophy (Profit) = 1 ∵(P=N).

Now lets find gain for Competition

Competition

	Pi	Ni	I(Pi, Ni)
Yes	0	2	0
NO	0	2	0

$I(P_i, N_i) = 0$ (Yes)

$I(P_i, N_i) = 0$ (NO)

(Koi bhi ek value P or N me 0 hai toh ast answer will be zero)

Entrophy (competition)

$$= \frac{2}{4}(0) + \frac{2}{4}(0) = 0$$

Gain = Entrophy (class) − Entrophy (competition)

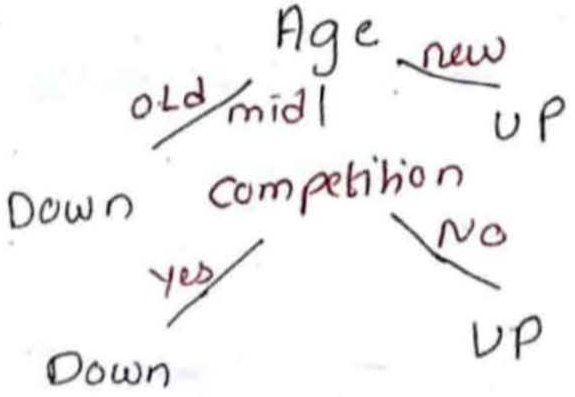
$= 1 - 0$

$= 1$

Gain

Competition → 1

Type → 0



obviously apko doubt aya hoga ki yaar fir Type kaha gaya

dekh bhai jab mai competition (yes) ki and Profit ki value compare kar raha hu toh I m Getting direct answer down toh Type ka jhanjat kya Palneko

similarly for NO = UP

competition	Profit	Competition	Profit
Yes	→ Down	NO	→ UP
Yes	→ Down	NO	→ UP

Toh agar app decision tree ko analyse karo toh we can see ki agar hum new person rakhte hai and voh age me jawan hai toh Profit up and budha hai toh Profit down

Agar age mid hai toh dekhne ka competition kitna hai if Yes toh Profit → down if NO toh Profit → up

Toh iss tarah se real life me decision tree kaam karta hai.

Note: Exams me sab likhne ki jarroorat nahi hai sirf jo ka calculation voh likhna hai baki apke understanding purpose ke liye hai.

Thank you so much to study from last moment tuitions.

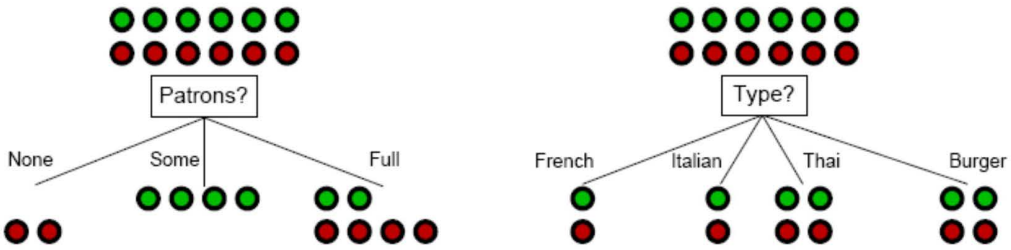
and please let us know if we can add some help to you from last moment tuitions aur ha

Attribute Selection Measures

- Information Gain / Entropy Reduction
- Gain Ratio
- Gini Index

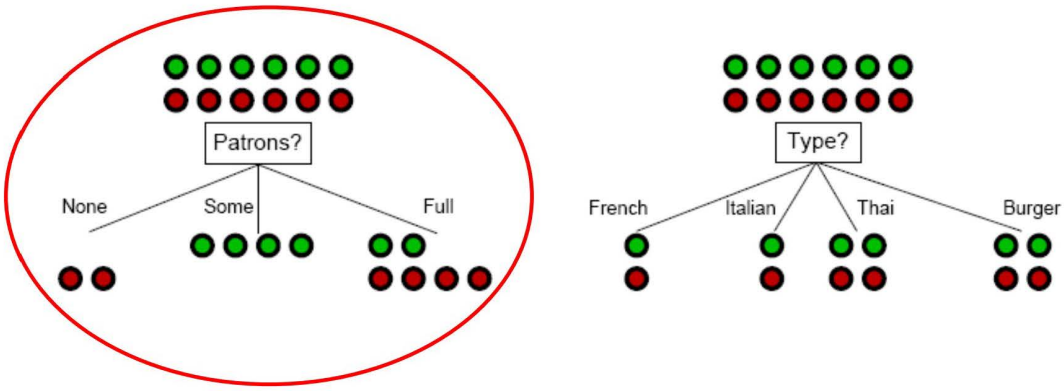
Decision Tree Induction: Attribute Selection

Intuitively: A *good attribute* splits the examples into subsets that are (ideally) *all positive* or *all negative*.

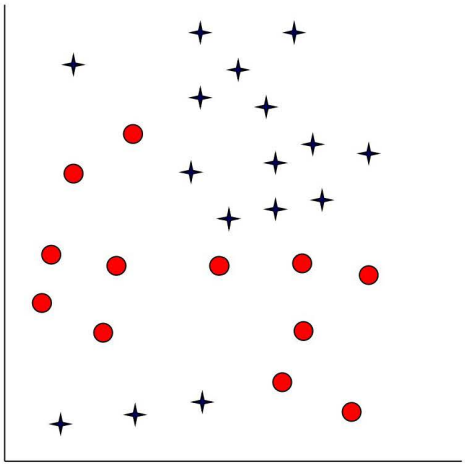


Decision Tree Induction: Attribute Selection

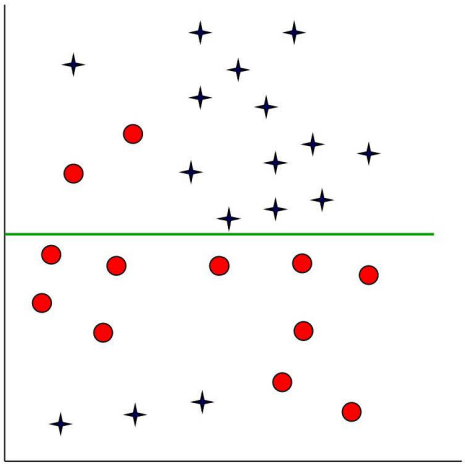
Intuitively: A *good attribute* splits the examples into subsets that are (ideally) *all positive* or *all negative*.



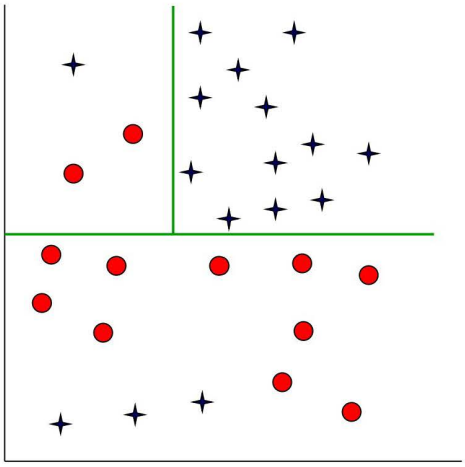
Decision Tree Induction: Decision Boundary



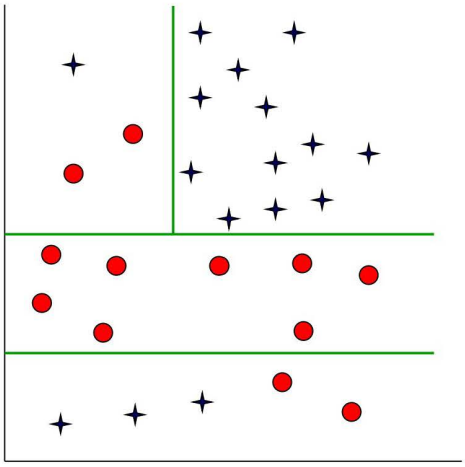
Decision Tree Induction: Decision Boundary



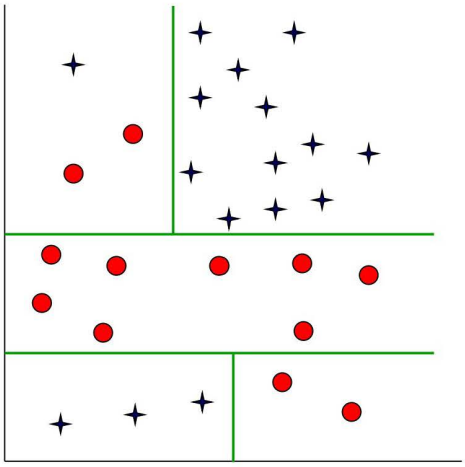
Decision Tree Induction: Decision Boundary



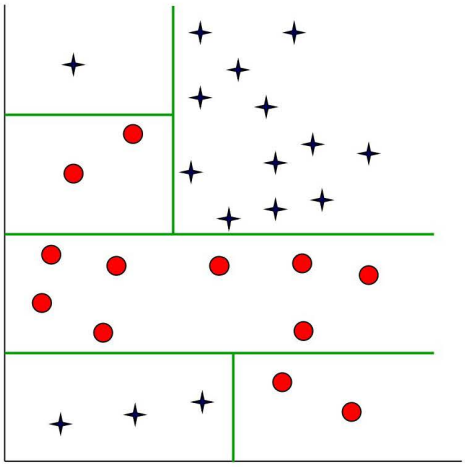
Decision Tree Induction: Decision Boundary



Decision Tree Induction: Decision Boundary



Decision Tree Induction: Decision Boundary



Basic Decision Tree Learning Algorithm

- Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split.
- The attribute having the best score for the measure is chosen as the splitting attribute for the given tuples.

Basic Decision Tree Learning Algorithm

- *Iterative Dichotomiser* 3 (*ID*3) *Algorithm* was the first of three Decision Tree implementations developed by Ross Quinlan.
- It is intended for use with nominal (categorical) inputs only.
- Real-valued variables are first binned into intervals, with each interval being treated as an unordered nominal attribute.
- It builds a decision tree for the given data in a top-down fashion, starting from a set of objects and a specification of properties Resources and Information.

Basic Decision Tree Learning Algorithm

- Each node of the tree, one property is tested based on maximizing information gain and minimizing entropy, and the results are used to split the object set.
- This process is recursively done until the set in a given sub-tree is homogeneous (i.e. it contains objects belonging to the same category).
- The ID3 algorithm uses a greedy search.
- It selects a test using the *Information Gain* criterion, and then never explores the possibility of alternate choices.

Basic Decision Tree Learning Algorithm

Disadvantages

- Data may be over-fitted or over-classified, if a small sample is tested.
- Only one attribute at a time is tested for making a decision.
- Does not handle numeric attributes and missing values.
- It maintains only a single current hypothesis as it searches through the space of decision trees.
- It does not have the ability to determine alternative decision trees.
- It does not perform backtracking in search. Hence, there are chances of getting stuck in local optima.
- It is less sensitive to error because information gain, which is a statistical property is used.

Basic Decision Tree Learning Algorithm

- *C4.5 Algorithm* developed by Ross Quinlan as a successor and refinement of ID3, is the most popular tree-based classification method.
- It uses *Gain Ratio* as an impurity measure.
- In C4.5, multiway splits for categorical variable are treated the same way as in ID3.
- Continuous-valued attributes have been incorporated by dynamically defining new discrete-valued attributes that partition the continuous-values into binary set of intervals.

Basic Decision Tree Learning Algorithm

The new features (versus ID3) are:

- accepts both continuous and discrete features.
- handles incomplete data points.
- solves over-fitting problem by (very clever) bottom-up technique usually known as “pruning”.
- different weights can be applied the features that comprise the training data.

Basic Decision Tree Learning Algorithm

Disadvantages

- C4.5 constructs empty branches with zero values.
- Over fitting happens when algorithm model picks up data with uncommon characteristics, especially when data is noisy.

Basic Decision Tree Learning Algorithm

- *Classification and Regression Trees* (*CART*) which uses *Gini Index* as impurity measure, is characterized by the fact that it constructs binary trees, namely each internal node has exactly two outgoing edges.
- CART can handle both numeric and categorical variables and it can easily handle outliers.

Basic Decision Tree Learning Algorithm

Disadvantages

- It can split on only one variable.
- Trees formed may be unstable.

ID3 Decision Tree

Simplified ID3 Algorithm

1. Compute the Entropy for training data set *S*.
2. For every attribute/feature:
   a. Calculate entropy for all categorical values.
   b. Take average entropy for the current attribute.
   c. Calculate Information Gain for the current Attribute.
3. Pick the highest Information Gain Attribute.
4. Repeat until the desired Decision Tree is generated.

Entropy

- Entropy is an entity that controls the split in data.
- It computes the homogeneity of examples.
- Entropy ranges between 0 and 1
- 0 if all members of S belong to the same class
- 1 if there are equal number of positive and negative examples

$$Entropy(S) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Information Gain / Entropy Reduction

Information gain is the measure of effectiveness of an attribute in classifying the training data.

$$Information\ Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} * Entropy(S_v)$$

Problem based on ID3

Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(1)}$	Sunny	Hot	High	Weak	No
$s^{(2)}$	Sunny	Hot	High	Strong	No
$s^{(3)}$	Overcast	Hot	High	Weak	Yes
$s^{(4)}$	Rain	Mild	High	Weak	Yes
$s^{(5)}$	Rain	Cool	Normal	Weak	Yes
$s^{(6)}$	Rain	Cool	Normal	Strong	No
$s^{(7)}$	Overcast	Cool	Normal	Strong	Yes
$s^{(8)}$	Sunny	Mild	High	Weak	No
$s^{(9)}$	Sunny	Cool	Normal	Weak	Yes
$s^{(10)}$	Rain	Mild	Normal	Weak	Yes
$s^{(11)}$	Sunny	Mild	Normal	Strong	Yes
$s^{(12)}$	Overcast	Mild	High	Strong	Yes
$s^{(13)}$	Overcast	Hot	Normal	Weak	Yes
$s^{(14)}$	Rain	Mild	High	Strong	No

Problem based on ID3

S is a collection of 14 samples.

S attributes are outlook, temperature, humidity and wind.

They can have the following values:

- Outlook = {Sunny, Overcast, Rain}
- Temperature = {Hot, Mild, Cool}
- Humidity = {High, Normal}
- Wind = {Strong, Weak}

Problem based on ID3

$S = [9^+, 5^-]$

Calculate $Entropy(S)$.

Problem based on ID3

$S = [9^+, 5^-]$

Calculate $Entropy(S)$.

$$Entropy(S) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

$$Entropy(S) = -\left(\frac{9}{14}\right) \log_2 \left(\frac{9}{14}\right) - \left(\frac{5}{14}\right) \log_2 \left(\frac{5}{14}\right)$$

$$Entropy(S) = 0.94$$

Problem based on ID3

$Values(Outlook) = Sunny, Overcast, Rain$

$S = [9^+, 5^-]$
$S_{Sunny} = [2^+, 3^-]$
$S_{Overcast} = [4^+, 0^-]$
$S_{Rain} = [3^+, 2^-]$

$Information\ Gain(S, Outlook) = ?$

Problem based on ID3

$Values(Outlook) = Sunny, Overcast, Rain$

$S = [9^+, 5^-]$
$S_{Sunny} = [2^+, 3^-]$
$S_{Overcast} = [4^+, 0^-]$
$S_{Rain} = [3^+, 2^-]$

$Information\ Gain(S, Outlook)$
$= Entropy(S) - \frac{5}{14} * Entropy(S_{Sunny}) - \frac{4}{14} * Entropy(S_{Overcast})$
$- \frac{5}{14} * Entropy(S_{Rain})$

Problem based on ID3

$Values(Outlook) = Sunny, Overcast, Rain$

$S = [9^+, 5^-]$
$S_{Sunny} = [2^+, 3^-]$
$S_{Overcast} = [4^+, 0^-]$
$S_{Rain} = [3^+, 2^-]$

$Information\ Gain(S, Outlook)$
$= Entropy(S) - \frac{5}{14} * Entropy(S_{Sunny}) - \frac{4}{14} * Entropy(S_{Overcast}) - \frac{5}{14} * Entropy(S_{Rain})$

$= 0.94 - \frac{5}{14} * 0.971 - \frac{4}{14} * 0 - \frac{5}{14} * 0.971 = 0.247$

Problem based on ID3

$Values(Temperature) = Hot, Mild, Cool$

$S = [9^+, 5^-]$
$S_{Hot} = [2^+, 2^-]$
$S_{Mild} = [4^+, 2^-]$
$S_{Cool} = [3^+, 1^-]$

$Information\ Gain(S, Temperature) = ?$

Problem based on ID3

$Values(Temperature) = Hot, Mild, Cool$

$S = [9^+, 5^-]$
$S_{Hot} = [2^+, 2^-]$
$S_{Mild} = [4^+, 2^-]$
$S_{Cool} = [3^+, 1^-]$

$Information\ Gain(S, Temperature)$
$= Entropy(S) - \frac{4}{14} * Entropy(S_{Hot}) - \frac{6}{14} * Entropy(S_{Mild}) - \frac{4}{14}$
$* Entropy(S_{Cool})$

$= 0.94 - \frac{4}{14} * 1 - \frac{6}{14} * 0.918 - \frac{4}{14} * 0.811 = 0.029$

Problem based on ID3

$Values(Humidity) = High, Normal$

$S = [9^+, 5^-]$
$S_{High} = [3^+, 4^-]$
$S_{Normal} = [6^+, 1^-]$

$Information\ Gain(S, Humidity) = ?$

Problem based on ID3

$Values(Humidity) = High, Normal$

$S = [9^+, 5^-]$
$S_{High} = [3^+, 4^-]$
$S_{Normal} = [6^+, 1^-]$

$Information\ Gain(S, Humidity)$
$= Entropy(S) - \frac{7}{14} * Entropy(S_{High}) - \frac{7}{14} * Entropy(S_{Normal})$

$= 0.94 - \frac{7}{14} * 0.985 - \frac{7}{14} * 0.592 = 0.151$

Problem based on ID3

$Values(Wind) = Strong, Weak$

$S = [9^+, 5^-]$
$S_{Strong} = [3^+, 3^-]$
$S_{Weak} = [6^+, 2^-]$

$Information\ Gain(S, Wind) =?$

Problem based on ID3

$Values(Wind) = Strong, Weak$

$S = [9^+, 5^-]$
$S_{Strong} = [3^+, 3^-]$
$S_{Weak} = [6^+, 2^-]$

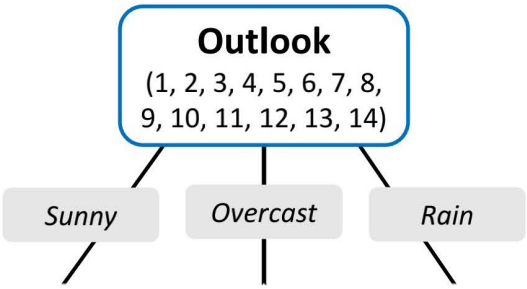
$Information\ Gain(S, Wind)$
$= Entropy(S) - \frac{6}{14} * Entropy(S_{Strong}) - \frac{8}{14} * Entropy(S_{Weak})$

$= 0.94 - \frac{6}{14} * 1 - \frac{8}{14} * 0.811 = 0.048$

Problem based on ID3

- The Outlook Attribute wins pretty handily, so it’s placed at the root of the Decision Tree.
- Afterwards, the Decision Tree is expanded to cover Outlook’s possible values.
- In Play, the Outlook can be Sunny, Overcast or Rain, so all of the examples that have a Sunny Outlook are funneled through the Sunny path, the examples with an Overcast Outlook are diverted to the Overcast path and so on.

Problem based on ID3



Problem based on ID3

Sunny outlook on decision

Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(1)}$	Sunny	Hot	High	Weak	No
$s^{(2)}$	Sunny	Hot	High	Strong	No
$s^{(8)}$	Sunny	Mild	High	Weak	No
$s^{(9)}$	Sunny	Cool	Normal	Weak	Yes
$s^{(11)}$	Sunny	Mild	Normal	Strong	Yes

Problem based on ID3

Gain(Outlook = Sunny|Temperature) = ?

Gain(Outlook = Sunny|Humidity) = ?

Gain(Outlook = Sunny|Wind) = ?

