Department of Computer Science And Engineering

Regulation 2021

II Year –IV Semester

CS3491-Artificial Intelligence And Machine Learning

UNIT II PROBABILISTICREASONING

Actingunderuncertainty–Bayesianinference–naïveBayesmodels.Probabilisticreasoning–Bayesiannetworks–exactinferenceinBN –approximateinferenceinBN–causalnetworks.

1. Actingunderuncertainty

Uncertainty:

Tillnow,wehavelearnedknowledgerepresentationusingfirst-orderlogicandpropositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes ofuncertainty:

Followingare someleading causesofuncertaintytooccurinthe realworld.

1. Informationoccurredfromunreliable sources.
2. ExperimentalErrors
3. Equipmentfault
4. Temperature variation
5. Climate change.

2. Bayesianinference

Bayes'theoreminArtificialintelligence

Bayes'theorem:

Bayes' theorem isalsoknown as **Bayes'rule,Bayes'law**,or **Bayesianreasoning**,which determinesthe probabilityof an event with uncertain knowledge.

Inprobabilitytheory,itrelatestheconditionalprobabilityandmarginalprobabilitiesoftworandomevents.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The**Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

ItisawaytocalculatethevalueofP(B|A)withtheknowledgeofP(A|B).

Bayes'theoremallowsupdatingtheprobabilitypredictionofaneventbyobservingnewinformationoftherealworld.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes'theoremcanbederivedusingproductruleandconditionalprobabilityofeventAwithknowneventB: As from

product rule we can write:

P(A∧B)=P(A|B)P(B)or

Similarly,theprobabilityofeventBwithknowneventA:

P(A∧B)=P(B|A)P(A)

Equatingrighthandsideofboththeequations,wewill get:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as **Bayes' rule** or**Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

Itshowsthesimplerelationshipbetweenjointandconditionalprobabilities.Here,

P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

P(A)iscalledthe**priorprobability**,probabilityofhypothesisbeforeconsideringtheevidence P(B) is

called **marginal probability**, pure probability of an evidence.

Intheequation(a),ingeneral,wecanwriteP(B)=P(A)*P(B|Ai),hencetheBayes'rulecanbewrittenas:

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

Where$A_1,A_2,A_3,..,A_n$isasetof mutuallyexclusiveandexhaustiveevents.

ApplyingBayes'rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$P(cause|effect) = \frac{P(effect|cause)\,P(cause)}{P(effect)}$$

Example-1:

Question:whatistheprobabilitythatapatienthasdiseasesmeningitiswithastiffneck? Given Data:

Adoctor isawarethatdiseasemeningitiscausesa patientto havea stiffneck,and itoccurs80%ofthetime. Heis also aware of some more facts, which are given as follows:

- TheKnownprobabilitythatapatienthasmeningitisdiseaseis1/30,000.
- TheKnown probabilitythatapatienthasastiffneckis2%.

Let a be the proposition that patient has stiff neck and b bethe proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b)=1/30000

P(a)= .02

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence,wecanassumethat1patientoutof750patientshasmeningitisdiseasewithastiffneck.

Example-2:

Question:From astandard deckof playing cards,asingle card is drawn.Theprobabilitythat the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$P(king|face) = \frac{P(Face|king)*P(King)}{P(Face)} \quad .....(i)$$

P(king):probabilitythatthecardisKing=4/52=1/13 P(face):

probability that a card is a face card= 3/13

P(Face|King):probabilityoffacecardwhenweassumeitisaking=1 Putting all

values in equation (i) we will get:

$$P(king|face) = \frac{1 * (\frac{1}{13})}{(\frac{3}{13})} = 1/3,$$ **it is a probability that a face card is a king card.**

ApplicationofBayes'theoreminArtificialintelligence:

FollowingaresomeapplicationsofBayes'theorem:

- Itisusedtocalculatethenextstepoftherobotwhenthealreadyexecutedstepisgiven.
- Bayes'theoremishelpfulinweatherforecasting.
- Itcan solvetheMontyHallproblem.

3. Probabilisticreasoning

Probabilisticreasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicatethe uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today,""behavior of someone for some situations,""Amatchbetween two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

NeedofprobabilisticreasoninginAI:

- Whentherareunpredictableoutcomes.
- Whenspecificationsorpossibilitiesofpredicatesbecomestoolargetohandle.
- Whenanunknownerroroccursduringan experiment.

Inprobabilisticreasoning,therearetwowaystosolveproblemswithuncertainknowledge:

- **Bayes'rule**
- **BayesianStatistics**

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

0≤P(A)≤1,whereP(A)istheprobabilityofaneventA.P(A) =

0,indicates total uncertainty in an event A.

P(A)=1,indicatestotalcertaintyinaneventA.

Wecanfindtheprobabilityofanuncertaineventbyusingthebelowformula.

$$\textbf{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- P(¬A)=probabilityofanothappeningevent.
- P(¬A)+P(A)=1.

Event: Eachpossibleoutcomeofavariableiscalledan event.

Samplespace: Thecollectionofallpossibleeventsiscalledsamplespace.

Randomvariables: Randomvariablesareusedtorepresenttheeventsandobjectsintherealworld.

Priorprobability: Thepriorprobabilityofaneventisprobabilitycomputedbeforeobservingnewinformation.

Posterior Probability: The probability that is calculated after all evidence or information has takeninto account. It isa combination of prior probability and new information.

Conditionalprobability:

Conditionalprobabilityisaprobabilityofoccurringaneventwhenanothereventhasalreadyhappened.

Let's suppose, we want to calculate the event Awhenevent B has already occurred, "the probability of Aunder the conditions of B", it can be written as:

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

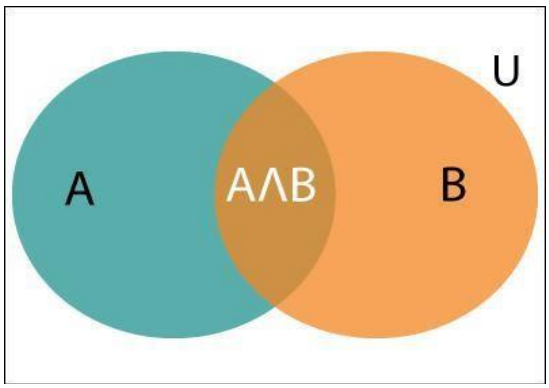
WhereP(*A*∧*B*)=JointprobabilityofaandBP(B)=

Marginal probability of B.

IftheprobabilityofA isgiven andweneedtofindtheprobabilityofB,thenitwillbegiven as:

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

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P(A∧*B*) by P(B)**.



Example:

In aclass, there are 70% ofthe students who likeEnglish and40% of the students who likesEnglish andmathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let,AisaneventthatastudentlikesMathematics B is

an event that a student likes English.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence,57%arethestudentswholikeEnglishalsolikeMathematics.

4. Bayesian networksorBelief networks

BayesianBeliefNetworkinartificialintelligence

Bayesianbeliefnetworkiskeycomputertechnologyfordealingwithprobabilisticeventsandtosolveaproblem which has uncertainty. We can define a Bayesian network as:

"ABayesiannetworkisaprobabilisticgraphicalmodelwhichrepresentsasetofvariablesandtheirconditional dependencies using a directed acyclic graph."

Itisalsocalled a**Bayesnetwork,beliefnetwork,decisionnetwork**,or**Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

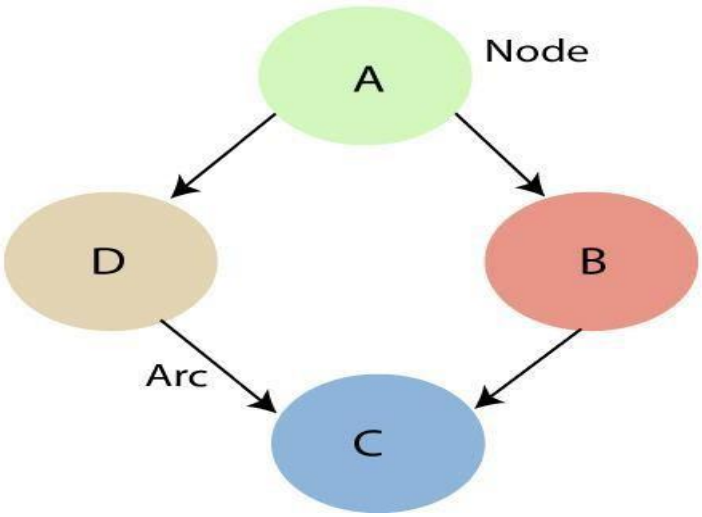
Realworldapplicationsareprobabilistic in nature,andtorepresenttherelationshipbetweenmultipleevents,weneed a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

BayesianNetworkcan beusedforbuildingmodelsfromdataandexpertsopinions,anditconsistsoftwoparts:

- **DirectedAcyclic Graph**
- **Tableofconditionalprobabilities.**

The generalized form of Bayesian network that represents and solve decision problems underuncertain knowledge is known as an **Influence diagram**.

ABayesiannetworkgraphismadeupofnodesandArcs(directedlinks),where:



- Each**node**correspondstotherandomvariables,andavariablecanbe**continuous**or**discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables.Thesedirectedlinksorarrowsconnectthepairofnodesinthegraph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
  - **In the above diagram, A, B, C, and D are random variables represented by the nodes of thenetwork graph.**
  - **If we are considering node B,which is connected with node A byadirected arrow,then node A is called the parent of Node B.**
  - **NodeCisindependentofnodeA.**

Note: The Bayesian network graph does not contain anycyclic graph. Hence, it is known as a directed acyclic graph or DAG

TheBayesiannetworkhasmainlytwocomponents:

- **CausalComponent**
- **Actualnumbers**

EachnodeintheBayesiannetworkhasconditionprobabilitydistribution **$P(X_i|Parent(X_i))$**,whichdeterminesthe effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's firstunderstand thejoint probability distribution:

Jointprobabilitydistribution:

Ifwehavevariablesx1,x2,x3,,xn,thentheprobabilitiesofadifferentcombinationofx1,x2,x3..xn,areknownas Jointprobabilitydistribution.

$P[x_1,x_2,x_3,,x_n]$,itcanbewrittenasthefollowingwayintermsofthejointprobabilitydistribution.

$=P[x_1|x_2,x_3,.....,x_n]P[x_2,x_3,,x_n]$

$=P[x_1|x_2,x_3,.....,x_n]P[x_2|x_3,.....,x_n].P[x_{n-1}|x_n]P[x_n].$

IngeneralforeachvariableXi,wecanwritethe equationas:

$$P(X_i|X_{i-1},.....,X_1)=P(X_i|Parents(X_i))$$

ExplanationofBayesiannetwork:

Let'sunderstandtheBayesiannetworkthroughanexamplebycreatingadirectedacyclic graph:

Example: Harry installed anew burglar alarmathishometodetect burglary. The alarm reliably respondsat detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at workwhen they hear the alarm. David always callsHarry whenhe hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

Calculatetheprobabilitythatalarmhassounded,butthereisneitheraburglary,noranearthquakeoccurred,andDavidandSophiabothcalledtheHarry.

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off,but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- TheconditionaldistributionsforeachnodearegivenasconditionalprobabilitiestableorCPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set ofcases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

Listofalleventsoccurringinthisnetwork:

- **Burglary(B)**
- **Earthquake(E)**
- **Alarm(A)**
- **DavidCalls(D)**

- **Sophiacalls(S)**

Wecanwritetheeventsofproblemstatementintheformofprobability: **P[D,S,A,B,E]**,canrewritetheabove probability statement using joint probability distribution:

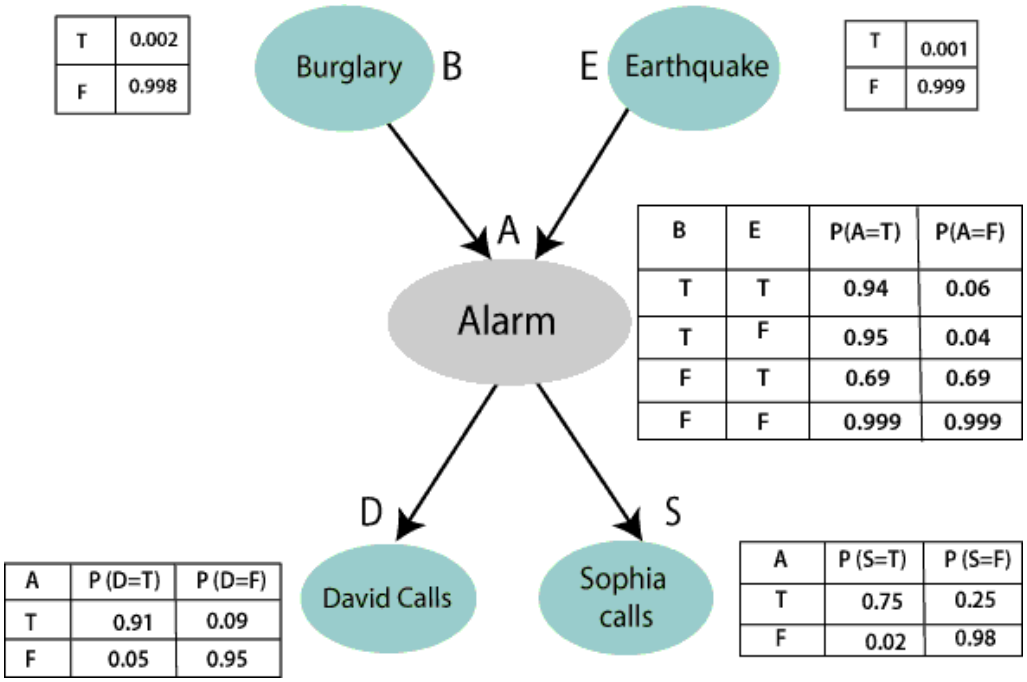
P[D,S,A,B,E]=P[D |S,A,B,E].P[S,A,B,E]

=P[D|S,A,B,E].P[S|A,B,E]. P[A,B,E]

=P[D|A].P[S|A, B,E].P[A,B,E]

=P[D|A].P[S|A]. P[A|B,E].P[B,E]

=P[D|A].P[S |A].P[A|B,E].P[B |E]. P[E]



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

Let'staketheobservedprobabilityfortheBurglaryandearthquakecomponent: P(B=

True) = 0.002, which is the probability of burglary.

P(B=False)=0.998,whichistheprobabilityofnoburglary.

P(E=True)=0.001,whichistheprobabilityofaminorearthquake

P(E=False)=0.999,Whichistheprobabilitythatanearthquakenotoccurred. We can

provide the conditional probabilities as per the below tables: **Conditional**

probability table for Alarm A:

TheConditionalprobabilityofAlarmAdependsonBurglar andearthquake:

B	E	P(A= True)	P(A=False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

ConditionalprobabilitytableforDavidCalls:

TheConditionalprobabilityofDavidthathewillcalldependson theprobabilityof Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

ConditionalprobabilitytableforSophiaCalls:

TheConditionalprobabilityofSophiathatshecallsisdependingon itsParentNode"Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

Fromtheformulaofjointdistribution,wecanwritetheproblemstatementintheformofprobabilitydistribution:

P(S,D,A,¬B,¬E)=P(S|A)*P(D|A)*P(A|¬B^¬E)*P(¬B)*P(¬E).

=0.75*0.91*0.001*0.998*0.999

=0.00068045.

Hence,aBayesiannetworkcananswer anyqueryaboutthedomainbyusing Jointdistribution. The semantics of Bayesian Network:

Therearetwowaystounderstandthesemanticsof theBayesiannetwork,whichisgiven below:

1. TounderstandthenetworkastherepresentationoftheJointprobabilitydistribution.

Itishelpfultounderstandhowtoconstructthenetwork.

2. Tounderstandthenetworkasanencodingofacollectionofconditionalindependencestatements.

Itishelpfulindesigninginference procedure.

1. InferenceinBayesianNetworks
1. Exactinference
2. Approximate inference

1. Exactinference:

Inexactinference,weanalyticallycomputetheconditionalprobabilitydistributionoverthe variables of interest.

Butsometimes,that's too hard to do, in which case we can use approximation techniques based on statistical sampling

GivenaBayesiannetwork,whatquestionsmightwewantto ask?

• Conditionalprobabilityquery:P(x|e)

• Maximumaposterioriprobability:WhatvalueofxmaximizesP(x|e)?

Generalquestion:What'sthewholeprobabilitydistributionovervariable Xgivenevidencee,P(X| e)?

Inourdiscreteprobabilitysituation,theonlywaytoanswerMAPqueryistocomputethe probability of x given e for all possible values of x and see which one is greatest

So,ingeneral,we'dliketobeabletocompute awholeprobabilitydistributionoversomevariableor variablesX,giveninstantiationsofasetofvariablese

Usingthejoint distribution

To answer any query involving a conjunction of variables,sum over the variables notinvolved in the query

Given the jointdistribution over the variables,we can easily answer any question aboutthe value of a single variable by summing (or marginalizing) over the other variables.

$$\Pr(d) = \sum_{ABC} \Pr(a,b,c,d)$$
$$= \sum_{a \in \mathrm{dom}(A)} \sum_{b \in \mathrm{dom}(B)} \sum_{c \in \mathrm{dom}(C)} \Pr(A = a \wedge B = b \wedge C = c)$$

So, in a domain with four variables, A, B, C, and D, the probability that variable D has value d is the sum over all possible combinations of values of the other three variables of the joint probability ofall four values. This is exactly the same as the procedure we went through in the last lecture, where to compute the probability of cavity, we added up the probability of cavity and toothache and the probability of cavity and not toothache.

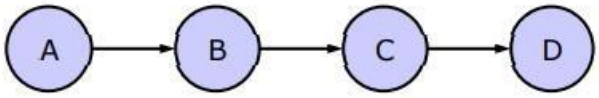
In general, we'll use the first notation, with a single summation indexed by a list of variable names, and a joint probability expression that mentions values of those variables. But here we can see the completely written-out definition, just so we all know what the shorthand is supposed to mean.

$$\Pr(d \mid b) = \frac{\Pr(b,d)}{\Pr(b)} = \frac{\sum_{AC} \Pr(a,b,c,d)}{\sum_{ACD} \Pr(a,b,c,d)}$$

To compute a conditional probability, we reduce it to a ratio of conjunctive queries using the definition of conditional probability,and then answer each of those queries by marginalizing outthe variables not mentioned.

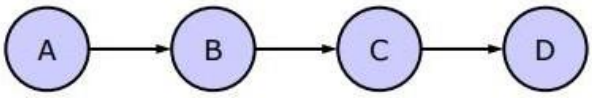
In the numerator, here, you can see that we're only summing over variables A and C,because b and d are instantiated in the query.

Simple Case



We're going tolearn a general purpose algorithm for answering these joint queries fairly efficiently. We'll start by looking at a very simple case to build up our intuitions, then we'll write down the algorithm, then we'll apply it to a more complex case.

Here'sourverysimplecase.It'sabayesnetwithfournodes,arrangedinachain.



$$\begin{aligned}
\Pr(d) &= \sum_{ABC} \Pr(a,b,c,d) \\
&= \sum_{ABC} \Pr(d \mid c)\Pr(c \mid b)\Pr(b \mid a)\Pr(a) \\
&= \sum_{C}\sum_{B}\sum_{A} \Pr(d \mid c)\Pr(c \mid b)\Pr(b \mid a)\Pr(a) \\
&= \sum_{C} \Pr(d \mid c) \sum_{B} \Pr(c \mid b) \sum_{A} \Pr(b \mid a)\Pr(a)
\end{aligned}$$

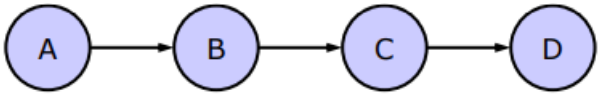
So, we know from before that the probability that variable D has some value little d is the sumover A, B, and C of the joint distribution, with d fixed.

Now, using the chain rule of Bayesian networks, we can write down the joint probability as aproduct over the nodes of the probability of each node's value given the values of its parents. So, in this case, we get P(d|c) times P(c|b) times P(b|a) times P(a).

This expression gives us a method for answering the query, given the conditional probabilities that are stored in the net. And this method can be applied directly to any other bayes net. But there's a problemwithit:itrequiresenumeratingall possiblecombinationsofassignmentstoA,B,andC, and then, for each one, multiplying the factors for each node. That's an enormous amount of work and we'd like to avoid it if at all possible.

So, we'll try rewriting the expression into something that might be more efficient to evaluate. First, we can make our summation into three separate summations, one over each variable.

Then, by distributivity of addition over multiplication, we can push the summations in, so that the sum over A includes all the terms thatmention A,but no others,and so on. It's pretty clear that this expression is the same as the previous one in value, but it can be evaluated more efficiently. We're still, eventually, enumerating all assignments to the three variables, but we're doing somewhat fewer multiplications than before. So this is still not completely satisfactory.



$$\Pr(d) = \sum_C \Pr(d \mid c) \sum_B \Pr(c \mid b) \sum_A \Pr(b \mid a) \Pr(a)$$

$$\begin{bmatrix} \Pr(b_1 \mid a_1)\Pr(a_1) & \Pr(b_1 \mid a_2)\Pr(a_2) \\ \Pr(b_2 \mid a_1)\Pr(a_1) & \Pr(b_2 \mid a_2)\Pr(a_2) \end{bmatrix}$$

Ifyoulook,for aminute,atthe termsinsidethesummationoverA,you'llseethatwe'redoingthese multiplications over for each value of C, which isn't necessary, because they're independent of C. Ouridea, here, istodo themultiplicationsonce and storethem forlater use. So,first,foreachvalue of A and B, we can compute the product, generating a two dimensional matrix.

$$\Pr(d)=\sum_{C}\Pr(d\mid c)\sum_{B}\Pr(c\mid b)\underbrace{\sum_{A}\Pr(b\mid a)\Pr(a)}_{\begin{bmatrix}\sum_{A}\Pr(b_1\mid a)\Pr(a)\\ \sum_{A}\Pr(b_2\mid a)\Pr(a)\end{bmatrix}}$$

Then,we can sum over the rows of the matrix,yielding one value of the sum for each possible value of b.

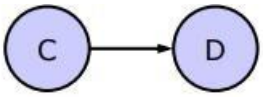
$$\Pr(d)=\sum_{C}\Pr(d\mid c)\sum_{B}\Pr(c\mid b)\underbrace{\sum_{A}\Pr(b\mid a)\Pr(a)}_{f_1(b)}$$

We'llcallthissetofvalues,whichdependsonb,f1ofb.

$$\Pr(d)=\sum_{C}\Pr(d\mid c)\underbrace{\sum_{B}\Pr(c\mid b)f_1(b)}_{f_2(c)}$$

Now, we can substitute f1 of b in for the sum over A in our previous expression.And,effectively,we can remove node A from our diagram. Now, we express the contribution of b, which takes the contribution of a into account, as f_1 of b.

We can continue the process in basically the same way. We can look at the summation over b and seethatthe only other variableitinvolvesisc.We can summarizethose productsas a set of factors, one for each value of c. We'll call those factors f_2 of c.



$$\Pr(d)=\sum_{C}\Pr(d\mid c)f_2(c)$$

Wesubstitutef_2of cintotheformula,remove nodeb from thediagram,and nowwe're down toa simpleexpressioninwhichdisknownandwehaveto sumovervaluesofc.

VariableEliminationAlgorithm

GivenaBayesiannetwork,andaneliminationorderforthenon-queryvariables,compute

$$\sum_{X_1}\sum_{X_2}\cdots\sum_{X_m}\prod_{j}\Pr(x_j \mid Pa(x_j))$$

Fori=mdownto 1

- removeallthefactorsthatmentionXi
- multiplythosefactors,gettingavalueforeachcombinationofmentionedvariables
- sumoverXi
- putthisnewfactorintothefactorset

That was a simple special case. Now we can look at the algorithm in the general case. Let's assume that we're given a Bayesian network and an ordering on the variables that aren't fixed in the query. We'll come back later to the question of the influence of the order, and how we might find a good one.

We can express the probability of the query variables as a sum over each value of each of the non-query variables of a product over each node in the network, of the probability that that variable has the given value given the values of its parents.

So,we'lleliminatethevariablesfromtheinsideout.StartingwithvariableXmandfinishingwith variableX1.

ToeliminatevariableXi,westartbygatheringupallofthefactorsthatmentionXi,andremoving themfromoursetof factors.Let'ssay thereareksuchfactors.