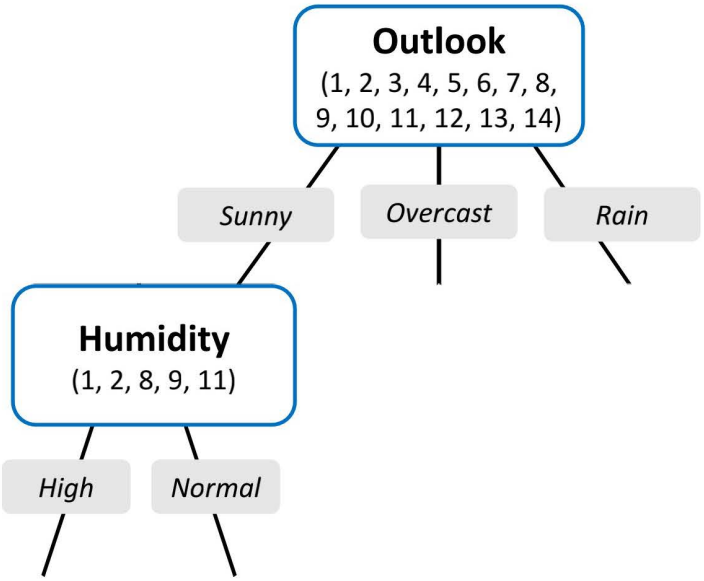
Problem based on ID3

Gain(Outlook = Sunny|Temperature) = 0.971 – (2/5)*0 – (2/5)*1 – (1/5)*0 = 0.571

Gain(Outlook = Sunny|Humidity) = 0.971 – (3/5)*0 – (2/5)*0 = 0.971

Gain(Outlook = Sunny|Wind) = 0.971 – (2/5)*1 – (3/5)*0.918 = 0.02

Problem based on ID3



Problem based on ID3

High humidity on decision

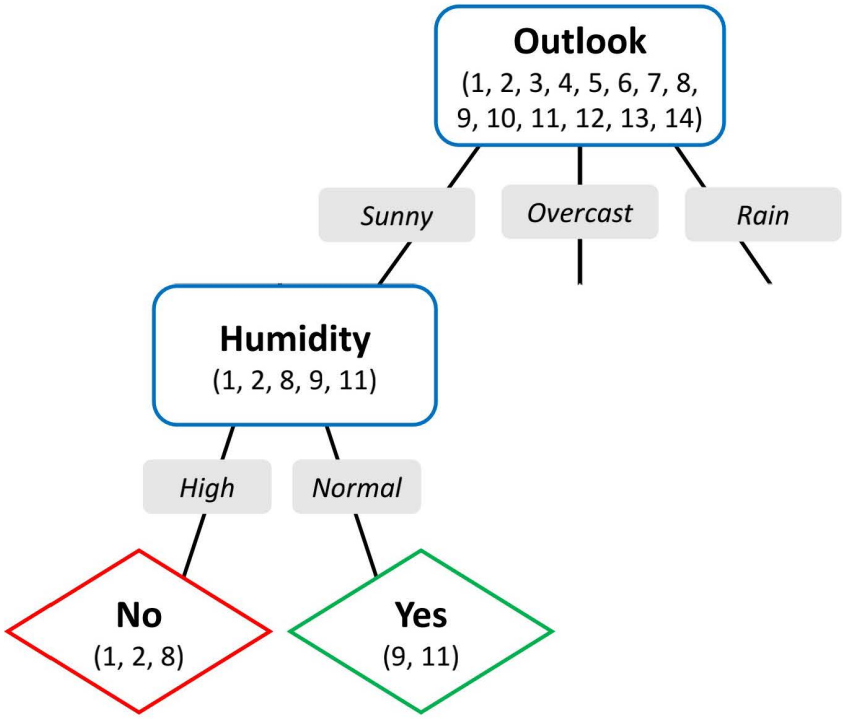
Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(1)}$	Sunny	Hot	High	Weak	No
$s^{(2)}$	Sunny	Hot	High	Strong	No
$s^{(8)}$	Sunny	Mild	High	Weak	No

Problem based on ID3

Normal humidity on decision

Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(9)}$	Sunny	Cool	Normal	Weak	Yes
$s^{(11)}$	Sunny	Mild	Normal	Strong	Yes

Problem based on ID3

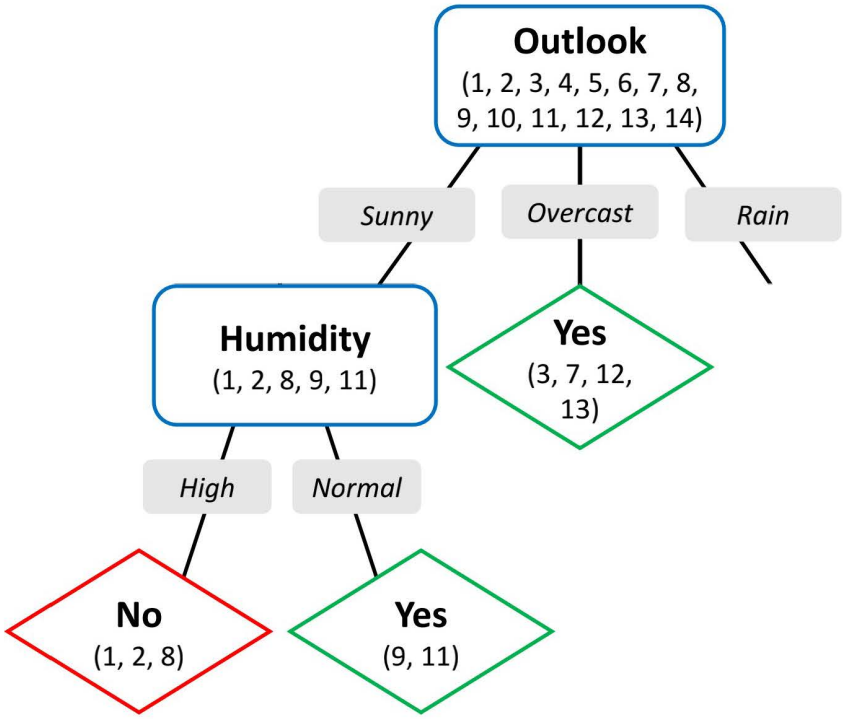


Problem based on ID3

Overcast outlook on decision

Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(3)}$	Overcast	Hot	High	Weak	Yes
$s^{(7)}$	Overcast	Cool	Normal	Strong	Yes
$s^{(12)}$	Overcast	Mild	High	Strong	Yes
$s^{(13)}$	Overcast	Hot	Normal	Weak	Yes

Problem based on ID3



Problem based on ID3

Rain outlook on decision

Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(4)}$	Rain	Mild	High	Weak	Yes
$s^{(5)}$	Rain	Cool	Normal	Weak	Yes
$s^{(6)}$	Rain	Cool	Normal	Strong	No
$s^{(10)}$	Rain	Mild	Normal	Weak	Yes
$s^{(14)}$	Rain	Mild	High	Strong	No

Problem based on ID3

Gain(Outlook = Rain|Temperature) = ?

Gain(Outlook = Rain|Humidity) = ?

Gain(Outlook = Rain|Wind) = ?

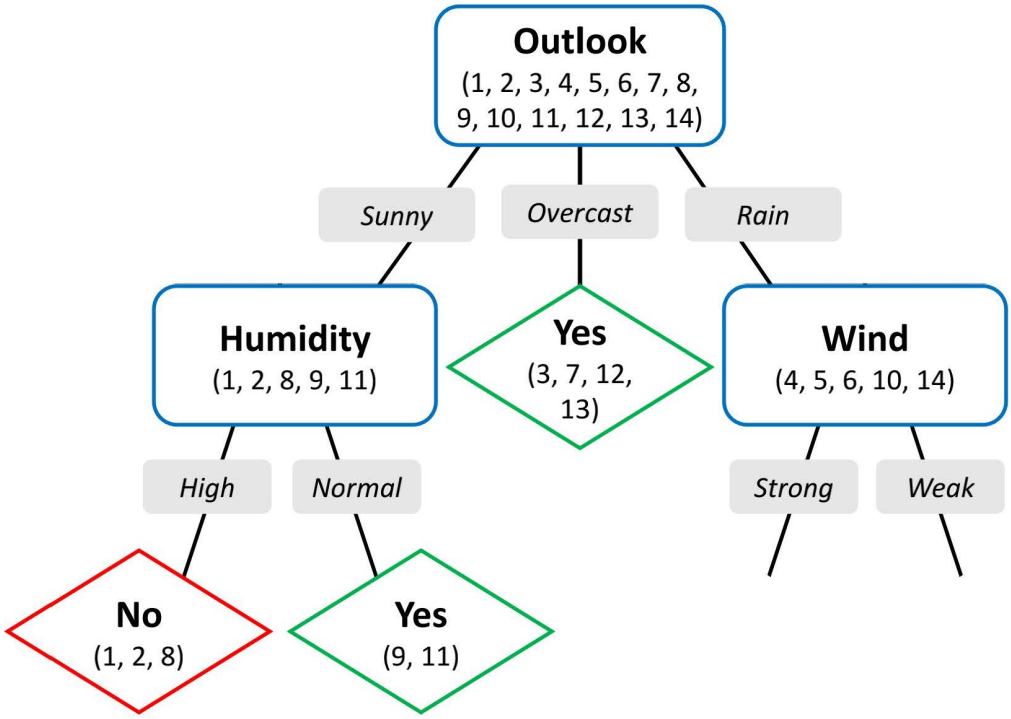
Problem based on ID3

Gain(Outlook = Rain|Temperature) = 0.971 – (3/5)*0.918 – (2/5)*1 = 0.02

Gain(Outlook = Rain|Humidity) = 0.971 – (2/5)*1 – (3/5)*0.918 = 0.02

Gain(Outlook = Rain|Wind) = 0.971 – (2/5)*0 – (3/5)*0 = 0.971

Problem based on ID3



Problem based on ID3

Strong wind on decision

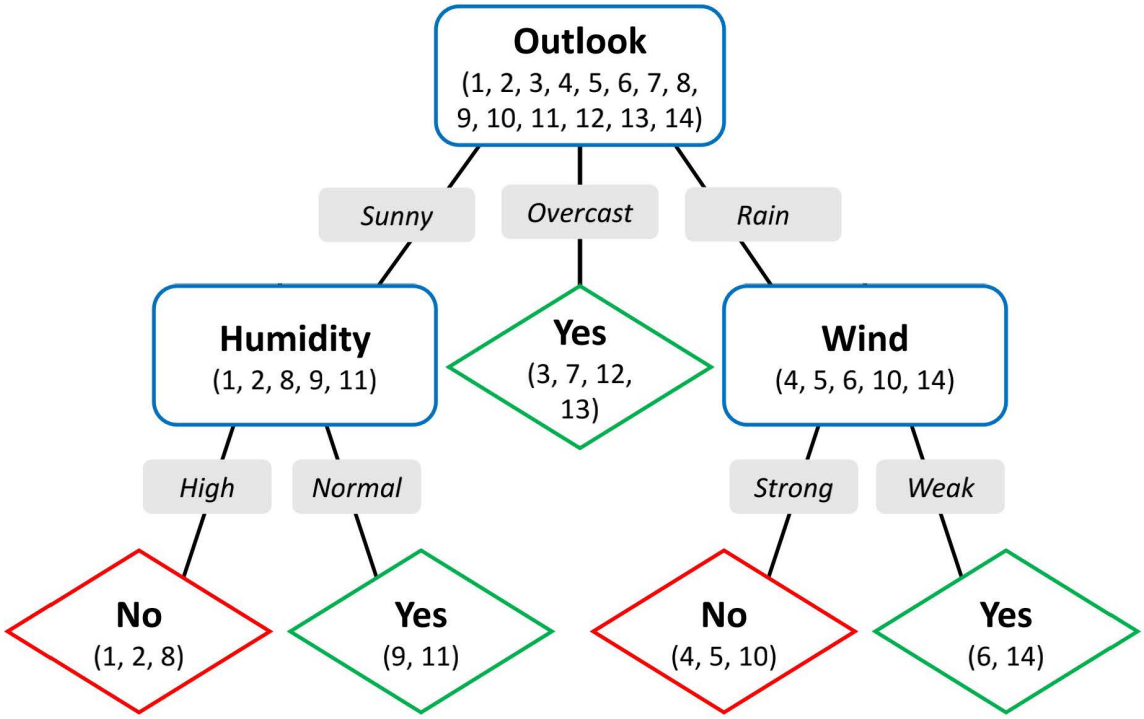
Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(6)}$	Rain	Cool	Normal	Strong	No
$s^{(14)}$	Rain	Mild	High	Strong	No

Problem based on ID3

Weak wind on decision

Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(4)}$	Rain	Mild	High	Weak	Yes
$s^{(5)}$	Rain	Cool	Normal	Weak	Yes
$s^{(10)}$	Rain	Mild	Normal	Weak	Yes

Problem based on ID3



Problem based on ID3

Instance	Eye-colour (x_1)	Married (x_2)	Gender (x_3)	Harilength (x_4)	Class $(\mathbf{y})$
$s^{(1)}$	Brown	Yes	Male	Long	Football
$s^{(2)}$	Blue	Yes	Male	Short	Football
$s^{(3)}$	Brown	Yes	Male	Long	Football
$s^{(4)}$	Brown	No	Female	Long	Netball
$s^{(5)}$	Brown	No	Female	Long	Netball
$s^{(6)}$	Blue	No	Male	Long	Football
$s^{(7)}$	Brown	No	Female	Long	Netball
$s^{(8)}$	Brown	No	Male	Short	Football
$s^{(9)}$	Brown	Yes	Female	Short	Netball
$s^{(10)}$	Brown	No	Female	Long	Netball
$s^{(11)}$	Blue	No	Male	Long	Football
$s^{(12)}$	Blue	No	Male	Short	Football

Problem based on ID3

Instance	Temperature (x_1)	Wind (x_2)	Humidity (y)
$s^{(1)}$	Hot	Weak	Normal
$s^{(2)}$	Hot	Strong	High
$s^{(3)}$	Mild	Weak	Normal
$s^{(4)}$	Mild	Strong	High
$s^{(5)}$	Cool	Weak	Normal
$s^{(6)}$	Mild	Strong	Normal
$s^{(7)}$	Mild	Weak	High
$s^{(8)}$	Hot	Strong	Normal
$s^{(9)}$	Mild	Strong	Normal
$s^{(10)}$	Cool	Strong	Normal

Problem based on ID3

Name	Hair	Height	Weight	Location	Class
Sunita	Blonde	Average	Light	No	Yes
Anita	Blonde	Tall	Average	Yes	No
Kavita	Brown	Short	Average	Yes	No
Sushma	Blonde	Short	Average	No	Yes
Xavier	Red	Average	Heavy	No	Yes
Balaji	Brown	Tall	Heavy	No	No
Ramesh	Brown	Average	Heavy	No	No
Swetha	Blonde	Short	Light	Yes	No

Problem based on ID3

Instance	x_1	x_2	x_3	x_4	y
$s^{(1)}$	Low	Medium	Medium	Low	False
$s^{(2)}$	Low	Medium	High	Medium	False
$s^{(3)}$	Low	Medium	Medium	High	False
$s^{(4)}$	High	Medium	Medium	Low	False
$s^{(5)}$	High	High	Medium	Medium	True
$s^{(6)}$	Low	High	Medium	Low	False
$s^{(7)}$	Low	Medium	High	Low	True
$s^{(8)}$	High	High	Low	Medium	False
$s^{(9)}$	Low	Medium	Medium	Medium	False
$s^{(10)}$	Low	Low	High	Medium	False
$s^{(11)}$	High	Low	Medium	Medium	True
$s^{(12)}$	Low	Low	Medium	High	False

Problem based on ID3

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Multiclass Classification

$$Entropy(S) = -\sum_{i=1}^{N} p_i \log_2 p_i$$

where,

N is the number of classes

p_i is the probability of an element to belong in class i

Problem based on ID3

Instance	Gender	Car Ownership (x_1)	Travel Cost (x_2)	Income Level (x_3)	Transportation (y)
$s^{(1)}$	Male	0	Cheap	Low	Bus
$s^{(2)}$	Male	1	Cheap	Medium	Bus
$s^{(3)}$	Female	1	Cheap	Medium	Train
$s^{(4)}$	Female	0	Cheap	Low	Bus
$s^{(5)}$	Male	1	Cheap	Medium	Bus
$s^{(6)}$	Male	0	Standard	Medium	Train
$s^{(7)}$	Female	1	Standard	Medium	Train
$s^{(8)}$	Female	1	Expensive	High	Car
$s^{(9)}$	Male	2	Expensive	Medium	Car
$s^{(10)}$	Female	2	Expensive	High	Car

Decision Tree Tutorial – Kardi Teknomo

Problem based on ID3

Instance	Weather (x_1)	Parents (x_2)	Money (x_3)	Decision (y)
$s^{(1)}$	Sunny	Yes	Rich	Cinema
$s^{(2)}$	Sunny	No	Rich	Tennis
$s^{(3)}$	Windy	Yes	Rich	Cinema
$s^{(4)}$	Rainy	Yes	Poor	Cinema
$s^{(5)}$	Rainy	No	Rich	Stay in
$s^{(6)}$	Rainy	Yes	Poor	Cinema
$s^{(7)}$	Windy	No	Poor	Cinema
$s^{(8)}$	Windy	No	Rich	Shopping
$s^{(9)}$	Windy	Yes	Rich	Cinema
$s^{(10)}$	Sunny	No	Rich	Tennis

C4.5 Decision Tree

Simplified Algorithm

1. Let *S* be the set of training instances.
2. Choose an attribute that best differentiates the instances contained in *S* (C4.5 uses the Gain Ratio to determine).
3. Create a tree node whose value is the chosen attribute:
   a. Create child links from this node where each link represents a unique value for the chosen attribute.
   b. Use the child link values to further subdivide the instances into subclasses.

Gain Ratio and Split Information

$$\text{Split Info}_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

$$\text{Gain Ratio}(A) = \frac{\text{Gain}(A)}{\text{Split Info}_A(D)}$$

Problem based on C4.5

Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(1)}$	Sunny	Hot	High	Weak	No
$s^{(2)}$	Sunny	Hot	High	Strong	No
$s^{(3)}$	Overcast	Hot	High	Weak	Yes
$s^{(4)}$	Rain	Mild	High	Weak	Yes
$s^{(5)}$	Rain	Cool	Normal	Weak	Yes
$s^{(6)}$	Rain	Cool	Normal	Strong	No
$s^{(7)}$	Overcast	Cool	Normal	Strong	Yes
$s^{(8)}$	Sunny	Mild	High	Weak	No
$s^{(9)}$	Sunny	Cool	Normal	Weak	Yes
$s^{(10)}$	Rain	Mild	Normal	Weak	Yes
$s^{(11)}$	Sunny	Mild	Normal	Strong	Yes
$s^{(12)}$	Overcast	Mild	High	Strong	Yes
$s^{(13)}$	Overcast	Hot	Normal	Weak	Yes
$s^{(14)}$	Rain	Mild	High	Strong	No

CART (Decision Tree)

Simplified Algorithm

1. Let *S* be the set of training instances.
2. Choose an attribute that best differentiates the instances contained in *S* (CART uses the Gini Index to determine).
3. Create a tree node whose value is the chosen attribute:
   a. Create child links from this node where each link represents a unique value for the chosen attribute.
   b. Use the child link values to further subdivide the instances into subclasses.

Gini Index

$$Gini\ Index(Attribute = Value) = 1 - \sum_{i=1}^{N} (p_i)^2$$

$$Gini\ Index(Attribute) = \sum_{v=Values} p_v * Gini\ Index(v)$$

Problem based on CART

Instance	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play? (y)
$s^{(1)}$	Sunny	Hot	High	Weak	No
$s^{(2)}$	Sunny	Hot	High	Strong	No
$s^{(3)}$	Overcast	Hot	High	Weak	Yes
$s^{(4)}$	Rain	Mild	High	Weak	Yes
$s^{(5)}$	Rain	Cool	Normal	Weak	Yes
$s^{(6)}$	Rain	Cool	Normal	Strong	No
$s^{(7)}$	Overcast	Cool	Normal	Strong	Yes
$s^{(8)}$	Sunny	Mild	High	Weak	No
$s^{(9)}$	Sunny	Cool	Normal	Weak	Yes
$s^{(10)}$	Rain	Mild	Normal	Weak	Yes
$s^{(11)}$	Sunny	Mild	Normal	Strong	Yes
$s^{(12)}$	Overcast	Mild	High	Strong	Yes
$s^{(13)}$	Overcast	Hot	Normal	Weak	Yes
$s^{(14)}$	Rain	Mild	High	Strong	No

Problem based on CART

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

CART (Regression Tree)

Standard Deviation

$$\sigma_x = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} (x_i - \overline{x})^2}$$

Problems based on CART

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
3	Overcast	Hot	High	Weak	46
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
7	Overcast	Cool	Normal	Strong	43
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
10	Rain	Mild	Normal	Weak	46
11	Sunny	Mild	Normal	Strong	48
12	Overcast	Mild	High	Strong	52
13	Overcast	Hot	Normal	Weak	44
14	Rain	Mild	High	Strong	30

Tutorials	Labs	Exam
All	Complete	74
Some	Partial	23
All	Complete	61
All	Complete	74
Some	Partial	25
All	Complete	61
Some	Complete	54
Some	Partial	42
Some	Complete	55
All	Complete	75
Some	Partial	13
All	Complete	73
Some	Partial	31
Some	Partial	12
Some	Partial	11

https://www.youtube.com/watch?v=nWuUahhK3Oc

Underfitting and Overfitting

Underfitting and Overfitting

- Underfitting occurs when a machine learning algorithm cannot capture the underlying trend of the data
- Overfitting occurs when a machine learning algorithm captures the noise of the data
- Given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

Avoiding Overfitting

- Approaches that stop the growth of tree before it reaches the point where it perfectly classifies the training data.
- Approaches that allow the tree to overfit the data, and then post-prune it.

Approaches to Identify Final Tree Size

- Use separate dataset for training and for evaluation. This is done by the *training and validation set approach*.
- Use the entire dataset for training but use a statistical test (*chi square test*) for estimation.
- Use an explicit measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This is done using the *minimum description length principle*.

- Continuous values attributes: Uses a threshold based Boolean attribute approach
- Missing attribute values: assign the value that is most common among training examples at node n.
- Handling Attributes with Differing Costs: prefer low-cost attributes are preferred over high-cost attributes

Rule-Based Classification

- Using IF–THEN Rules for Classification:
- Define coverage and accuracy of a rule. It is given by

$$\text{Coverage}(R1) = \frac{m_{\text{covers}}}{|D|}$$

$$\text{Accuracy}(R1) = \frac{m_{\text{correct}}}{m_{\text{covers}}}$$

- Properties of rule generated by rule based classifier
  - Mutually exclusive rules
  - Exhaustive rules

Rule Ordering Schemes

- The rule ordering scheme does the prioritization of the rules
- Ordering based on class: the classes are sorted in the order of decreasing “importance”.
- With rule ordering, the triggering rule that appears first in the list has the highest priority and it fires the class prediction

Rule Extraction from a Decision Tree

- To extract rules from a decision tree, one rule is created for each path from the root node to the leaf node.
- Each splitting criterion in each path is logically ANDed to form the rule antecedent. The leaf node holds the class prediction.

Pruning the Rule Set

- For a given rule antecedent, any condition that does not improve the estimated accuracy of the rule can be pruned
- Sequential covering algorithm can be used for the same

Algorithm

1. Start with an empty cover
2. Find the best hypothesis using learn-one-rule
3. If the just-learnt-rule satisfies the threshold then
   (a) Put just-learnt-rule to the cover.
   (b) Remove examples covered by just-learnt-rule.
   (c) Go to step 2.
4. Sort the cover according to its performance.
5. Return: cover.