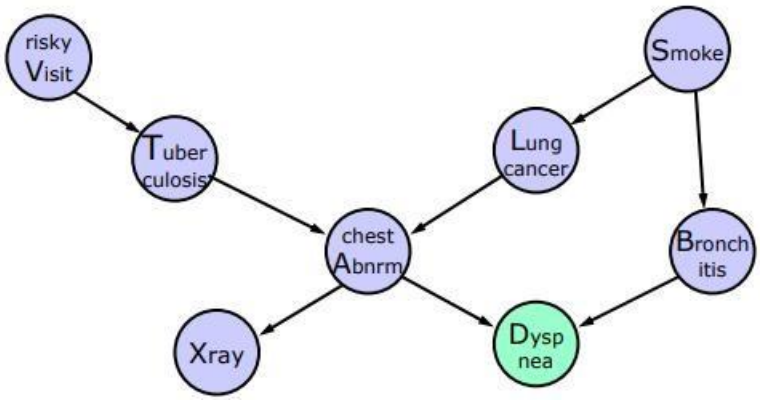
Now, we make a k+1 dimensional table, indexed by Xi as well as each of the other variables that is mentioned in our set of factors.

Wethensumthe tableovertheXidimension,resultinginak-dimensionaltable.

Thistableisournewfactor,and weputa termforit backintoour setof factors. Once

we've eliminated all the summations, we have the desired value.

Onemoreexample



$$\Pr(d) = \sum_{A,B,L,T} \Pr(d \mid a,b)\Pr(a \mid t,l) f_1(t) \underbrace{\sum_{S} \Pr(b \mid s)\Pr(l \mid s)\Pr(s)}$$

$$\Pr(d) = \sum_{A,B,L,T} \Pr(d \mid a,b)\Pr(a \mid t,l) f_1(t) \underbrace{\sum_{S} \Pr(b \mid s)\Pr(l \mid s)\Pr(s)}_{f_2(b,l)}$$

$$\Pr(d) = \sum_{A,B,L,T} \Pr(d \mid a,b)\Pr(a \mid t,l) f_1(t) f_2(b,l)$$

$$\Pr(d) = \sum_{A,B,L} \Pr(d \mid a,b) f_1(b,l) \underbrace{\sum_{T} \Pr(a \mid t,l) f_1(t)}_{f_3(a,l)}$$

$$\Pr(d) = \sum_{A,B,L} \Pr(d \mid a,b) f_2(b,l) f_3(a,l)$$

Here's a more complicated example, to illustrate the variable elimination algorithm in a more general case. We have this big network that encodes a domain for diagnosing lung disease. (Dyspnea, as I understand it, is shortness of breath).

We'lldovariableeliminationonthisgraphusingeliminationorderA,B,L,T,S,X, V.

So, we start by eliminating V. We gather the two terms that mention V and see that they alsoinvolve variable T. So, we compute the product for each value of T, and summarize those in the factor f1 of T.

Nowwecansubstitutethatfactor inforthe summation,andremovethenodefromthenetwork.

The next variable to be eliminated is X. There is actually only one term involving X, and it also involves variable A. So, for each value of A, we compute the sum over X of P(x|a). But wait! Weknow what this value is! If we fix a and sum over x, these probabilities have to add up to 1.

So, rather than adding another factor to our expression, we can just remove the whole sum. In general, the only nodes that will have an influence on the probability of D are its ancestors.

Now, it's time to eliminate S. We find that there are three terms involving S, and we gather them into the sum. These three terms involve two other variables, B and L. So we have to make a factor that specifies, for each value of B and L, the value of the sum of products.

We'llcallthatfactorf_2ofband l.

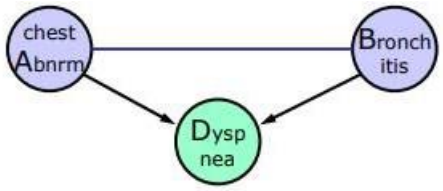
Now we can substitute that factor back into our expression. We can also eliminate node S. But in eliminating S, we've added a direct dependency between L and B (they used to be dependent via S, butnow thedependencyisencodedexplicitlyinf2(b).We'llshowthatinthe graphbydrawingaline between the two nodes. It's not exactly a standard directed conditional dependence, but it's still useful to show that they're coupled.

Now we eliminate T. It involves two terms, which themselves involve variables A and L. So we make a new factor f3 of A and L.

WecansubstituteinthatfactorandeliminateT.We'regettingclose!

$$\Pr(d) = \sum_{A,B} \Pr(d \mid a,b) \underbrace{\sum_{L} f_2(b,l) f_3(a,l)}_{f_4(a,b)}$$

Nextwe eliminate L.Itinvolves thesetwo factors,which depend on variablesA and B. Sowemake a new factor, f4 of A and B, and substitute it in. We remove node L, but couple A and B.



$$\Pr(d) = \sum_{A,B} \Pr(d \mid a,b) f_4(a,b)$$

Atthispoint,wecouldjustdothesummationsoverAandBandbedone.Buttofinishoutthe algorithmthewayacomputerwould,it'stimetoeliminatevariableB.

$$\Pr(d) = \sum_{A} \underbrace{\sum_{B} \Pr(d \mid a,b) f_4(a,b)}_{f_5(a)}$$

It involves both of our remaining terms, and it seems to depend on variables A and D. However, in this case, we're interested in the probability of a particular value, little d of D, and so the variable dis instantiated. Thus,we cantreatitas a constantin this expression,and we only need togenerate a factor over a, which we'll call f5 of a. And we can now, in some sense, remove D from our networkas well (because we've already factored it into our answer).

$$\Pr(d) = \sum_{A} f_5(a)$$

Finally, to get the probability that variable D has value little d, we simply sum factor f5 over allvalues of a. Yay! We did it.

PropertiesofVariableElimination

Let'sseehowthevariableeliminationalgorithmperforms,bothintheoryandinpractice.

- Timeisexponentialinsizeoflargestfactor
- Badeliminationordercangeneratehugefactors
- NPHardtofindthebestelimination order

- Eventhebesteliminationordermaygeneratelargefactors
- Thereareareasonableheuristicsforpickinganeliminationorder(suchaschoosingthe variable that results in the smallest next factor)
- Inferenceinpolytrees(netswithnocycles)islinearinsizeofthenetwork(thelargestCPT)
- Manyproblemswithverylargenetshaveonlysmallfactors,andthusefficientinference

First of all, it's pretty easy to see that it runs in time exponential in the number of variables involved in the largestfactor. Creating a factor with k variablesinvolvesmaking a k+1dimensional table. If you have b values per variable, that's a table of size b^(k+1). To make each entry, you have to multiply at most n numbers, where n is the number of nodes. We have to do this for each variable to be eliminated (which is usually close to n). So we have something like time = O(n^2 b^k).

How big the factors are depends on the elimination order. You'll see in one of the recitation exercises just how dramatic the difference in factor sizes can be. A bad elimination order can generate huge factors.

So,we'd like touse the elimination order thatgenerates the smallestfactors. Unfortunately,itturns out to be NP hard to find the best elimination order.

At least, there are some fairly reasonable heuristics for choosing an elimination order. It's usually done dynamically. So, rather than fixing the elimination order in advance, as we suggested in the algorithm description, you can pick the next variable to be eliminated depending on the situation. In particular,one reasonable heuristic is to pick the variable to eliminate next that will result in the smallest factor. This greedy approach won't always be optimal, but it's not usually too bad.

There is one case where Bayes net inference in general, and the variable elimination algorithm in particular is fairly efficient, and that's when the network is a polytree. A polytree is a network with no cycles. That is, a network in which, for any two nodes, there is only one path between them. In a polytree,inferenceislinearinthesizeof thenetwork, wherethesizeof thenetwork isdefinedto be the size of the largest conditional probability table (or exponential in the maximum number of parents ofany node). In a polytree, the optimal elimination order is to start at the root nodes, and work downwards, always eliminating a variable that no longer has any parents. In doing so, we never introduce additional connections into the network.

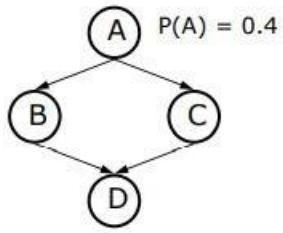
So,inferenceinpolytreesisefficient,andeveninmanylargenon-polytreenetworks,it'spossibleto keepthefactorssmall,andthereforetodoinferencerelativelyefficiently.

Whenthenetworkissuchthatthefactorsare,ofnecessity,large,we'llhavetoturntoadifferentclass ofmethods.

2. Approximateinference:

Sampling

Togetapproximateanswerwecandostochasticsimulation(sampling).



A	B	C	D
T	F	T	T
...			

- Flip a coin where P(T)=0.4, assume we get T, use that value for A
- Given A=T, lookup P(B|A=T) and flip a coin with that prob., assume we get F
- Similarly for C and D
- We get one sample from joint distribution of these four vars

Another strategy, which is a theme that comes up also more and more in AI actually, is to say, well, we didn'treallywanttherightanswer anyway.Let'stry to doan approximation.Andyou canalsoshowthat it's computationally hard to get an approximation that's within epsilon of the answer that you want, but again that doesn't keep us from trying.

So, the other thing that we can do is the stochastic simulation or sampling. In sampling, what we do iswelookattherootnodesofourgraph,and attached tothisroot nodeissomeprobabilitythat Aisgoing to be true, right? Maybe it's .4. So we flip a coin that comes up heads with probability .4 and see if we get true or false.

We flip our coin,let's say,and weget true for A -- thistime.And now, given the assignmentof true to A, we look in the conditional probability table for B given A = true, and that gives us a probability for B.

Now,weflipacoinwiththatprobability.SaywegetFalse.Weenterthatintothetable.

WedothesamethingforC,andlet'ssaywegetTrue.

Now,welookintheCPTfor DgivenBandC,forthe casewhereBisfalseandC istrue,and weflipacoin with that probability, in order to get a value for D.

So, there's one sample from the joint distribution of these four variables. And you can just keep doing this, all day and all night, and generate a big pile of samples, using that algorithm. And now you can ask various questions.

Estimate:

P*(D|A)=#D,A/#A

Let's say you want to know the probability of D given A. How would you answer- - given all the examples -- what would you do to compute the probability of D given A? You would just count. You'd count the number of cases in which A and D were true, and you'd divide that by the number of cases in

whichAwastrue,andthatwouldgiveyouanunbiasedestimateoftheprobabilityofDgivenA.The moresamples,themoreconfidenceyou'dhavethattheestimatedprobabilityisclosetothetrueone.

Estimation

- Someprobabilitiesareeasierthanothersto estimate
- Ingeneratingthetable,therareeventswillnotbewellrepresented
- P(Disease|spots-on-your-tongue,soretoe)
- If spots-on-your-tongue and sore toe are not root nodes, you would generate a huge table but the cases of interest would be very sparse in the table
- Importancesamplingletsyoufocusontheset ofcasesthatareimportanttoansweringyour question

It'sgoingtoturnoutthatsomeprobabilitiesareeasierthanotherones toestimate.

Exactly because of the process we're using to generate the samples, the majority of them will be the typical cases. Oh, it's someone with a cold, someone with a cold, someone with a cold, someone with a cold, someone with a cold, someone with malaria, someone with a cold, someone with a cold. So the rare results are not going to come up very often. And so doing this sampling naively can make it really hard to estimate the probability of a rare event. If it's something that happens one in ten thousand times, well, you know for sure you're going to need, some number of tens of thousands of samples toget even a reasonable estimate of that probability.

Imagine that you want to estimate the probability of some disease given -- oh, I don't know -- spots on your tongue and a sore toe. Somebody walks in andthey have a really peculiar set of symptoms, andyou want to know what's the probability that they have some disease.

Well, if the symptoms are root nodes, it's easy. If the symptoms were root nodes, you could just assign the root nodes to have their observed values and then simulate the rest of the network as before.

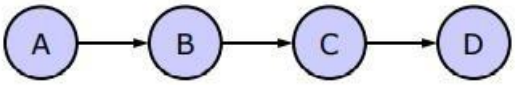
But if the symptoms aren't root nodes then if you do naïve sampling, you would generate a giant tableof samples, and you'd have to go and look and say, gosh, how many cases do I have where somebody has spots on their tongue and a sore toe; and the answer would be, well, maybe zero or not very many.

There's a technique called importance sampling, which allows you to draw examples from a distribution that's going tobe more helpful and then reweightthem sothatyou canstill getanunbiased estimate of thedesiredconditionalprobability.It'sa bit beyond thescope of this classtoget into the details,but it's an important and effective idea.

RecitationProblem

• Dothevariableeliminationalgorithmonthenetbelowusingtheeliminationorder A,B,C(thatis, eliminate node C first). In computing P(D=d), what factors do you get?

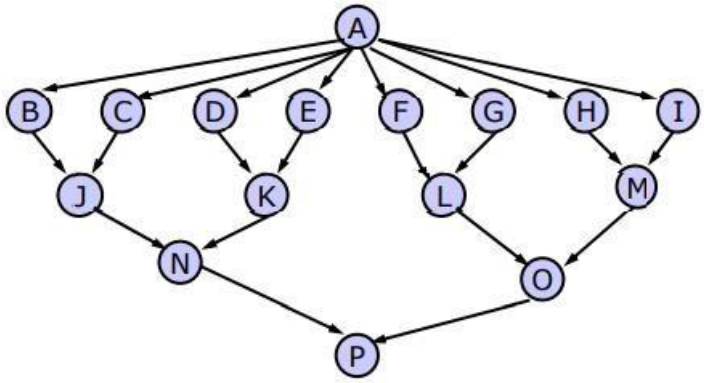
• WhatifyouwantedtocomputethewholemarginaldistributionP(D)?



Here's the network we started with. We used elimination order C, B, A (we eliminated A first). Now we're going to explore what happens when we eliminate the variables in the opposite order. First, work on the case we did, where we're trying to calculate the probability that node D takes on a particular value, little d. Remember that little d is a constant in this case. Now, do the case where we're trying to find the whole distribution over D, so we don't know a particular value for little d.

AnotherRecitationProblem

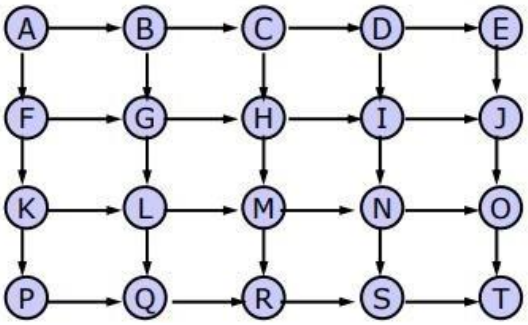
Find an elimination order that keeps the factors small for the net below, or show that there is no such order.



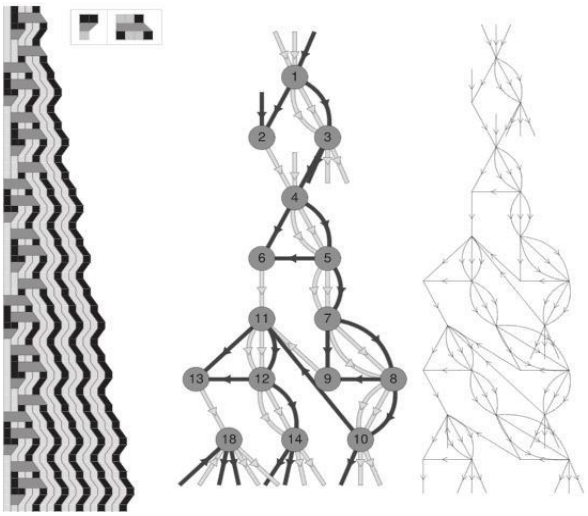
Here's a pretty complicated graph. But notice that no node has more than 2 parents, so none of theCPTsarehuge.The questionis,is thisgraph hardfor variableelimination?More concretely,can you find an elimination order thatresults only in fairly small factors? Is there an elimination order thatgenerates a huge factor?

TheLastRecitationProblem

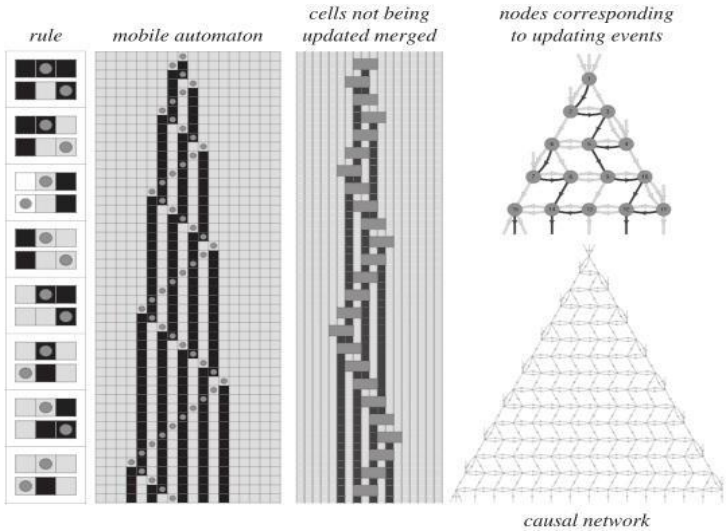
Bayesian networks (or related models) are often used in computer vision, but they almost alwaysrequiresampling.Whathappenswhenyoutry todovariableeliminationonamodel likethegridbelow?



6. CasualNetworks:



Acausalnetworkisan[acyclicdigraph](#)arisingfromanevolutionofa [substitutionsystem](#),and representingitshistory.Theillustrationaboveshowsacausalnetworkcorrespondingtotherules $\{B\ B \to A,\ A\ A\ B \to B\ A\ A\ B\}$ (appliedina left-to-right scan)andinitialcondition $A\ B\ A\ A\ B$.



Thefigureaboveshowstheprocedurefordiagrammaticallycreatingacausalnetworkfro m a [mobile automaton](#).

In an evolution [of a multiway system](#), each substitution event is a vertex in a causal network. Two events which are related by causal dependence, meaning one occurs just before the other, haveanedgebetweenthecorresponding vertices inthecausalnetwork.Moreprecisely, theedge is a directed edge leading from the past event to the future event.

What is Machine Learning

In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning**.

Machine Learning is said as a subset of **artificial intelligence** that is mainly concerned with the development of algorithms which allow a computer to learn from the data and past experiences on their own. The term machine learning was first introduced by **Arthur Samuel** in **1959**. We can define it in a summarized way as:

> Machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed.

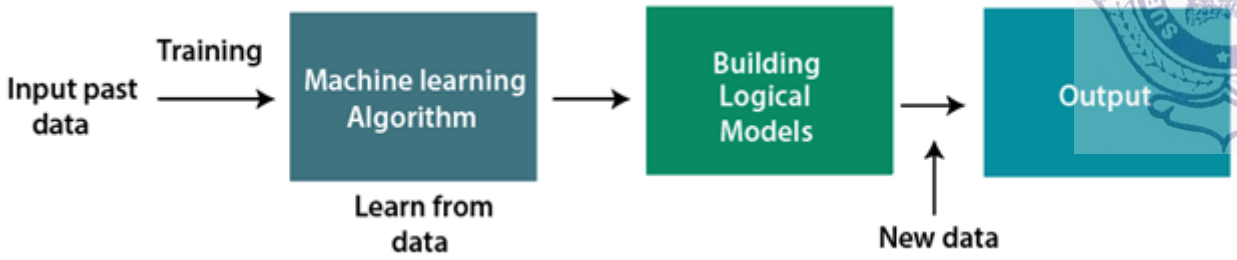
With the help of sample historical data, which is known as **training data**, machine learning algorithms build a **mathematical model** that helps in making predictions or decisions without being explicitly programmed. Machine learning brings computer science and statistics together for creating predictive models. Machine learning constructs or uses the algorithms that learn from historical data. The more we will provide the information, the higher will be the performance.

A machine has the ability to learn if it can improve its performance by gaining more data.

How does Machine Learning work

A Machine Learning system **learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it**. The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.

Suppose we have a complex problem, where we need to perform some predictions, so instead of writing a code for it, we just need to feed the data to generic algorithms, and with the help of these algorithms, machine builds the logic as per the data and predict the output. Machine learning has changed our way of thinking about the problem. The below block diagram explains the working of Machine Learning algorithm:



Features of Machine Learning:

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

Need for Machine Learning

The need for machine learning is increasing day by day. The reason behind the need for machine learning is that it is capable of doing tasks that are too complex for a person to implement directly. As a human, we have some limitations as we cannot access the huge amount of data manually, so for this, we need some computer systems and here comes the machine learning to make things easy for us.

We can train machine learning algorithms by providing them the huge amount of data and let them explore the data, construct the models, and predict the required output automatically. The performance of the machine learning algorithm depends on the amount of data, and it can be determined by the cost function. With the help of machine learning, we can save both time and money.

The importance of machine learning can be easily understood by its uses cases, Currently, machine learning is used in **self-driving cars**, **cyber fraud detection**, **face recognition**, and **friend suggestion by Facebook**, etc. Various top companies such as Netflix and Amazon have build machine learning models that are using a vast amount of data to analyze the user interest and recommend product accordingly.

Following are some key points which show the importance of Machine Learning:

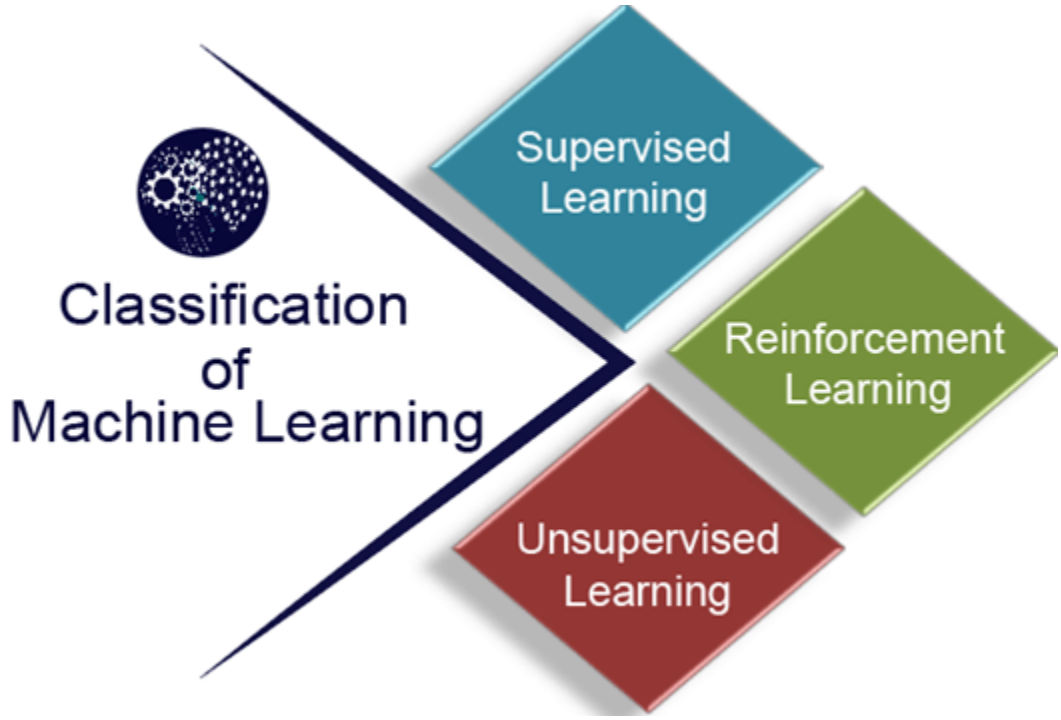
- Rapid increment in the production of data
- Solving complex problems, which are difficult for a human
- Decision making in various sector including finance

- Finding hidden patterns and extracting useful information from data.

CLASSIFICATION OF MACHINE LEARNING

At a broad level, machine learning can be classified into three types:

1. **Supervised learning**
2. **Unsupervised learning**
3. **Reinforcement learning**



1) Supervised Learning

Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.

The system creates a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing a sample data to check whether it is predicting the exact output or not.

The goal of supervised learning is to map input data with the output data. The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning is **spam filtering**.

Supervised learning can be grouped further in two categories of algorithms:

- **Classification**
- **Regression**

2) Unsupervised Learning

Unsupervised learning is a learning method in which a machine learns without any supervision.

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.

In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data. It can be further classifieds into two categories of algorithms:

- **Clustering**
- **Density Estimation**
- **Dimensionality Reduction**

3) Reinforcement Learning

Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.

The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning.

APPLICATIONS OF MACHINE LEARNING

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:

Image Recognition: Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, Automatic friend tagging suggestion:

Facebook provides us a feature of auto friend tagging suggestion. Whenever we upload a photo with our Facebook friends, then we automatically get a tagging suggestion with name, and the technology behind this is machine learning's face detection and recognition algorithm.

It is based on the Facebook project named "Deep Face," which is responsible for face recognition and person identification in the picture.

Speech Recognition: While using Google, we get an option of "Search by voice," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "Speech to text", or "Computer speech recognition." At present, machine learning algorithms are widely used by various applications of speech recognition. Google assistant, Siri, Cortana, and Alexa are using speech recognition technology to follow the voice instructions.