Comparison

	ID3	C4.5	CART
Splitting Criteria	Information Gain	Gain Ratio	Gini Index
Attribute Type	Handles only categorical values	Handles both categorical and numerical values	Handles both categorical and numerical values
Missing Values	Do not handle missing values	Handles missing values	Handles missing values
Pruning Strategy	No pruning is done	Error based pruning is used	Cost-complexity pruning is used
Outlier Detection	Susceptible to outliers	Susceptible to outliers	Can handle outliers

Module 4

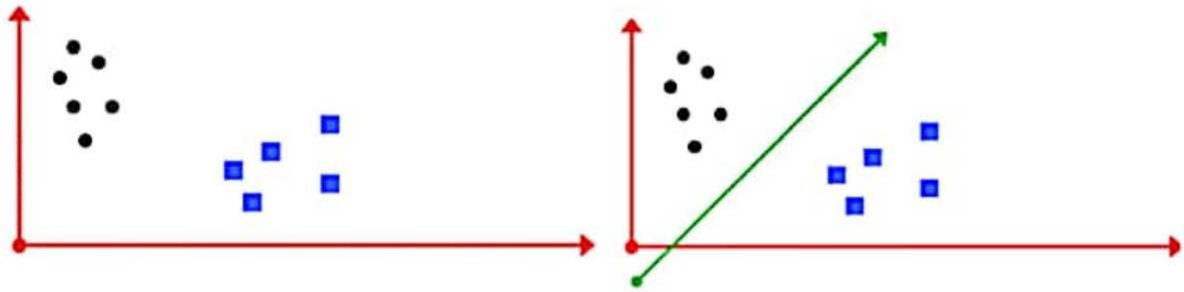
Support Vector Machines

Support Vector Machines:

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labelled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

Let's have a simple approach to it……

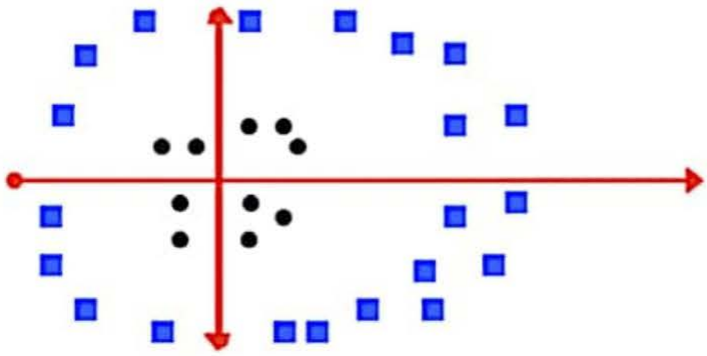
Suppose you are given plot of two label classes on graph as shown in image 1 below. Can you decide a separating line for the classes?



You might have come up with something similar to *image 2*. It fairly separates the two classes. Any point that is left of line falls into black circle class and on right falls into blue square class. ***Separation of classes. That's what SVM does.*** It finds out a line/ hyper-plane (in multidimensional space that separate outs classes).

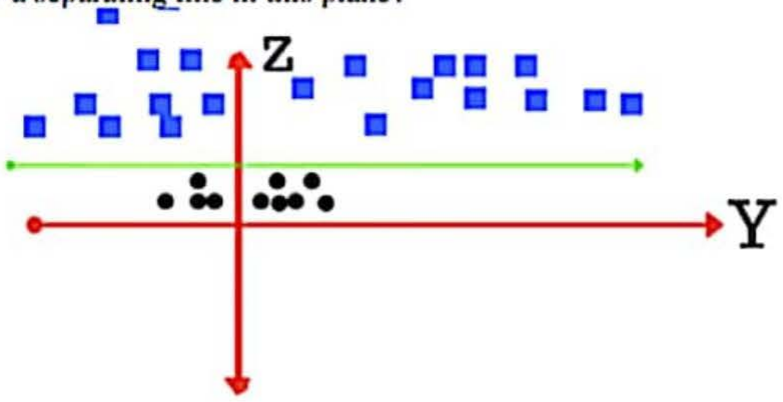
1. Making it a Bit complex…

Now consider what if we had data as shown in image below?



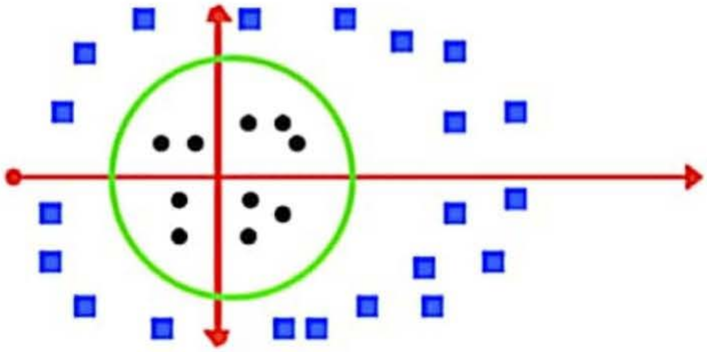
Clearly, there is no line that can separate the two classes in this x-y plane. For this we apply transformation and add one more dimension as we call it z-axis. Let's assume value of points on z plane, $w = x^2 + y^2$. In this case we can manipulate it as distance of point from z-origin. Now if we plot in z-axis, a clear separation is visible and a line can be drawn.

Can you draw a separating line in this plane?



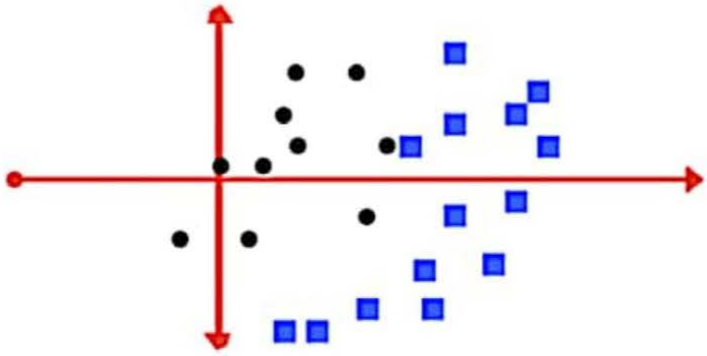
plot of zy axis. A separation can be made here.

When we transform back this line to original plane, it maps to circular boundary as shown in *image below*. These transformations are called ***kernels.***



Transforming back to x-y plane, a line transforms to circle.
Thankfully, you don't have to guess/ derive the transformation every time for your data set. The sklearn library's SVM implementation provides it inbuilt.

2. Making it a little more complex… What if data plot overlaps?



What in this case?

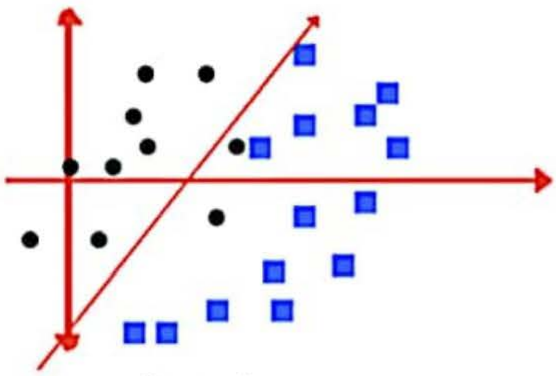


Image 1

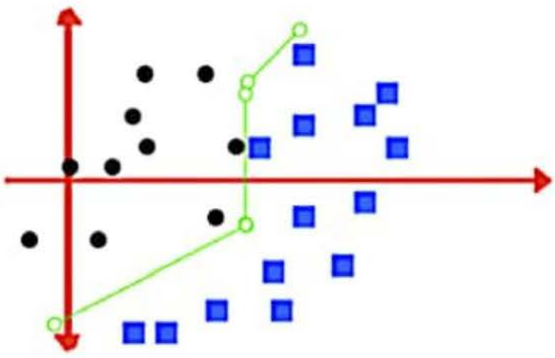


Image 2

Which one do you think? Well, both the answers are correct. The first one tolerates some outlier points. The second one is trying to achieve 0 tolerance with perfect partition.

But, there is trade off. In real world application, finding perfect class for millions of training data set takes lot of time. This is called **regularization parameter.**

Tuning parameters:

Kernel

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role. For **linear kernel** the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B(0) + sum(ai * (x,xi))

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

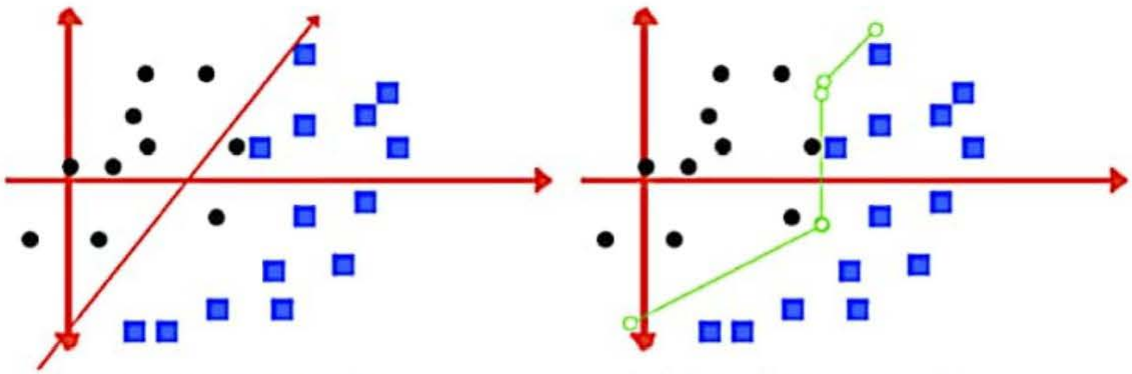
The **polynomial kernel** can be written as $K(x,xi) = 1 + sum(x * xi)^d$ and **exponential** as $K(x,xi) = exp(-gamma * sum((x—xi^2))$.

Polynomial and exponential kernels calculate separation line in higher dimension. This is called **kernel trick.**

Regularization

The Regularization parameter (C) tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

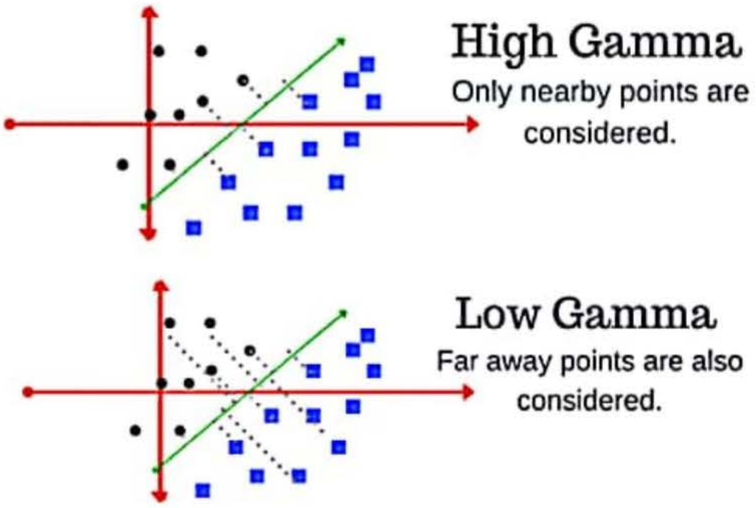
The images below are example of two different regularization parameter. Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.



Low regularization value **High regularization value**

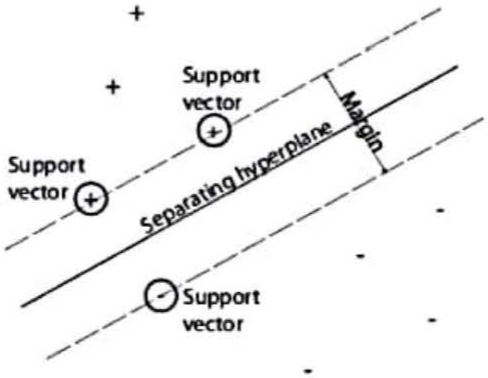
Gamma

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. In other words, with low gamma, points far away from reasonable separation line are considered in calculation for the separation line. Whereas high gamma means the points close to reasonable line are considered in calculation.



Support Vector

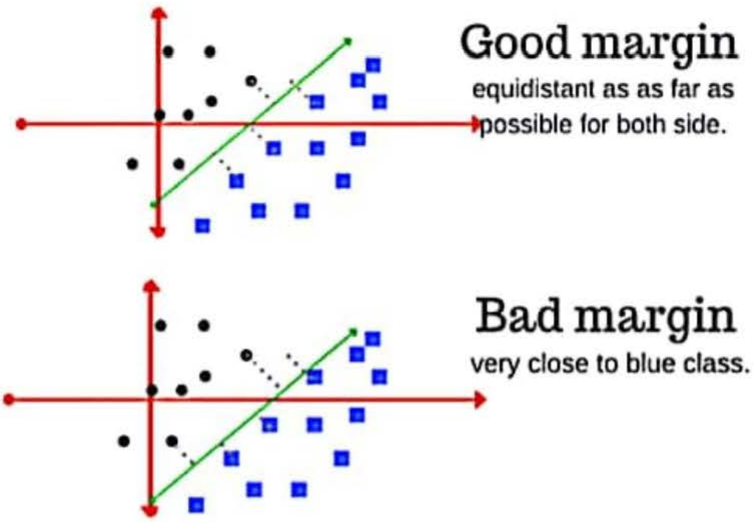
It can be defined as the distance from a class element to the line / hyperplane.



Margin

A margin is a separation of line to the closest class points.

A ***good margin*** is one where this separation is larger for both the classes. Images below gives to visual example of good and bad margin. A good margin allows the points to be in their respective classes without crossing to other class.



The target is to determine the decision boundary (Good Margin). This can be done using Lagrange Multiplier as follows –

The Lagrangian is

$$\mathcal{L} = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \alpha_i \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\right)$$

- Note that $||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$

Setting the gradient of $\mathcal{L}$ w.r.t. **w** and b to zero, we have

$$\mathbf{w} + \sum_{i=1}^{n} \alpha_i(-y_i)\mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

Where **W** is a unit vector with class labels.

- If we substitute $\mathbf{w} = \sum_{i=1} \alpha_i y_i \mathbf{x}_i$ to $\mathcal{L}$, we have

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \left(1 - y_i(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b)\right)$$

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i - b\sum_{i=1}^{n} \alpha_i y_i$$

$$= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i$$

  - Note that $\sum_{i=1}^{n} \alpha_i y_i = 0$

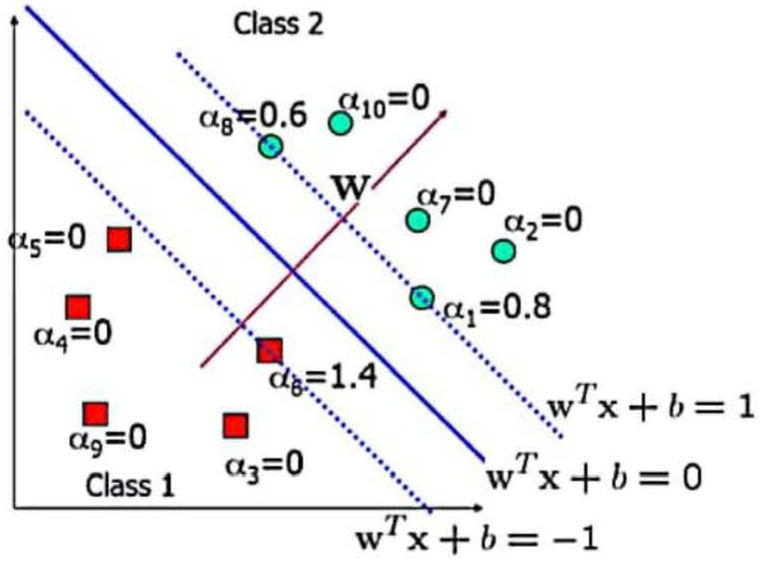
If we know **w**, we know all ***ai***; if we know all ***ai***, we know **w.**

Quadratic Programming Problem

For SVM, sequential minimal optimization (SMO) seems to be the most popular.

A QP with two variables is trivial to solve. Each iteration of SMO picks a pair of **(*ai*,*aj*)** and solve the QP with these two variables; repeat until convergence.

In practice, we can just regard the QP solver as a "blackbox" without bothering how it works.



Module 5
Learning with Classification

Rule Based Classification

In Rule based Classification, the learned model is represented as a set of IF-THEN rules, which is an easy way of representing information or bits of knowledge. The rules are less expensive for computing and interpretable by human. An IF-THEN rule is expressed as **IF condition THEN conclusion.**
Where the "IF" part (or left side) of a rule is known as the rule antecedent or precondition and the "THEN" part (or right side) is the rule consequent. If the condition consists of one or more attribute tests, then all those attributes are logically ANDed.

Example

Consider the following dataset (Table:1).

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

From the above dataset a rule R1 can be derived as follows,
R1: IF age = youth AND student = yes THEN buys computer = yes.
In the rule R1, the rule antecedent consists of attribute tests as *"age = youth" "student = yes"* which are joined together using AND operator.

Characteristics of Rule based Classifier

The rules generated for rule-based classification can be of either of the following types

- Mutually Exclusive Rules: The rules are said to be mutually exclusive rules if all the rules generated for a dataset are independent of each other, that is each record in the dataset are covered by atmost one rule. Mutually exclusive means that we cannot have rule conflicts here because no two rules will be triggered for the same tuple.
- Exhaustive Rules: The rules are said to be exhaustive in nature if it accounts for every possible combination of attribute values. In this method there exist multiple rules that covers the same tuple, and every record is covered by atmost one rule. This does not require a default rule as it covers every set of tuples.

Conflict Resolution Strategy

1. Size Ordering scheme assigns the highest priority to the triggering rule that has the "toughest" requirements, where toughness is measured by the rule antecedent size. That is, the triggering rule with the most attribute tests is fired.
2. Rule ordering scheme prioritizes the rules beforehand. The ordering may be class-based or rule-based.

- ***Class-based ordering***, the classes are sorted in order of decreasing order of prevalence (or importance). All the rules for the most prevalent class come first, the rules for the next prevalent class come next, and so on. The rules are then sorted based on the misclassification cost per class. Within each class, the rules are not ordered because they all predict the same class and no class conflict occurs in there.
- ***Rule-based ordering***, the rules are organized into one long priority list, according to some measure of rule quality, such as accuracy, coverage, or size or based on advice from domain experts. The rule set here is known to be a **decision list.**

Any other rule that satisfies X is ignored.

If there is no rule satisfied by X a fallback or default rule can be set up to specify a **default class**, based on a training set. The class used as default class can be the class in majority or the majority class of the tuples that were not covered by any rule.

Approaches for Creating Rules

Direct Method: This method directly learns the required classification rules from the training dataset. Eg. Sequential Covering Algorithm

Indirect Method: These types of approach first create a decision tree or neural network using the training dataset. The rules for classification are later assumed from the structure of decision trees or from the neural network epochs.

Rule Induction using Sequential Covering Algorithm (Direct Method)

IF-THEN rules can be extracted directly from the training data using a sequential covering algorithm. The rules are learned sequentially one at a time, where each rule for a given class will ideally cover many of the class's tuples. Each time a rule is learned, the tuples covered by the rule are removed, and the process repeats on the remaining tuples. When learning a rule for a class, C, the rule cover all (or many) of the training tuples of class C and none (or few) of the tuples for other classes. So, the learned rules possess high accuracy. AQ, CN2, RIPPER are the common sequential covering algorithms.

The Sequential covering to learn a set of IF-THEN rules for classification is as follows

Input:

D: A data set of class-labeled tuples;

Att vals: The set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Algorithm:

```
(1)  Rule_set = {}; // initial set of rules learned is empty
(2)  for each class c do
(3)      repeat
(4)          Rule = Learn_One_Rule(D, Att_vals, c);
(5)          remove tuples covered by Rule from D;
(6)          Rule_set = Rule_set + Rule; // add new rule to rule set
(7)      until terminating condition;
(8)  endfor
(9)  return Rule_Set;
```

Learn_One_Rule

1. *Consider one class*
2. *Pass through training data*
3. *Find attribute that increase accuracy of current empty set*
4. *Append the attribute into rule*

Rules are grown in a general-to-specific manner. The Learn One Rule procedure finds the "best" rule for the current class, given the current set of training tuples. Initially the algorithm starts with an empty rule and then gradually keep appending attribute tests to it. Appending is done by adding the attribute test as a logical conjunct to the existing condition of the rule antecedent.

Rule Extraction from a Decision Tree (Indirect Method)

Decision tree classifiers are a popular method of classification, easily understandable and having high accuracy. But decision trees can become large and difficult to interpret. A rule-based classifier can be created from decision tree by extracting IF-THEN rules which are easier for human to interpret.

To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically ANDed to form the rule antecedent ("IF" part). The leaf node holds the class prediction, forming the rule consequent ("THEN" part). A disjunction (logical OR) is implied between each of the extracted rules. Because the rules are extracted directly from the tree, they are mutually exclusive and exhaustive. The set of rules generated In this manner from a Decision tree may contain subtree repetition and replication which has to be removed by rule pruning.

Rule Pruning: Any rule that does not contribute to the overall accuracy of the entire rule set can also be pruned. The rules are individually pruned by removing any condition that does not improve the rule accuracy.

One problem of Pruning is that the Rules may not be mutually exclusive and exhaustive after pruning. This can be avoided using Conflict Resolution strategies (e.g. Class-Based Ordering).

Rule Quality Measures

There needs to be some measure that checks for the improvement in the rule efficiency by appending the attribute for a rule-set. Some of the measures to evaluate the rule quality are as follows:

Refer Module 3 for explanation on Entropy, Information Gain.

Coverage & Accuracy:

Coverage of a rule can be defined as the fraction of tuples that satisfy the antecedent of a rule, whereas the **accuracy** of the rule is the fraction of tuples that satisfies both the antecedent and consequent of the rule.

Given a tuple '***X***', from a class labelled data set '***D***', let $\mathbf{n_{covers}}$ be the number of tuples covered by R; $\mathbf{n_{correct}}$ be the number of tuples correctly classified by R; and **|D|** be the number of tuples in D. The coverage and accuracy of R can be defined as follows:

$$coverage(R) = \frac{n_{covers}}{|D|}$$

$$accuracy(R) = \frac{n_{correct}}{n_{covers}}$$

In other words, a rule's coverage is the percentage of tuples that are covered by the rule and rule's accuracy is the percentage of the tuples covered and are correctly classified.

First Order Inductive Learner (FOIL)

FOIL is a type of sequential covering algorithm that learns first-order logic rules. Learning first-order rules is complex because those rules contain variables, where as in normal case rules are propositional in nature which are variable free.

The tuples of the class for which the rules are being learned are called positive tuples, while the remaining tuples are negative.

Pos : be the number of positive tuples covered by R.

Neg : be the number of negative tuples covered by R.

Pos' : be the number of positive tuples covered by R'.

Neg' : be the number of negative tuples covered by R'.

FOIL assesses the information gained by extending condition as

$$FOIL\_Gain = pos' \times \left(\log_2 \frac{pos'}{pos' + neg'} - log_2 \frac{pos}{pos + neg}\right)$$

It favours rules that have high accuracy and cover many positive tuples.

Likelihood Statistics:

Likelihood ratio statistic is a statistical test of significance to determine if the apparent effect of a rule is not attributed to chance, but instead indicates a genuine correlation between attribute values and classes. The test compares the observed distribution among classes of tuples covered by a rule with the expected distribution that would result if the rule made predictions at random.

$$Likelihood\_Ratio = 2\sum_{i=1}^{m} f_i \log\left(\frac{f_i}{e_i}\right)$$

For tuples satisfying the rule, f_i is the observed frequency of each class i among the tuples. e_i is expected frequency of each class i to be if the rule made random predictions.

Rule Pruning

To avoid the overfitting **(Overfitting means the rules may perform well on the training data, but less well on subsequent data)** of rules on data a rule is pruned by removing a conjunct (attribute test). A Rule R is chosen for pruning, if the pruned version of R has greater quality, as assessed on an independent set of tuples which is known as Pruning Set.

Pruning can be done in several ways as in decision trees. In FOIL, the quality of pruning is evaluated as Foil prune for given rules R

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg},$$

Metrics for Evaluating Classifier Performance

Positive and Negative Tuples:

Let ***Positive Tuples*** be the set of tuples of dataset under consideration by any rule. ***Negative Tuples*** are all the remaining tuples other than *Positive tuples.*

There are four basic terms that are used in computing many evaluation measures.

True positives (TP): This refers to the positive tuples that were correctly labelled by the classifier.

True negatives (TN): These are the negative tuples that were correctly labelled by the classifier.

False positives (FP): These are the negative tuples that were incorrectly labelled as positive.

False negatives (FN): These are the positive tuples that were incorrectly labelled as negative.

Based on the above four measures (TP, TN, FP, FN) the classifier's performance can be evaluated using the following equations

accuracy, recognition rate	$\frac{TP+TN}{P+N}$
error rate, misclassification rate	$\frac{FP+FN}{P+N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP+FP}$

Confusion matrix: The confusion matrix is a useful tool for analysing how well the classifier can recognize tuples of different classes. TP and TN tell when the classifier is getting things right, while FP and FN tell us when the classifier is getting things wrong.

		Predicted class		
		yes	*no*	Total
Actual class	*yes*	*TP*	*FN*	*P*
	no	*FP*	*TN*	*N*
	Total	*P′*	*N′*	*P* + *N*

Classifiers can also be compared with respect to the following **additional aspects:**

Speed: This refers to the computational costs involved in generating and using the given classifier.
Robustness: This is the ability of the classifier to make correct predictions, given noisy data or data with missing values.
Scalability: This refers to the ability to construct the classifier efficiently given large amounts of data which keeps on increasing.
Interpretability: This refers to the level of understanding and insight that is provided by the classifier or predictor.

Bayesian Classification

Bayesian classifiers are statistical classifiers that predict class membership probabilities. The probability that a given tuple belongs to a particular class. A simple Bayesian classifier is also known as the naive Bayesian classifier which is comparable in performance with decision tree and selected neural network classifiers. Bayesian classifiers have exhibited high accuracy and speed when applied to large databases. Naive Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class-conditional independence.

Bayes' theorem

Bayesian classification is based on Bayes' theorem. Let X be a data tuple considered as Evidence in Bayesian terms. X is described based on measurements made on a set of n attributes. Let 'H' be some hypothesis such as that the data tuple 'X' belongs to a specified class 'C'. In-order to perform a classification, the conditional probability P(H|X) has to be determined. P(H|X) is the probability that tuple X belongs to class C, given that the attribute description of X is known. It is given as –

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

where,
P(H|X) is known as the posterior probability of H conditioned on X.
P(X|H) is the posterior probability of X conditioned on H.
P(H) and P(X) is the prior probability.

Naive Bayesian Classification

The naive Bayesian classifier or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. Each tuple is represented by an n-dimensional attribute vector, $X = (x_1, x_2, \ldots, x_n)$, depicting 'n' measurements made on the tuple from n attributes, respectively, $A_1, A_2, \ldots, A_n$.
2. Suppose that there are m classes, $C_1, C_2, \ldots, C_m$. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X.

i.e., the naive Bayesian classifier predicts that tuple X belongs to the class C_i if and only if

$P(C_i|X) > P(C_j|X)$ for $1 \le j \le m,\ j \ne i$.

Thus $P(C_i|X)$ has to be maximized. The class C_i for which $P(C_i|X)$ is maximized is called the *maximum posteriori hypothesis.*

Using Bayes theorem, $P(C_i|X)$ can be calculated as

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

3. P(X) is constant for all classes, only P(X|Ci)*P(Ci) needs to be maximized.
If the class prior probabilities are not known, assume same probability for all class that is,
$P(C_1) = P(C_2) = \ldots\ldots = P(C_m)$, and only maximize $P(X|C_i)$.
Otherwise,
Maximize P(X|Ci)*P(Ci).

The class prior probabilities may be estimated by $P(C_i) = |C_i, D|/|D|$, where $|C_i, D|$ is the number of training tuples of class C_i in D.

4. If the dataset have too many attributes then computation is expensive to compute P(X|Ci).

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

$$= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i)$$ Thus

The probabilities $P(x_1|C_i)$, $P(x_2|C_i)$, . . . , $P(x_n|C_i)$ from the training tuples. x_k refers to the value of attribute A_k for tuple X.

The attributes can be either Categorial or Continuous.

A. If A_k is categorical, then $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k, divided by $|C_i,D|$, the number of tuples of class C_i in D.

B. If A_k is continuous-valued, A continuous-valued attribute have a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$ So that,

5. To predict the class label of X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of tuple X is the class Ci if and only if $P(X|C_i)P(C_i) > P(X|C_j)P(C_j)$ for $1 \le j \le m, j = i$.

i.e. predicted class label is the class C_i for which $P(X|C_i)P(C_i)$ is the maximum.

Bayesian Belief Networks

Bayesian belief networks are classifiers that makes use of probabilistic graphical models. Naive Bayesian classifier includes Class-conditional independence that makes computations easy. But the attributes are not always independent from each other in real world classification problems. There exist some relationships between different attributes of a dataset. Bayesian belief networks specify joint conditional probability distributions that allows to represent class conditional dependencies between attributes. Bayesian Belief

Network provide a graphical model of relationships, which can be trained and can be used for classification. Bayesian belief networks are also known as belief networks, Bayesian networks, and probabilistic networks.

A belief network is defined by two components

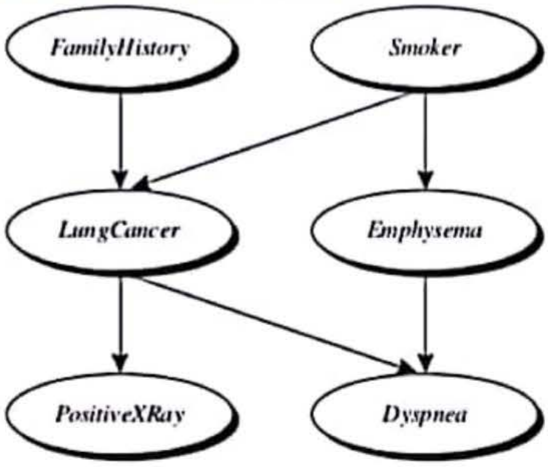
- Directed Acyclic Graph
- Set of Conditional Probability Tables (CPT)

Directed Acyclic Graph

Each node in the directed acyclic graph represents a random variable. The variables may be discrete- or continuous-valued. The variables may be either actual attributes which is given in the dataset or "hidden variables". A hidden variable can be an undefined entity which is not specified in the dataset, but have a influence in decision making or it can be a collective effect of actual variables which result in a new attribute.

If an arc is drawn from a node Y to a node Z, then Y is a parent or immediate predecessor of Z, and Z is a descendant of Y. Each variable is conditionally independent of its non-descendants in the graph, given its parents

Example: A directed acyclic graph showing the Six attributes of a person resulting in Lung Cancer is generated from the dataset. For example, having lung cancer is influenced by a person's family history of lung cancer, as well as whether or not the person is a smoker. Note that the variable PositiveXRay is independent of whether the patient has a family history of lung cancer or is a smoker, given that we know the patient has lung cancer. In other words, once we know the outcome of the variable LungCancer, then the variables FamilyHistory and Smoker do not provide any additional information regarding PositiveXRay. The arcs also show that the variable LungCancer is conditionally independent of Emphysema, given its parents, FamilyHistory and Smoker.



Conditional probability table (CPT)

A belief network has one conditional probability table (CPT) for each variable. The CPT for a variable Y specifies the conditional distribution P(Y |Parents(Y)), where Parents(Y) are the parents of Y .

	FH, S	*FH, ~S*	*~FH, S*	*~FH, ~S*
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

The above table represents the CPT for the variable LungCancer. The conditional probability for each known value of LungCancer is given for each possible combination of the values of its parents. From the above CPT it is clear that the upper leftmost and lower rightmost entries represent

$$P(LungCancer = yes \mid FamilyHistory = yes,\ Smoker = yes) = 0.8$$

$$P(LungCancer = no \mid FamilyHistory = no,\ Smoker = no) = 0.9$$

Let $X = (x_1, \ldots, x_n)$ be a data tuple described by the variables or attributes $Y_1, \ldots, Y_n$, respectively. Each variable is conditionally independent of its non-descendants in the network graph, given its parents. The joint probability distribution is represented as

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | Parents(Y_i))$$

where $P(x_1, \ldots, x_n)$ is the probability of a particular combination of values of X, and the values for $P(x_i | Parents(Y_i))$ correspond to the entries in the CPT for Y_i.

A node within the network can be selected as an "output" node, representing a class label attribute. There may be more than one output node. The classification process can return a probability distribution that gives the probability of each class.

Classification by Backpropagation

Backpropagation is a neural network learning algorithm. A neural network is a set of connected input/output units in which each connection has a weight associated with it. During the learning phase, the neural network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples.

Disadvantage of Neural Network

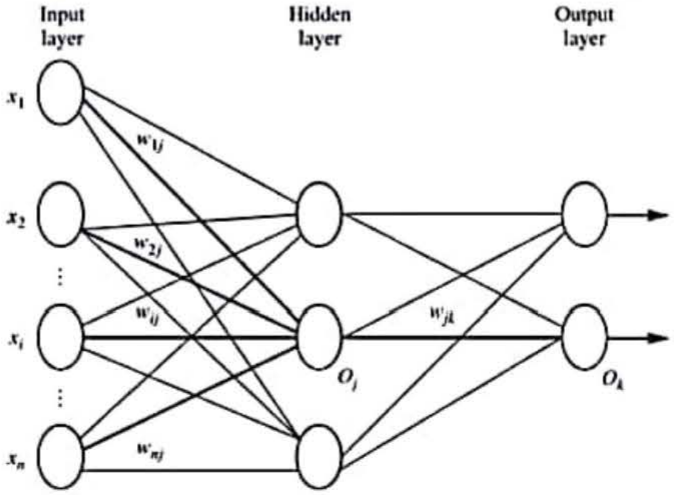
- Long training time required in order to make the neural network efficient for performing a classification or Prediction.
- As compared to Decision tree, Neural networks are less interpretable to human.

Advantages of Neural Network

- Highly tolerant to noisy data.
- Ability to classify patterns that are not trained.
- Well suited for continuous valued output, unlike Decision trees.
- Decision trees are sequential in nature where, as parallelization technique of neural network speeds up the computation.

Multilayer Feed-Forward Neural Network

A multilayer feed-forward neural network consists of an input layer, one or more hidden layers, and an output layer. It iteratively learns a set of weights for prediction of the class label of tuples.



The attributes measured for each tuple are fed into the input layer simultaneously. These inputs along with the weights associated with each link will be the input for the next layer called hidden layer. The output of the first hidden layer may be input to other hidden layer or to the output layer. The number of hidden layers is determined by the complexity of the problem to be solved. Usually a single Hidden layer is used. The output produced by the output layer corresponds to the network prediction for the given tuples.

Input Units: Units in input layer that takes attributes from outside into the network.
Neurodes: The hidden units present between input unit and output unit.

Two-Layer Network: Have three layers (Input layer, Hidden layer and output layer). Input layer is not counted as it does not perform any computation.
Feed forward Network: A network which have connection only in forward direction. There won't be any loops or backward connections.
Fully connected network: Each unit provides input to each unit in the next forward layer.

Back Propagation Algorithm

Backpropagation learns by iteratively processing a data set of training tuples, comparing the network's prediction for each tuple with the actual known target value. The target value may be the known class label of the training tuple (for classification problems) or a continuous value (for numeric prediction). For each training tuple, the weights are modified so as to minimize the mean-squared error between the network's prediction and the actual target value. These modifications are made in the "backwards" direction (i.e., from the output layer) through each hidden layer down to the first hidden layer (hence the name backpropagation). The weights will eventually converge, and the learning process stops.

Inputs

D: A data set consisting of the training tuples and their associated target values;
l: the learning rate;
Network: A multilayer feed-forward network.

Algorithm

(1) Initialize all weights and biases in *network*;
(2) **while** terminating condition is not satisfied {
(3) **for** each training tuple $\boldsymbol{X}$ in D {
(4) // Propagate the inputs forward:
(5) **for** each input layer unit j {
(6) $O_j = I_j$; // output of an input unit is its actual input value
(7) **for** each hidden or output layer unit j {
(8) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
(9) $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
(10) // Backpropagate the errors:
(11) **for** each unit j in the output layer
(12) $Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error
(13) **for** each unit j in the hidden layers, from the last to the first hidden layer
(14) $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
(15) **for** each weight w_{ij} in *network* {
(16) $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
(17) $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
(18) **for** each bias θ_j in *network* {
(19) $\Delta \theta_j = (l) Err_j$; // bias increment
(20) $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
(21) } }

The steps can be described as follows:
Initialize the weights: The weights in the network are initialized to small random numbers (e.g., ranging from −1.0 to 1.0, or −0.5 to 0.5). Each unit has a bias associated with it. The biases are initialized to small random numbers.
Each training tuple, X, is processed by the following steps.

Propagate the inputs forward: The training tuple is fed to the network's input layer. The inputs pass through the input units, unchanged. For an input unit, j, its output, Oj , is equal to its input value, Ij . The net input and output of each unit in the hidden and output layers are computed. The net input to a unit in the hidden or output layers is computed as a linear combination of its inputs. Given a unit, j in a hidden or output layer, the net input, Ij , to unit j is

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

where wij is the weight of the connection from unit i in the previous layer to unit j; Oi is the output of unit i from the previous layer; and j is the bias of the unit. The bias acts as a threshold (θ) in that it serves to vary the activity of the unit. Each unit in the hidden and output layers takes its net input and then applies an activation function. The logistic, or sigmoid, function is used as activation function. Given the net input Ij to unit j, then Oj , the output of unit j, is computed as

$$O_j = \frac{1}{1 + e^{-I_j}}$$

This function is also referred to as a squashing function, because it maps a large input domain onto the smaller range of 0 to 1. The logistic function is nonlinear and differentiable, allowing the backpropagation algorithm to model classification problems that are linearly inseparable. output values, Oj , for each hidden layer, up to and including the output layer, which gives the network's prediction.

Backpropagate the error: The error is propagated backward by updating the weights and biases to reflect the error of the network's prediction. For a unit j in the output layer, the error Errj is computed by Errj = Oj (1 − Oj)(Tj − Oj)
where Oj is the actual output of unit j, and Tj is the known target value of the given training tuple. Oj (1 − Oj) is the derivative of the logistic function. To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit j in the next layer are considered. The error of a hidden layer unit j is

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$$

where wjk is the weight of the connection from unit j to a unit k in the next higher layer and Errk is the error of unit k. The weights and biases are updated to reflect the propagated errors.
Weights are updated as

$$\Delta w_{ij} = (l) Err_j O_i$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

The variable l is the learning rate, a constant typically having a value between 0.0 and 1.0. Backpropagation learns using a gradient descent method to search for a set of weights that fits the training data so as to minimize the mean-squared distance between the network's class prediction and the known target value of the tuples. The learning rate helps avoid getting stuck at a local minimum in decision space (i.e., where the weights appear to converge, but are not the optimum solution) and encourages finding the global minimum. If the learning rate is too small, then learning will occur at a very slow pace. If the learning rate is too large, then oscillation between inadequate solutions may occur. Biases are updated

$$\Delta \theta_j = (l) Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

Updating the weights and biases after the presentation of each tuple is referred to as case updating. The weight and bias increments could be accumulated in variables and updated after all the tuples in the training set have been presented. This strategy is called epoch updating, where one iteration through the training set is an epoch.

Terminating condition: Training stops when

- All wij in the previous epoch are so small as to be below some specified threshold.
- The percentage of tuples misclassified in the previous epoch is below some threshold.
- A prespecified number of epochs has expired.

The computational efficiency depends on the time spent in training the network. Given |D| tuples and w weights, each epoch requires O(|D| × w) time.

Hidden Markov Model

A hidden Markov model (HMM) is a statistical model based on the Markov process with unobserved (*hidden*) states. A Markov process is referred as a memoryless process which satisfies Markov property. The property states that if one can make predictions for the future of the process based solely on its present state. Markov property states that the conditional probability distribution of future states of a process depends only upon the present state, not on the sequence of events that preceded it.
The Hidden Markov Model (HMM) is a variant of a *finite state machine* having a set of hidden *states* (***W***), an output *alphabet* (observations) or visible states (**V**), transition probabilities (***A***), output (emission) probabilities (***B***), and initial state probabilities (**Π**). The current state is not observable. Instead, each state produces an output with a certain probability (***B***). Usually the states, ***W***, and outputs, ***V***, are understood, so an HMM is said to be a triple, (***A***, ***B***, ***Π***).

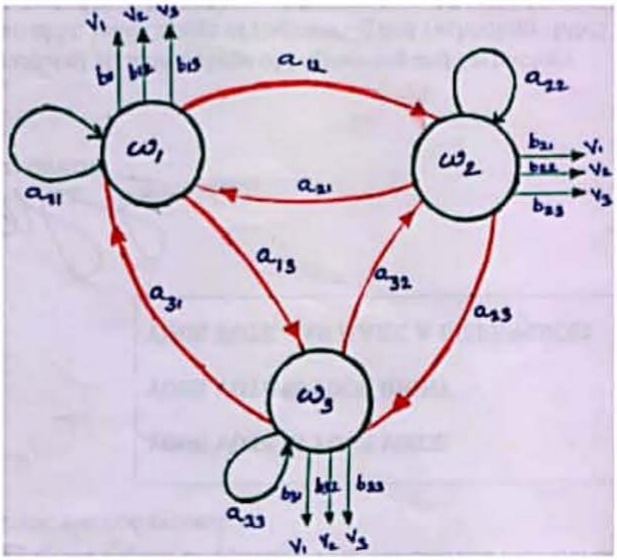
Examples of Markov Process

- Measurement of weather pattern.
- Daily Stock market price.

Explanation of Hidden Markov Model

- HMM consist of two types of states, the Hidden State (w) and Visible State (v).
- **Transition probability:** is the probability of transitioning from one state to another in a single step.
- **Emission Probability:** The conditional distributions of the observed variables $P(V_n \mid Z_n)$ from a specific state.

Consider a hidden Markov model with three visible states and three hidden states
W= w_1 w_2, w_3
$V = V_1$ V_2, V_3



In each hidden state W, the system will emits a visible output V=V_1 V_2, V_3. The probability of emitting any visible output by the hidden states are represented by emission probability which is represented as b_{ij}. i.e

State w_1 will emits the output $V_1 V_2 V_3$ with probability b_{11}, b_{12}, b_{13} respectively.
State w_2, *will emits the output with probability* b_{21}, b_{22}, b_{23} *respectively.*
State w_3, will also emit the output V_1 V_2, V_3 *with probability* b_{31}, b_{32}, b_{33} *respectively.*

From any hidden state there is always a transition to any other state. The probability of such transition between states is represented as transition probability A_{ij}.
The transition probability from state w_1 *to* w_2, *is represented as* a_{12} *and from* w_2, *to* w_1 *is* a_{21}.
Similarly between w_2 *and* , w_3.

It is also possible to have a transition from a state to itself which is represented as a_{ii}. The sum of probability to have transition from one state to other will be one. i.e.

$$\sum_j Aij = 1$$

for all j

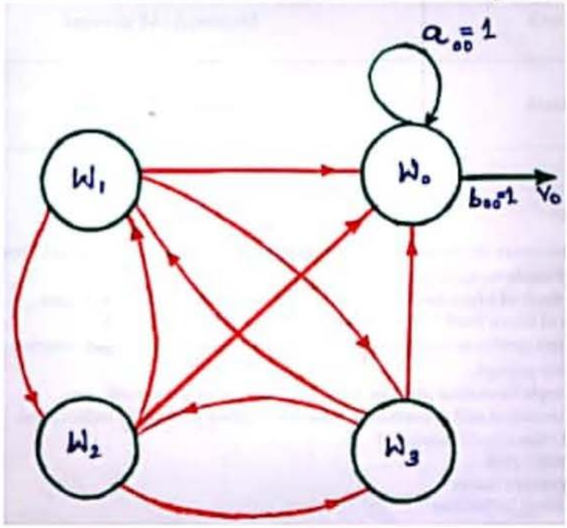
similarly, the sum of probability to emit any of the visible output by a state is also one. i.e.

$$\sum_k Bjk = 1$$

for all k.