Traffic prediction: If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow-moving, or heavily congested with the help of two ways:

- Real Time location of the vehicle form Google Map app and sensors
- Average time has taken on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

Product recommendations: Machine learning is widely used by various e-commerce and entertainment companies such as Amazon, Netflix, etc., for product recommendation to the user. Whenever we search for some product on Amazon, then we started getting an advertisement for the same product while internet surfing on the same browser and this is because of machine learning.

Google understands the user interest using various machine learning algorithms and suggests the product as per customer interest.

As similar, when we use Netflix, we find some recommendations for entertainment series, movies, etc., and this is also done with the help of machine learning.

Self-driving cars: One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

Email Spam and Malware Filtering: Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- Content Filter
- Header filter
- General blacklists filter
- Rules-based filters
- Permission filters

ISSUES IN MACHINE LEARNING.

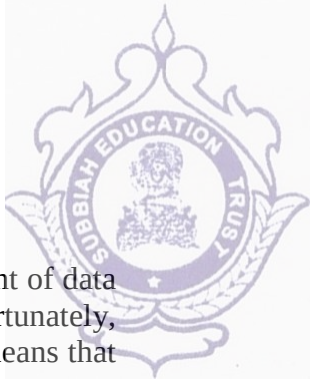
a. Poor Quality of Data

Data plays a significant role in the machine learning process. One of the significant issues that machine learning professionals face is the absence of good quality data. Unclean and noisy data can make the whole process extremely exhausting. We don't want our algorithm to make inaccurate or faulty predictions. Hence the quality of data is essential to enhance the output. Therefore, we need to ensure that the process of data preprocessing which includes removing outliers, filtering missing values, and removing unwanted features, is done with the utmost level of perfection.

b. Underfitting of Training Data

This process occurs when data is unable to establish an accurate relationship between input and output variables. It simply means trying to fit in undersized jeans. It signifies the data is too simple to establish a precise relationship. To overcome this issue:

- *Maximize the training time*
- *Enhance the complexity of the model*
- *Add more features to the data*
- *Reduce regular parameters*
- *Increasing the training time of model*



c. Overfitting of Training Data

Overfitting refers to a machine learning model trained with a massive amount of data that negatively affect its performance. It is like trying to fit in Oversized jeans. Unfortunately, this is one of the significant issues faced by machine learning professionals. This means that the algorithm is trained with noisy and biased data, which will affect its overall performance. Let's understand this with the help of an example. Let's consider a model trained to differentiate between a cat, a rabbit, a dog, and a tiger. The training data contains 1000 cats, 1000 dogs, 1000 tigers, and 4000 Rabbits. Then there is a considerable probability that it will identify the cat as a rabbit. In this example, we had a vast amount of data, but it was biased; hence the prediction was negatively affected.

It can be solved with:

- *Analyzing the data with the utmost level of perfection*
- *Use data augmentation technique*
- *Remove outliers in the training set*
- *Select a model with lesser features*

d. Machine Learning is a Complex Process

The machine learning industry is young and is continuously changing. Rapid hit and trial experiments are being carried on. The process is transforming, and hence there are high chances of error which makes the learning complex. It includes analyzing the data, removing data bias, training data, applying complex mathematical calculations, and a lot more. Hence it is a really complicated process which is another big challenge for Machine learning professionals.

e. Lack of Training Data

The most important task you need to do in the machine learning process is to train the data to achieve an accurate output. Less amount training data will produce inaccurate or too biased predictions. A machine-learning algorithm needs a lot of data to distinguish. For complex problems, it may even require millions of data to be trained. Therefore we need to ensure that Machine learning algorithms are trained with sufficient amounts of data.

f. Slow Implementation

This is one of the common issues faced by machine learning professionals. The machine learning models are highly efficient in providing accurate results, but it takes a tremendous amount of time. Slow programs, data overload, and excessive requirements usually take a lot of time to provide accurate results. Further, it requires constant monitoring and maintenance to deliver the best output.

g. Imperfections in the Algorithm When Data Grows

So you have found quality data, trained it amazingly, and the predictions are really concise and accurate. The best model of the present may become inaccurate in the coming Future and require further rearrangement. So we need regular monitoring and maintenance to keep the algorithm working. This is one of the most exhausting issues faced by machine learning.

MACHINE LEARNING WORKFLOW | PROCESS STEPS

We have discussed-

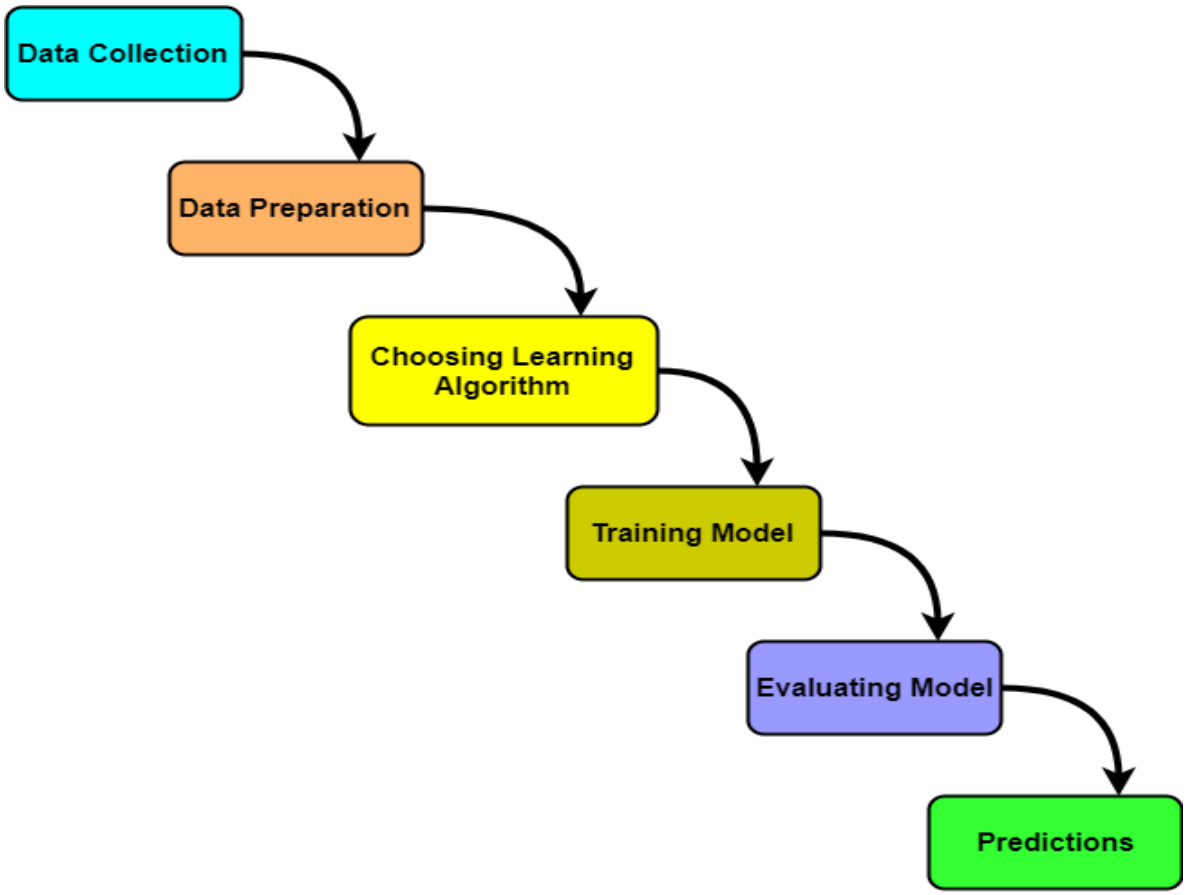
- Machine learning is building machines that can adapt and learn from experience.
- Machine learning systems are not explicitly programmed.

In this article, we will discuss machine learning workflow.

Machine Learning Workflow

Machine learning workflow refers to the series of stages or steps involved in the process of building a successful machine learning system.

The various stages involved in the machine learning workflow are-



Machine Learning Workflow

1. Data Collection
2. Data Preparation
3. Choosing Learning Algorithm
4. Training Model
5. Evaluating Model
6. Predictions

Let us discuss each stage one by one.

1. Data Collection-

In this stage,

- Data is collected from different sources.
- The type of data collected depends upon the type of desired project.
- Data may be collected from various sources such as files, databases etc.
- The quality and quantity of gathered data directly affects the accuracy of the desired system.

2. Data Preparation-

In this stage,

- Data preparation is done to clean the raw data.
- Data collected from the real world is transformed to a clean dataset.
- Raw data may contain missing values, inconsistent values, duplicate instances etc.
- So, raw data cannot be directly used for building a model.

Different methods of cleaning the dataset are-

- Ignoring the missing values
- Removing instances having missing values from the dataset.
- Estimating the missing values of instances using mean, median or mode.
- Removing duplicate instances from the dataset.
- Normalizing the data in the dataset.

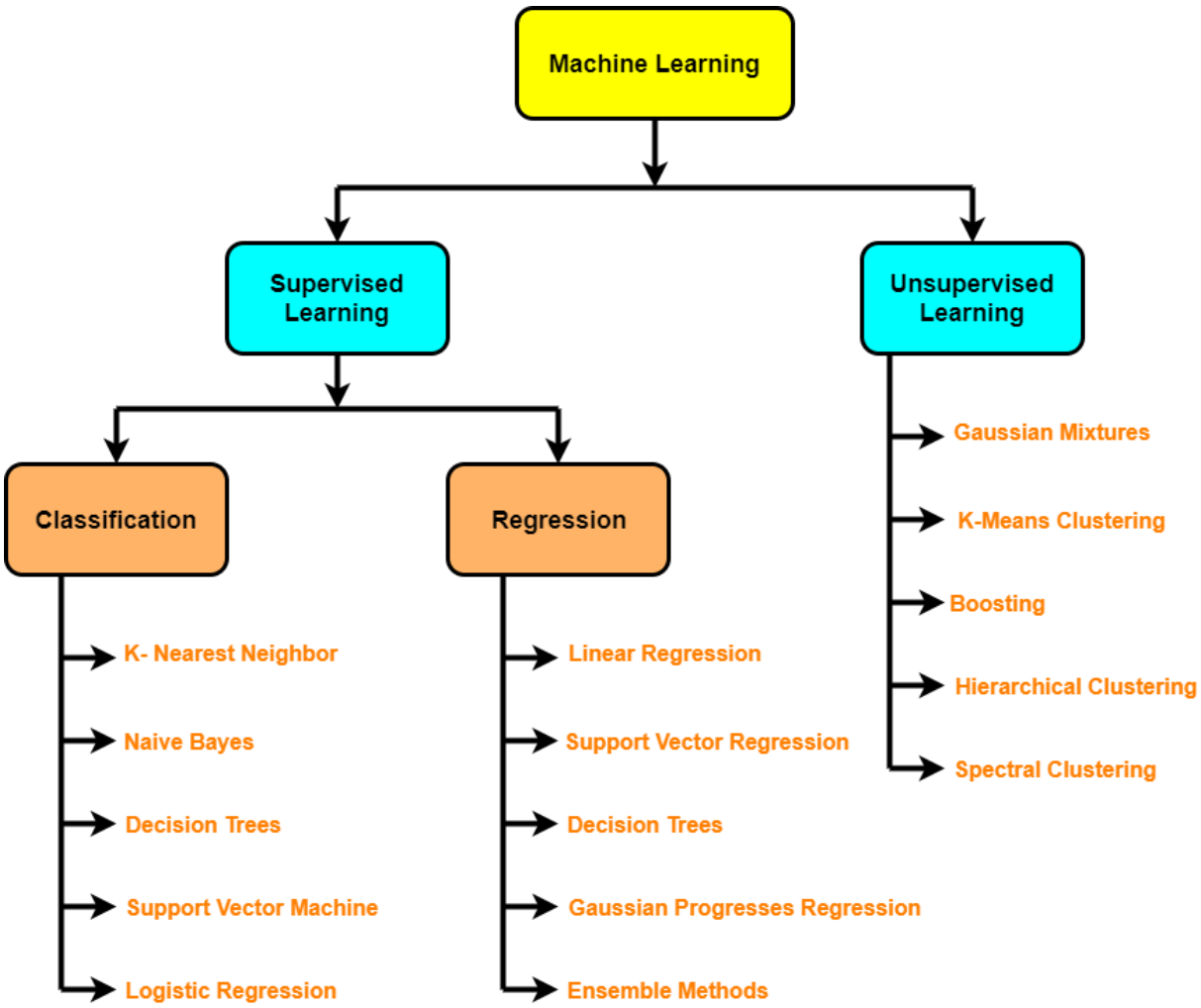
This is the most time consuming stage in machine learning workflow.

3. Choosing Learning Algorithm-

In this stage,

- The best performing learning algorithm is researched.
- It depends upon the type of problem that needs to solved and the type of data we have.
- If the problem is to classify and the data is labeled, classification algorithms are used.
- If the problem is to perform a regression task and the data is labeled, regression algorithms are used.
- If the problem is to create clusters and the data is unlabeled, clustering algorithms are used.

The following chart provides the overview of learning algorithms-



4. Training Model-

In this stage,

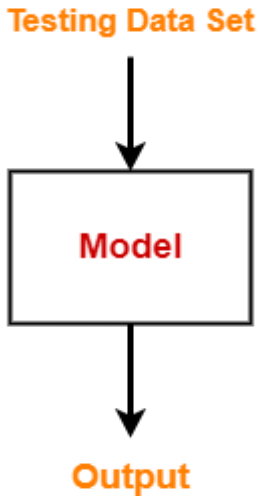
- The model is trained to improve its ability.
- The dataset is divided into training dataset and testing dataset.
- The training and testing split is order of 80/20 or 70/30.

- It also depends upon the size of the dataset.
- Training dataset is used for training purpose.
- Testing dataset is used for the testing purpose.
- Training dataset is fed to the learning algorithm.
- The learning algorithm finds a mapping between the input and the output and generates the model.

5. Evaluating Model-

In this stage,

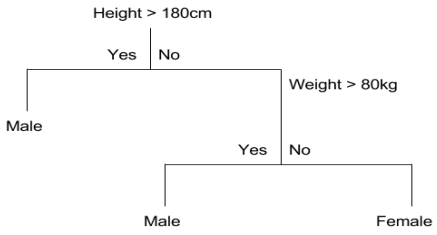
- The model is evaluated to test if the model is any good.
- The model is evaluated using the kept-aside testing dataset.
- It allows to test the model against data that has never been used before for training.
- Metrics such as accuracy, precision, recall etc are used to test the performance.
- If the model does not perform well, the model is re-built using different hyper parameters.
- The accuracy may be further improved by tuning the hyper parameters.



6. Predictions-

In this stage,

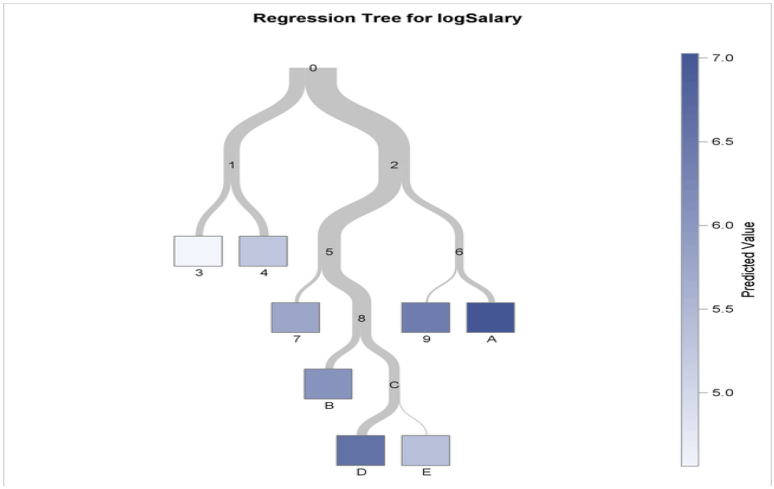
- The built system is finally used to do something useful in the real world.
- Here, the true value of machine learning is realized.



2.1.2.2. Regression Trees

A regression tree refers to an algorithm where the target variable is and the algorithm is used to predict it's value. As an example of a regression type problem, you may want to predict the selling prices of a residential house, which is a continuous dependent variable.

This will depend on both continuous factors like square footage as well as categorical factors like the style of home, area in which the property is located and so on.



When to use Classification and Regression Trees

Classification trees are used when the dataset needs to be split into classes which belong to the response variable. In many cases, the classes Yes or No.

In other words, they are just two and mutually exclusive. In some cases, there may be more than two classes in which case a variant of the classification tree algorithm is used.

Regression trees, on the other hand, are used when the response variable is continuous. For instance, if the response variable is something like the price of a property or the temperature of the day, a regression tree is used.

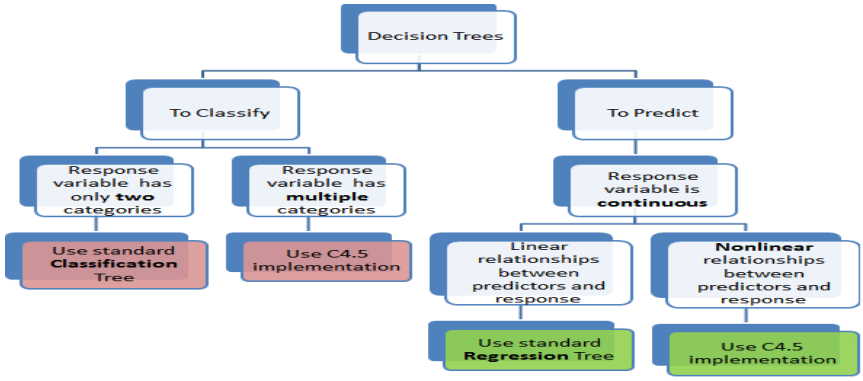
In other words, regression trees are used for prediction-type problems while classification trees are used for classification-type problems.

How Classification and Regression Trees Work

A classification tree splits the dataset based on the homogeneity of data. Say, for instance, there are two variables; income and age; which determine whether or not a consumer will buy a particular kind of phone.

If the training data shows that 95% of people who are older than 30 bought the phone, the data gets split there and age becomes a top node in the tree. This split makes the data "95% pure". Measures of impurity like entropy or Gini index are used to quantify the homogeneity of the data when it comes to classification trees.

In a regression tree, a regression model is fit to the target variable using each of the independent variables. After this, the data is split at several points for each independent variable.
At each such point, the error between the predicted values and actual values is squared to get "A Sum of Squared Errors" (SSE). The SSE is compared across the variables and the variable or point which has the lowest SSE is chosen as the split point. This process is continued recursively.



Advantages of Classification and Regression Trees

The purpose of the analysis conducted by any classification or regression tree is to create a set of if-else conditions that allow for the accurate prediction or classification of a case.

(i) The Results are Simplistic

The interpretation of results summarized in classification or regression trees is usually fairly simple. The simplicity of results helps in the following ways.

- It allows for the rapid classification of new observations. That's because it is much simpler to evaluate just one or two logical conditions than to compute scores using complex nonlinear equations for each group.
- It can often result in a simpler model which explains why the observations are either classified or predicted in a certain way. For instance, business problems are much easier to explain with if-then statements than with complex nonlinear equations.

(ii) Classification and Regression Trees are Nonparametric & Nonlinear

The results from classification and regression trees can be summarized in simplistic if-then conditions. This negates the need for the following implicit assumptions.

- The predictor variables and the dependent variable are linear.
- The predictor variables and the dependent variable follow some specific nonlinear link function.
- The predictor variables and the dependent variable are monotonic.

Since there is no need for such implicit assumptions, classification and regression tree methods are well suited to data mining. This is because there is very little knowledge or assumptions that can be made beforehand about how the different variables are related.
As a result, classification and regression trees can actually reveal relationships between these variables that would not have been possible using other techniques.

(iii) Classification and Regression Trees Implicitly Perform Feature Selection

Feature selection or variable screening is an important part of analytics. When we use decision trees, the top few nodes on which the tree is split are the most important variables within the set. As a result, feature selection gets performed automatically and we don't need to do it again.

Limitations of Classification and Regression Trees

Classification and regression tree tutorials, as well as classification and regression tree ppts, exist in abundance. This is a testament to the popularity of these decision trees and how frequently they are used. However, these decision trees are not without their disadvantages.

There are many classification and regression trees examples where the use of a decision tree has not led to the optimal result. Here are some of the limitations of classification and regression trees.

(i) Overfitting

Overfitting occurs when the tree takes into account a lot of noise that exists in the data and comes up with an inaccurate result.

(ii) High variance

In this case, a small variance in the data can lead to a very high variance in the prediction, thereby affecting the stability of the outcome.

(iii) Low bias

A decision tree that is very complex usually has a low bias. This makes it very difficult for the model to incorporate any new data.

What is a CART in Machine Learning?

A Classification and Regression Tree (CART) is a predictive algorithm used in machine learning. It explains how a target variable's values can be predicted based on other values.

It is a decision tree where each fork is a split in a predictor variable and each node at the end has a prediction for the target variable.

The CART algorithm is an important decision tree algorithm that lies at the foundation of machine learning. Moreover, it is also the basis for other powerful machine learning algorithms like bagged decision trees, random forest and boosted decision trees.

Summing up

The Classification and regression tree (CART) methodology is one of the oldest and most fundamental algorithms. It is used to predict outcomes based on certain predictor variables.

They are excellent for data mining tasks because they require very little data pre-processing. Decision tree models are easy to understand and implement which gives them a strong advantage when compared to other analytical models.

2.2. Regression

Regression Analysis in Machine learning

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as **temperature, age, salary, price,** etc.

We can understand the concept of regression analysis using the below example:

Example: Suppose there is a marketing company A, who does various advertisement every year and get sales on that. The below list shows the advertisement made by the company in the last 5 years and the corresponding sales:

Advertisement	Sales
$90	$1000
$120	$1300
$150	$1800
$100	$1200
$130	$1380
$200	??

Now, the company wants to do the advertisement of $200 in the year 2019 **and wants to know the prediction about the sales for this year**. So to solve such type of prediction problems in machine learning, we need regression analysis.

Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables. It is mainly used for **prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables**.

In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data. In simple words, ***"Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum."*** The distance between datapoints and line tells whether a model has captured a strong relationship or not.

Some examples of regression can be as:

- Prediction of rain using temperature and other factors
- Determining Market trends
- Prediction of road accidents due to rash driving.

Terminologies Related to the Regression Analysis:

- **Dependent Variable:** The main factor in Regression analysis which we want to predict or understand is called the dependent variable. It is also called **target variable**.
- **Independent Variable:** The factors which affect the dependent variables or which are used to predict the values of the dependent variables are called independent variable, also called as a **predictor**.
- **Outliers:** Outlier is an observation which contains either very low value or very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.
- **Multicollinearity:** If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.
- **Underfitting and Overfitting:** If our algorithm works well with the training dataset but not well with test dataset, then such problem is called **Overfitting**. And if our algorithm does not perform well even with training dataset, then such problem is called **underfitting**.

Why do we use Regression Analysis?

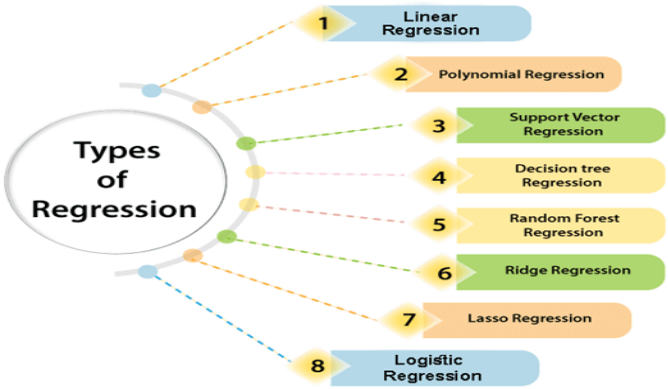
As mentioned above, Regression analysis helps in the prediction of a continuous variable. There are various scenarios in the real world where we need some future predictions such as weather condition, sales prediction, marketing trends, etc., for such case we need some technology which can make predictions more accurately. So for such case we need Regression analysis which is a statistical method and used in machine learning and data science. Below are some other reasons for using Regression analysis:

- Regression estimates the relationship between the target and the independent variable.
- It is used to find the trends in data.
- It helps to predict real/continuous values.
- By performing the regression, we can confidently determine the **most important factor, the least important factor, and how each factor is affecting the other factors**.

Types of Regression

There are various types of regressions which are used in data science and machine learning. Each type has its own importance on different scenarios, but at the core, all the regression methods analyze the effect of the independent variable on dependent variables. Here we are discussing some important types of regression which are given below:

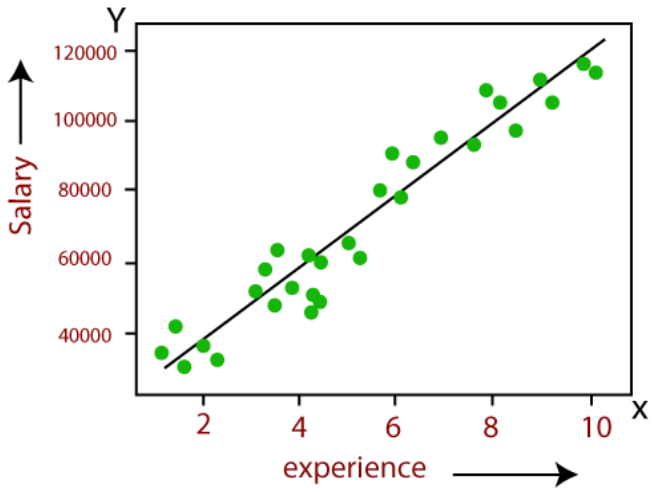
- **Linear Regression**
- **Logistic Regression**
- **Polynomial Regression**
- **Support Vector Regression**
- **Decision Tree Regression**
- **Random Forest Regression**
- **Ridge Regression**
- **Lasso Regression**



2.2.1. Linear Regression:

- Linear regression is a statistical regression method which is used for predictive analysis.
- It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.
- It is used for solving the regression problem in machine learning.
- Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.

- If there is only one input variable (x), then such linear regression is called **simple linear regression**. And if there is more than one input variable, then such linear regression is called **multiple linear regression**.
- The relationship between variables in the linear regression model can be explained using the below image. Here we are predicting the salary of an employee on the basis of **the year of experience**.



Below is the mathematical equation for Linear regression:

$Y = aX+b$

Here, Y = dependent variables (target variables),
X= Independent variables (predictor variables),
a and b are the linear coefficients

Some popular applications of linear regression are:

- Analyzing trends and sales estimates
- Salary forecasting
- Real estate prediction
- Arriving at ETAs in traffic.

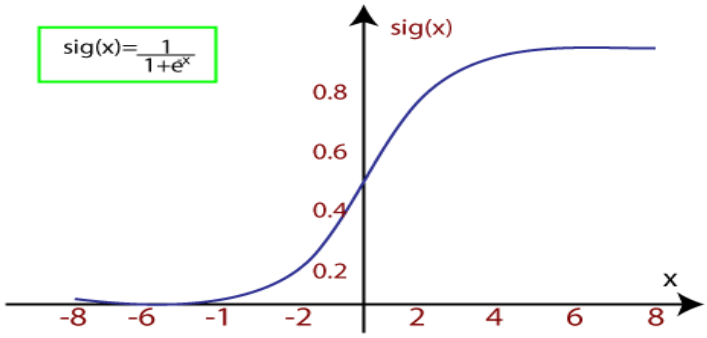
2.2.2. Logistic Regression:

- Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In **classification problems**, we have dependent variables in a binary or discrete format such as 0 or 1.
- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.
- It is a predictive analysis algorithm which works on the concept of probability.
- Logistic regression is a type of regression, but it is different from the linear regression algorithm in the term how they are used.
- Logistic regression uses **sigmoid function** or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression. The function can be represented as:

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

- f(x)= Output between the 0 and 1 value.
- x= input to the function
- e= base of natural logarithm.

  When we provide the input values (data) to the function, it gives the S-curve as follows:



- It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.

There are three types of logistic regression:

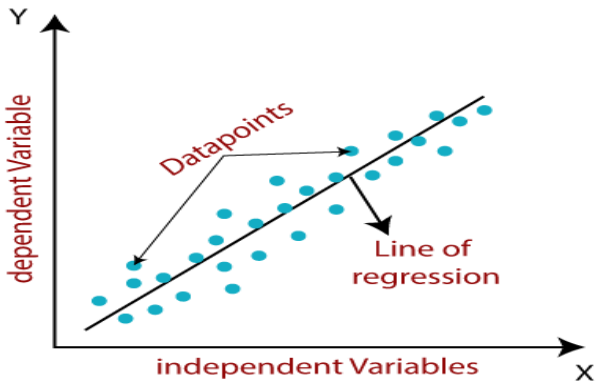
- **Binary(0/1, pass/fail)**
- **Multi(cats, dogs, lions)**
- **Ordinal(low, medium, high)**

Linear Regression in Machine Learning

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$$y= a_0+a_1x+ \varepsilon$$

Here,
Y= Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error
The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

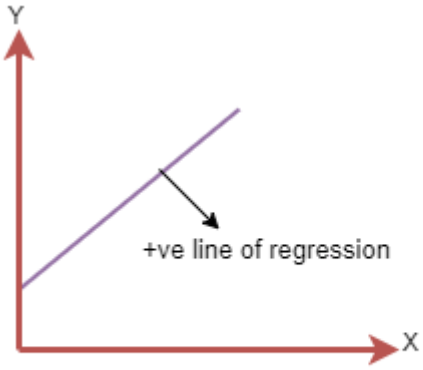
Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

- **Multiple Linear regression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

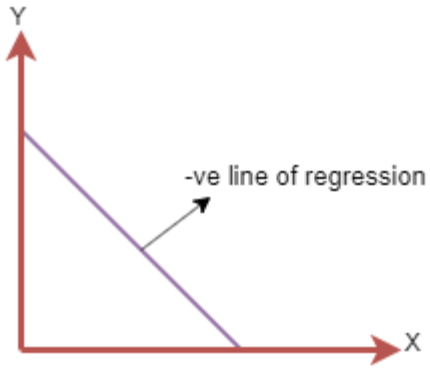
Linear Regression Line:
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y= a_0+a_1x$

- **Negative Linear Relationship:**
  If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation
Yi = Actual value
$(a1x_i+a_0)$= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:**
  Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:**
  Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is

difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.

- **Homoscedasticity Assumption:**
  Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.

- **Normal distribution of error terms:**
  Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
  It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.

- **No autocorrelations:**
  The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

Simple Linear Regression in Machine Learning

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the ***dependent variable must be a continuous/real value***. However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

- **Model the relationship between the two variables.** Such as the relationship between Income and expenditure, experience and Salary, etc.

- **Forecasting new observations.** Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

Simple Linear Regression Model:

The Simple Linear Regression model can be represented using the below equation:

$y= a_0+a_1x+ \varepsilon$

Where,

a0= It is the intercept of the Regression line (can be obtained putting x=0)
a1= It is the slope of the regression line, which tells whether the line is increasing or decreasing.
ε = The error term. (For a good model it will be negligible)

2.2.3. Multiple Linear Regressions

In the previous topic, we have learned about Simple Linear Regression, where a single Independent/Predictor(X) variable is used to model the response variable (Y). But there may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used.

Moreover, Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable.

We can define it as:
"*Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.*"

Example:
Prediction of CO_2 emission based on engine size and number of cylinders in a car.

Some key points about MLR:

- For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.
- Each feature variable must model the linear relationship with the dependent variable.
- MLR tries to fit a regression line through a multidimensional space of data-points.

MLR equation:

In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables $x_1, x_2, x_3, ...,x_n$. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

Y= b₀+b₁x₁+ b₂x₂+ b₃x₃+..... bnxn ..... (a)

Where,

Y= Output/Response variable
b_0, b_1, b_2, b_3 , b_n....= Coefficients of the model.
x_1, x_2, x_3, x_4,...= Various Independent/feature variable

Assumptions for Multiple Linear Regression:

- A linear relationship should exist between the Target and predictor variables.
- The regression residuals must be normally distributed.
- MLR assumes little or no multicollinearity (correlation between the independent variable) in data.

2.3. Neural Networks (ANN - Artificial Neural Network**)**

2.3.1. Introduction

The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.

UNIT-IVENSEMBLETECHNIQUESANDUNSUPERVISEDLEARNING

Combining multiple learners: Model combination schemes, Voting, Ensemble Learning - bagging, boosting, stacking, Unsupervised learning: K-means, Instance Based Learning: KNN, Gaussian mixture models and Expectation maximization.

Combiningmultiplelearners:

In any application, we can use one of several learning algorithms, and with certain algorithms, there are hyper parameters that affect thefinal learner. For example, in a classification setting, we can usea parametricclassifieror amultilayerperceptron,and,forexample, witha multilayer perceptron,we should also decide on the number of hidden units. The No Free Lunch Theorem states that there is no singlelearningalgorithmthatinanydomainalwaysinducesthemostaccuratelearner.Theusualapproach is to try many and choose the one that performs the best on a separate validation set.

Each learning algorithm dictates a certain model that comes with a set of assumptions. This inductive biasleadstoerrorifthe assumptions donot holdforthe data.Learningisanill-posed problem and with finite data, each algorithm converges to a different solution and fails under different circumstances.The performance of a learner maybe fine-tunedto get the highest possible accuracyon a validation set, but this finetuning is a complex task and still there are instances on which even the best learner is not accurate enough. The idea isthat there maybe another base-learner learner that is accurate on these. By suitably combining multiple *baselearners* then, accuracy can be improved. Recently with computation and memory getting cheaper, such systems composed of multiple learners havebecome popular.

Therearebasicallytwoquestionshere:

1. Howdowegeneratebase-learnersthatcomplementeachother?
2. Howdowecombinetheoutputsofbase-learnersformaximumaccuracy?

Our discussion in this chapter will answer these two related questions. We will see that model combination is not a trick that always increases accuracy; model combination does always increase time and space complexity of training and testing, and unless base-learners are trained carefully and their decisions combined smartly, we will only pay for this extra complexity without any significant gain in accuracy.

GeneratingDiverseLearners

Since there is no point in combining learners that always make similar diversity decisions, the aim is to be able to find a set of *diverse* learners who differ in their decisions so that they complement each other. At the same time, there cannot be a gain in overall success unless the learners are accurate,atleastintheirdomainofexpertise.Wethereforehavethisdoubletaskofmaximizingindividual accuracies and the diversity between learners.

Letusnowdiscussthedifferentwaystoachieve this.

DifferentAlgorithms

We can use different learning algorithms to train different base-learners. Different algorithms makedifferentassumptionsaboutthedataandleadtodifferentclassifiers.Forexample,onebase-learner may be parametric and another may be nonparametric. When we decide on a single algorithm,wegive emphasis to a single method and ignore all others. Combining multiple learners based on multiple algorithms, we free ourselves from taking a decision and we no longer put all our eggs in one basket.

DifferentHyperparameters

Wecanusethesamelearningalgorithmbutuseitwithdifferenthyperparameters.
Examples are the number of hidden units in a multilayer perceptron, k in k-nearest neighbor, error thresholdindecisiontrees,thekernelfunctioninsupportvectormachines,andsoforth.WithaGaussian parametric classifier, whether the covariance matrices are shared or not is a hyperparameter. Ifthe optimizationalgorithmusesaniterativeproceduresuchasgradient descentwhosefinalstatedependson theinitialstate,suchasinbackpropagationwithmultilayerperceptrons,theinitialstate,for example,the initial weights, is another hyperparameter. When we train multiple base-learners with different hyperparameter values, we average over this factor and reduce variance, and therefore error.

DifferentInputRepresentations

Separate base-learners may be usingdifferent *representations* of the same input object or event, making it possible to integrate different types of sensors/measurements/modalities. Different representationsmakedifferentcharacteristicsexplicitallowingbetteridentification.Inmanyapplications, there are multiple sources of information, and it is desirable to use all of these data to extract more information and achieve higher accuracy in prediction.

For example, in speech recognition, to recognize the uttered words, in addition to the acoustic input, we can also use the video image of the speaker's lips and shape of the mouth as the words are spoken. This is similar to *sensor fusion* where the data from different sensors are integrated to extract moreinformationforaspecificapplication.Anotherexampleisinformation,forexample,imageretrieval where in addition to the image itself, we may also have text annotation in the formofkeywords.

Insuchacase,wewanttobeabletocombinebothofthesesourcestofindtherightsetofimages; this is also sometimes called *multi-view learning*.

The simplest approach is to concatenate all data vectors and treat it as one large vector from a singlesource,but thisdoesnot seemtheoreticallyappropriatesincethiscorresponds to modelingdataas sampled from one multivariate statistical distribution. Moreover, larger input dimensionalities make the systemsmorecomplexandrequirelargersamplesfortheestimatorstobeaccurate.Theapproachwetake istomakeseparatepredictionsbasedondifferentsourcesusingseparatebase-learners,thencombinetheir predictions.

Even if there is a single input representation, by choosing random subsets from it, we can have classifiers using different input features; this is called the *random subspace method*. This has the effect

that different learners will look at the same problem from different points of view and will be robust; it will also help reduce the curse of dimensionality because inputs are fewer dimensional.

DifferentTrainingSets

Another possibilityistotrain different base-learnersbydifferentsubsets ofthetrainingset.This canbedonerandomlybydrawingrandomtrainingsetsfromthegivensample;thisiscalled*bagging*.Or, thelearnerscanbetrainedseriallysothatinstancesonwhichtheprecedingbase-learnersarenotaccurate are given more emphasis in training later base-learners; examples are *boosting* and *cascading*, which actively try to generate complementary learners, instead of leaving this to chance.

The partitioning of the training sample can also be done based on locality in the input space so that each base-learner is trained on instances in a certain local part of the input space; Similarly, it is possible to define the main task in terms of a number of subtasks to be implemented by thebase-learners,asisdoneby*error-correctingoutputcodes.*

Diversityvs.Accuracy

One important note is that when we generate multiple base-learners, we want them to be reasonably accurate but do not require them to be very accurate individually, so they are not, and need not be, optimized separately for best accuracy. The base-learners are not chosen for their accuracy, but fortheirsimplicity.Wedorequire,however,thatthebaselearnersbediverse,thatis,accurateon different instances, specializing in subdomains of the problem. What we care for is the final accuracy whenthe base-learners are combined, rather than the accuracies of thebase-learnerswe started from. Let ussaywehaveaclassifierthatis80percent accurate.Whenwedecideonasecondclassifier,wedo not care for the overall accuracy;we care only about how accurate it is on the 20percent that the first classifier misclassifies, as long as we know when to use which one.

This implies that the required accuracy and diversity of the learnersalso depend on how theirdecisions are to be combined, If, as in a votingscheme, a learner is consulted for all inputs, it should be accurate everywhere anddiversityshouldbeenforcedeverywhere; if we havea partitioningof theinput space into regions of expertise for different learners, diversity is already guaranteed by this partitioning and learners need to be accurate only in their own local domains.

ModelCombinationSchemes

Thereare alsodifferentwaysthemultiplebase-learnersarecombinedtogeneratethefinal output:

*Multiexpertcombination*methodshavebase-learnersthatworkin*parallel*.Thesemethodscan inturnbedividedintotwo:

Inthe*global*approach,alsocalled*learnerfusion*,givenaninput,allbase-learners generate an output and all these outputs are used.

Examplesare*voting*and*stacking*.

In the *local* approach, or *learner selection*, for example, in *mixture of experts*, there is a *gating* model, which looks at the input and chooses one (or very few) of the learners as responsible for generating the output.

Multistage combination methods use a *serial* approach where the next base-learner is trained with or tested on only the instances where the previous base-learners are not accurate enough. The idea is that the base-learners (or the different representations they use) are sorted in increasing complexity so that a complex base-learner is not used (or its complex representation is not extracted) unless the preceding simpler base-learners are not confident. An example is *cascading*.

Let us say that we have L base-learners. We denote by $d_j(x)$ the prediction of base-learner Mj given the arbitrary dimensional input x. In the case of multiple representations, each Mj uses a different input representation x_j . The final prediction is calculated from the predictions of the base-learners:

$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

where $f(\cdot)$ is the combining function with Φ denoting its parameters.

Base-learners are d_j and their outputs are combined using $f(\cdot)$.

This is for a single output; in the case of classification, each base-learner has K outputs that are separately used to calculate y_i , and then we choose the maximum.

Note that here all learners observe the same input; it may be the case that different learners observe different representations of the same input object or event.

Voting

The simplest way to combine multiple classifiers is by *voting*, which corresponds to taking a linear combination of the learners :

$$Y_i = \sum_j w_j d_{ji} \text{ where } w_j \geq 0, \sum_j w_j = 1$$

This is also known as *ensembles* and *linear opinion pools*. In the simplest case, all learners are given equal weight and we have *simple voting* that corresponds to taking an average. Still, taking a (weighted) sum is only one of the possibilities and there are also other combination rules. If the outputs are not posterior probabilities, these rules require that outputs be normalized to the same scale.

An example of the use of these rules is shown in table, which demonstrates the effects of different rules. Sum rule is the most intuitive and is the most widely used in practice. Median rule is more robust to outliers; minimum and maximum rules are pessimistic and optimistic, respectively.

With the product rule, each learner has veto power; regardless of the other ones, if one learner has an output of 0, the overall output goes to 0. Note that after the combination rules, y_i do not necessarily sum up to 1.

In weighted sum, d_{ji} is the vote of learner j for class C_i and w_j is the weight of its vote. Simple voting is a special case where all voters have equal weight, namely, $w_j = 1/L$. In classification, this is called *plurality voting* where the class having the maximum number of votes is the winner. When there are two classes, this is *majority voting* where the winning class gets more than half of the votes. If the

voters can also supply the additional information of how much they vote for each class (e.g., by the posteriorprobability),thenafternormalization,thesecanbeusedasweightsina*weightedvoting*scheme. Equivalently, if *dji* are the class posterior probabilities, *P(*C*i*|*x*,M*j)*, then we can just sum them up(*wj* = 1*/L*) and choose the class with maximum *yi* .

Inthecaseofregression,simpleorweightedaveragingormediancanbeusedtofusetheoutputs ofbase-regressors.Medianismorerobusttonoisethantheaverage.Anotherpossiblewaytofind*wj*isto assess the accuracies of the learners (regressor or classifier) on a separate validation set and use that information to compute the weights, so that we give more weights to more accurate learners.

Voting schemes can be seen as approximations under a Bayesian framework with weights approximating prior model probabilities, and model decisions approximating model-conditional likelihoods. This is *Bayesian* combination *model combination*.

For example, in classification we have *wj* ≡ *P(*M*j)*, *dji* = *P(*C*i*|*x*,M*j)*, and equation corresponds to *P(*C*i*|*x)* =

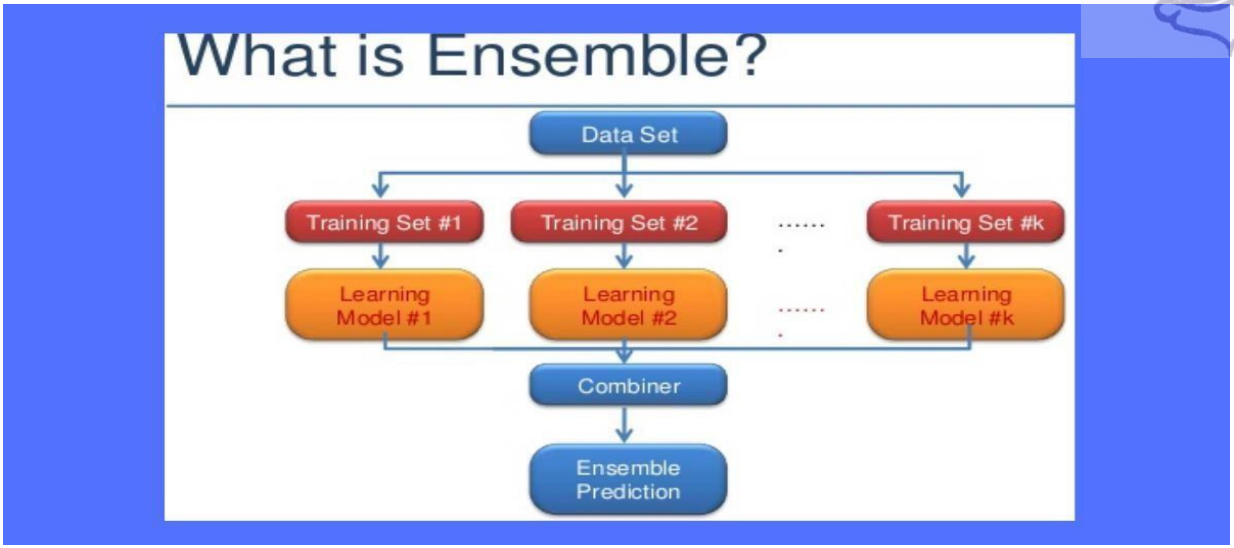
all models M*j*

*P(*C*i*|*x*,M*j)P(*M(17.3)*j)*

Simple votingcorrespondsto a uniformprior. If we have a prior distribution preferring simpler models, this would give larger weights to them. We cannot integrate over all models; we only choose a subsetfor which we believe *P(*M*j)* is high, or we can have another Bayesian step and calculate *P(*M*j*|X*)*, theprobability of a model given the sample, and sample high probable models from this density.Hansen and Salamon (1990) have shown that given independent twoclassclassifierswith success probabilityhigher than 1/2, namely, better than random guessing, by taking a majority vote, the accuracy increases as the number of voting classifiers increases.

EnsembleLearning:

- EnsembleLearningisaTechniquethatcreatemultiplemodelsandthencombinethemto produce improved results.
- Itproducesmoreaccuratesolutionsthanasinglemodel.



BAGGING

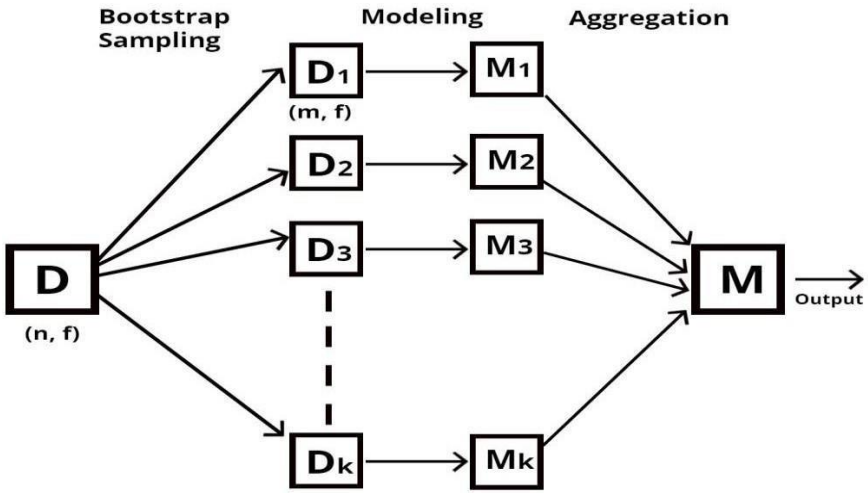
Bagging is a voting method whereby base-learners are made different by training them over slightly different training sets. Generating$L$slightly different samplesfromagivensampleisdonebybootstrap,wheregivenatrainingsetX of size$N$,wedraw$N$instancesrandomlyfromX*withreplacement*.Becausesampling is done with replacement, it is possible that some instances are drawn more than once and that certain instances are not drawn at all.

When this is done to generate$L$samples X_j,j□1,…,L, these samples are similarbecausetheyarealldrawnfromthesameoriginalsample,buttheyarealso slightly different due to chance. The base-learners$d_j$are trained with these$L$ samples X_j .

Alearningalgorithmisan*unstablealgorithm*ifsmallchangesinthetraining setcausesalargedifferenceinthegeneratedlearner,namely,thelearningalgorithm hashighvariance.Bagging,shortforbootstrapaggregating,uses

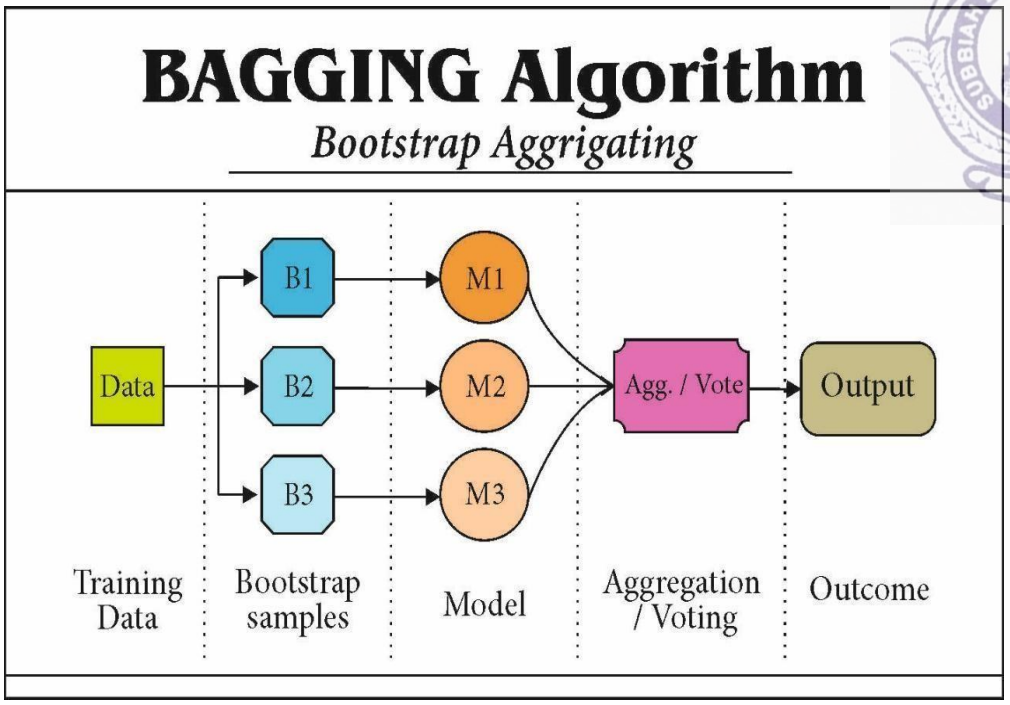
bootstraptogenerate$L$trainingsets,trains$L$base-learnersusinganunstablelearning procedure, and then, during testing, takes an average.