After several transitions between hidden states, the system will reach a final state represented by w_0. Once the system reaches final state, then there is transition to only itself. This transition is represented as a_{00} with probability 1. It emits only one type of visible symbol, V_0. Also, the probability of emitting the visible symbol V_0 is given by b_{00} where $b_{00}=1$.

The above state diagram can be drawn with the final state and the output as shown below.



The hidden markov model have to address three central issues.

- **Evaluation Problem:** The evaluation problem states that for a given Model θ, with

W: Hidden states V: Visible symbols A_{ij}: Transition Probability B_{jk}: Emission Probability
V^T: Sequence of visible symbols emitted
what is the probability that the visible symbol sequence V^T will be emitted by the model θ.

i.e. $P(V^T|\theta) = ?$

- **Decoding Problem:** In decoding problem, W^T, the sequence of states generated by the visible symbol sequence V^T has to be calculated.

i.e. $W^T=?$

- **Training Problem:** For a known set of hidden states, W and visible symbols V, the training problem is to find the transition probability A_{ij}, and emission probability B_{jk} from the training set.

i.e. $A_{ij} = ?$
$B_{jk} = ?$

Evaluation Problem

For a given HMM θ, and sequence of visible symbols V^T, the conditional probability of getting the visible symbol sequence have to be estimated.

$$P(V^T|\theta) = ?$$

So $P(V^T|\theta)$ can be written as

$$P(V^T|\theta) = P(V^T|W_r^T) \,.\, P(W_r^T) \quad \text{----------(1)}$$

where W_r^T is one of those possible sequence of W^T and $P(W_r^T)$ is the probability of getting W_r^T selected.

Since there are different number of possible sequence of W, the $P(V^T|\theta)$ can be represented as

$$\sum_{r=1}^{rmax} P(V^T|(W_r^T).P(W_r^T)$$

where r_{max} is the number of possible sequence of W.

$W_{rT} = \{W_{(1)}, W_{(2)}, \ldots\ldots W_{(T)}\}$

If N is the number of Hidden states then $r_{max} = N_T$

In equation (1) the term $P(W_r^T)$ can be written as

$P(W_r^T) = P(W_T|\ W_{t-1})$ as the system is a markov proecess. In order to compute the probability of a state for entire time sequence, the individual probabilities have to be multiplied.

$$\prod_{t=1}^{T}[P(W^T)|Wt-1] \quad \text{----------(2)}$$

i.e. product of transition probabilities at consequent time sequence in consecutive time. The $P(V^T|W_r^T)$ term of the equation (1) can be rewritten as

$$P(V^T|Wr^T) = \prod_{t=1}^{T}[P(V^T)|W(t)] \quad \text{----------(3)}$$

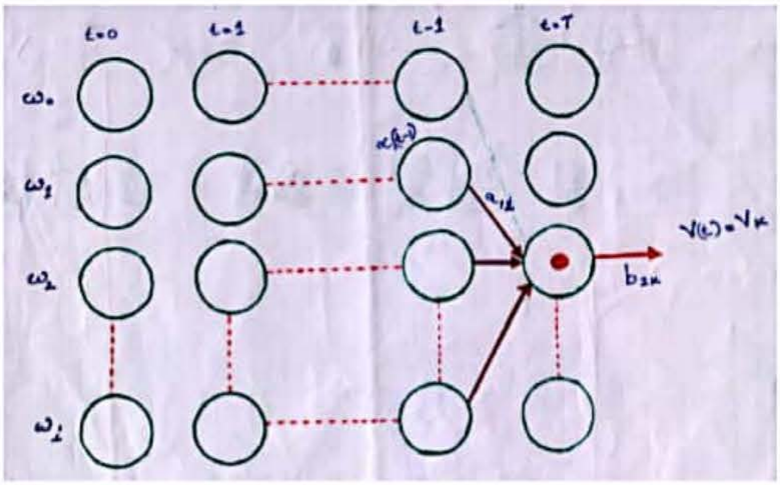
i.e. the product of all possible values of V

Substituting (2) and (3) back in equation (1) results in the final equation of $P(V^T|\theta)$ as

$$P(V^T)|\Theta) = \sum_{r=1}^{rmax}\prod_{t=1}^{T} P(V(t)|W(t)).P(W(t)|w(t-1)) \quad \text{----------(4)}$$

The above equation (4) have high computational complexity which is of the order $O(N^TN)$

Consider the following Trellis diagram that shows number of states at different instances of times.



At time t=0, the state of the system is w_1.
If the visible symbol emitted at time t=1 i.e. v(1)=v0, then there is a transition from w1 to w0.
From the above diagram, the probability that a machine is at state W_i at time T=t can be represented as $\alpha_i(t)$.
So $\alpha_1(t-1)$ is the probability that the machine is at state W1 at time t-1. Similarly, for $\alpha_2(t-1)$, $\alpha_1(t-1)$ etc.
The probability of transition from one state to other can be rewritten as a product of probability of being in a state and probability of transition to other state.
i.e. probability of transition from w_i to w_j can be represented as $\alpha_i(t-1)a_{ij}$
Example:
a_{12} = Probability of transition from $w_1(t)$ to $w_2(t)$ at time instant t $=\alpha_1(t-1)a_{12}$.

a_{22} = Probability of transition from $w_2(t)$ to $w_2(t)$ at time instant t $=\alpha_2(t-1)a_{22}$.

Therefore, the probability of a visible state is the product of sum of all the probabilities and their emission probabilities.

$$\alpha_j[\sum\alpha_i(t-1)a_{ij}]b_{jk}v(t).\text{----------(5)}$$

when sum is calculated for the product of probabilities of transition from a state w_{t-1} to wt it gives the probability that there will be a transition from one of the state to another in a time instant t.

Probability that machine make a transition from a state at time t-1 to another state at time t is given by $\alpha_i(t-1)a_{ij}$.

The probability that the machine is in state w_2 at time t after emitting t number of visible symbols from sequence of visible symbols V^T is given by sum of product terms multiplied by emission probability. In Simplified Terms it can be written as

=	0	if t=0 & j≠ initial state
$\alpha_j(t)$=	1	if t=0 & j= initial state
	$\alpha_j[\sum\alpha_i(t-1)a_{ij}]b_{jk}v(t)$	Otherwise

Using this an algorithm can be generated to find the probability of HMM θ has generated a sequence V^T.

Forward Algorithm

Initialize: t=0, a_{ij}, b_{jk}, V^T, $\alpha_j(0)$
For t=t+1

$$\alpha j(t) = bjkV(t).\sum_{i=1}^{N} \alpha i(t-1)aij$$

until t=T
Return $P(V^T|\theta) = \alpha_o(T)$ for final state
end

Decoding Problem

The decoding problem deals with finding the most probable sequence of set through which the machine passed to reach the final state.

Algorithm for Decoding Problem

Let V^T be the visible symbol sequence.
Aim: Find the probable sequence of hidden states.

Initialize: Path = {}; t=0; j=0;
For t=t+1
For j=j+1

$$\alpha j(t) = bjkV(t).\sum_{i=1}^{N} \alpha i(t-1)aij$$

Until j=N
j' = arg max □ (t□
Append j' to path
Repeat until t=T
Return Path.

Training Problem

The hidden markov model can be trained using supervised learning method. It uses a number of known visible symbols for training. The main motive of training a HMM is to estimate the transition probability and the emission probability a_{ij} and b_{jk} which are the unknown terms in equation (5).

Example

Consider a visible symbol sequence $<v_1,v_3,v_1,v_5,v_7,v_2,v_0>$
Then $\alpha_3(4)$=the probability that the machine is in states W_3 after generating the first four visible symbols of the sequence, v_1,v_3,v_1,v_5 .
$\beta_3(4)$=The probability that the machine remains in the state W_3 and it will generate the remaining four visible symbols v_5,v_7,v_2,v_0.

In the above example a new term is introduced $\beta_i(t)$ **represent the probability that model will be in $w_i(t)$ and will generate remaining of the given target sequence V^T. i.e. all the symbols from V(t+1) to V(t).**

In Simplified Terms it can be written as

$\beta_j(t)=$

$$\begin{cases} 0 & \text{if } t=T \ \& \ wi(t) \neq w0 \\ 1 & \text{if } t=T \ \& \ wi(t) = w0 \\ bjkV(t+1).\sum_j \beta ij(t+1)aij & \text{Otherwise} \end{cases}$$

Using this an algorithm can be generated to find the probability of HMM θ that will generate the remaining symbols of V^T by continuing in the present state. This algorithm is known as backward algorithm.

Backward Algorithm

Initialize: t=T, a_{ij}, b_{jk}, V^T, $\beta_i(t)$

For t=t-1

$\beta_j(t)=$

$$bjkV(t+1).\sum_j \beta ij(t+1)aij$$

until t=T

Return $P(V^T|\theta) = \beta_j(0)$ for the known initial state.

end

Both $\alpha_j(t)$ and $\beta_i(t)$, the forward probability and backward probability will be used to calculate the a_{ij} and b_{ij}.

Using the above terms, the probability of transition from $w_i(t-1)$ to $w_j(t)$ which emits visible symbols V^T can be written as

$$\gamma_{ij}(t) = \frac{\alpha_j(t-1)a_{ij} \cdot b_{jk} \cdot \beta_i(t)}{P(V^T|\theta)}$$

Still, in the above equation of $\gamma_{ij}(t)$, there are terms of a_{ij} **and** b_{jk} which are unknown. So, in order to solve the above equation, assume random values for a_{ij} and b_{jk} initially.
By using random values of a_{ij} and b_{jk} the $\gamma_{ij}(t)$ can be estimated, and later improve the values of a_{ij} and b_{jk}.

The expected number of transitions from Wi(t-1) to Wj(t) at any time in the sequence V^T is

$$\sum_{t=1}^{T} \gamma ij(T)$$

The total expected number of transition from Wi to any other state is

$$\sum_{t=1}^{T} \sum_{k} \gamma ik(T)$$

The estimated value of transition probability a_{ij} can be given as

$$\hat{a}ij = \frac{\sum_{t=1}^{T} \gamma ik(T)}{\sum_{t=1}^{T} \sum_{k} \gamma ik(T)}$$

Similarly, the estimated value of transition probability a_{ij} can be given as

$$\widehat{bij} = \frac{\sum_{t=1}^{T} \sum_i \gamma jl(T)(V^T = Vk)}{\sum_{t=1}^{T} \sum_l \gamma jl(T)}$$

where the numerator is the transition which emitted symbol V_k and the denominator is the transition of all symbols emitted.

Using the above sequence, the value for $P(V^T|\theta)$ can be estimated for a particular model θ. Similarly, for other models θ_i also. If the conditional probability of one model θ_i is greater than other θ_j then the model points the corresponding class of θ_i.

$$P(V^T|\theta_i) > P(V^T|\theta_j) \text{ then } V^T \varepsilon \ \theta_i$$

Example

Consider a model with 4 hidden states and 4 visible states

$W = W_1, W_2, W_3, W_4$

$V = V_1, V_2, V_3, V_4$ and final symbol V_0

Let the Transition probability and Emission probability as follows.

$$A_{ij} = \begin{array}{c} \begin{matrix} W_1 & W_2 & W_3 & W_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ .2 & .3 & .1 & .4 \\ .2 & .5 & .2 & .1 \\ .7 & .1 & .1 & .1 \end{bmatrix} \end{array} \begin{matrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{matrix}$$

$$B_{ij} = \begin{array}{c} \begin{matrix} V_0 & V_1 & V_2 & V_3 & V_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & .3 & .4 & .1 & .2 \\ 0 & .1 & .1 & .7 & .1 \\ 0 & .5 & .2 & .1 & .2 \end{bmatrix} \end{array} \begin{matrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{matrix}$$

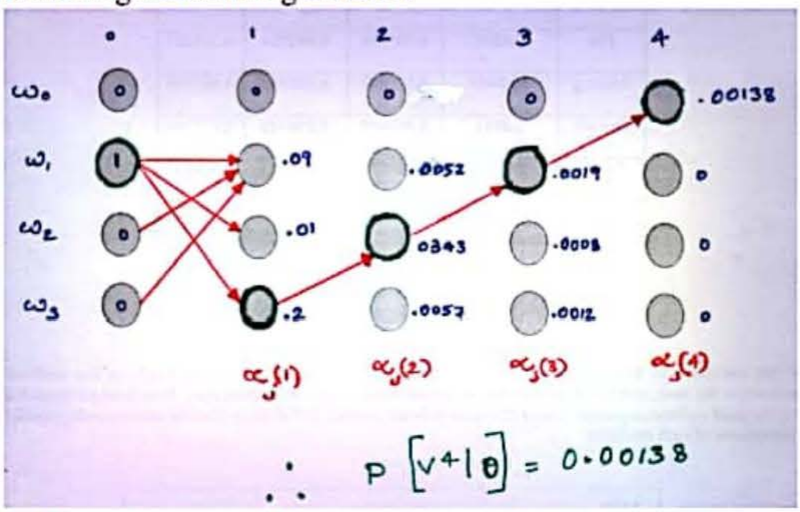
where

$$\sum_j Aij = 1 \quad \sum_k Bjk = 1$$

Let $V^4 = \{v_1, v_3, v_2, v_0\}$ and the machine is at W_1 at t=0. Find $P(V^T|\theta)$

Solution:

From above data, the trellis diagram can be generated.



Given Initial state is W_1. So, probability of W_1 is 1 at t=0. And remaining states have probability zero. From initial state transition is possible to W_1, W_2 and W_3 at t=1 Considering the emission probability for the first symbol given in sequence v_1, at state W_1, using the equation of $\mathbf{a_j(t)}$, the probability will be obtained as 0.09. i.e. .3x.3. Similarly, for the other states W_2 and W_3.

So, at time instant t=1, the state with highest probability will be firing for the subsequent states i.e. state W_3 have 0.2 probability which is the highest. Same process is continued till all visible symbols are produced as per the sequence given. At t=4 the system reaches final state W_0 and emits v_0 with a probability 0.00138. which is $P(V^T|\theta)$ for given HMM.

Why naive bayes is called naive?

(Naive bayes ek classifier hai jo classify karta hai cheezo ko naive ka matlab hota hai a Person/action showing a lack of experience toh isko is classifier ko naive kyu kehte hai voh iss answer me bataya hai)

- Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.
- The Naive Bayes Classifier assumes that all the features of a class are independent of each other.
- This means that anyone of the feature is missing then classification may not be perfect.
- For example, a fruit may be considered to be an apple if it is red, round, and about 4" in diameter, or a ball if we don't give the red colour classification.
- So even if the features are inter-dependent on each other, the Naive Bayes Classifier will consider them independently.

It's called naive also because it makes the assumption that all attributes are independent of each other. This assumption is why it's called naive as in lots of real world situations this does not fit. Despite this the classifier works extremely well in lots of real world situations and has comparable performance to neural networks and SVM's in certain cases (though not all).

Naive Bayes Classifier

S.No.	Age	Income	Student	Credit	Buy
1	<30	High	No	Fair	No
2	<30	High	No	Excellent	No
3	31-40	High	No	Fair	Yes
4	>40	Medium	No	Fair	Yes
5	>40	Low	Yes	Fair	Yes
6	>40	Low	Yes	Excellent	No
7	31-40	Low	Yes	Excellent	Yes
8	<30	Medium	No	Fair	No
9	<30	Low	Yes	Fair	Yes
10	>40	Medium	Yes	Fair	Yes
11	<30	Medium	Yes	Excellent	Yes
12	31-40	Medium	No	Excellent	Yes
13	31-40	High	Yes	Fair	Yes
14	>40	Medium	No	Excellent	No

X = (age = <30, income = medium, student = yes, credit = fair)

$P(buy = Yes) = 9/14 = 0.643$

$P(buy = No) = 5/14 = 0.357$

- Age.

$P(age = <30 \mid buy = Yes) = 2/9 = 0.222$

$P(age = <30 \mid buy = No) = 3/5 = 0.600$

- Income

$P(Income = med \mid buy = Yes) = 4/9 = 0.44$

$P(income = med \mid buy = No) = 2/5 = 0.40$

- Student

$P(stud = Yes \mid buy = Yes) = 6/9 = 0.667$

$P(stud = Yes \mid Buy = No) = 1/5 = 0.20$

- Credit

$P(credit = fair \mid buy = Yes) = 6/9 = 0.667.$

$P(credit = fair \mid buy = No) = 2/5 =$

$P(X \mid Yes) \cdot P(Yes) = P(age \mid Yes) \cdot P(med \mid Yes) \cdot P(Yes \mid buy) \cdot P(fair \mid Yes) \cdot P(Yes)$

$= 0.222 \times 0.44 \times 0.667 \times 0.667$

$= 0.0435$

$P(X \mid No) \cdot P(No) = P(age \mid No) \cdot P(med \mid No) \; P(Yes \mid No) \; P(fair \mid No) \cdot P(No)$

$= 0.60 \times 0.40 \times 0.20 \times 0.40$

$= 0.0192$

$P(X \mid Yes)$ is greater than $P(X \mid No)$,

$\therefore$ X will buy the product.

Naive Baye's

Person ID	Name	Gender	Height	class
1	A	Female	1.6m	Short
2	B	Male	2.0m	tall
3	C	Female	1.9m	Medium
4	D	Female	1.85m	medium
5	E	Male	2.8m	tall
6	F	Male	1.7m	Short
7	G	Male	1.8m	Medium
8	H	Female	1.6m	Short
9	I	Female	1.65m	Short

Sabse pelhe jo class attribute hai uski probability nikallo.

∴ class me 3 value hai
- Short
- Medium
- tall

teeno ki Probability nikalni

formula of Probability is

$$P(\text{Short}) = \frac{\text{no. of time short came}}{\text{total no. of item}}$$

$$= \frac{4}{9}$$

Its for your understand

(sab formula nikhlu ki jaroorat nahi hai)

similarly for medium and tall we will find

$$P(\text{medium}) = \frac{3}{9}$$

$$P(\text{tall}) = \frac{2}{9}$$

* Ab humko height ko ranges me divide karna app apne hisaab se isko divide kar sakte ho

Height → 6 ranges

[0 - 1.6]
[1.61 - 1.7]
[1.71 - 1.8]
[1.81 - 1.9]
[1.91 - 2.0]
[2.0 - ∞]

Gender: Male
Female

yeh sab karne ke baad hume banana hai probability table

attribute	value			Probability		
Gender	short	Medium	tall	short	Medium	tall
male	1	1	2	1/4	1/3	2/2
female	3	2	0	3/4	2/3	0
Height						
[0-1.6]	2	0	0	2/4	0	0
[1.61-1.7]	2	0	0	2/4	0	0
[1.71-1.8]	0	1	0	0	1/3	0
[1.81-1.9]	0	2	0	0	2/3	0
[1.91-2.0]	0	0	1	0	0	1/2
[2.0-∞]	0	0	1	0	0	1/2

Ab table apna taiyaar

/* Exams me most of time yeh jo humne table banaya hai voh diya rehta hai */

* (Qushon puchta hai hume ek new tuple diya rahenga aur puchenge classify given touple is short, medium, tall) *

example

m = male (isko medium consider mat Karna)

t = { Hola, M, 2.2m }

Step 1: touple ki har class attribute ke saath Probability nikalo

P(t/short) = P[m/short) * P((2.0 — ∞) /short)

$= \frac{1}{4} \times 0$

$= 0$

Step 2 :- Home likelihood Nikalna

Yahi jo meri new value hai voh short ke kitna karib, medium ke kitna karib hai, tall ke kitna karib hai

Likelyhood for short

$$= P(t/short) \times P(short)$$

$$= 0 \times \frac{4}{9}$$

$$= 0$$

Likelihood for medium

$$= P(t/medium) \times P(medium)$$

$$= 0 \times \frac{3}{9}$$

$$= 0$$

Likelihood for tall

$$= P(t/tall) \times P(tall)$$

$$= \frac{1}{2} \times \frac{2}{9}$$

$$= \frac{2}{18} = \frac{1}{9} = 0.11$$

ab nikalo estimate value

Estimate = Addition of all likelihood

$$= 0 + 0 + 0.11$$

$$P(t) = 0.11$$

ab achual Probility nikalni hai

formula

$$P(x/y) = \frac{P(y/x)\,P(x)}{P(y)}$$

∴

$$P(\text{Short}/t) = \frac{P(t/\text{Short}) * P(\text{Short})}{P(t)}$$

$$= \frac{0 * \frac{4}{9}}{0.11} = 0$$

$$P(\text{Medium}/t) = \frac{P(t/\text{Medium}) * P(\text{Medium})}{P(t)}$$

$$= \frac{0 * \frac{3}{9}}{0.11} = 0$$

$$P(\text{tall}/t) = \frac{P(t/\text{tall}) \cdot P(\text{tall})}{P(t)}$$

$$= \frac{\frac{1}{2} * \frac{2}{9}}{0.11} = \frac{0.11}{0.11} = 1$$

ab tino ki value compare karo

tall sabse bada hai toh

Thus new tuple is tall.

Tall

Module 6
Dimensionality Reduction

Dimensionality reduction or dimension reduction is the process of reducing the number of random variables under consideration, via obtaining a set of "uncorrelated" principle variables. Dimension Reduction refers to the process of converting a set of data having vast dimensions into data with lesser dimensions ensuring that it conveys similar information concisely.

Advantages of Dimensionality Reduction

- It helps in data compressing and reducing the storage space required.
- Reduces the time required for performing computations.
- Allow usage of algorithms unfit for a large number of dimensions.
- Removal of multi-collinearity improves the performance of the machine learning model.
- It becomes easier to visualize the data when reduced to very low dimensions such as 2D or 3D.
- It is helpful in noise removal

Methods for Reducing Dimensionality

There are two main methods for reducing dimensionality:

- **Feature selection:** Feature selection deals with finding ***k*** of the ***d*** dimensions that give the most information and discard the other **(d − k)** dimensions. Feature selection approaches try to find a subset of the original variables. There are three strategies; *filter* (e.g. information gain) and *wrapper* (e.g. search guided by accuracy) approaches and *embedded* (features are selected to add or be removed while building the model based on the prediction errors).
- **Feature extraction:** Feature extraction deals in finding a new set of **k** dimensions that are combinations of the original **d** dimensions. (It does not eliminate the remaining dimension like feature selection). Feature extraction transforms the data in the high-dimensional space to a space of fewer dimensions. The data transformation may be linear, as in principal component analysis and linear discriminant analysis or non-linear method like isometric feature mapping, Laplacian eighen map etc.