Bagging can be used both for classification and regression. In the case of regression, to be more robust, one can take the median instead of the average when combining predictions.



We saw before that averaging reduces variance only if the positive correlation is small; an algorithm is stable if different runs of the same algorithmonresampledversionsofthesamedatasetleadtolearnerswithhigh positive correlation. Algorithms such as decision trees and multi- layer perceptrons are unstable.

Nearestneighbor is stable, but condensed nearestneighbor is unstable (Alpaydın 1997). If the original training set is large, then we may want to generatesmallersetsofsize$N' < N$fromthemusingbootstrap,sinceotherwise thebootstrapreplicates$X_j$willbetoosimilar,and$d_j$willbehighlycorrelated.



BOOSTING

In bagging, generating complementary base-learners is left to chance andto the unstability of the learning method. In boosting, we actively try to generate complementarybase-learnersbytrainingthenextlearneronthemistakesoftheprevious learners. The original*boosting*algorithmcombines three weak learners to generate a stronglearner.A*weaklearner*haserrorprobabilitylessthan1/2,which makesitbetter thanrandomguessingonatwo-classproblem,anda*stronglearner*hasarbitrarilysmall error probability.

Given a large training set, we randomly divide it into three. We use $X_1$and traind_1.WethentakeX$_2$andfeedittod_1.Wetakeallinstancesmisclassifiedby $d_1$andalsoasmanyinstancesonwhich$d_1$iscorrectfromX$_2$,andthesetogetherform thetrainingsetofd_2.WethentakeX$_3$andfeeditto$d_1$andd_2.Theinstancesonwhich d_1 and d_2 disagree form the training set ofd_3. During testing, given an instance,we giveitto$d_1$andd_2;iftheyagree,thatistheresponse,otherwisethe

response of d_3 istaken as the output. Schapire (1990) has shown that this overall system has reducederror rate, and the error rate can arbitrarily be reduced by using such systems recursively, that is, a boosting system of three modelsused as$d_j$in a higher system.

Though it is quite successful,the disadvantage of the original boosting method is thatitrequiresa verylargetrainingsample. Thesampleshouldbedividedinto three and furthermore, the second and third clas- sifiers are only trained on a subsetonwhichthepreviousoneserr.Sounlessonehasaquitelargetrainingset, d_2 and d_3 will not have trainingsets of reasonable size. It use a set of 118,000in- stances in boosting multilayer perceptrons for optical handwritten digit recognition.

Training:

For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$

For all base-learners $j = 1, \ldots, L$

Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities p_j^t

Train d_j using $\mathcal{X}_j$

For each (x^t, r^t), calculate $y_j^t \leftarrow d_j(x^t)$

Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$

If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop

$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$

For each (x^t, r^t), decrease probabilities if correct:

If $y_j^t = r^t$, then $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$

Normalize probabilities:

$Z_j \leftarrow \sum_t p_{j+1}^t$; $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$

Testing:

Given x, calculate $d_j(x), j = 1, \ldots, L$

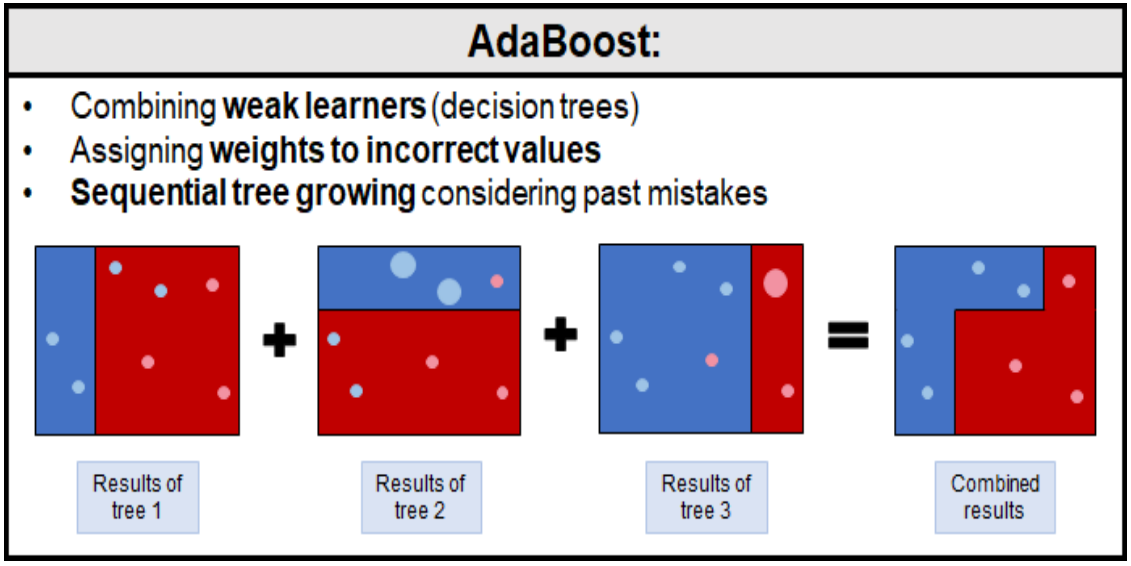
Calculate class outputs, $i = 1, \ldots, K$:

$$y_i = \sum_{j=1}^{L} \log \frac{1}{\beta_j} d_{ji}(x)$$

Freund and Schapire (1996) proposed a variant, named*AdaBoost*, short for adaptive boosting, that uses the same training set over and over and thus need not be large, but theclassifiersshouldbesimplesothattheydonotoverfit.AdaBoostcanalsocombine an arbitrary number of base- learners, not three.

InAdaBoost, although differentbase-learnershave slightly differenttraining sets,thisdifferenceisnotlefttochanceasinbagging,butisafunctionoftheerror of the previous base-learner. The actual performance of boosting on a particular problem is clearly dependent on the data and the base-learner. There should be enough training data and the base-learner should be weak but not too weak, and boosting is especially susceptible to noise and outliers.

AdaBoost has also been generalized to regression: Onestraightforwardway, proposed by Avnimelech and Intrator (1997), checks for whether the predictionerror islargerthanacertainthreshold,andifsomarks it as error, then uses AdaBoost proper.In another version, probabilities are modified based on the magnitude of error, such that instances where the previous base-learner commits a large error, havea higher probability of being drawn to train the next base-learner. Weighted average, ormedian,isusedtocombinethepredictionsofthebase-learners.



STACKING

*Stacked generalization*is a technique proposed by Wolpert (1992) that extends votinginthatthewaytheoutputofthebase-learnersiscombinednednotbelinearbut islearnedthroughacombinersystem,$f(\Phi)$

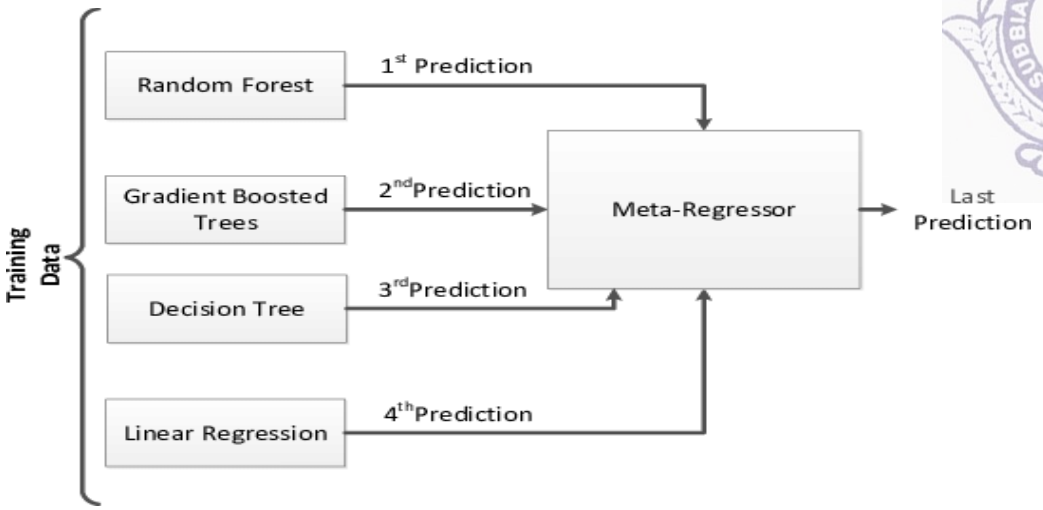
$$y=f(d_1,d_2,...,d_L\Phi)$$

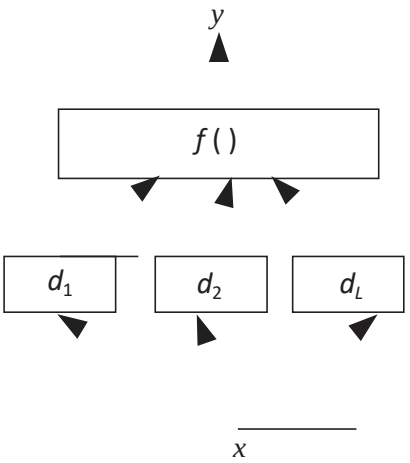
The combiner learns what the correct output is when the base-learners give a certain output combination. We cannot train the combiner functionon the training data because the base-learners may be memorizing the training set; the combiner system should actually learn how the base- learners make errors.Stacking is a means of estimatingandcorrectingfor the biases of the base-learners. Therefore, thecombiner shouldbe trained on data unused intraining the base-learners.

If $f(\cdot|w1, \ldots, wL)$is a linear model with constraints, $wi \geq 0, j\ wj = 1$, the optimal weights can be found by constrained regression, but of course we do not need to enforce this; in stacking, there is no restriction on the combiner function and unlike voting, $f(\cdot)$ can be nonlinear. For example, it may be implemented as a multilayer perceptron with Φ its connection weights. The outputs of the base-learners dj define a new L-dimensional space in which the output discriminant/regression function is learned by the combiner function.

In stacked generalization, we would like the base-learners to be as different as possible so that theywillcomplementeachother,and,forthis,itisbestiftheyarebasedondifferentlearningalgorithms. Ifwearecombiningclassifiersthatcangeneratecontinuousoutputs,forexample,posterior probabilities, it is better that they be the combined rather than hard decisions.

Whenwecompareatrainedcombineraswehaveinstacking,withafixedrulesuchasinvoting, we see that both have their advantages: A trained rule is more flexible and may have less bias, but adds extra parameters, risks introducing variance, and needs extra time and data for training. Note also that there is no need to normalize classifier outputs before stacking.


Training Data
Random Forest
Gradient Boosted Trees
Decision Tree
Linear Regression
1st Prediction
2nd Prediction
3rd Prediction
4th Prediction
Meta-Regressor
Last Prediction



FigureIn stacked generalization, the combiner is another learnerandisnotrestrictedtobeingalinearcombinationas in voting.

Unsupervisedlearning:

Manyof the learningalgorithms that wehave seen todate have made use ofa trainingset that consists of acollection of labelledtarget data,oratleast(for evolutionaryandreinforcement learning) some scoring system that identifies whether or not a prediction is good or not. Targets are obviously useful, since they enable us to show the algorithm the correct answer to possible inputs, but in many circumstances they are difficult to obtain they could, for instance, involve somebody labelling each instance by hand.

In addition, it doesn't seem to be very biologically plausible: most of the time when we arelearning, we don't get told exactly what the right answer should be. In this chapter we will consider exactlythe opposite case,where there is no information about the correct outputs available at all, and the algorithm is left to spot some similarity between different inputs for itself. Unsupervised learning isaconceptuallydifferentproblemtosupervisedlearning.Obviously,wecan'thopetoperform regression: we don't knowthe outputs for any points, so we can't guess what the function is. Can we hope to do classification then? The aim of classification is to identify similarities between inputs that belong to the same class. There isn't any information about the correctclasses, but if the algorithmcan exploit similarities between inputs in order to cluster inputs that are similar together, this might perform classification automatically. So the aim of unsupervised learning is to find clusters of similar inputs in the data without being explicitlytold that these datapoints belong to oneclass and those toa different class. Instead, the algorithm has to discover the similarities for itself. We have already seen some unsupervised learning algorithms, where the focus was on dimensionality reduction, and hence clustering of similar datapoints together.

Thesupervisedlearningalgorithmsthatwehavediscussedsofarhaveaimedtominimisesomeexternal errorcriterion—mostlythesum-of-squareserror—basedonthedifferencebetweenthetargetsandthe outputs. Calculating and minimising this error was possible because we had target datato calculate it from, which is not true for unsupervised learning. This means that we need to find something else to drive the learning. The problem is more general than sum-of-squares error: we can'tuse any error criterionthatreliesontargetsorotheroutsideinformation(anexternalerrorcriterion), weneedtofind something internal to the algorithm. This means that the measure has to be independent of the task, becausewe can't keep on changing the whole algorithm every time a new task is introduced. In supervisedlearningtheerrorcriterionwastask-specific,becauseit wasbasedonthetarget datathatwe provided.Theninputsthatareclosetogetherareidentifiedasbeingsimilar,sothattheycanbeclustered, while inputs that are far apart are not clustered together. We can extend thisto the nodes of a network by aligning weight space with input space. Now if the weight values of a node aresimilarto the elementsofaninputvectorthenthatnodeshouldbeagoodmatchforthe

input, and any other inputs thatare similar. In order tostart to see these ideas in practice we'll look ata simple clustering algorithm, the *k*-Means Algorithm, which has been around in statistics for a long time.

THE*K*-MEANSALGORITHM

If you have ever watched a group of tourists with a couple of tour guides who hold umbrellas up so that everybody can see them and follow them, then you have seen a dynamic version of the *k*-means algorithm. Our version is simpler, because the data (playing the part of the tourists) does notmove, only the tour guides.

Supposethat we want to divide ourinput datainto *k*categories, wherewe know the value of *k* (for example, we have a set of medical test results from lots of people for three diseases, and we want to see how well the tests identify the three diseases). We allocate *k* cluster centres to our input space, and we would like to position these centres so that there is one cluster centre in the middle of each cluster. However, we don't know where the clusters are, let alone where their 'middle' is, so we need an algorithm that will find them.

Learning algorithms generally try to minimise some sort of error, so we need to think of an error criterion that describes this aim. The idea of the 'middle' is the first thing that we need to think about. How do we define the middle of a set of points? There are actually two things that we need to define:

A distance measure In order to talk about distances between points, we need some way to measure distances. It is often the normal Euclidean distance.

The mean average Once wehave a distance measure,we can compute the centralpointof a set of datapoints, which is the mean average (if you aren't convinced, think what the mean of two numbers is, it is the point halfway along the line between them). Actually, this is only true in Euclidean space, which is the one you are used to, whereeverything is nice and flat. Everything becomes a lot trickierifwehavetothinkaboutcurvedspaces;whenwehavetoworryaboutcurvature,theEuclidean distance metric isn't the right one, and there are at least two different definitions of the mean. So we aren'tgoingtoworryaboutanyofthesethings,andwe'llassumethatspaceisflat.Thisiswhat statisticians do all the time.

Wecannowthinkaboutasuitablewayofpositioningtheclustercentres:wecomputethe meanpoint of eachcluster, $\boldsymbol{\mu}c(i)$,and put the cluster centrethere. This isequivalent tominimisingthe Euclidean distance (which is the sum-of-squares error again) fromeach datapoint to its cluster centre. How do we decide which points belongto which clusters? It is important to decide, since we will use that to position the cluster centres. The obvious thing is to associate each point with the cluster centre that it is closest too. This might change as the algorithm iterates, but that's fine.

We start by positioning the cluster centres randomly through the input space, since we don't know where to put them, and then we update their positions according to the data. We decide which clustereachdatapointbelongstobycomputingthedistancebetweeneachdatapointandallofthecluster centres, and assigning it to the cluster that is the closest.

Notethatwecanreducethecomputationalcost ofthisprocedurebyusingtheKD-Tree algorithm. For all of the points that are assigned to a cluster, we then compute the mean of them, and move the cluster centre to that place.

Weiteratethealgorithmuntiltheclustercentresstopmoving.Hereisthealgorithmic description:

The*k*-MeansAlgorithm Initialisation

– chooseavaluefor*k*

– choose*k*randompositionsintheinput space

– assigntheclustercentres$\boldsymbol{\mu}j$tothose positions

• **Learning**

– repeat

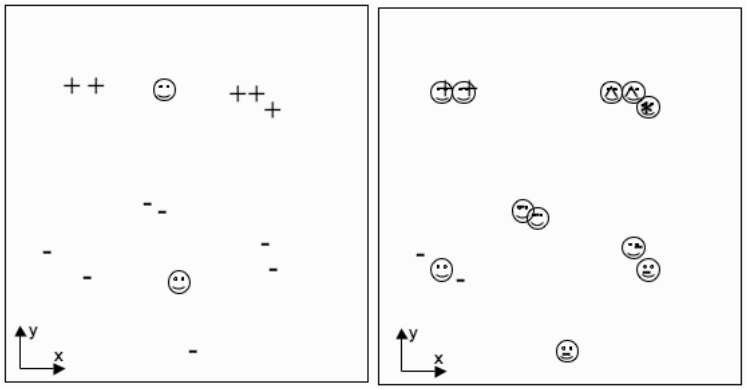
* foreachdatapoint$\mathbf{x}i$:

· computethedistancetoeachclustercentre

· assignthedatapointtothenearestclustercentrewithdistance

di=min*j* *d*(**x***i*,***μ****j*)
* foreachclustercentre:
· movethepositionofthecentretothemeanofthepointsinthatcluster (*Nj* is the number of points in cluster *j*):
μ*j*=1
Nj
X*Nj*
i=1
x*i*
– untiltheclustercentresstopmoving
• **Usage**
– foreachtest point:
* computethedistancetoeachclustercentre
* assignthedatapointtothenearestclustercentrewithdistance
di=min
j
d(**x***i*,***μ****j*).

```
#Compute distances
distances=np.ones((1,self.nData))*np.sum((data-
self.centres[0,:])**2,'axis=1)
forjinrange(self.k-1):
distances=np.append(distances,np.ones((1,self.nData))*np.sum((data-
'self.centres[j+1,:])**2,axis=1),axis=0)
#Identifytheclosestcluster
cluster=distances.argmin(axis=0)
cluster=np.transpose(cluster*np.ones((1,self.nData)))
#Updatetheclustercentres
forjinrange(self.k):
thisCluster=np.where(cluster==j,1,0) if
sum(thisCluster)>0:
self.centres[j,:]=np.sum(data*thisCluster,axis=0)/np.sum(thisCluster)
```



*Left:*Asolutionwithonly2classes,whichdoesnotmatchthedatawell.

*Right:*Asolutionwith11classes,showingsevereoverfitting

weset*k*tobeequaltothenumberofdatapoints,wecanpositiononecentreoneverydatapoint,and the sum-of-squares error willbe zero (in fact, this won't happen,sincetherandominitialisationwill mean that several clusters will end up coinciding). However, there is no generalisation in this solution:it isa case of serious overfitting. However, by computing the error on a validation set and multiplying the error by *k* we can see something about the benefit of adding each extra cluster centre.

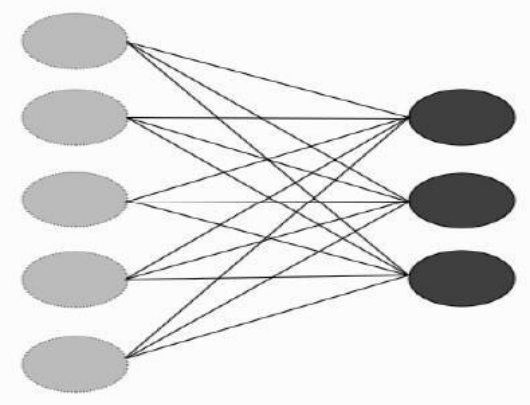
DealingwithNoise

Therearelotsofreasonsforperformingclustering,butoneofthemorecommononesistodeal withnoisydatareadings.Thesemightbeslightlycorrupted,oroccasionallyjustplainwrong.If wecan choose the clusters correctly, then we have effectively removed the noise, because we replace each noisy datapoint by the cluster centre.

Unfortunately,themeanaverage,whichiscentraltothe*k*-meansalgorithm,isverysusceptible tooutliers,i.e.,verynoisymeasurements.Onewaytoavoidtheproblemistoreplacethe meanaverage withthemedian,whichiswhatisknownasarobust statistic,meaningthatitisnot affectedbyoutliers (themeanof(1,2,1,2,100)is21.2,whilethemedianis2).Theonlychangethatisneededtothe algorithmistoreplacethecomputationofthemeanwiththecomputationofthemedian.Thisis computationally more expensive, as we've discussed previously, but it does remove noise effectively. The *k*-Means Neural Network

The *k*-means algorithm clearly works, despite its problems with noise and the difficulty with choosing the number of clusters. Interestingly, while it might seem a long way from neural networks,it isn't. If we think about the cluster centres that we optimise the positions of as locations in weight space,thenwecouldpositionneuronsinthoseplacesanduseneuralnetworktraining.Thecomputation that happenedinthe *k*-meansalgorithmwasthat eachinput decidedwhichclustercentre it wasclosest to by calculating the distance to all of the centres. We could do this inside a neural network, too: the location of each neuron is its position in weight space, which matches the values of its weights.

Asingle-layerneuralnetworkcanimplementthe*k*-meanssolution



So,wecanimplementthe*k*-meansalgorithmusingasetofneurons.Wewillusejustonelayer ofneurons,togetherwithsomeinputnodes,andnobiasnode.Thefirstlayerwillbetheinputs, which don't do any computation, as usual, and the second layer will be a layer of competitive neurons,thatis,neuronsthat'compete'tofire,withonlyoneofthemactuallysucceeding.Only oneclustercentrecanrepresentaparticularinputvector,andsowewillchoosetheneuronwith the highest activation *h* to be the one that fires. This is known as winner-takes-all activation, and it is an example of competitive learning, since the set of neurons compete with each other tofire,withthewinnerbeingtheonethatbestmatches(i.e.,isclosestto)theinput.Competitive learning is sometimes said to lead to grandmother cells, because each neuron in the network will learn to recognise one particular feature, and will fireonly when that input is seen. You would then have a specificneuron that was trained to recognise your grandmother (and others for anybody else/anything else that you see often).

Wewillchoose*k*neurons(forhopefullyobviousreasons)andfullyconnecttheinputs to the neurons, as usual. We will use neurons with a linear transfer function, computingthe activation of the neurons as simply the product of the weights and inputs:

$$hi=\sum wijxj$$

Providing that the inputs are normalised so that their absolute size is the same, this effectively measures the distance between the input vector and the cluster centre represented by that neuron, with larger numbers (higher activations) meaning that the two points are closer together.

Sothewinningneuronistheonethatisclosesttothecurrentinput.

TheOn-Line*k*-MeansAlgorithm

• **Initialisation**

– chooseavaluefor*k*,whichcorrespondstothenumberofoutputnodes

– initialisetheweightstohavesmallrandomvalues

• **Learning**

– normalisethedatasothatallthepointslieontheunit sphere

– repeat:

* foreachdatapoint:

· computetheactivationsofallthenodes

· pickthewinnerasthenodewiththehighestactivation

· updatetheweightsusingEquation

* untilnumberofiterationsisaboveathreshold

• **Usage**

–foreachtest point:

* computetheactivationsofallthe nodes

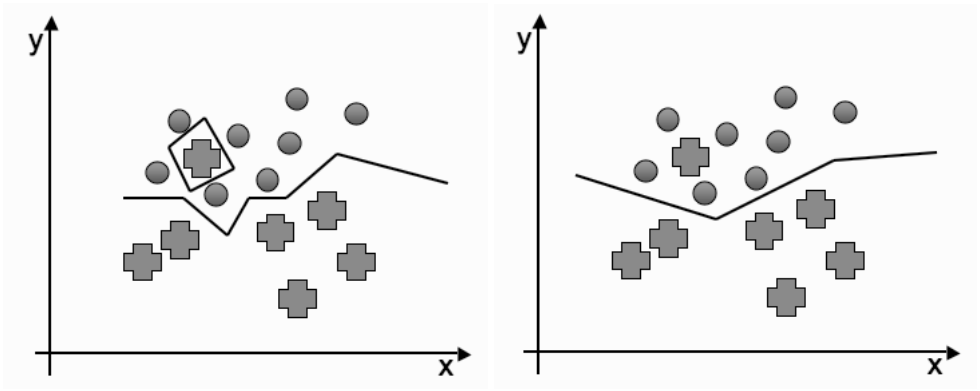
* pickthewinnerasthenodewiththehighestactivation

NEARESTNEIGHBOURMETHODS

Suppose that you are in a nightclub and decide to dance. It is unlikely that you will know the dancemoves fortheparticular songthat is playing, soyou will probablytrytoworkout what to do by lookingatwhatthepeopleclosetoyouaredoing.Thefirstthingyoucoulddowouldbejusttopickthe person closest to you and copy them. However, since most of the people who are in the nightclub are also unlikely to know all the moves, you might decide to look at a few more people and do what most of them are doing. This is pretty much exactlythe idea behind nearest neighbour methods: if we don't haveamodelthatdescribesthedata,thenthebestthingtodoistolookatsimilardataandchoosetobe in the same class as them.