Some other methods of Dimensionality reduction are as follows.

- **Missing Values:** Given a training sample with a set of features. Among the available features, a particular feature has many missing values. The feature or dimension with more missing values contribute less to the process of classification. So, such a feature with missing values can be completely eliminated.
- **Low Variance:** Consider that a particular feature has constant value for all the training sample. That means the variance of that feature for different sample is comparatively less. This implies that the feature with low variance or constant in nature have less impact in the process of classification, there by resulting in elimination of such a feature.
- **High Correlation:** if there are two features having high correlation with each other, then it implies that both the features contribute almost same for the classification process. In such a case, instead of using two features which is more or less same, it can be represented by a single feature.
- **Principal Component Analysis (PCA):** In this technique, variables are transformed into a new set of variables, which are linear combination of original variables. These new set of variables are known as principle components.
- **Backward Feature Elimination.** In this technique, at a given iteration, the selected classification algorithm is trained on *n* input features. Then we remove one input feature at a time and train the same model on *n-1* input features *n* times. The input feature whose removal has produced the smallest increase in the error rate is removed.

- **Forward Feature Construction.** This is the inverse process to the Backward Feature Elimination. This method starts with a single feature only, progressively adding one feature at a time, i.e. the feature that produces the highest increase in performance.
- **Random Forests / Ensemble Trees.** Decision Tree Ensembles, also referred to as random forests generate a large and carefully constructed set of trees against a target attribute and then use each attribute's usage statistics to find the most informative subset of features.

Subset Selection

Subset selection, is the process of finding the best subset of the set of features. The best subset contains the least number of dimensions that contribute most to the accuracy by discarding the remaining, unimportant dimensions. Once the necessary dimensions are found out, it can be used in both regression and classification problems using a suitable error function. Suppose there are ***d*** dimensions or variables, then there exist 2^d possible subsets. Unless if d is small, it is difficult to test all the variables. There are two approaches:

- Sequential Forward Approach
- Sequential Backward Approach

Sequential Forward Approach

Forward selection, start with no variables and then adds variable one by one with least possible error, until any further addition does not decrease the error (or decreases it only slightly).
Let F= a feature set of input dimensions, x_i , i = 1, . . . , d.
E(F) = the error incurred on the validation sample only when the inputs in F are used.
Depending on the application, the error is either the mean square error or misclassification error.

In sequential forward selection,
Initially there is no features: $F = \emptyset$
At each step, for all possible x_i, train the model on training set and calculate $E(F \cup x_i)$ on the validation set.
Choose that input x_j that causes the least error

$$j = \arg\min_i E(F \cup x_i)$$

and

$$\text{add } x_j \text{ to } F \text{ if } E(F \cup x_j) < E(F)$$

Feature adding stops when any feature does not decrease E (error). If there exist a user-defined threshold that depends on the application constraints, then also adding of features will be stopped.
The process of sequential forward selection may be costly because to decrease the dimensions from d to k, training and testing the system is $O(d^2)$. This is a local search procedure and does not guarantee finding the optimal subset, namely, the minimal subset causing the smallest error. For example, xi and xj by themselves may not be good but together may decrease the error a lot, but because this algorithm is greedy and adds attributes one by one, it may not be able to detect this. To avoid this drawback floating search methods is used where the number of added features and removed features can also change at each step.

Sequential Backward Approach

The backward selection, starts with all variables and remove them one by one, at each step removing the one that decreases the error the most (or increases it only slightly), until any further removal increases the error significantly. In sequential backward selection, the process starts with F containing all features and then remove one attribute from F at a time that causes the least error.

$$j = \arg\min_i E(F - x_i)$$

and

$$\text{remove } x_j \text{ from } F \text{ if } E(F - x_j) < E(F)$$

The process stops if removing a feature does not decrease the error. To decrease complexity, it decides to remove a feature if its removal causes only a slight increase in error. The complexity of backward search has the same order of complexity as forward search, except that training a system with more features is costlier than training a system with fewer features, and forward search may be preferable especially if many useless features are included. Subset selection is supervised in that outputs are used by the regressor or classifier to calculate the error.

Principal Component Analysis (PCA)

PCA is a useful statistical technique that has found application in fields such as face recognition and image compression and is a common technique for finding patterns in data of high dimension. **Principal component analysis** is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly **correlated variables** into a set of values of **linearly uncorrelated variables** called principal components. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.

The Principal Component Analysis is a projection method that finds a mapping from input in original d dimensional space to new k (k<d) dimensional space, with minimum loss of information. Projection of X on W is given by $\mathbf{Z=W^TX}$. PCA is an **unsupervised** method that does not use output information.

Some of the mathematical concepts used in PCA are summarized below.

Statistics: Statistics is based around the idea that there exist a big set of data, and it has to be analysed in terms of the relationships between the individual points in that data set. Some of the measures used for finding the relationships among data points are as follows.

Mean

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

The mean is the average of the numbers of a given set or a calculated "central" value of a set of numbers.

Standard Deviation

The standard deviation of the dataset is a measure of how spread out data is. It is the average distance from mean of dataset to a point. The standard deviation can be calculated by computing the squares of the distance from each data point to the mean of the set, add them all up, it is to compute the squares of the distance from each data point to the mean of the set, add them all up, divide by n-1, and take the positive square root. That is

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

where s is the symbol representing Standard Deviation. Here n-1 is used instead of n in order to represent a sample smaller in size than the population, where the result obtained by considering a small sample will always be more or less same as that of considering the actual sample.

Variance

Variance is another measure of the spread of data in a data set. In fact, it is almost identical to the standard deviation. The formula is

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

Both standard deviation and Variance are used to find the relationships among data points in a One Dimension Space.

Covariance

Covariance is always measured between 2 dimensions. If the covariance between one dimension and itself is calculated then variance is obtained. Covariance is a measure to find out how much the dimensions vary from the mean with respect to each other. The covariance between any two dimensions in a multi-dimensional space can be given as

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

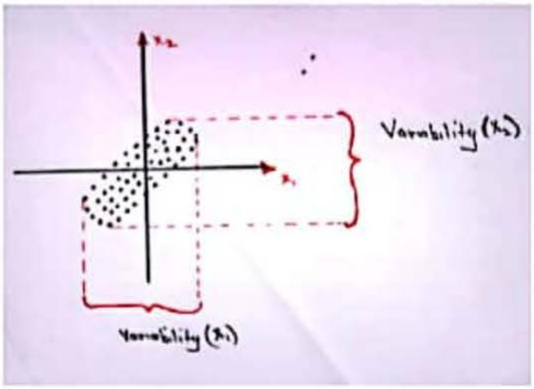
Covariance Matrix

Covariance is always measured between 2 dimensions. If a data set contains more than 2 dimensions, there is more than one covariance measurement that can be calculated. The matrix representation of all the possible calculated values of covariance is known as covariance matrix. A covariance matrix three dimensions is given as follows

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

Two important points to be noted about Principal Component Analysis are that, it lowers the dimension of the input features and orthogonality of the new dimension is always maintained.

Consider the data plot given below, which has two dimensions, one along axis X1 and other along X2.



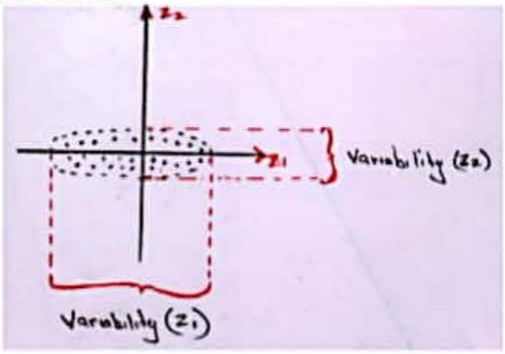
In the above data plot, the variability of data points along both the axis is large. That means, both the dimension provide significant information about the data points in the plot. So, each data point can be represented using the corresponding coordinate values of both dimensions.

$$X = \begin{array}{c} \begin{matrix} X_1 & X_2 \end{matrix} \\ \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \\ X_{31} & X_{32} \\ \vdots & \vdots \\ X_{n1} & X_{n2} \end{bmatrix} \end{array}$$

And the covariance between both dimensions can be denoted as

$$Cov(X) = \begin{array}{c} \begin{matrix} S_{11} & S_{12} \end{matrix} \\ \begin{bmatrix} S_{21} & S_{22} \end{bmatrix} \end{array}$$

The main aim of PCA is to find new dimensions such that the variability across atleast one of the dimension will be so small enough to be neglected. Inorder to find new dimensions, rotate the axis X1 and X2 by and angle ɵ. After rotaing both axis, the new axis obtained are Z1 and Z2. The data plot after transformation is shown below.

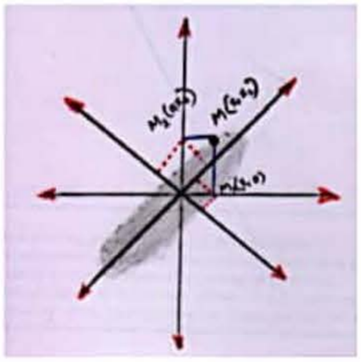


Now it can be noted that the variability of data along the axis Z2 is very small. That means it does not provide much information about the data points in the plot. So, ignoring such a dimension will not affect the quality of the classification process. The two new dimensions obtained by transforming the initial axis are known as Principal Components. Here the principal components are Z1 and Z2.

$$Var(Z1) >> Var(Z2)$$

In original data X1 and X2 are highly co-related. So, getting an ellipse where major and minor axis are not parallel to X1 and X2 which shows the dependencies between x1 and x2. After transformation major and minor axis are along Z1 and Z2 and it is known as orthogonality. Variability across Z2 is very less as compared to Z1 so that dimension Z2 can be ignored.

Consider a single data point in the plot say M(x1,x2). When the axis are changed to Z1 and z2 from x1 and x2, the corresponding coordinates have to be found out for each point in the data plot. Consider the below diagram



The new coordinates can be written in terms of old as

Z1: x1cos ɵ + x2 sin ɵ Z2:

-x1 sin ɵ + x2 cos ɵ

which can be written as $Z=A^T.X$

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

$$A^T = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \quad A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}$$

where

If the number of dimension is 'p' then

$$\begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_p \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \ldots\ldots & a_{1p} \\ a_{21} & a_{22} \ldots\ldots & a_{2p} \\ \vdots & \vdots & \vdots \\ a_{p1} & a_{p2} \ldots\ldots & a_{pp} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_2 \end{bmatrix}$$

From the above equation

$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ which can be written as

$\Rightarrow [a_1 \; a_2]$ where $a_1 = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$

So $a_1^T \cdot a_1 \Rightarrow \begin{bmatrix} \cos\theta & \sin\theta \end{bmatrix} \cdot \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$

$\Rightarrow \cos^2\theta + \sin^2\theta = 1$

So in general

$A^T \cdot A \Rightarrow \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow A \cdot A^T$

This proves the orthogonality of dimensions

If there are 'p' dimensions then each of the new principal component can be written as

$Z_1 = a_1^T X \Rightarrow a_{11}X_1 + a_{12}X_2 \cdots a_{1p}X_p$

$Z_2 = a_2^T X \Rightarrow a_{21}X_1 + a_{22}X_2 \cdots a_{2p}X_p$

$\vdots$

$Z_p = a_p^T X \Rightarrow a_{p1}X_1 + a_{p2}X_2 \cdots a_{pp}X_p$

where $a_p^T a_p = 1$ and

$$\boxed{Var(Z_1) \geq Var(Z_2) \geq \cdots \geq Var(Z_p)}$$

Principles of Principal Components

- Each PC is a linear combination of x, which is a $P \times 1$ variable vector ie $a_p^T x$
- First PC is $a_1^T x$, subjected to $a_1^T a_1 = 1$, that maximizes $\text{Var}(a_1^T x)$
- Second PC is $a_2^T x$ that maximizes $\text{Var}[a_2^T x]$ and subjected to $a_2^T a_2 = 1$ and

$$\text{cov}\left[a_1^T x, a_2^T x\right] = 0$$

- j^{th} Principal component is $a_j^T x$ that maximizes $\text{Var}(a_k^T x)$ and subjected to $a_j^T a_j = 1$ and

$$\text{cov}\left(a_j^T x, a_k^T x\right) = 0 \quad \text{for } k < j$$

In statistical terms the mean and standard deviation of a population is represented as M and Σ respectively. The mean and standard deviation of a sample is denoted by $\bar{x}$ and s. Now the principal components can be written in terms of standard deviation

For J^{th} principal component variance

$$V(z_j) = V(a_j^T x) = a_j^T V(x)\, a_j$$

$$= a_j^T \Sigma a_j$$

$$\text{Var}(x) \Rightarrow \text{cov}(x) = \Sigma$$

Expected value of z_j

$$E(z_j) = E\left[a_j^T x\right] = a_j^T E(x) = a_j^T M$$

$$a_j^T x = \left[a_j^T M,\ a_j^T \Sigma a_j\right]$$

In case of a sample the expected value of Z_j

$$E(z_j) = E\left[a_j^T x\right] = a_j^T E[x] = \underline{a_j^T \bar{x}}$$

$$\text{Cov}\left[Z_j\right] = \text{Var}\left[a_j^T x\right] = a_j^T \text{cov}[x]\, a_j = a_j^T S a_j$$

$$\boxed{a_j^T x = \left[a_j^T \bar{x},\ a_j^T S a_j\right]}$$

Now the problem of analysing, the principal components can be written in term of a lagrangian function which has a function to be maximized subjected to some constraints.

$$V(z_j) = a_j^T S . a_j \quad \text{subjected to} \quad a_j^T . a_j = 1 \ \text{or}\ a_j^T a_j - 1 = 0$$

$a_j^T S . a_j$ ⇑ function to be maximized

$a_j^T a_j - 1 = 0$ ⇑ constraint

So the function using Lagrangian multiplier is

$$L = a_j^T S a_j - \lambda\left[a_j^T . a_j - 1\right]$$ where λ is the lagrangian multiplier

Inorder to solve a lagrangian fn, compute the partial derivative and equate to zero.

$$\text{i.e } \frac{\partial L}{\partial a_j} = 0 \Rightarrow \left[S - \lambda I\right] a_j = 0$$

where S is a PxP matrix and $\underline{I}$ is a PxP matrix.

If the matrix is PxP then while calculating $[S-\lambda \underline{I}] = 0$ we get 'P' roots for λ.

Example.

Consider the data set and corresponding plot.

Profit	Sales
10	100
12	110
11	105
9	94
9	95
10	99
11	104
12	108
11	105
10	98
11	103
12	110

$$S = \text{cov}[Y] = \begin{bmatrix} 1.15 & 5.76 \\ 5.76 & 29.54 \end{bmatrix} \qquad \underline{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$[S-\lambda \underline{I}] = 0 \Rightarrow \left| \begin{bmatrix} 1.15 & 5.76 \\ 5.76 & 29.54 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = 0$$

$$(1.15-\lambda)(29.54-\lambda) - 5.76^2 = 0$$

The function is in the form of a quadratic eqn

This quadratic eqn can be solved for λ using eqn

$$\frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

Two roots of λ : λ_1 & λ_2 will be calculated

$$\lambda^2 - 30.96\lambda + C$$

where c is the constant part.

$$\lambda = -(-30.69) \pm \frac{\sqrt{(-30.69)^2 - 4.1.C}}{2.1}$$

$\lambda_1 = 30.60$

$\lambda_2 = 0.03$.

When $\lambda = \lambda_1 \Rightarrow 30.60$ then substitute in $[S - \lambda I] a_i = 0$

$$\left[\begin{bmatrix} 1.15 & 5.76 \\ 5.76 & 29.54 \end{bmatrix} - 30.66 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right] \times a_i = 0$$

here $a_i = \begin{bmatrix} a_{11} \\ a_{12} \end{bmatrix}$

we know $a_1^T a_1 = 1 \Rightarrow \begin{bmatrix} a_{11} & a_{12} \end{bmatrix} \cdot \begin{bmatrix} a_{11} \\ a_{12} \end{bmatrix} = 1$

$$a_{11}^2 + a_{12}^2 = 1$$

Substituting the above eqn.

we get

PC1	PC2
0.19	0.98
0.98	-0.19

PC1 $\Rightarrow Z_1 = 0.19X_1 + 0.98X_2$

$Z_2 = 0.98X_1 + 0.19X_2$.

Independent Component Analysis (ICA)

ICA is a statistical and computational technique for revealing hidden factors that underlie sets of random variables, measurements, or signals. It is a statistical technique for decomposing a complex dataset into independent sub-parts. One of the major application of ICA is for solving the Blind Source Separation (BSS) problem. The BSS problem is to determine the source signals given only the mixtures. The linear mixtures of two source signals which are known to be independent of each other, i.e. observing the value of one signal does not give any information about the value of the other. The above problem can be mathematically represented as

$$\mathbf{x} = \mathbf{As}$$

where 's' is a two-dimensional random vector containing the independent source signals, **A** is the two-by-two mixing matrix, and **x** contains the observed (mixed) signals. The main aim of independent component analysis is to find an unmixed matrix **w** where

$$\mathbf{W}=\mathbf{A}^{-1}$$

Motivation

Imagine that you are in a room where two people are speaking simultaneously. You have two microphones, which you hold in different locations. The microphones give you two recorded time signals, which we could denote by x1 (t) and x2 (t), with x1 and x2 the amplitudes, and t the time index. Each of these recorded signals is a weighted sum of the speech signals emitted by the two speakers, which we denote by s1 (t) and s2 (t). We could express this as a linear equation:

$$\mathbf{x_1(t) = a_{11}\, s_1 + a_{12} s_2} \quad \mathbf{(1)}$$

$$\mathbf{x_2(t) = a_{21}\, s_1 + a_{22} s_2} \quad \mathbf{(2)}$$

where a_{11}, a_{12}, a_{21}, and a_{22} are some parameters that depend on the distances of the microphones from the speakers. It would be very useful if you could now estimate the two original speech signals s_1 (t) and s_2(t), using only the recorded signals x_1 (t) and x_2 (t). This is called the **cocktail-party problem.**

One approach to solving this problem would be to use some information on the statistical properties of the signals s_i (t) to estimate the a_{ij} . Actually, and perhaps surprisingly, it turns out that it is enough to assume that s_1 (t) and s_2 (t), at each time instant t, are statistically independent. Independent Component Analysis, or ICA, can be used to estimate the a_{ij} based on the information of their independence, which allows us to separate the two original source signals s_1 (t) and s_2 (t) from their mixtures x_1 (t) and x_2 (t).

Definition of ICA

Assume that we observe n linear mixtures x_1 , ..., x_n of n independent components x $_j = a_{j1}\, s_1 + a_{j2}\, s_2 + ... + a_{jn} s_n$, for all j.

Using vector-matrix notation let us denote by **x** the random vector whose elements are the mixtures x_1 , ..., x_n. and by s the random vector with elements s_1 , ..., s_n . Let **A** the matrix with elements a_{ij}. All vectors are understood as column vectors; thus xT , or the transpose of x, is a row vector. Using vector matrix notation this can be represented as

$$\mathbf{x} = \mathbf{As}. \qquad \mathbf{(3)}$$

$$\mathbf{x} = \sum_{i=1}^{n} \mathbf{a}_i s_i.$$

The statistical model in Eq. 3 is called independent component analysis, or ICA model. The ICA model is a generative model, which means that it describes how the observed data are generated by a process of mixing the components s_i . The independent components are latent variables, meaning that they cannot be directly observed. Also the mixing matrix is assumed to be unknown. All we observe is the random vector x, and we must estimate both **A** and **s** using it. The starting point for ICA is the very simple assumption that the components si are statistically independent and the independent

component must have non-gaussian distributions. After estimating the matrix, A, compute its inverse, say W, and obtain the independent component simply by: **s = Wx.**

Ambiguities of ICA

1. Cannot determine the variances of the independent components. Because both s and A being unknown, any scalar multiplier in one of the sources s_i could always be cancelled by dividing the corresponding column a_i of A by the same scalar
2. Cannot determine the order of the independent components.

Definition and fundamental properties

What is independence?

Consider two scalar-valued random variables y1 and y2 . Basically, the variables y_1 and y_2 are said to be independent if information on the value of y_1 does not give any information on the value of y_2, and vice versa. independence can be defined by the probability densities. Let us denote by $p(y_1, y_2)$ the joint probability density function (pdf) of y_1 and y_2 . Let us further denote by $p_1(y_1)$ the marginal pdf of y_1 , i.e. the pdf of y_1 when it is considered alone.

$$p_1(y_1) = \int p(y_1, y_2) dy_2$$

and similarly for y_2 . Thus y_1 and y_2 are independent if and only if the joint pdf is factorizable in the following way:

$$p(y_1, y_2) = p_1(y_1) p_2(y_2)$$

Uncorrelated variables

A weaker form of independence is uncorrelatedness. Two random variables y1 and y2 are said to be uncorrelated if their covariance is zero.

$E\{y_1 y_2\} - E\{y_1\}E\{y_2\} = 0$

If the variables are independent, they are uncorrelated, On the other hand, uncorrelatedness does not imply independence. Since independence implies uncorrelatedness, many ICA methods constrain the estimation procedure so that it always gives uncorrelated estimates of the independent components. This reduces the number of free parameters and simplifies the problem.

Non-Gaussianity and Independence

Sum of two independent signal have a distribution closer to Gaussian. So gaussian can be considered as a linear combination of many independent signals. So, separation of signals can be accomplished by making linear signal transformation as non-gaussian as possible.

Measures of non-gaussianity

Kurtosis: The classical measure of non-gaussianity is kurtosis or the fourth-order cumulant. The kurtosis of y is classically defined by

$$\text{kurt}(y) = E\{y^4\} - 3(E\{y^2\})^2$$

If assuming that y is of unit variance, the right-hand side simplifies to $E\{y^4\} - 3$. This shows that kurtosis is simply a normalized version of the fourth moment $E\{y^4\}$. For a gaussian y, the fourth moment equals $3(E\{y^2\})^2$. Thus, kurtosis is zero for a Gaussian random variable. For most (but not quite all) non-gaussian random variables, kurtosis is non-zero. Kurtosis can be both positive or negative. Random variables that have a negative kurtosis are called sub- gaussian, and those with positive kurtosis are called super-gaussian.

Module 7
Learning with Clustering

K-means Clustering

The k-means algorithm takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intra-cluster similarity is high but the inter-cluster similarity is low. The cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster's centroid or centre of gravity.

Steps:

- First, it randomly selects k of the objects, each of which initially represents a cluster mean or centre.
- For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean.
- It then computes the new mean for each cluster.
- This process iterates until the criterion function converges.
- Typically, the square-error criterion is used, defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2,$$

where E is the sum of the square error for all objects in the data set; p is the point in space representing a given object; and 'mi' is the mean of cluster Ci (both p and mi are multidimensional)

Algorithm:

The k-means algorithm for partitioning, where each cluster's centre is represented by the mean value of the objects in the cluster.

Input:

k: the number of clusters,

D: a data set containing n objects.

Output: A set of k clusters.

Method:

(1) arbitrarily choose k objects from D as the initial cluster centres;

(2) **repeat**

(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;

(4) update the cluster means, i.e., calculate the mean value of the objects for each cluster;

(5) **until** no change;

Hierarchical Clustering

Hierarchical Clustering

A hierarchical clustering method works by grouping data objects into a tree of clusters. The quality of a pure hierarchical clustering method suffers from its inability to perform adjustment once a merge or split decision has been executed. That is, if a particular merge or split decision later turns out to have been a poor choice, the method cannot backtrack and correct it.
In general, there are two types of hierarchical clustering methods:

Agglomerative hierarchical clustering:

This bottom-up strategy starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters, until all of the objects are in a single cluster or until certain termination conditions are satisfied. Most hierarchical clustering methods belong to this category. They differ only in their definition of inter-cluster similarity.

Divisive hierarchical clustering:

This top-down strategy does the reverse of agglomerative hierarchical clustering by starting with all objects in one cluster. It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the diameter of each cluster is within a certain threshold.

In either agglomerative or divisive hierarchical clustering, the user can specify the desired number of clusters as a termination condition.

Difficulties with Hierarchical Clustering:

- The hierarchical clustering method, though simple, often encounters difficulties regarding the selection of merge or split points.
- Such a decision is critical because once a group of objects is merged or split, the process at the next step will operate on the newly generated clusters.
- It will neither undo what was done previously nor perform object swapping between clusters.
- Thus, merge or split decisions, if not well chosen at some step, may lead to low-quality clusters.
- Moreover, the method does not scale well, because each decision to merge or split requires the examination and evaluation of a good number of objects or clusters.

Solution to above Problems:

One promising direction for improving the clustering quality of hierarchical methods is to integrate hierarchical clustering with other clustering techniques, resulting in multiple-phase clustering. There are three of them –

- The first, called BIRCH, begins by partitioning objects hierarchically using tree structures, where the leaf or low-level non-leaf nodes can be viewed as "micro-clusters" depending on the scale of resolution. It then applies other clustering algorithms to perform macro-clustering on the micro-clusters.
- The second method, called ROCK, merges clusters based on their interconnectivity.
- The third method, called Chameleon, explores dynamic modelling in hierarchical clustering.

clustering.

Expectation Maximization Algorithm

The EM algorithm is an efficient iterative procedure to compute the Maximum Likelihood (ML) estimate in the presence of missing or hidden data. In ML estimation, we wish to estimate the model parameter(s) for which the observed data are the most likely.
Each iteration of the EM algorithm consists of two processes: The E-step, and the M-step. In the expectation, or E-step, the missing data are estimated given the observed data and current estimate of the model parameters. This is achieved using the conditional expectation, explaining the choice of terminology. In the M-step, the likelihood function is maximized under the assumption that the missing data are known. The estimate of the missing data from the E-step are used in lieu of the actual missing data. Convergence is assured since the algorithm is guaranteed to increase the likelihood at each iteration.

Derivation

Let $\mathbf{X}$ be random vector which results from a parameterized family. We wish to find θ such that $\mathcal{P}(\mathbf{X}|\theta)$ is a maximum. This is known as the Maximum Likelihood (ML) estimate for θ. In order to estimate θ, it is typical to introduce the *log likelihood function* defined as,

$$L(\theta) - \ln \mathcal{P}(\mathbf{X}|\theta). \tag{7}$$

The likelihood function is considered to be a function of the parameter θ given the data $\mathbf{X}$. Since $\ln(x)$ is a strictly increasing function, the value of θ which maximizes $\mathcal{P}(\mathbf{X}|\theta)$ also maximizes $L(\theta)$.

The EM algorithm is an iterative procedure for maximizing $L(\theta)$. Assume that after the n^{th} iteration the current estimate for θ is given by θ_n. Since the objective is to maximize $L(\theta)$, we wish to compute an updated estimate θ such that,

$$L(\theta) > L(\theta_n) \tag{8}$$

Equivalently we want to maximize the difference,

$$L(\theta) - L(\theta_n) - \ln \mathcal{P}(\mathbf{X}|\theta) - \ln \mathcal{P}(\mathbf{X}|\theta_n). \tag{9}$$

So far, we have not considered any unobserved or missing variables. In problems where such data exist, the EM algorithm provides a natural framework for their inclusion. Alternately, hidden variables may be introduced purely as an artifice for making the maximum likelihood estimation of θ tractable. In this case, it is assumed that knowledge of the hidden variables will make the maximization of the likelihood function easier. Either way, denote the hidden random vector by $\mathbf{Z}$ and a given realization by $\mathbf{z}$. The total probability $\mathcal{P}(\mathbf{X}|\theta)$ may be written in terms of the hidden variables $\mathbf{z}$ as,

$$\mathcal{P}(\mathbf{X}|\theta) - \sum_{\mathbf{z}} \mathcal{P}(\mathbf{X}|\mathbf{z},\theta)\mathcal{P}(\mathbf{z}|\theta). \tag{10}$$

We may then rewrite Equation (9) as,

$$L(\theta) - L(\theta_n) = \ln \sum_{\mathbf{z}} \mathcal{P}(\mathbf{X}|\mathbf{z},\theta)\mathcal{P}(\mathbf{z}|\theta) - \ln \mathcal{P}(\mathbf{X}|\theta_n). \tag{11}$$

Notice that this expression involves the logarithm of a sum. In Section (2) using Jensen's inequality, it was shown that,

$$\ln \sum_{i=1}^{n} \lambda_i x_i \geq \sum_{i=1}^{n} \lambda_i \ln(x_i)$$

for constants $\lambda_i \geq 0$ with $\sum_{i=1}^{n} \lambda_i = 1$. This result may be applied to Equation (11) which involves the logarithm of a sum provided that the constants λ_i can be identified. Consider letting the constants be of the form $\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)$. Since $\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)$ is a probability measure, we have that $\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n) \geq 0$ and that $\sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n) = 1$ as required.

Then starting with Equation (11) the constants $\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)$ are introduced as,

$$\begin{aligned}
L(\theta) - L(\theta_n) &= \ln \sum_{\mathbf{z}} \mathcal{P}(\mathbf{X}|\mathbf{z}, \theta)\mathcal{P}(\mathbf{z}|\theta) - \ln \mathcal{P}(\mathbf{X}|\theta_n) \\
&= \ln \sum_{\mathbf{z}} \mathcal{P}(\mathbf{X}|\mathbf{z}, \theta)\mathcal{P}(\mathbf{z}|\theta) \cdot \frac{\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)}{\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)} - \ln \mathcal{P}(\mathbf{X}|\theta_n) \\
&= \ln \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n) \left(\frac{\mathcal{P}(\mathbf{X}|\mathbf{z}, \theta)\mathcal{P}(\mathbf{z}|\theta)}{\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)} \right) - \ln \mathcal{P}(\mathbf{X}|\theta_n) \\
&\geq \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n) \ln \left(\frac{\mathcal{P}(\mathbf{X}|\mathbf{z}, \theta)\mathcal{P}(\mathbf{z}|\theta)}{\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)} \right) - \ln \mathcal{P}(\mathbf{X}|\theta_n) \\
&= \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n) \ln \left(\frac{\mathcal{P}(\mathbf{X}|\mathbf{z}, \theta)\mathcal{P}(\mathbf{z}|\theta)}{\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)\mathcal{P}(\mathbf{X}|\theta_n)} \right) \\
&\triangleq \Delta(\theta|\theta_n) \qquad (12)
\end{aligned}$$

Equivalently we may write,

$$L(\theta) \geq L(\theta_n) + \Delta(\theta|\theta_n) \qquad (13)$$

and for convenience define,

$$l(\theta|\theta_n) \triangleq L(\theta_n) + \Delta(\theta|\theta_n)$$

so that the relationship in Equation (13) can be made explicit as,

$$L(\theta) \geq l(\theta|\theta_n).$$

We have now a function, $l(\theta|\theta_n)$ which is bounded above by the likelihood function $L(\theta)$. Additionally, observe that,

$$\begin{aligned}
l(\theta_n|\theta_n) &= L(\theta_n) + \Delta(\theta_n|\theta_n) \\
&= L(\theta_n) + \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n) \ln \frac{\mathcal{P}(\mathbf{X}|\mathbf{z}, \theta_n)\mathcal{P}(\mathbf{z}|\theta_n)}{\mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n)\mathcal{P}(\mathbf{X}|\theta_n)} \\
&= L(\theta_n) + \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n) \ln \frac{\mathcal{P}(\mathbf{X}, \mathbf{z}|\theta_n)}{\mathcal{P}(\mathbf{X}, \mathbf{z}|\theta_n)} \\
&= L(\theta_n) + \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X}, \theta_n) \ln 1 \\
&= L(\theta_n), \qquad (14)
\end{aligned}$$

Our objective is to choose a values of θ so that $L(\theta)$ is maximized. We have shown that the function $l(\theta|\theta_n)$ is bounded above by the likelihood function $L(\theta)$ and that the value of the functions $l(\theta|\theta_n)$ and $L(\theta)$ are equal at the current estimate for $\theta = \theta_n$. Therefore, any θ which increases $l(\theta|\theta_n)$ in turn increase the $L(\theta)$. In order to achieve the greatest possible increase in the value of $L(\theta)$, the EM algorithm calls for selecting θ such that $l(\theta|\theta_n)$ is maximized. We denote this updated value as θ_{n+1}. This process is illustrated in Figure (2).

Formally we have,

$$\begin{aligned}
\theta_{n+1} &= \arg\max_{\theta} \{l(\theta|\theta_n)\} \\
&= \arg\max_{\theta} \left\{ L(\theta_n) + \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X},\theta_n) \ln \frac{\mathcal{P}(\mathbf{X}|\mathbf{z},\theta)\mathcal{P}(\mathbf{z}|\theta)}{\mathcal{P}(\mathbf{X}|\theta_n)\mathcal{P}(\mathbf{z}|\mathbf{X},\theta_n)} \right\} \\
&\quad \text{Now drop terms which are constant w.r.t. } \theta \\
&= \arg\max_{\theta} \left\{ \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X},\theta_n) \ln \mathcal{P}(\mathbf{X}|\mathbf{z},\theta)\mathcal{P}(\mathbf{z}|\theta) \right\} \\
&= \arg\max_{\theta} \left\{ \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X},\theta_n) \ln \frac{\mathcal{P}(\mathbf{X},\mathbf{z},\theta)}{\mathcal{P}(\mathbf{z},\theta)} \frac{\mathcal{P}(\mathbf{z},\theta)}{\mathcal{P}(\theta)} \right\} \\
&= \arg\max_{\theta} \left\{ \sum_{\mathbf{z}} \mathcal{P}(\mathbf{z}|\mathbf{X},\theta_n) \ln \mathcal{P}(\mathbf{X},\mathbf{z}|\theta) \right\} \\
&= \arg\max_{\theta} \{ \mathrm{E}_{\mathbf{Z}|\mathbf{X},\theta_n} \{ \ln \mathcal{P}(\mathbf{X},\mathbf{z}|\theta) \} \} \qquad (15)
\end{aligned}$$

In Equation (15) the expectation and maximization steps are apparent. The EM algorithm thus consists of iterating the:

1. *E-step*: Determine the conditional expectation $\mathrm{E}_{\mathbf{Z}|\mathbf{X},\theta_n} \{ \ln \mathcal{P}(\mathbf{X},\mathbf{z}|\theta) \}$
2. *M-step*: Maximize this expression with respect to θ.

At this point it is fair to ask what has been gained given that we have simply traded the maximization of $L(\theta)$ for the maximization of $l(\theta|\theta_n)$. The answer lies in the fact that $l(\theta|\theta_n)$ takes into account the unobserved or missing data $\mathbf{Z}$. In the case where we wish to estimate these variables the EM algorithms provides a framework for doing so. Also, as alluded to earlier, it may be convenient to introduce such hidden variables so that the maximization of $L(\theta|\theta_n)$ is simplified given knowledge of the hidden variables. (as compared with a direct maximization of $L(\theta)$)

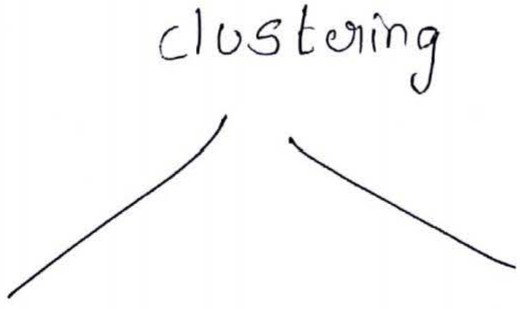
What is Clustering?

Clustering is the process of making a group of abstract object into classes of similar object.

(matlab abstract object bole toh bahut saare object me se similar object ka group banana isko kehte hai clustering.)

Application of clustering

- Clustering analysis is broadly used in many application such as market research, pattern recognization, data analysis and image Processing



K-mean clustering

- This is used in ~~creative~~ creating central cluster

- K mean clustering aims to partition n observation into K cluster in which each observation belongs to the cluster with nearest mean, serving as a prototype of the cluster. This result in partition of the data space into cell

(K mean ka maksat hota hai partition karna (n = jitna data hai) in k cluster)

- ~~Procedure~~ Steps to solve the ~~solve~~ sums

Step 1) Take mean value

step 2) find nearest number of mean and Pot in cluster

step 3) Repeat Process one and two untill we get same mean

Tension mat lo jab sum solve karenge toh zyaada achse samjhega.

Given Problem:-

Page No.

$K = \{2, 3, 4, 10, 11, 12, 20, 25, 30\}$ ← given data set

$K = 2$ ← jitna K diya hai utna cluster banane ke.

Step 1) Take mean value

K=2 hai dekho abhi hamara start hai so pehle se agar 2 mean value nahi di rahengi toh koi bhi 2 random value ko le lene ka 2 kya bcoz K=2 hai isliye.

$m_1 = 4$ $m_2 = 12$

Step 2) Find nearest number of mean and Put in cluster

$K_1 = \{2, 3, 4\}$

jo 4 ke zyaada kareeb hai usko K_1 me daalo

$K_2 = \{10, 11, 12, 20, 25, 30\}$

jo 12 ke zyaad kareeb hai usko K_2 me daalo

ab hrsme mean nikalo Pan asli wala K_1 ka mean and K 2 ka mean

$$m_1 = \frac{2+3+4}{3} = 3$$

$$m_2 = \frac{10+11+12+20+25+30}{6}$$

$$m_2 = \frac{108}{6} = 18$$

$\therefore m_1 = 3$ $\quad$ $m_2 = 18$

same Process cluster banao

$K_1 = \{2, 3, 4, 10\}$ $\quad$ $K_2 = \{11, 12, 20, 25, 30\}$

$m_1 = 4.75$ $\quad$ $m_2 = 19.6$

$K_1 = \{2, 3, 4, 10, 11, 12\}$ $\quad$ $K_2 = \{20, 25, 30\}$

$m_1 = 7$ $\quad$ $m_2 = 25$

$K_1 = \{2, 3, 4, 10, 11, 12\}$ $\quad$ $K_2 = \{20, 25, 30\}$

$m_1 = 7$ $\quad$ $m_2 = 25$

Thus we are getting same mean we have to stop K_1 and K_2 are new cluster.

Part (1)

Agglomerative clustering

Example 1 (How to draw adjacency Matrix)

object	Attribute(x)	Attribute (Y)
A	2	2
B	3	1
C	1	1
D	3	0.5
E	1.5	

Before creating the adjacency matrix, we need to find the distance between each object to the other object

By Eucledian Distance formula

$$d(A,B) = \sqrt{(3-2)^2 + (2-2)^2} = 1$$

$$d(A,C) = \sqrt{(1-2)^2 + (1-2)^2} = \sqrt{2} = 1.41$$

$$d(A,D) = \sqrt{(3-2)^2 + (1-2)^2} = \sqrt{2} = 1.41$$

$$d(A,E) = \sqrt{(1.5-2)^2 + (0.5-2)^2} = 1.58$$

$$d(B,C) = \sqrt{(1-3)^2 + (1-2)^2} = \sqrt{5} = 2.24$$

$$d(B,D) = \sqrt{(3-3)^2 + (1-2)^2} = 1$$

$$d(B,E) = \sqrt{(1.5-3)^2 + (0.5-2)^2} = 2.12$$

$$d(C,D) = \sqrt{(3-1)^2 + (1-1)^2} = 2$$

$$d(C,E) = \sqrt{(1.5-1)^2 + (0.5-1)^2} = 6.75$$

$$d(D,E) = \sqrt{(1.5-3)^2 + (0.5-1)^2} = 1.58$$

The adjacency matrix is given as

object	A	B	C	D	E
A	0				
B	1	0			
C	1.41	2.24	0		
D	1.41	1	2	0	
E	1.58	2.12	0.71	1.58	0

Agglomerative with Dendrogram.

Item	E	A	C	B	D
E	0	1	2	2	3
A	1	0	2	5	3
C	2	2	0	1	6
B	2	5	1	0	3
D	3	3	6	3	0

Consider only one part (lower trangular part)

Item	E	A	C	B	D
E	0				
A	1	0			
C	2	2	0		
B	2	5	1	0	
D	3	3	6	3	0

Now find minimum distance

minimum distance = 1 for E, A

Hence merge EA

Dendrogram

Distance

Find min distance from EA to other

$\min [\text{dist}\{(E,A), C\}]$

$= \min [\text{dist}(E,C), \text{dist}(A,C)]$

$= \min [2, 2] = 2$

$\min [\text{dist}\{(E,A), D\}]$ dist (ED, AD)

$= \min [3, 3] = 3$ = 3

$\min [\text{dist}\{(E,A), B\}]$ dist (EB, AB)

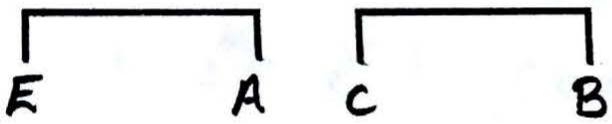
$= \min [2, 5]$

$= 2$

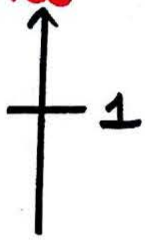
Item	EA	C	B	D
EA	0			
C	2	0		
B	2	1	0	
D	3	6	3	0

In this minimum Distance = 1 for C, B.

Dendrogram



Distance



Now again Find min distance From CB to other Point.

min dist [(E,A), (C,B)]

= min dist [(E,C), (E,B), (A,C), (A,B)]

= min dist [2, 2, 2, 5] = 2.

min dist [(C,B), D]

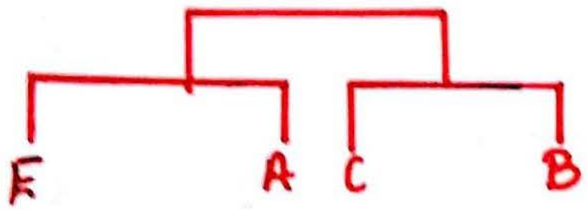
= min. dist [(C,B), D]

= min dist [(C,D), (B,D)]

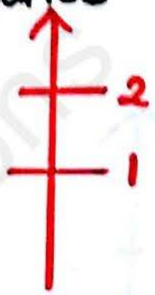
= min. dist [6, 3] = 3

Item	EA	CB	D
EA	0		
CB	2	0	
D	3	3	0

Dendrogram



Distance



Now minimum distance = 2 for EA, CB

min. dist [(E,A,C,B), D]

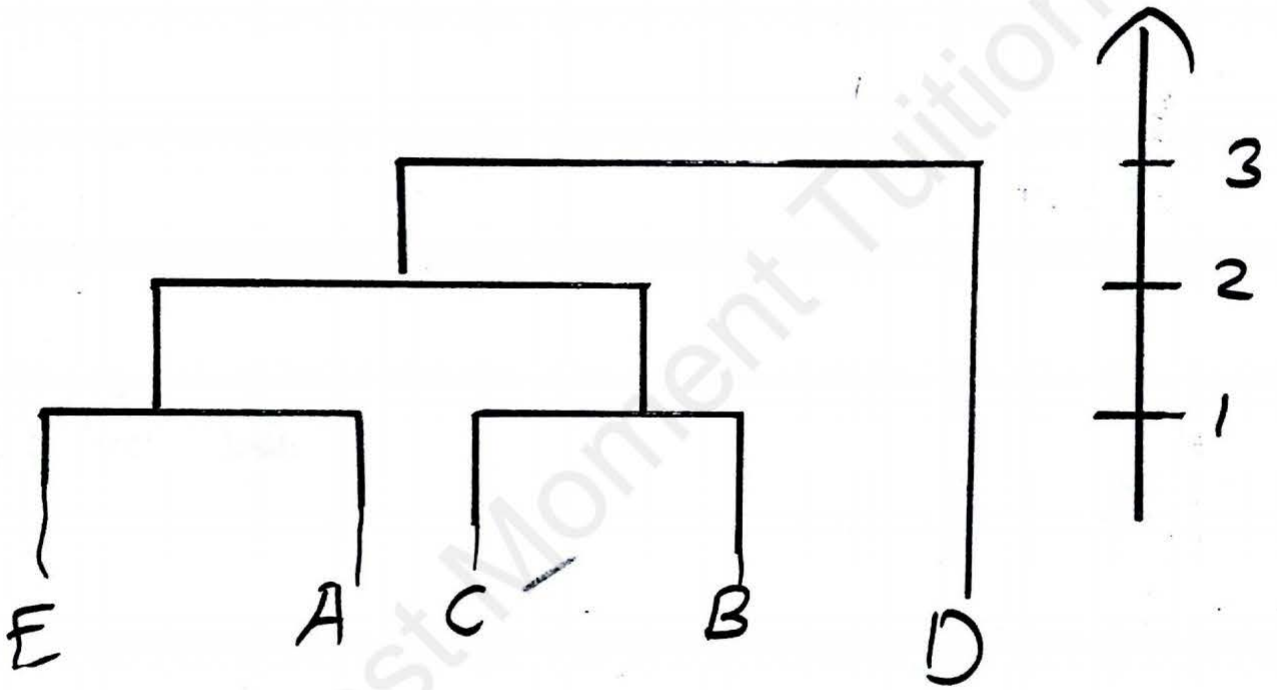
= min. dist [(E,D), (A,D), (C,D), (B,D)]

= min dist [3, 3, 6, 3]

= 3

	EA CB	D
EA CB	0	
D	3	0

Dendrogram



Radial Basis Function

A radial basis function (RBF) is a term that describes any real valued function whose output depends exclusively on the distance of its input from some origin. Typically, radial basis functions are defined in terms of the standard Euclidean norm of the input vector, but technically speaking one can use any other norm as well.