Wehavethedatapointspositionedwithininputspace,sowejustneedtoworkoutwhichof the training data are close to it. This requires computing the distance to each datapoint in the training set, which is relatively expensive: if we are in normal Euclidean space, then we have to compute d subtractions and d squarings (we can ignore the square root since we only want to know which points are the closest, not the actualdistance) and this has tobe done O(N2) times. We can then identify the$k$nearestneighbourstothetestpoint,andthensettheclassofthetestpointtobethemostcommon one out of those for the nearestneighbours. The choice of k is not trivial. Make it too small and nearest neighbour methods are sensitive to noise, too large and the accuracy reduces as points that are too far awayareconsidered.Somepossibleeffectsofchangingthesizeof$k$onthedecisionboundaryareshown in Figure.

Thismethodsuffersfromthecurseofdimensionality.First,asshown above,thecomputationalcostsgethigherasthenumberofdimensionsgrows.Thisisnotas bad as it might appear at first: there are sets of methods such as KD-Trees that compute this in O(N log$N$)time.However,moreimportantly,asthenumberofdimensionsincreases,sothedistancetoother datapoints tends to increase.



Thenearestneighboursdecisionboundarywith*left:*oneneighbourand*right:*
twoneighbours.

In addition, they can be far away in a variety of different directions—there might be points that are relatively close in some dimensions, but a long way in others. There are methods fordealingwiththeseproblems,knownas adaptivenearestneighbourmethods,andthereis areferencetothemintheFurtherReadingsectionattheendofthechapter.

Theonlypart of thisthatrequires anycare duringtheimplementationis whatto do whenthereismorethanoneclassfoundintheclosestpoints,butevenwiththatthe implementationisniceandsimple:

```
def knn(k,data,dataClass,inputs):
    nInputs=np.shape(inputs)[0]
    closest = np.zeros(nInputs)
for ninrange(nInputs):
    #Computedistances
    distances=np.sum((data-inputs[n,:])**2,axis=1)
    #Identifythenearestneighbours
    indices = np.argsort(distances,axis=0)
    classes=np.unique(dataClass[indices[:k]]) if
    len(classes)==1:
        closest[n]=np.unique(classes)
    else:
        counts=np.zeros(max(classes)+1)
        for i in range(k):
            counts[dataClass[indices[i]]]+=1 closest[n]
            = np.max(counts)

    returnclosest
```

We are going to look next at how we can use these methods for regression, before we turn to the question of how to perform the distance calculations as efficiently as possible, something that is done simply but inefficiently in the code above. We will then consider briefly whether or not the Euclideandistanceisalwaysthemostusefulwaytocalculatedistances,andwhatalternativesthereare.

Forthe*k*-nearestneighboursalgorithmthebias-variancedecompositioncanbecomputed as:

The way to interpret this is that when k is small, so that there are few neighbours considered, the model has flexibility and can represent the underlying model well, but that it makes mistakes (has high variance) because there is relatively little data. As k increases, the variance decreases, but at the cost of less flexibility and so more bias.

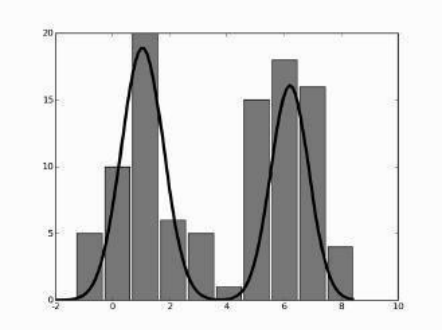
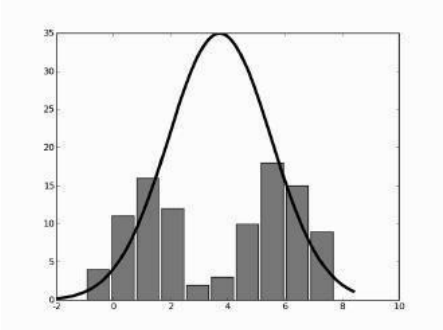
GAUSSIANMIXTUREMODELS

For the Bayes' classifier thatwe saw in previousSection, the data had target labels, andso we could do supervised learning, learning the probabilities from the labelled data. However, supposethatwehavethesamedata,butwithouttargetlabels.Thisrequiresunsupervisedlearning, but here we will look at one special case. Suppose that the different classes each come from their ownGaussiandistribution.Thisisknownasmulti-modaldata,sincethereisonedistribution(mode) for each different class. We can't fit one Gaussian to the data, because it doesn't look Gaussian overall.

There is, however, something we can do. If we know how many classes there are in the data, then we can try to estimate the parameters for that many Gaussians, all at once. If we don't know, then we can try different numbers and see which one works best. We will talk about this issue more for a different method (the *k*-means algorithm) in Section 14.1. It is perfectly possible to use any other probability distribution instead of a Gaussian, but Gaussians are by far the most common choice. Then the output for any particular datapoint that is input to the algorithm will be the sum of the values expected by all of the

*M*Gaussians:

$f(\mathbf{x})=\sum m=1\alpha m|\emptyset(\mathbf{x};\boldsymbol{\mu}m,\sum m),$

Histograms of training data from a mixture of two Gaussians and two fitted models, shown as the line plot. The model shown on the left fits well, but the one on the right produces two Gaussians right on top of each other that do not fit the data well.

where_(**x**;$\boldsymbol{\mu}m$,_*m*)isaGaussianfunctionwithmean $\boldsymbol{\mu}m$andcovariancematrix_*m*,and the _*m* are weights with the constraint that *M*P*m*=1.

Figure shows two examples, where the data (shown by the histograms) comes from two different Gaussians, and the model is computed as a sum or mixture of the two Gaussians together. The figure alsogivesyousomeideaofhowtousethemixturemodelonceithasbeencreated.Theprobabilitythat input**x***i*belongstoclass*m*canbewrittenas(whereahatonavariable(ˆ·)meansthatweareestimating the value of that variable):

$$p(\mathbf{x}i2cm)=\hat{}\_m\_(\mathbf{x}i;\hat{}\boldsymbol{\mu}m;\hat{}\_m)M\text{P}$$
$$k=1$$
$$\hat{}\_m\_(\mathbf{x}i;\hat{}\boldsymbol{\mu}k;\hat{}\_k)$$

The problem is how to choose the weights _*m*. The common approach is to aim for the maximum likelihood solution (the likelihood is the conditional probability of the data given the model, and the maximum likelihood solution varies the model to maximise this conditional probability). In fact, it is commontocomputetheloglikelihoodandthentomaximisethat;itisguaranteedtobenegative,since probabilitiesarealllessthan1,andthelogarithmspreadsoutthevalues,makingtheoptimisationmore effective. The algorithm that is used is an example of a very general one known as the expectation-maximisation(ormorecompactly,EM)algorithm.Thereasonfrthenamewill become clearer below. We will see another example of an EM algorithm, but here we see how to useit for fitting Gaussian mixtures, and get a very approximate idea of how the algorithm works for more general examples.

TheExpectation-Maximisation(EM)Algorithm

The basic ideaof theEM algorithm is that sometimesit is easier to add extra variables thatarenotactuallyknown(calledhiddenorlatentvariables)andthentomaximisethefunctionover those variables. This might seem to be making a problem much more complicated than it needs to be, but it turns out for many problems that it makes finding the solution significantly easier.

In order to see how it works, we will consider the simplest interesting case of the Gaussian mixture model: a combination of just two Gaussian mixtures. The assumption now is that data were created by randomly choosing one of two possible Gaussians, and then creating a sample from that Gaussian. If the probability of picking Gaussian one is *p*, then the entire model looks like this (where N(***μ***,_2) specifies a Gaussian distribution with mean ***μ*** and standard deviation _):

*G*1=N(***μ***1,_21)

*G*2=N(***μ***2,_22)

y= *pG*1+(1 −*p*)*G*2.

Iftheprobabilitydistributionof*p*iswrittenas_,thentheprobabilitydensityis:

P(**y**)=___(**y**;***μ***1,_1)+(1−_)_(**y**;***μ***2,_2).

Findingthemaximumlikelihoodsolution(actuallythemaximumloglikelihood)tothisproblemis then a case of computing the sum of the logarithm of Equation (7.4) over all of the training data, and differentiatingit,whichwouldberatherdifficult.Fortunately,thereisawayaroundit.Thekeyinsight that we need is that if we knew which of the two Gaussian components the datapoint came from, then the computation would be easy. Themean and standard deviation for each component could be computed from the datapoints that belong to that component, and there would not be a problem. Although we don't know which component each datapoint came from, we can pretend wedo, by introducinga new variable *f*. If *f* = 0 thenthe data came fromGaussian one, if*f* =1 thenit came from Gaussian two.

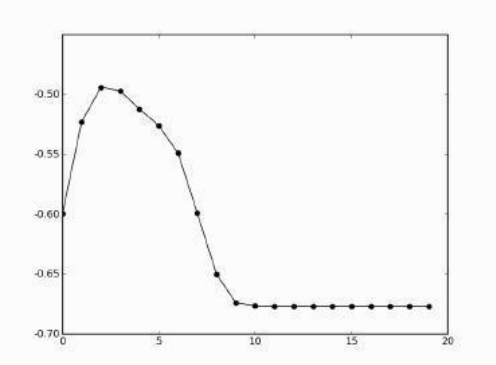
This is the typical initial step of an EM algorithm: adding latent variables. Now we just need to work out how to optimise over them. This is the time when the reason for the algorithm being called expectation-maximisation becomes clear. We don't know much about variable *f* (hardly surprising, since we invented it), but we can compute its expectation (that is, the value that we 'expect' to see, which is the mean average) from the data:

i(^***μ***1,^***μ***2,^_1,^_2,^_)=*E*(*f*|^***μ***1,^***μ***2,^_1,^_2,^_,*D*)

=*P*(*f*=1|^***μ***1,^***μ***2,^_1,^_2,^_,*D*),

where*D*denotesthedata.Notethatsincewehaveset *f*=1thismeansthatweare choosing Gaussian two.

Computing the value of this expectation is known as the E-step. Then this estimate of the expectation is maximised over the model parameters (the parameters of the two Gaussians and the mixing parameter _), the M-step. This requires differentiating the expectation with respect to each of the model parameters. These two steps are simplyiterated until the algorithmconverges. Note that the estimate never gets any smaller, and it turns out that EM algorithms are guaranteed to reach a local maxima.



PlotoftheloglikelihoodchangingastheGaussianMixtureModelEMalgorithm

learnstofitthetwoGaussians

```
s2=np.sum(gamma*(y-mu2)**2)/np.sum(gamma) pi
= np.sum(gamma)/N
ll[count-1]=np.sum(np.log(pi*np.exp(-(y[i]-mu1)**2/(2*s1))+(1-pi)'
*np.exp(-(y[i]-mu2)**2/(2*s2))))
```

shows the log likelihood dropping as the algorithm learns for the example on the left of Figure. The computationalcostsofthismodelareverygoodforclassifyinganewdatapoint,sinceitisO(*M*),where *M*isthenumberofGaussians,whichisoftenoftheorderoflog*N*(where*N*isthenumberofdatapoints). The training is, however, fairly expensive: O(*NM*2 +*M*3).

Thegeneral algorithmhasprettymuch exactlythe same steps(theparameters ofthe model are written as _, _0 is a dummy variable, *D* is the original dataset, and *D*0 is the dataset with the latent variables included):

TheGeneralExpectation-Maximisation(EM)Algorithm

- Initialisation
  - –guessparameters
- Repeatuntilconvergence:
  - – (E-step)computetheexpectation
  - – (M-step)estimatethenewparameters

The trick with applying EM algorithms to problems is in identifying the correct latent variables to include,andthensimplyworkingthroughthesteps.Theyareverypowerfulmethodsforawide variety of statistical learning problems. We are now going to turn our attention to something much simpler, which is how we can use information about nearby datapoints to decide on classification output. For this we don't use a model of the data at all, but directly use the data that is available.

UNIT V NEURAL NETWORKS

Multilayer perceptron, activation functions, network training – gradient descent optimization –stochastic gradient descent, error backpropagation, from shallow networks to deep networks –Unit saturation (aka the vanishing gradient problem) – ReLU, hyperparameter tuning, batch normalization, regularization, dropout.

Multilayer perceptron:

In the last chapter we saw that while linear models are easy to understand and use, they come with the inherent cost that is implied by the word 'linear'; that is, they can only identify straight lines, planes, or hyperplanes. And this is not usually enough, because the majority of interesting problems are not linearly separable. In Section we saw that problems can be made linearly separable if we can work out how to transform the features suitably.

We have pretty much decided that the learning in the neural network happens in the weights. So, to perform more computation it seems sensible to add more weights. There are two things that we can do: add some backwards connections, so that the output neurons connect to the inputs again, or add more neurons. The first approach leads into recurrent networks. These have been studied, but are not that commonly used. We will instead consider the second approach. We can add neurons between the input nodes and the outputs, and this will make more complex neural networks, such as the one shown in Figure.1.

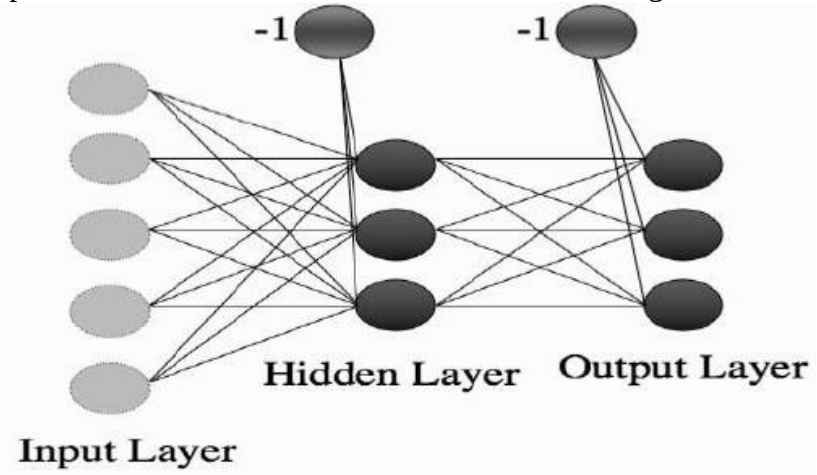


FIGURE:1 The Multi-layer Perceptron network, consisting of multiple layers of connected neurons.

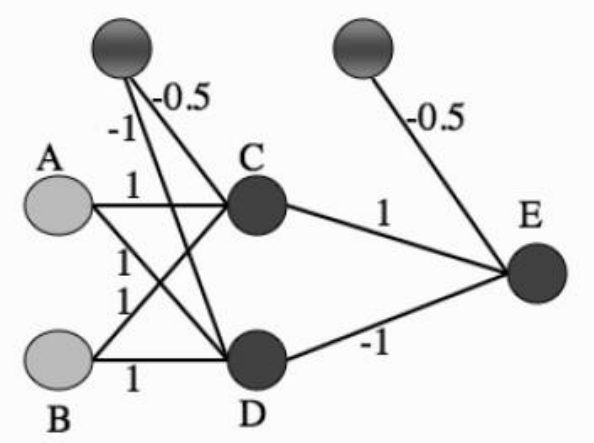


Figure:2 A Multi-layer Perceptron network showing a set of weights that solve the XOR problem.

We will think about why adding extra layers of nodes makes a neural network more powerful, but for now, to persuade ourselves that it is true, we can check that a prepared network can solve the two-dimensional XOR problem, something that we have seen is not

possible for a linear model like the Perceptron. A suitable network is shown in Figure2. To check that it gives the correct answers, all that is required is to put in each input and work through the network, treating it as two different Perceptrons, first computing the activations of the neurons in the middle layer (labelled as C and D in Figure 2) and then using those activations as the inputs to the single neuron at the output. As an example, I'll work out what happens when you put in (1, 0) as an input; the job of checking the rest is up to you. Input (1, 0) corresponds to node A being 1 and B being 0. The input to neuron C is therefore $-1 \times 0.5 + 1 \times 1 + 0 \times 1 = -0.5 + 1 = 0.5$. This is above the threshold of 0, and so neuron C fires, giving output 1. For neuron D the input is $-1 \times 1 + 1 \times 1 + 0 \times 1 = -1 + 1 = 0$, and so it does not fire, giving output 0. Therefore the input to neuron E is $-1 \times 0.5 + 1 \times 1 + 0 \times -1 = 0.5$, so neuron E fires. Checking the result of the inputs should persuade you that neuron E fires when inputs A and B are different to each other, but does not fire when they are the same, which is exactly the XOR function (it doesn't matter that the fire and not fire have been reversed).

Since this network can solve a problem that the Perceptron cannot, it seems worth looking into further. However, now we've got a much more interesting problem to solve, namely how can we train this network so that the weights are adapted to generate the correct (target) answers? If we try the method that we used for the Perceptron we need

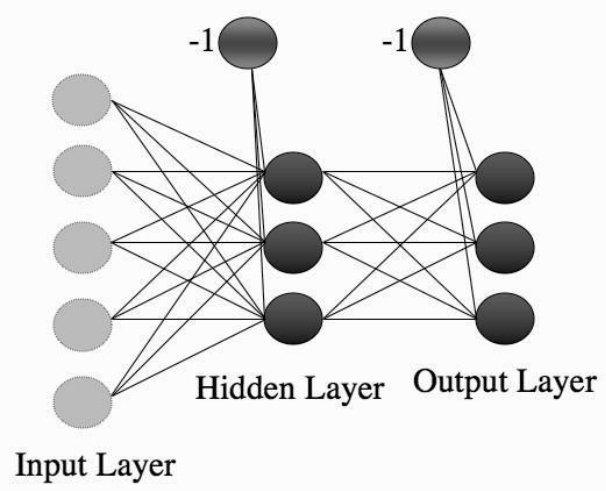


Fig: The Multi-layer Perceptron network, consisting of multiple layers of connected neurons.

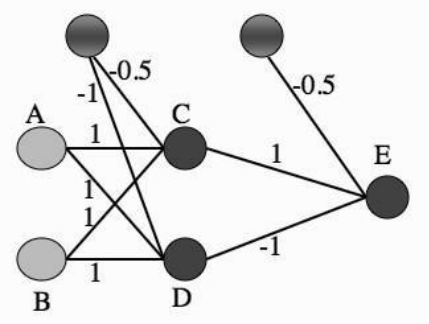


Fig:The Multi-layer Perceptron network, consisting of multiple layers of connectedneurons.

A Multi-layer Perceptron network showing a set of weights that solve the XOR problem. to compute the error at the output. That's fine, since we know the targets there,

so we can compute the difference between the targets and the outputs. But now we don't know which weights were wrong: those in the first layer, or the second? Worse, we don't know what the correct activations are for the neurons in the middle of the network. This fact gives the neurons in the middle of the network their name; they are called the hidden layer (or layers), because it isn't possible to examine and correct their values directly. It took a long time for people who studied neural networks to work out how to solve this problem. In fact, it wasn't until 1986 that Rumelhart, Hinton, and McClelland managed it.

However, a solution to the problem was already known by statisticians and engineers—they just didn't know that it was a problem in neural networks! In this chapter we are going to look at the neural network solution proposed by Rumelhart, Hinton, and McClelland, the Multi-layer Perceptron (MLP), which is still one of the most commonly used machine learning methods around. The MLP is one of the most common neural networks in use. It is often treated as a 'black box', in that people use it without understanding how it works, which often results in fairly poor results. Getting to the stage where we understand how it works and what we can do with it is going to take us into lots of different areas of statistics, mathematics, and computer science, so we'd better get started.

ACTIVATION FUNCTIONS

Activation functions are functions used in neural networks to computes the weighted sum of input and biases, of which is used to decide if a neuron can be fired or not. It manipulates the presented data through some gradient processing usually gradient descent and afterwards produce an output for the neural network, that contains the parameters in the data. These AFs are often referred to as a transfer function in some literature.

Activation function can be either linear or non-linear depending on the function it represents, and are used to control the outputs of out neural networks, across different domains from object recognition and classification to speech recognition, segmentation, scene understanding and description, machine translation test to speech systems, cancer detection systems, finger print detection, weather forecast, self-driving cars, and other domains to mention a few, with early research results by, validating categorically that a proper choice of activation function improves results in neural network computing.

For a linear model, a linear mapping of an input function to an output, as performed in the hidden layers before the final prediction of class score for each label is given by the affine transformation in most cases [5]. The input vectors x transformation is given by

$$f(x) = w\ T\ x + b - (1.1)$$

where x = input, w = weights, b = biases.

Furthermore, the neural networks produce linear results from the mappings from equation (1.1) and the need for the activation function arises, first to convert these linear outputs into non-linear output for further computation, especially to learn patterns in data. The output of these models are given by

$$y = (w1\ x1 + w2\ x2 + .. + wn\ xn + b) - (1.2)$$

These outputs of each layer is fed into the next subsequent layer for multilayered networks like deep neural networks until the final output is obtained, but they are linear by default. The expected output determines the type of activation function to be deployed in a given network.

However, since the output are linear in nature, the nonlinear activation functions are required to convert these linear inputs to non-linear outputs. These AFs are transfer functions that are applied to the outputs of the linear models to produce the transformed non-linear outputs, ready for further processing.

The non-linear output after the application of the AF is given by

$$y = \alpha(w1\ x1 + w2\ x2 + .. + wn\ xn + b) - (1.3)$$

Where α is the activation function.

The need for these AFs include to convert the linear input signals and models into non-linear output signals, which aids the learning of high order polynomials beyond one degree for deeper networks. A special property of the non-linear activation functions is that they are differentiable else they cannot work during backpropagation of the deep neural networks.

The deep neural network is a neural network with multiple hidden layers and output layer. An understanding of the makeup of the multiple hidden layers and output layer is our interest. A typical block diagram of a deep learning model is shown in Figure 3, which shows the three layers that make up a DL based system with some emphasis on the positions of activation functions, represented by the dark shaded region in the respective blocks.

Fig. 3. Block diagram of a DL based system model showing the activation function

The input layer accepts the data for training the neural network which comes in various formats from images, videos, texts, speech, sounds, or numeric data, while the hidden layers are made up of mostly the convolutional and pooling layers, of which the convolutional layers detect the local the patterns and features in data from the previous layers, presented in array-like forms for images while the pooling layers semantically merges similar features into one. The output layer presents the network results which are often controlled by AFs, specially to perform classifications or predictions, with associated probabilities. The position of an AF in a network structure depends on its function in the network thus when the AF is placed after the hidden layers, it converts the learned linear mappings into non-linear forms for propagation while in the output layer, it performs predictions.

The deep architectures are composed of several processing layers with each, involving both linear and non-linear operations, that are learned together to solve a given task. These deeper networks come with better performances though common issues likes vanishing gradients and exploding gradient arises, as a result of the derivative terms which are usually less than 1. With successive multiplication of this derivative terms, the value becomes smaller and smaller and tends to zero, thus the gradient vanishes. Consequently, if the values are greater than 1, successive multiplication will increase the values and the gradient tends to infinity thereby exploding the gradient. Thus, the AFs maintains the values of these gradients to specific limits. These are achieved using different mathematical functions and some of the early proposals of activation functions, used for neural network computing were explored by Elliott, 1993 as he studied the usage of the AFs in neural network.

The compilation of the existing activation functions is outlined with the advantages offered by most of the respective functions as highlighted by the authors as found in the literature.

Fundamentals of ANN

Neural computing is an information processing paradigm, inspired by biological system composed of a large number of highly interconnected processing elements(neurons) working in unison to solve specific problems.

Artificial neural networks (ANNs), like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

The Biological Neuron

The human brain consists of a large number, more than a billion of neural cells that process information. Each cell works like a simple processor. The massive interaction between all cells and their parallel processing only makes the brain's abilities possible. Figure 1 represents a human biological nervous unit. Various parts of biological neural network(BNN) is marked in Figure 1.

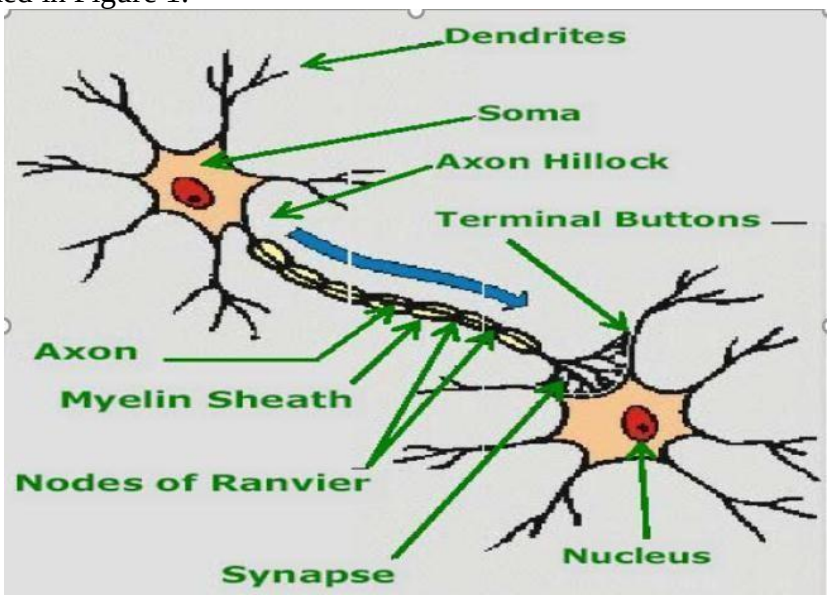


Figure 1: Biological Neural Network

Information flow in a neural cell The input/output and the propagation of information are shown below.

Artificial neuron model

An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.

□ The McCulloch-Pitts Neuron

This is a simplified model of real neurons, known as a Threshold Logic Unit.

□ A set of input connections brings in activations from other neuron.

□ A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing/transfer/threshold function).

□ An output line transmits the result to other neurons.

Basic Elements of ANN:

Neuron consists of three basic components –weights, thresholds and a single activation function. An Artificial neural network(ANN) model based on the biological neural sytems is shown in figure 2.

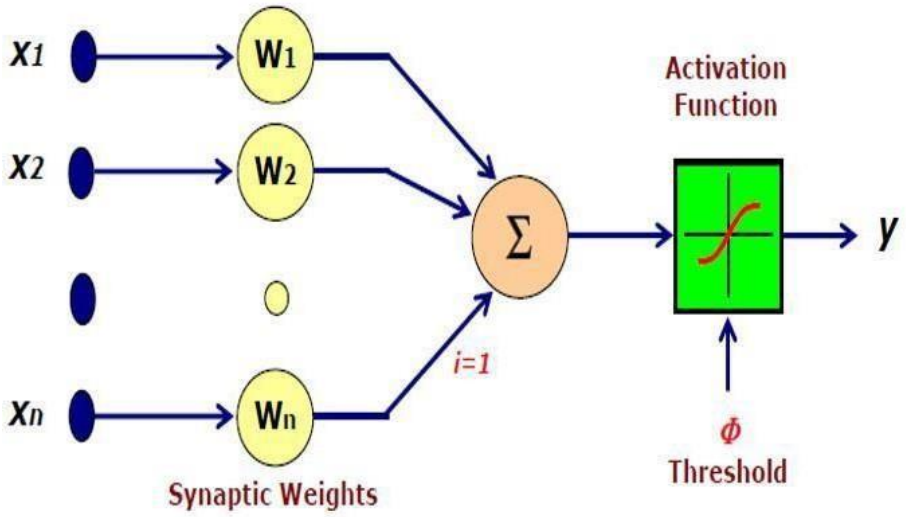


Figure 2: Basic Elements of Artificial Neural Network

Different Learning Rules

A brief classification of Different Learning algorithms is depicted in figure 3.

⊓ Training: It is the process in which the network is taught to change its weight and bias.

⊓ Learning: It is the internal process of training where the artificial neural system learns to update/adapt the weights and biases.

Different Training /Learning procedure available in ANN are

⊓ **Supervised learning**
⊓ **Unsupervised learning**
⊓ **Reinforced learning**
⊓ **Hebbian learning**
⊓ **Gradient descent learning**
⊓ **Competitive learning**
⊓ **Stochastic learning**

Requirements of Learning Laws:

• Learning Law should lead to convergence of weights

• Learning or training time should be less for capturing the information from the training Pairs • Learning should use the local information

• Learning process should able to capture the complex non linear mapping available between the input & output pairs

• Learning should able to capture as many as patterns as possible

• Storage of pattern information's gathered at the time of learning should be high for the given network

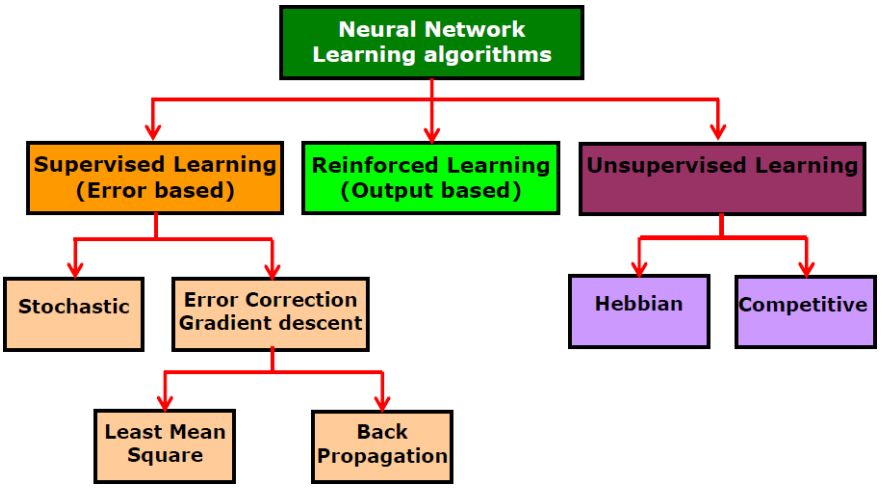


Figure 3: Different Training methods of Artificial Neural Network

Supervised learning :

Every input pattern that is used to train the network is associated with an output pattern which is the target or the desired pattern. A teacher is assumed to be present during the training process, when a comparison is made between the network's computed output and the correct expected output, to determine the error. The error can then be used to change network parameters, which result in an improvement in performance.

Unsupervised learning:

In this learning method the target output is not presented to the network. It is as if there is no teacher to present the desired patterns and hence the system learns of its own by discovering and adapting to structural features in the input patterns.

Reinforced learning:

In this method, a teacher though available, does not present the expected answer but only indicates if the computed output correct or incorrect. The information provided helps the network in the learning process**.**

Hebbian learning:

This rule was proposed by Hebb and is based on correlative weight adjustment. This is the oldest learning mechanism inspired by biology. In this, the input-output pattern pairs (xi , yi) are associated by the weight matrix W, known as the correlation matrix.

It is computed as W = Σ $xiyi$

T ni=1

Here yi T is the transpose of the associated output vector yi .Numerous variants of therule have been proposed.

1.4.1.5 Gradient descent learning:

This is based on the minimization of error E defined in terms of weights and activation function of the network. Also it is required that the activation function employed by the network is differentiable, as the weight update is dependent on the gradient of the error E.

Thus if Δwij is the weight update of the link connecting the ith and jth neuron of the twoneighbouring layers, then Δwij is defined as,

$$\Delta wij = \eta \frac{\partial E}{\partial wij}$$

Where, η is the learning rate parameter and ∂E ∂wij is the error gradient with reference to the weight wij.

Perceptron Model

Simple Perceptron for Pattern Classification

Perceptron network is capable of performing pattern classification into two or more categories. The perceptron is trained using the perceptron learning rule. We will first consider classification into two categories and then the general multiclass classification later. For classification into only two categories, all we need is a single output neuron. Here we will use bipolar neurons. The simplest architecture that could do the job consists of a layer of N input neurons, an output layer with a single output neuron, and no hidden layers. This is the same architecture as we saw before for Hebb learning. However, we will use a different transfer function here for the output neurons as given below in eq (7). Figure 7 represents a single layer perceptron network.

$$y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

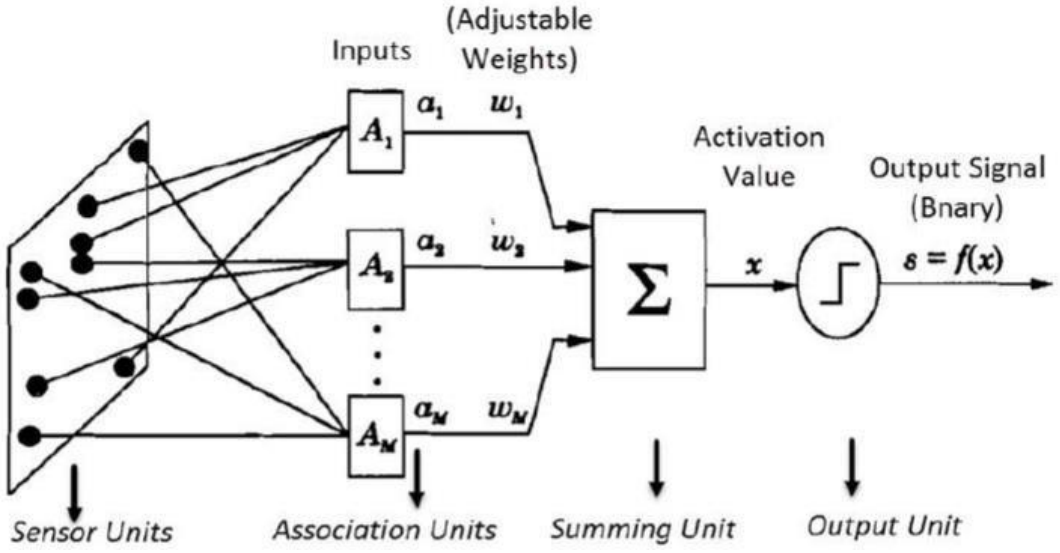


Figure 4: Single Layer Perceptron

Equation 7 gives the bipolar activation function which is the most common function used in the perceptron networks. Figure 7 represents a single layer perceptron network. The inputs arising from the problem space are collected by the sensors and they are fed to the association units. Association units are the units which are responsible to associate the inputs based on their similarities. This unit groups the similar inputs hence the name association unit. A single input from each group is given to the summing unit.Weights are randomnly fixed intially and assigned to this inputs. The net value is calculate by using the expression

$x = \Sigma\, wiai - \theta$ _____eq(8)

This value is given to the activation function unit to get the final output response. The actual output is compared with the Target or desired .If they are same then we can stop training else the weights has to be updated .It means there is error .Error is given as **δ = b-s** , where b is the desired / Target output and S is the actual outcome of the machine here the weights are updated based on the perceptron Learning law as given in equation 9.

Weight change is given as **Δw= η δ ai.** So new weight is given as

Wi (new) = Wi (old) + Change in weight vector (**Δw**)_____eq(9)

Perceptron Algorithm

Step 1: Initialize weights and bias. For simplicity, set weights and bias to zero. Set learning rate in the range of zero to one.

• Step 2: While stopping condition is false do steps 2-6
• Step 3: For each training pair s:t do steps 3-5
• Step 4: Set activations of input units *xi = ai*
• Step 5: Calculate the summing part value Net = Σ aiwi-θ
• Step 6: Compute the response of output unit based on the activation functions
• Step 7: Update weights and bias if an error occurred for this pattern(if *yis not equal to t)*
Weight (new) = wi(old) + atxi , & bias (new) = b(old) + at
Else *wi(new) = wi(old) & b(new) = b(old)*
• Step 8: Test Stopping Condition

Limitations of single layer perceptrons:

• Uses only Binary Activation function
• Can be used only for Linear Networks
• Since uses Supervised Learning ,Optimal Solution is provided
• Training Time is More
• Cannot solve Linear In-separable Problem

Multi-Layer Perceptron Model:

Figure 8 is the general representation of Multi layer Perceptron network. In between the input and output Layer there will be some more layers also known as Hidden layers.

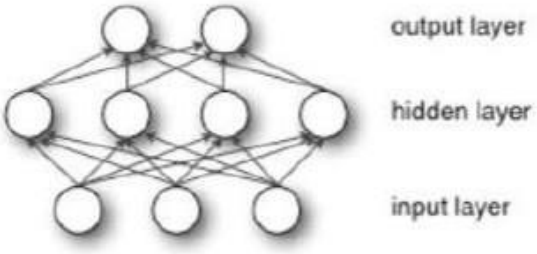


Figure 5: Multi-Layer Perceptron

Multi Layer Perceptron Algorithm

1. Initialize the weights (Wi) & Bias (B0) to small random values near Zero
2. Set learning rate η or α in the range of "0" to "1"
3. Check for stop condition. If stop condition is false do steps 3 to 7
4. For each Training pairs do step 4 to 7
5. Set activations of Output units: xi = si for i=1 to N
6. Calculate the output Response
yin = b0 + Σ xiwi
7. Activation function used is Bipolar sigmoidal or Bipolar Step functions
For Multi Layer networks, based on the number of layers steps 6 & 7 are repeated
8. If the Targets is (not equal to) = to the actual output (Y), then update weights and bias based on Perceptron Learning Law
Wi (new) = Wi (old) + Change in weight vector
Change in weight vector = ηtixi
Where η = Learning Rate
ti = Target output of ith unit
xi = ith Input vector

b0(new) = b0 (old) + Change in Bias
Change in Bias = ηti
Else Wi (new) = Wi (old)
b0(new) = b0 (old)
9. Test for Stop condition

Linearly separable & Linear in separable tasks:

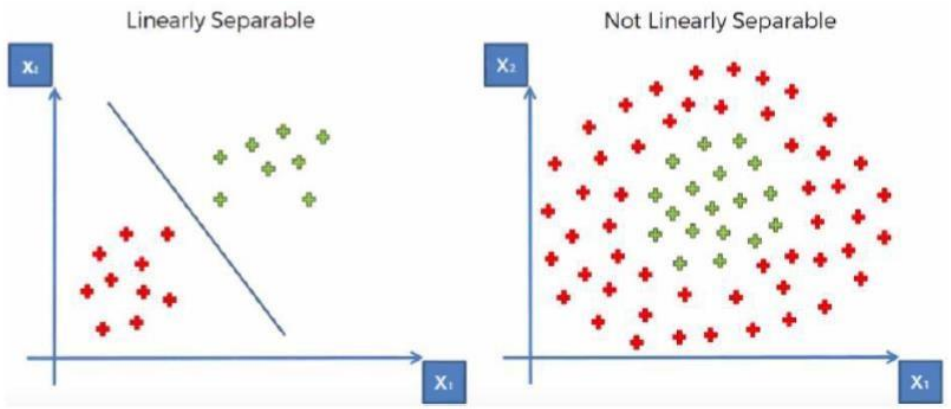


Figure 6: Representation of Linear separable & Linear-in separable Tasks

Perceptron are successful only on problems with a linearly separable solution space .Figure 9 represents both linear separable as well as linear in seperable problem. Perceptron cannot handle, in particular, tasks which are not linearly separable.(Known as linear inseparable problem).Sets of points in two dimensional spaces are linearly separable if the sets can be seperated by a straight line. Generalizing, a set of points in n-dimentional space are that can be seperated by a straight line is called Linear seperable as represented in figure 9.
Single layer perceptron can be used for linear separation.Example AND gate.But it cant be used for non linear ,inseparable problems.(Example XOR Gate).Consider figure 10.

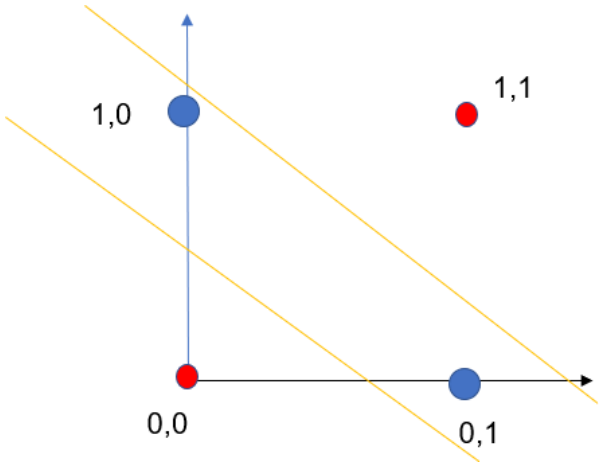


Figure 7: XOR representation (Linear-in separable Task)

Here a single decision line cannot separate the Zeros and Ones Linearly. At least Two lines are required to separate Zeros and Onesas shown in Figure 10. Hence single layer

networks can not be used to solve inseparable problems. To over come this problem we go for creation of **convex regions**.

Convex regions can be created by multiple decision lines arising from multi layer networks. Single layer network cannot be used to solve inseparable problem. Hence we go for multilayer network there by creating convex regions which solves the inseparable problem.

Convex Region:

Select any Two points in a region and draw a straight line between these two points. If the points selected and the lines joining them both lie inside the region then that region is **known as convex regions.**

Types of convex regions

(a) Open Convex region (b) Closed Convex region

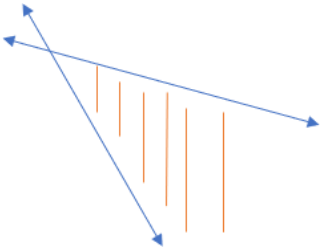


Figure 8: Open convex region

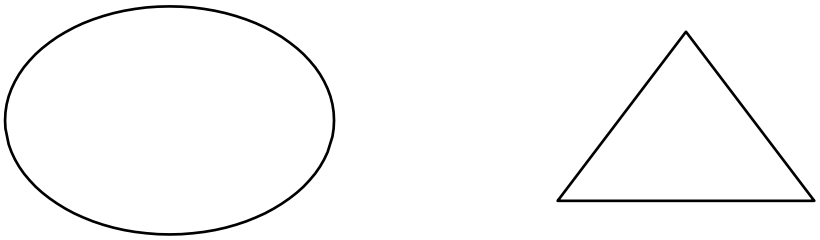


Figure 9 A: Circle - Closed convex region Figure 9 B: Triangle - Closed convex region

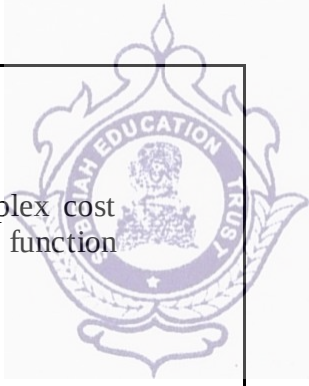
Logistic Regression

Logistic regression is a probabilistic model that organizes the instances in terms of probabilities. Because the classification is probabilistic, a natural method for optimizing the parameters is to ensure that the predicted probability of the observed class for each training occurrence is as large as possible. This goal is achieved by using the notion of maximumlikelihood estimation in order to learn the parameters of the model. The likelihood of the training data is defined as the product of the probabilities of the observed labels of each training instance. Clearly, larger values of this objective function are better. By using the negative logarithm of this value, one obtains a loss function in minimization form. Therefore, the output node uses the negative log-likelihood as a loss function. This loss function replaces the squared error used in the Widrow-Hoff method.

The output layer can be formulated with the sigmoid activation function, which is very common in neural network design.

Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In classification problems, we have dependent variables in a binary or discrete format such as 0 or 1.

- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.
- It is a predictive analysis algorithm which works on the concept of probability.
- Logistic regression is a type of regression, but it is different from the linear regression algorithm in the term how they are used.



□ Logistic regression uses sigmoid function or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression. The function can be represented as:

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

Where f(x)= Output between the 0 and 1 value.
x= input to the function
e= base of natural logarithm.

When we provide the input values (data) to the function, it gives the Scurve as follows: It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.

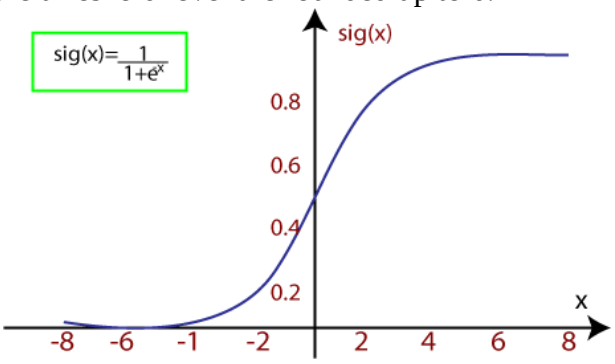


Figure 10: Circle – Logistic Function

Support Vector Machines

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

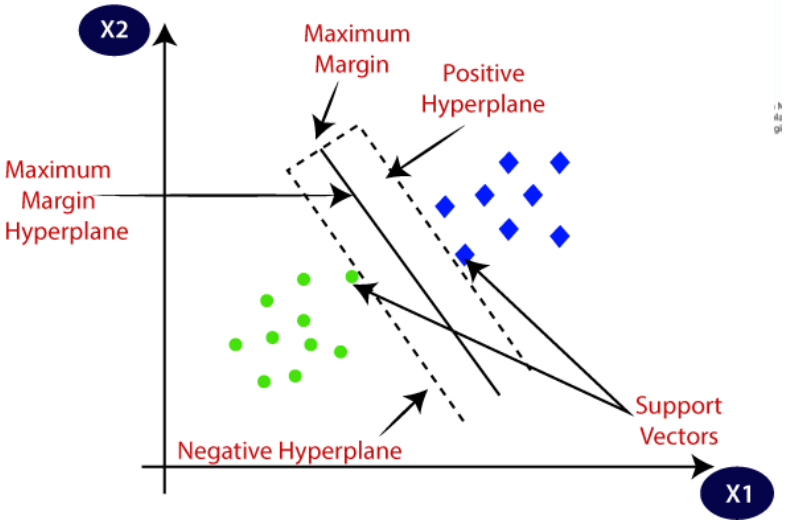


Figure 11: SVM – Classification

SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair (x1, x2) of coordinates in either green or blue. Consider the below image figure11. It is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

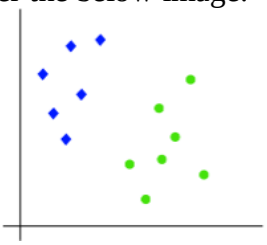


Figure 12A: SVM – Input Space

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.

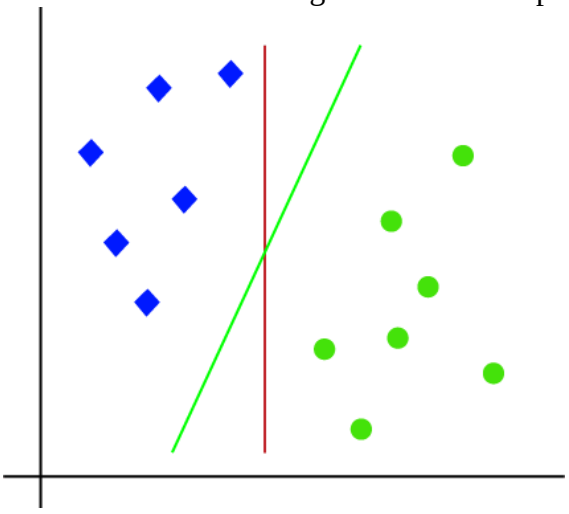


Figure 12B: SVM – Linear Classification

Gradient Descent:

Gradient Descent is a popular optimization technique in Machine Learning and Deep Learning, and it can be used with most, if not all, of the learning algorithms. A gradient is the slope of a function. It measures the degree of change of a variable in response to the changes of another variable. Mathematically, Gradient Descent is a convex function whose output is the partial derivative of a set of parameters of its inputs. The greater the gradient, the steeper the slope. Starting from an initial value, Gradient Descent is run iteratively to find the optimal values of the parameters to find the minimum possible value of the given cost function.

Types of Gradient Descent:

Typically, there are three types of Gradient Descent:

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-batch Gradient Descent

Stochastic Gradient Descent (SGD):

The word 'stochastic' means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big.

Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

History of Deep Learning [DL]:

- The chain rule that underlies the back-propagation algorithm was invented in the seventeenth century (Leibniz, 1676; L'Hôpital, 1696)
- Beginning in the 1940s, the function approximation techniques were used to motivate machine learning models such as the perceptron
- The earliest models were based on linear models. Critics including Marvin Minsky pointed out several of the flaws of the linear model family, such as its inability to learn the XOR function, which led to a backlash against the entire neural network approach
- Efficient applications of the chain rule based on dynamic programming began to appear in the 1960s and 1970s
- Werbos (1981) proposed applying chain rule techniques for training artificial neural networks. The idea was finally developed in practice after being independently rediscovered in different ways (LeCun, 1985; Parker, 1985; Rumelhart et al., 1986a)
- Following the success of back-propagation, neural network research gained popularity and reached a peak in the early 1990s. Afterwards, other machine learning techniques became more popular until the modern deep learning renaissance that began in 2006
- The core ideas behind modern feedforward networks have not changed substantially since the 1980s. The same back-propagation algorithm and the same approaches to gradient descent are still in use.