A radial basis function gives value to each point based on its distance from the origin or a fixed centre, usually on a Euclidean space. It is spherically symmetrical.

In machine learning, radial basis functions are most commonly used as a kernel for classification with the support vector machine (SVM). For this application of RBFs, the Gaussian Radial Basis Function is virtually always selected.

One of the commonly-used types, Gaussian, is usually chosen as the activation function to compute the derived features (units in the middle of the network) in neural networks known as radial basis function networks.

RBFs can also occasionally be used as activation functions in neural networks to form what is known as a Radial Basis Function Network, but this application is uncommon.

MODULE 8

Reinforcement Learning

Introduction

- Reinforcement Learning is a type of *Machine Learning*, and thereby also a branch of *Artificial Intelligence*.
- It allows machines and software agents to automatically determine the ideal behaviour within a specific context, in order to maximize its performance.
- Simple reward feedback is required for the agent to learn its behaviour; this is known as the reinforcement signal.
- Reinforcement Learning allows the machine or software agent to learn its behaviour based on feedback from the environment.
- If the problem is modelled with care, some Reinforcement Learning algorithms can converge to the global optimum; this is the ideal behaviour that maximises the reward.

Functioning

- In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives reward (or penalty) for its actions in trying to solve a problem.
- After a set of trial and error runs, it should learn the best policy, which is the sequence of actions that maximize the total reward.

Core Concept

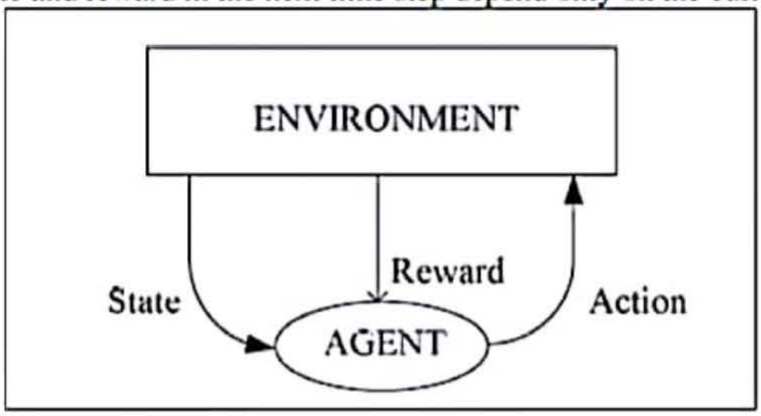
- Reinforcement learning differs from standard supervised learning in which correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.
- There is a focus on 'on-line' performance, which involves finding a balance between exploration (of uncharted territory) and exploitation.
- It is called "learning with a critic," as opposed to learning with a teacher which we have in supervised learning.
- A critic differs from a teacher in that it does not tell us what to do but only how well we have been doing in the past; the critic never informs in advance.
- The feedback from the critic is scarce and when it comes, it comes late. This leads to the credit assignment problem.
- The reinforcement learning generates internal values or the intermediate actions in terms of how good it is in leading to the goal and getting us to the real reward.
- Once such an internal reward mechanism is learned, the agent can just take the local actions to maximize it.
- The solution to the task requires a sequence of actions which resembles a markov process. A Markov decision process is used to model the agent.
- The difference is that in the case of Markov models, there is an external process that generates a sequence of signals. In this case it is the agent that generates the sequence of actions.
- Sometimes a partially observable Markov decision process is seen in cases where the agent does not know its state exactly but should infer it with some uncertainty through observations using sensors.

Elements of Reinforcement Learning

The main components of reinforcement learning are

- States and Actions
- Environment and Agent
- Reward

The learning decision maker is called the **agent**. The agent interacts with the **environment** that includes everything outside the agent. The agent has sensors to decide on its state in the environment and takes an **action** that modifies its state. When the agent takes an action, the environment provides a **reward.** Time is discrete as t = 0, 1, 2, . . ., and $s_t \in S$ denotes the **state** of the agent at time t where S is the set of all possible states. $a_t \in A(s_t)$ denotes the action that the agent takes at time t where A(st) is the set of possible actions in state s_t. When the agent in state s_t takes the action 'at', the clock ticks, reward $r_{t+1} \in R$ is received, and the agent moves to the next state, Markov decision s_{t+1}. The problem is modelled using a Markov process decision process (MDP). The reward and next state are sampled from their respective probability distributions, $p(r_{t+1}|s_t, at)$ and $P(s_{t+1}|s_t, a_t)$. This is similar to Markov system where the state and reward in the next time step depend only on the current state and action.



Episode/Trail: The sequence of actions from the start to the terminal state is an episode or a trial. Based on the application the any state may be designated as the initial state. There is also an absorbing terminal (goal) state where the search ends; all actions in this terminal state transition to itself with probability 1 and without any reward.

Policy: The policy, defines the agent's behaviour and is a mapping from the states of theπ environment to actions: $S \pi \rightarrow A$. The policy defines the action to be taken in any state $s_t : a_t = (s\pi_t)$. The value of a policy, $V \pi^{\pi}(s_t)$, is the expected cumulative reward that will be received while the agent follows the policy, starting from state s_t.

Finite Horizon: In the finite horizon or episodic model, the agent tries to maximize the expected reward for the next T steps:

$$V^{\pi}(s_t) = E[r_{t+1} + r_{t+2} + \cdots + r_{t+T}] = E\left[\sum_{l=1}^{T} r_{t+l}\right]$$

Infinite Horizon: Tasks that are continuing, and there is no prior fixed limit to the episode. In the infinite horizon model, there is no sequence limit, but future rewards are discounted.

$$V^{\pi}(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots] = E\left[\sum_{l=1}^{\infty} \gamma^{l-1} r_{t+l}\right]$$

where $0 \leq \gamma < 1$ is the discount rate

Discount Rate: The discount rate is used to keep the return finite. If = 0, then only the 'γ' immediate reward counts. As approaches 1, rewards further in the future count more. 'γ' is less than 1 because there generally is a time limit to the sequence of actions needed to solve the task.

Optimal Policy: For each policy, there is a $V\pi^*(s_t)$, and we want to find the optimal policy π^* such that

$$\boxed{V^*(s_t) = \max_{\pi} V^{\pi}(s_t), \forall s_t}$$

In some cases instead of working with the values of states, $V(s_t)$, it is preferred to work with the values of state-action pairs, $Q(s_t, a_t)$. $V(s_t)$ denotes how good it is for the agent to be in state s_t, whereas $Q(s_t, a_t)$ denotes how good it is to perform action a_t when in state s_t. We define $Q^*(s_t, a_t)$ as the value, that is, the expected cumulative reward, of action a_t taken in state s_t and then obeying the optimal policy afterwards. The value of a state is equal to the value of the best possible action:

$$\begin{aligned}
V^*(s_t) &= \max_{a_t} Q^*(s_t, a_t) \\
&= \max_{a_t} E\left[\sum_{l=1}^{\infty} \gamma^{l-1} r_{t+l}\right] \\
&= \max_{a_t} E\left[r_{t+1} + \gamma \sum_{l=1}^{\infty} \gamma^{l-1} r_{t+l+1}\right] \\
&= \max_{a_t} E\left[r_{t+1} + \gamma V^*(s_{t+1})\right] \\
V^*(s_t) &= \max_{a_t} \left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V^*(s_{t+1})\right) \text{ ----------(1)}
\end{aligned}$$

The equation (1) is known as Bellman's Equation. Similarly, the value of expected cumulative reward can be represented by

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

Once the value of $Q^*(s_t, a_t)$ is calculated the policy can be defined by taking the action $a\pi_t$, which has the highest value among all $Q^*(s_t, a_t)$

$$\pi^*(s_t) : \text{Choose } a_t^* \text{ where } Q^*(s_t, a_t^*) = \max_{a_t} Q^*(s_t, a_t)$$

If values of $Q^*(s_t, a_t)$ are known, then using greedy approach at each local step, the optimal sequence of steps that maximises the cumulative reward can be calculated.

Learning in Reinforcement learning can be done by two ways

- Model Based Learning
- Model Free Learning

Model-Based Learning

In model-based learning, the environment model parameters, $P(r_{t+1}|s_t, a_t)$ and $P(s_{t+1}|s_t, a_t)$ are known. If the environmental parameters are known optimal value function and policy can be solved directly using dynamic programming. The optimal value function is unique and is the solution to the simultaneous equations (Bellman's equation). The optimal policy is to choose the action that maximizes the value of optimal value function, in the next state:

$$\pi^*(s_t) = \arg\max_{a_t} \left(E[r_{t+1}|s_t, a_t] + \gamma \sum_{s_{t+1} \in S} P(s_{t+1}|s_t, a_t) V^*(s_t + 1)\right)$$

The optimal policy can be evaluated using either

- Value iteration Algorithm
- Policy iteration Algorithm

Value Iteration

To find the optimal policy, use the optimal value function and apply the iterative algorithm called value iteration that will converge to the correct V* values.

Initialize $V(s)$ to arbitrary values
Repeat
  For all $s \in S$
    For all $a \in \mathcal{A}$
      $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
    $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

If the maximum value difference between two iterations is less than a certain threshold δ, then the values can be assumed to be converged.

$$\max_{s \in S} |V^{(l+1)}(s) - V^{(l)}(s)| < \delta$$

where l is the iteration counter. It is possible that the policy converges to the optimal one even before the values converge to their optimal values because only the actions with maximum values are considered.

Policy Iteration

In policy iteration, the policies are stored and updated, rather than doing this indirectly over the values. The process starts with a policy and improve it repeatedly until there is no change. The value function can be calculated by solving for the linear equations. Improvement in the policy by is checked. This step is guaranteed to improve the policy, and when no improvement is possible, the policy is guaranteed to be optimal.

Initialize a policy π' arbitrarily
Repeat
  $\pi \leftarrow \pi'$
  Compute the values using π by
    solving the linear equations
      $V^{\pi}(s) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s))V^{\pi}(s')$
  Improve the policy at each state
    $\pi'(s) \leftarrow \arg\max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V^{\pi}(s'))$
Until $\pi = \pi'$

Temporal Difference Learning

Temporal difference (TD) learning is a prediction-based machine learning method. It has primarily been used for the reinforcement learning problem and is said to be "a combination of Monte Carlo ideas and dynamic programming (DP) ideas. TD resembles a Monte Carlo method because it learns by sampling the environment according to some *policy*, and is related to dynamic programming techniques as it approximates its current estimate based on previously learned estimates.

Temporal Difference learning is one of the most used approaches for policy evaluation. It is a central part of solving reinforcement learning tasks. For deriving optimal control, policies have to be evaluated. This task requires value function approximation.

Optimal policy can be solved using dynamic programming, but it need good knowledge about the environment. The environment is explored to get values of next state and rewards, and these information is used to update values of current state. These algorithms are called temporal difference algorithms because the difference between our current estimate of the value of a state (or a state-action pair) and the discounted value of the next state and the reward received are considered.

Exploration Strategies

ε-greedy search: ε-greedy search is where with probability ε, we choose one action uniformly randomly among all possible actions reffered to as explore, and with probability 1-ε, we choose the best action, reffered as exploit. Start with a high value and gradually decrease it inorder to exploit.

Soft Policy: Policy is soft, if the probability of choosing any action a ∈ A in state s ∈ S is greater than 0. A **Soft-Max** function is a function used to convert values to probabilities

$$P(a|s) = \frac{\exp Q(s,a)}{\sum_{b \in \mathcal{A}} \exp Q(s,b)}$$

and then sample according to these probabilities. The above equation can be rewritten use a "temperature" variable T and define the probability of choosing action a as

$$P(a|s) = \frac{\exp[Q(s,a)/T]}{\sum_{b \in \mathcal{A}} \exp[Q(s,b)/T]}$$

When T is large, all probabilities are equal and we have exploration. When T is small, better actions are favored. The strategy is to start with a large T and decrease it gradually, a procedure named annealing, which in this case moves from exploration to exploitation

Model Free Learning is divided into

- Deterministic Rewards and Actions
- Non-Deterministic Rewards and Actions

Deterministic Rewards and Actions

In deterministic case, at any state-action pair, there is a single reward and next state possible.

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

use this as an assignment to update Q(st , at). At state s_t , action a_t is chosen, which returns a reward r_{t+1} and takes us to state s_{t+1} . The value of previous action can be updated as

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

where the hat denotes that the value is an estimate Q(st+1 , at+1) is a later value and has a higher chance of being correct. We discount this by and add the immediate reward (ifγ any) and take this as the new estimate for the previous Q(st , at). This is called a backup because it can be viewed as taking the estimated value of an action in the next time step and "backing it up" to revise the estimate for the value of a current action.

Assume that all Q(s, a) values are stored in a table. Initially all (s t , a t) are 0, and theyQQ are updated in time as a result of trial episodes. If there is a sequence of moves and at each move the estimate of the Q value of the previous state-action pair is updated using the Q value of the current state-action pair by the formula

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

In the intermediate states, all rewards have values as 0, so no update is done. When the goal state is reached, the reward r is obtained and update the Q value of the previous stateaction pair as r. As for the preceding state-action pair, its immediate reward is 0 and the γ contribution from the next state-action pair is discounted by γ because it is one step later. Then in another episode, if we reach this state, we can update the one preceding that as γ to r, and so on. This way, after many episodes, this information is backed up to earlier stateaction pairs. Q values increase until they reach their optimal values as we find paths with higher cumulative reward.

Nondeterministic Rewards and Actions

If the rewards and the result of actions are not deterministic, then there is a probability distribution for the reward $p(r_{t+1}|s_t,a_t)$ from which rewards are sampled, and there is a probability distribution for the next state $P(s_{t+1}|s_t,a_t)$. These help us model the uncertainty in the system that may be due to forces that cannot control in the environment.

For example, we may have an imperfect robot which sometimes fails to go in the intended direction

$$Q(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

and deviates or advances shorter or longer than expected. we can write

A direct assignment is not possible because for the same state and action, different rewards or move to different next states are received. This type of problem can be handled by keeping the Average, and the method is known as Q-Learning.

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \eta(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

In the above equation, the term

$$r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

has values as a sample of instances for each (s_t,a_t) pair and $Q(s_t,a_t)$ converge to its mean. η is gradually decreased in time for convergence, and it has been shown that this algorithm converges to the optimal Q* values. The pseudocode of the Q learning algorithm is given below

Initialize all $Q(s,a)$ arbitrarily
For all episodes
  Initalize s
  Repeat
    Choose a using policy derived from Q, e.g., ϵ-greedy
    Take action a, observe r and s'
    Update $Q(s,a)$:
      $Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
    $s \leftarrow s'$
  Until s is terminal state

Q learning, which is an off-policy temporal difference algorithm.

Temporal difference (TD) algorithms are those reducing the difference between the current Q value and the backed-up estimate, from one-time step later. This is an off-policy method as the value of the best next action is used without using the policy. In an on-policy method, the policy is used to determine also the next action. The on-policy version of Q learning is the Sarsa algorithm.

Initialize all $Q(s,a)$ arbitrarily
For all episodes
  Initalize s
  Choose a using policy derived from Q, e.g., ϵ-greedy
  Repeat
    Take action a, observe r and s'
    Choose a' using policy derived from Q, e.g., ϵ-greedy
    Update $Q(s,a)$:
      $Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma Q(s',a') - Q(s,a))$
    $s \leftarrow s'$, $a \leftarrow a'$
  Until s is terminal state

Sarsa algorithm, which is an on-policy version of Q learning.

The on-policy Sarsa uses the policy derived from Q values to choose one next action a' and uses its Q value to calculate the temporal difference. On-policy methods estimate the value of a policy while using it to take actions. In off-policy methods, these are separated, and the policy used to generate behaviour, called the **behaviour policy** is different from the policy that is evaluated and improved, called the **estimation policy**.

Eligibility Traces: The temporal difference is used to update only the previous value (of the state or state-action pair). An eligibility trace is a record of the occurrence of past visits that enables us to implement temporal credit assignment, allowing us to update the values of previously occurring visits as well. To store the eligibility trace, an additional memory variable is required associated with each state-action pair, e(s, a), initialized to 0. When the state-action pair (s, a) is visited, i.e. an action a is taken in state s, its eligibility is set to 1; the eligibilities of all other state-action pairs are multiplied by $\gamma\lambda$. $0 \leq \lambda \leq 1$ is the trace decay parameter.

$$e_t(s,a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases}$$

If a state-action pair has never been visited, its eligibility remains 0; if it has been, as time passes and other state-actions are visited, its eligibility decays depending on the value of γ and λ. In Sarsa, the temporal error at time t is

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

In Sarsa with an eligibility trace, named Sarsa(λ), all state-action pairs are updated as

$$Q(s,a) \leftarrow Q(s,a) + \eta\delta_t e_t(s,a), \ \forall s,a$$

The algorithm for Sarsa(λ) is as follows:

```
Initialize all Q(s,a) arbitrarily, e(s,a) ← 0, ∀s,a
For all episodes
    Initalize s
    Choose a using policy derived from Q, e.g., ϵ-greedy
    Repeat
        Take action a, observe r and s'
        Choose a' using policy derived from Q, e.g., ϵ-greedy
        δ ← r + γQ(s',a') − Q(s,a)
        e(s,a) ← 1
        For all s,a:
            Q(s,a) ← Q(s,a) + ηδe(s,a)
            e(s,a) ← γλe(s,a)
        s ← s', a ← a'
    Until s is terminal state
```

Sarsa(λ) algorithm.

Generalization

In all reinforcement learning algorithms its assumed that the Q(s, a) values (or V (s), if values of states are estimated) are stored in a lookup table, and the algorithms are called tabular algorithms. Some issues with this tabular approach are:

- when the number of states and the number of actions is large, the size of the table may become quite large.
- States and actions may be continuous, and to use a table, they should be discretized which may cause error.
- When the search space is large, too many episodes may be needed to fill in all the entries of the table with acceptable accuracy.

Instead of storing the Q values, consider a regression problem, which is a supervised learning problem with a regressor $Q(s,a|\ )\theta$, taking s and a as inputs and parameterized by a vector of parameters, θ, to learn Q values. This can be an artificial neural network with s and a as its inputs, one output, and θ its connection weights.

A good approximation may be achieved with a simple model without explicitly storing the training instances; it can use continuous inputs; and it allows generalization. If we know that similar (s,a) pairs have similar Q values, we can generalize from past cases and come up with good Q(s,a) values even if that state-action pair has never been encountered before.

To be able to train the regressor, a training set is required. In the case of Sarsa(0), we saw before that we would like Q(st , at) to get close to rt+1 + Q(st+1 , at+1). So, we can form a set of training samples where the input is the state-action pair (st , at) and the required output is

$$r_{t+1} + \gamma Q(s_{\gamma\,t+1}, a_{t+1}).$$

The squared error is as follows.

$$E'(\theta) = [r\theta_{t+1} + \lambda Q(s_{\gamma\,t+1}, a_{t+1}) - Q(s_t, a_t)]^2$$

If using a gradient-descent method, as in training neural networks, the parameter vector is updated as

$$\Delta\boldsymbol{\theta} = \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t)$$

This is a one-step update. In the case of Sarsa( ), the eligibility trace is also taken into account

$$\Delta\boldsymbol{\theta} = \eta\delta_t \boldsymbol{e}_t$$

where the temporal difference error is

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

and the vector of eligibilities of parameters are updated as

$$\boldsymbol{e}_t = \gamma\lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t)$$

with e 0 all zeros. In the case of a tabular algorithm, the eligibilities are stored for the stateaction pairs because they are the parameters (stored as a table). In the case of an estimator, eligibility is associated with the parameters of the estimator. any regression method can be used to train the Q function, but the particular task has a number of requirements. First, it should allow generalization; that is, need to guarantee that similar states and actions have similar Q values. This also requires a good coding of s and a, as in any application, to make the similarities apparent. Second, reinforcement learning updates provide instances one by one and not as a whole training set, and the learning algorithm should be able to do individual updates to learn the new instance without forgetting what has been learned before.

Well Posed Learning Problem:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

It incudes 3 features

Learning Task : The thing that needs to be learnt.

Training Experience : The fetching and processing of data for training the model.

Performance Measure : The critic that helps us to recognize the result were correct or incorrect.

- Example

Task T: Playing checkers

Training Experience E: Playing games against itself

Performance Measure P: Percentage of games won against opponents.