Most of the improvement in neural network performance from 1986 to 2015 can be attributed to two factors. First, larger datasets have reduced the degree to which statistical generalization is a challenge for neural networks. Second, neural networks have become much

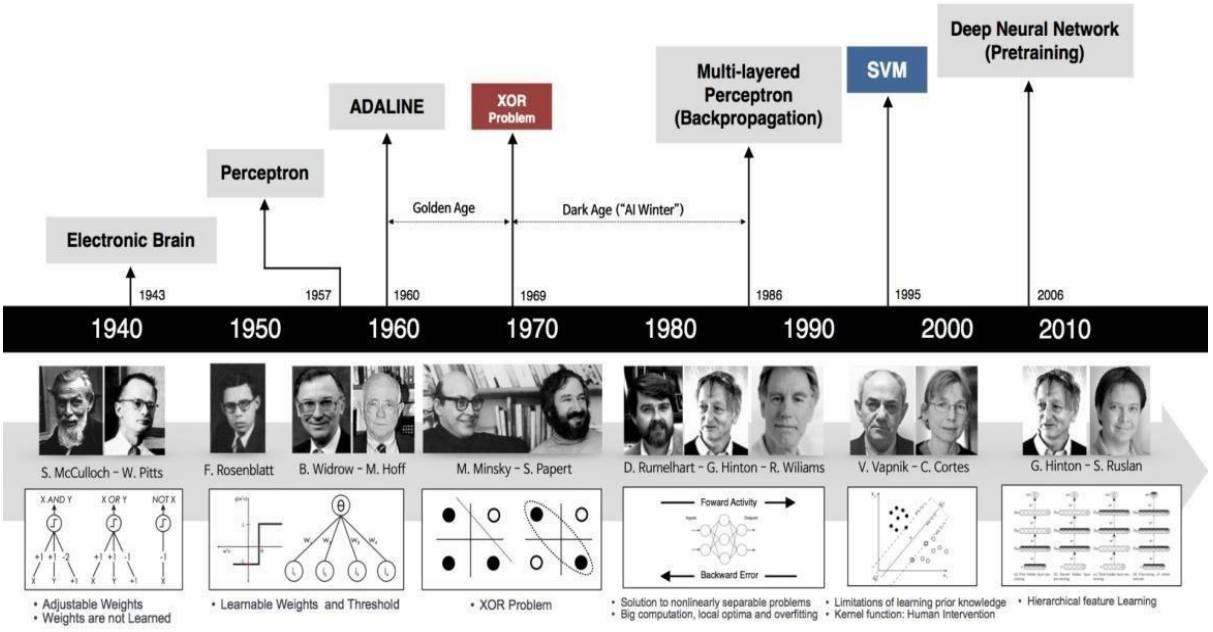
larger, because of more powerful computers and better software infrastructure.

A small number of algorithmic changes have also improved the performance of neural networks noticeably. One of these algorithmic changes was the replacement of mean squared error with the cross-entropy family of loss functions. Mean squared error was popular in the 1980s and 1990s but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learning community.

The other major algorithmic change that has greatly improved the performance of feedforward networks was the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units. Rectification using the max{0, z} function was introduced in early neural network models and dates back at least as far as the Cognitron and Neo-Cognitron (Fukushima, 1975, 1980).

For small datasets, Jarrett et al. (2009) observed that using rectifying nonlinearities is even more important than learning the weights of the hidden layers. Random weights are sufficient to propagate useful information through a rectified linear network, enabling the classifier layer at the top to learn how to map different feature vectors to class identities. When more data is available, learning begins to extract enough useful knowledge to exceed the performance of randomly chosen parameters.2011a) showed that learning is far easier in deep rectified linear networks than in deep networks that have curvature or two-sided saturation in their activation functions.

When the modern resurgence of deep learning began in 2006, feedforward networks continued to have a bad reputation. From about 2006 to 2012, it was widely believed that feedforward networks would not perform well unless they were assisted by other models, such as probabilistic models. Today, it is now known that with the right resources and engineering practices, feedforward networks perform very well. Today, gradient-based learning in feedforward networks is used as a tool to develop probabilistic models. Feedforward networks continue to have unfulfilled potential. In the future, we expect they will be applied to many more tasks, and that advances in optimization algorithms and model design will improve their performance even further.



A Probabilistic Theory of Deep Learning

Probability is the science of quantifying uncertain things. Most of machine learning and deep learning systems utilize a lot of data to learn about patterns in the data. Whenever data is

utilized in a system rather than sole logic, uncertainty grows up and whenever uncertainty grows up, probability becomes relevant.

By introducing probability to a deep learning system, we introduce common sense to the system. In deep learning, several models like Bayesian models, probabilistic graphical models,

Hidden Markov models are used. They depend entirely on probability concepts. Real world data is chaotic. Since deep learning systems utilize real world data, they require a tool to handle the chaoticness.

Back Propagation Networks (BPN)

Need for Multilayer Networks

- Single Layer networks cannot used to solve Linear Inseparable problems & can only be used to solve linear separable problems
- Single layer networks cannot solve complex problems
- Single layer networks cannot be used when large input-output data set is available
- Single layer networks cannot capture the complex information's available in the training pairs.Hence to overcome the above said Limitations we use Multi-Layer Networks.

Multi-Layer Networks

- Any neural network which has at least one layer in between input and output layers is called Multi-Layer Networks
- Layers present in between the input and out layers are called Hidden Layers
- Input layer neural unit just collects the inputs and forwards them to the next higher layer
- Hidden layer and output layer neural units process the information's feed to them and produce an appropriate output
- Multi -layer networks provide optimal solution for arbitrary classification problems
- Multi -layer networks use linear discriminants, where the inputs are non linear

Back Propagation Networks (BPN)

Introduced by Rumelhart, Hinton, & Williams in 1986. BPN is a Multilayer Feedforward Network but error is back propagated, Hence the name Back Propagation Network (BPN). It uses Supervised Training process; it has a systematic procedure for training the network and is used in Error Detection and Correction. Generalized Delta Law /Continuous Perceptron Law/ Gradient Descent Law is used in this network. Generalized Delta rule minimizes the mean squared error of the output calculated from the output. Delta law has faster convergence rate when compared with Perceptron Law. It is the extended version of Perceptron Training Law. Limitations of this law is the Local minima problem. Due to this the convergence speed reduces, but it is better than perceptron's. Figure 1 represents a BPN network architecture. Even though Multi level perceptron's can be used they are flexible and efficient that BPN. In figure 1 the weights between input and the hidden portion is considered as Wij and the weight between first hidden to the next layer is considered as Vjk. This network is valid only for Differential Output functions. The Training process used in backpropagation involves three stages, which are listed as below

1. Feedforward of input training pair
2. Calculation and backpropagation of associated error
3. Adjustments of weights

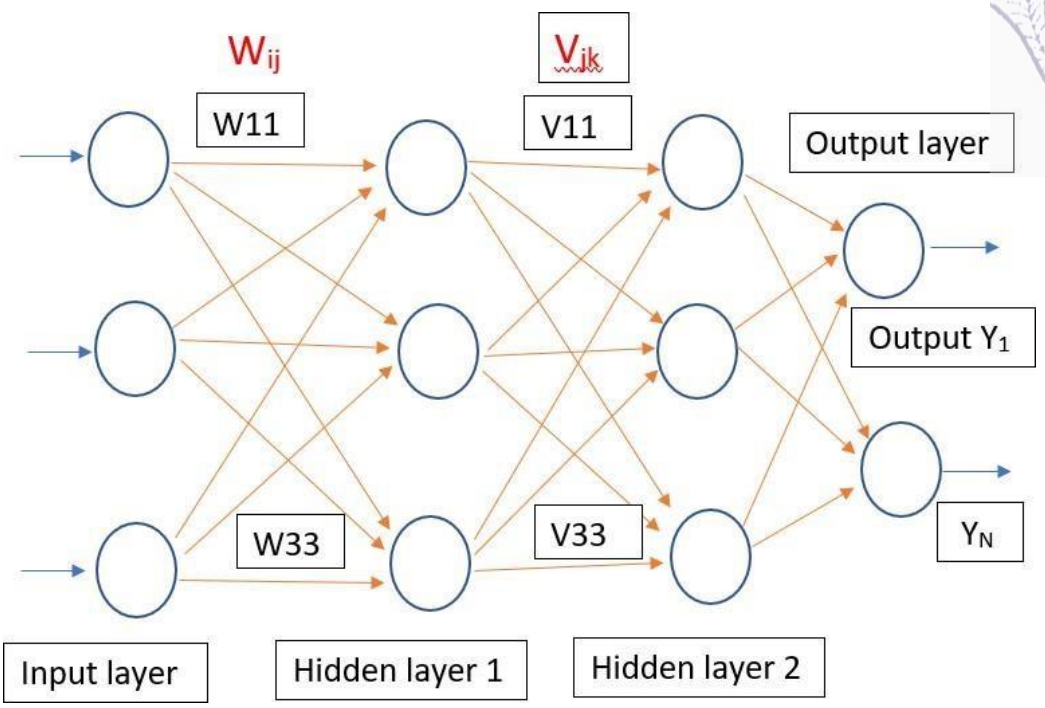


Figure 1: Back Propagation Network

BPN Algorithm

The algorithm for BPN is as classified int four major steps as follows:

1. Initialization of Bias, Weights
2. Feedforward process
3. Back Propagation of Errors
4. Updating of weights & biases

Algorithm:

I. Initialization of weights:

Step 1: Initialize the weights to small random values near zero

Step 2: While stop condition is false , Do steps 3 to 10

Step 3: For each training pair do steps 4 to 9

II. Feed forward of inputs

Step 4: Each input xi is received and forwarded to higher layers (next hidden)

Step 5: Hidden unit sums its weighted inputs as follows

$Zinj = Woj + \Sigma xiwij$

Applying Activation function

$Zj = f(Zinj)$

This value is passed to the output layer

Step 6: Output unit sums it's weighted inputs

$yink = Voj + \Sigma ZjVjk$

Applying Activation function

$Yk = f(yink)$

III. Backpropagation of Errors

Step 7: $\delta k = (tk - Yk)f(yink)$

Step 8: $\delta inj = \Sigma \delta jVjk$

IV. Updating of Weights & Biases

Step 8: Weight correction is $\Delta wij = \alpha \delta kZj$

bias Correction is $\Delta woj = \alpha \delta k$

V. Updating of Weights & Biases

Step 9: continued:

New Weight is

Wij(new) = Wij(old) + Δwij

Vjk(new) = Vjk(old) + ΔVjk

New bias is

Woj(new) = Woj(old) + Δwoj

Vok(new) = Vok(old) + ΔVok

Step 10: Test for Stop Condition

2.2.5 Merits

- Has smooth effect on weight correction
- Computing time is less if weight's are small
- 100 times faster than perceptron model
- Has a systematic weight updating procedure

Demerits

- Learning phase requires intensive calculations
- Selection of number of Hidden layer neurons is an issue
- Selection of number of Hidden layers is also an issue
- Network gets trapped in Local Minima
- Temporal Instability
- Network Paralysis
- Training time is more for Complex problems

Shallow Networks:

Shallow neural networks give us basic idea about deep neural network which consist of only 1 or 2 hidden layers. Understanding a shallow neural network gives us an understanding into what exactly is going on inside a deep neural network A neural network is built using various hidden layers. Now that we know the computations that occur in a particular layer, let us understand how the whole neural network computes the output for a given input *X*. These can also be called the *forward-propagation* equations.

$$Z^{[1]} = W^{[1]T} X + b^{[1]}$$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

$$Z^{[2]} = W^{[2]T} A^{[1]} + b^{[2]}$$

$$\hat{y} = A^{[2]} = \sigma\left(Z^{[2]}\right)$$

1. The first equation calculates the intermediate output ***Z[1]*** of the first hidden layer.
2. The second equation calculates the final output ***A[1]*** of the first hidden layer.
3. The third equation calculates the intermediate output ***Z[2]*** of the output layer.
4. The fourth equation calculates the final output ***A[2]*** of the output layer which is also the final output of the whole neural network.

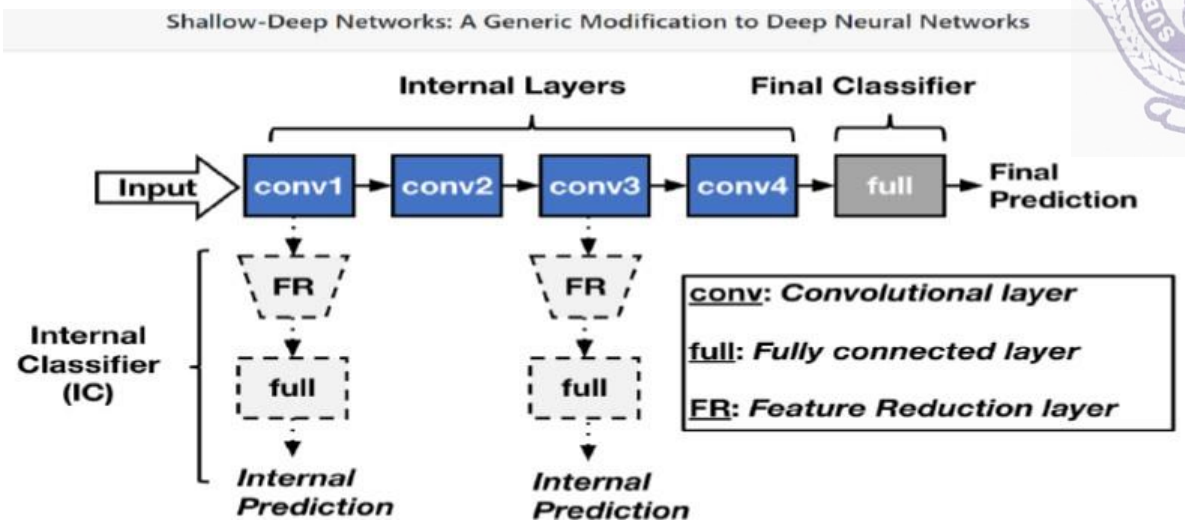


Figure 2: Shallow Networks – Generic Model

Unit saturation (aka the vanishing gradient problem) – ReLU

Introduction to Vanishing Gradient Problem

In Machine Learning, the Vanishing Gradient Problem is encountered while training Neural Networks with gradient-based methods (example, Back Propagation). This problem makes it hard to learn and tune the parameters of the earlier layers in the network.

The vanishing gradients problem is one example of unstable behaviour that you may encounter when training a deep neural network.

It describes the situation where a deep multilayer feed-forward network or a recurrent neural network is unable to propagate useful gradient information from the output end of the model back to the layers near the input end of the model.

The result is the general inability of models with many layers to learn on a given dataset, or for models with many layers to prematurely converge to a poor solution.

Methods proposed to overcome vanishing gradient problem

- Multi-level hierarchy
- Long short – term memory
- Faster hardware
- Residual neural networks (ResNets)
- ReLU

Rectified Linear Unit (ReLU) Function

The rectified linear unit (ReLU) activation function was proposed by Nair and Hinton 2010, and ever since, has been the most widely used activation function for deep learning applications with state-of-the-art results to date. The ReLU is a faster learning AF, which has proved to be the most successful and widely used function. It offers the better performance and generalization in deep learning compared to the Sigmoid and tanh activation functions. The ReLU represents a nearly linear function and therefore preserves the properties of linear models that made them easy to optimize, with gradient-descent methods.

The ReLU activation function performs a threshold operation to each input element where values less than zero are set to zero thus the ReLU is given by

$$f(x) = \max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases}$$

This function rectifies the values of the inputs less than zero thereby forcing them to zero and eliminating the vanishing gradient problem observed in the earlier types of activation function. The ReLU function has been used within the hidden units of the deep neural networks with another AF, used in the output layers of the network with typical examples found in object classification and speech recognition applications.

The main advantage of using the rectified linear units in computation is that, they guarantee faster computation since it does not compute exponentials and divisions, with overall speed of computation enhanced. Another property of the ReLU is that it introduces sparsity in the hidden units as it squishes the values between zero to maximum. However, the ReLU has a limitation that it easily overfits compared to the sigmoid function although the dropout technique has been adopted to reduce the effect of overfitting of ReLUs and the rectified networks improved performances of the deep neural networks.

The ReLU and its variants have been used in different architectures of deep learning, which include the restricted Boltzmann machines and the convolutional neural network architectures, although (Nair and Hinton, 2010) outlined that the ReLU has been used in numerous architectures because of its simplicity and reliability.

The ReLU has a significant limitation that it is sometimes fragile during training thereby causing some of the gradients to die. This leads to some neurons being dead as well, thereby causing the weight updates not to activate in future data points, thereby hindering learning as dead neurons gives zero activation. To resolve the dead neuron issues, the leaky ReLU was proposed.

Hyperparameter Optimization:

Hyperparameter optimization in machine learning intends to find the hyperparameters of a given machine learning algorithm that deliver the best performance as measured on a validation set. Hyperparameters, in contrast to model parameters, are set by the machine earning engineer before training. The number of trees in a random forest is a hyperparameter while the weights in a neural network are model parameters learned during training.

Hyperparameter optimization finds a combination of hyperparameters that returns an optimal model which reduces a predefined loss function and in turn increases the accuracy on given independent data.

Hyperparameter Optimization methods

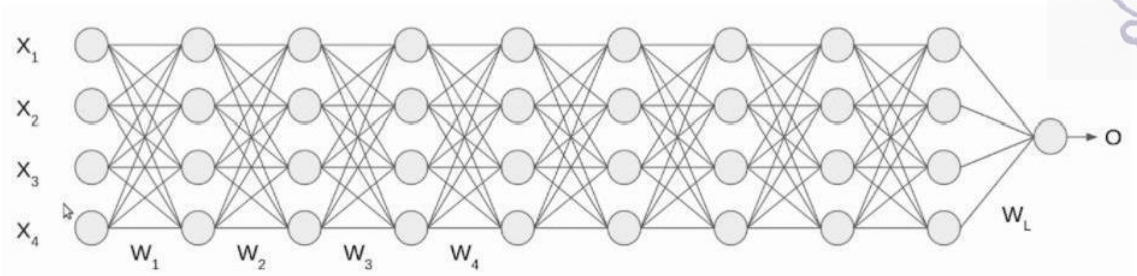
- Manual Hyperparameter Tuning
- Grid Search
- Random Search
- Bayesian Optimization
- Gradient-based Optimization

Batch Normalization:

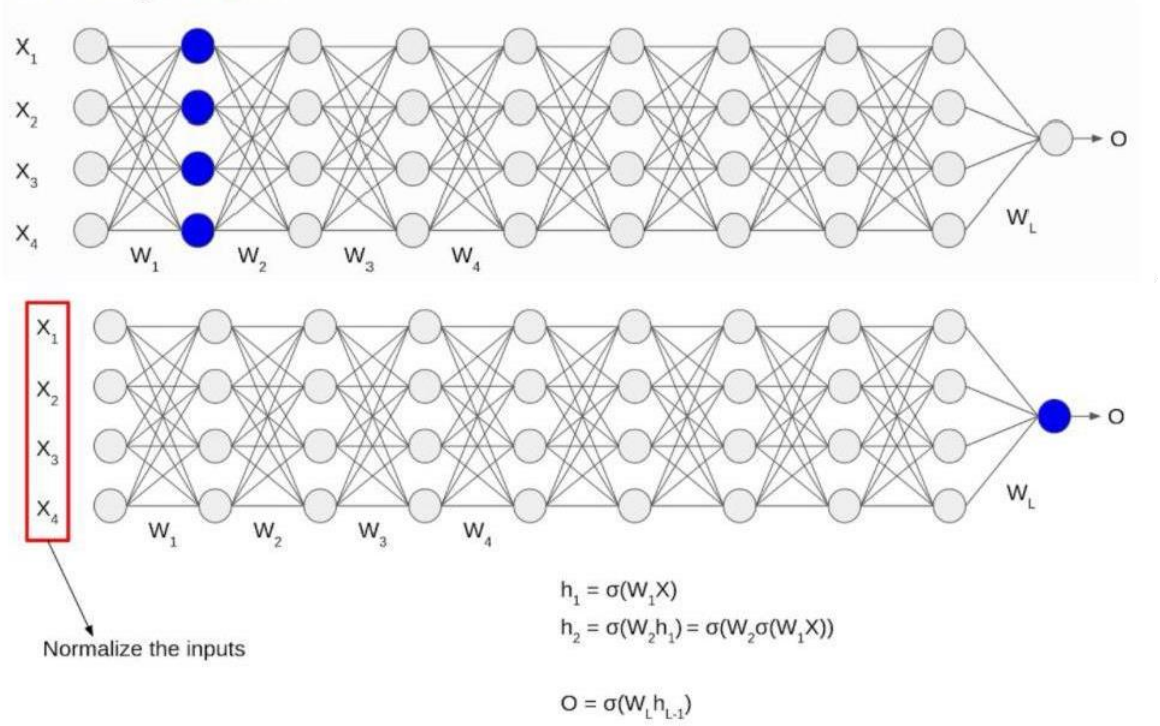
It is a method of adaptive reparameterization, motivated by the difficulty of training very deep models. In Deep networks, the weights are updated for each layer. So the output will no longer be on the same scale as the input (even though input is normalized). Normalization - is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.

when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale because, we ensure that our model can generalize appropriately.(Normalization is used to bring the input into a balanced scale/ Range).

Let's understand this through an example, we have a deep neural network as shown in the following image.



Initially, our inputs X1, X2, X3, X4 are in normalized form as they are coming from the pre-processing stage. When the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of input X and the weight matrix W.



Even though the input X was normalized but the output is no longer on the same scale. The data passes through multiple layers of network with multiple times(sigmoidal) activation functions are applied, which leads to an internal co-variate shift in the data.

This motivates us to move towards Batch Normalization. Normalization is the process of altering the input data to have mean as zero and standard deviation value as one.

Procedure to do Batch Normalization:

(1) Consider the batch input from layer h, for this layer we need to calculate the mean of this hidden activation.

(2) After calculating the mean the next step is to calculate the standard deviation of the hidden activations.

(3) Now we normalize the hidden activations using these Mean & Standard Deviation values. To do this, we subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).

(4) As the final stage, the re-scaling and offsetting of the input is performed. Here two components of the BN algorithm is used, γ(gamma) and β (beta).

These parameters are used for re-scaling (γ) and shifting(β) the vector contains values from the previous operations.

These two parameters are learnable parameters, Hence during the training of neural network the optimal values of γ and β are obtained and used. Hence we get the accurate normalization of each batch.

Regularization

A fundamental problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs. Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization.

Definition: - "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

□ In the context of deep learning, most regularization strategies are based on regularizing estimators.

□ Regularization of an estimator works by trading increased bias for reduced variance.

An effective regularizer is one that makes a profitable trade, reducing variance significantly while not overly increasing the bias.

□ Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J. We denote the regularized objective function by J˜

$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$

where $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, relative to the standard objective function J. Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.

□ The parameter norm penalty Ω that penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized.

Dropout

• Dropout is one of the most popular regularization techniques for deep neural networks.

» Even the state-of-the-art neural networks get a 1–2% accuracy boost simply by adding dropout.

• At every training step, every neuron (including the input neurons, but always excluding the output neurons) has a probability p of being temporarily "dropped out,"

» It will be entirely ignored during this training step, but it may be active during the next step.

• The hyperparameter p is called the dropout rate

» typically set between 10% and 50%

» closer to 20– 30% in recurrent neural nets

» closer to 40–50% in convolutional neural networks

• After training, neurons don't get dropped anymore.

Neurons trained with dropout cannot co-adapt with their neighbouring neurons.

» Since each neuron can be either present or absent, there are a total of 2N possible networks (where N is the total number of droppable neurons).

» The resulting neural network can be seen as an averaging ensemble of all these smaller neural networks.

• Suppose p = 50%, in which case during testing a neuron would be connected to twice as many input neurons as it would be (on average) during training.

» To compensate for this fact, we need to multi- ply each neuron's input connection weights by 0.5 after training.
• More generally, we need to multiply each input connection weight by the keep probability (1 – p) after training.

• If you observe that the model is overfitting, you can increase the dropout rate.
• If you observe that the model is underfitting, you can decrease the dropout rate.
• It can also help to increase the dropout rate for large layers, and reduce it for small ones.
• Many state-of-the-art architectures only use dropout after the last hidden layer, so you may want to try this if full dropout is too strong.
• Dropout does tend to significantly slow down convergence.
• If you want to regularize a self-normalizing network based on the SELU activation function (as discussed earlier), you should use alpha dropout.