Semester-II

19CSP205B MACHINE LEARNING 4H – 4C

Instruction Hours / week: L: 4 T: 0 P: 0 **Marks:** Internal:**40** External:**60** Total: **100**

End Semester Exam : 3 Hours

Course Objectives

- To enable the student to learn techniques in machine learning.
- To understood the techniques in machine learning and apply machine learning techniques

Course Outcomes(COs)

On successful completion of the course the student should be

1. Presents the foundations of learning, linear models, distance based models, and tree and rule based model and reinforcement learning.

Unit I - FOUNDATIONS OF LEARNING

Components of learning – learning models – geometric models – probabilistic models – logic models – grouping and grading – learning versus design – types of learning – supervised – unsupervised – reinforcement – theory of learning – feasibility of learning – error and noise – training versus testing – theory of generalization – generalization bound – bias and variance – learning curve

Unit II - LINEAR MODELS

Linear classification – univariate linear regression – multivariate linear regression – regularized regression – Logistic regression – perceptrons – multilayer neural networks – learning neural networks structures – support vector machines – soft margin SVM – generalization and over fitting – regularization – validation

Unit III - DISTANCE-BASED MODELS

Nearest neighbor models – K-means – clustering around medoids – silhouttes – hierarchical clustering – k- d trees – locality sensitive hashing – non - parametric regression – ensemble learning – bagging and random forests – boosting – meta silhouttes – hierarchical clustering – k- d trees – locality sensitive hashing – non - parametric regression – ensemble learning – bagging and random forests – boosting – meta learning

Unit IV - TREE AND RULE MODELS

Decision trees – learning decision trees – ranking and probability estimation trees – Regression trees – clustering trees – learning ordered rule lists – learning unordered rule lists – descriptive rule learning – association rule mining – first -order rule learning

Unit V - REINFORCEMENT LEARNING

Passive reinforcement learning – direct utility estimation – adaptive dynamic programming – temporal - difference learning – active reinforcement learning – exploration – learning an action utility function – Generalization in reinforcement learning – policy search – applications in game playing – applications in robot control

SUGGESTED READINGS

1. Y. S. Abu - Mostafa, M. Magdon-Ismail, and H.-T. Lin. (2012).Learning from Data, AMLBook Publishers.
2. P. Flach. (2012). “Machine Learning: The art and science of algorithms that make sense of data”, Cambridge University Press.
3. K. P. Murphy. (2012). Machine Learning: A probabilistic perspective, MIT Press,
4. C. M. Bishop. (2007). Pattern Recognition and Machine Learning. Springer.
5. D. Barber. (2012). Bayesian Reasoning and Machine Learning, Cambridge University Press.

WEB SITES

1. https://machinelearningmastery.com/linear-regression-for-machine-learning/
2. https://www.cambridge.org/core/books/machine-learning/distancebased-models/
3. https://dzone.com/articles/machine-learning-with-decision-trees
4. http://reinforcementlearning.ai-depot.com/

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2017 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

SUBJECT : MACHINE LEARNING

SEMESTER :II **L T P C**

SUBJECT CODE:19CSP205B CLASS :I M.Sc CS **4 0 0 4**

LECTURE PLAN

STAFF NAME: D.MANJULA

S.No	Lecture Duration (Hr)	Topics	Support Materials
		UNIT-I	
1.	1	FOUNDATIONS OF LEARNING : Components of learning – learning models	T1:1-5
2.	1	geometric models – probabilistic models – logic models – grouping and grading	T2:24
3.	1	learning versus design	T1:9
4.	1	types of learning – supervised – unsupervised – reinforcement	T1:11-14
5.	1	– theory of learning – feasibility of learning	T1:24
6.	1	– error and noise – training versus testing – theory of generalization	T1:27,39-53
7.	1	generalization bound –bias and variance – learning curve	T1:55-68
8.	1	Recapitulation and Discussion of Important Questions	
		Total No of Hours Planned For Unit – I	**8**

UNIT-II			
1.	1	LINEAR MODELS : Linear classification – univariate linear regression	**T1:77-82**
2.	1	multivariate linear regression – regularized regression – Logistic regression	T2:253
3.	1	perceptrons – multilayer neural networks	T2:262
4.	1	– learning neural networks structures – support vector machines	T2:270
5.	1	soft margin SVM – generalization and over fitting	T2:278
6.	1	regularization – validation	T1:126-153
7.	1	Recapitulation and Discussion of Important Questions	
		Total No of Hours Planned For Unit – II	**7**
UNIT-III			
1.	1	DISTANCE-BASED MODELS : Nearest neighbor models	W1
2.	1	K-means	W2
3.	1	clustering around medoids	W3
4.	1	silhouttes – hierarchical clustering	W4
5.	1	k- d trees – locality sensitive hashing	W5,W6
6	1	non - parametric regression, ensemble learning	W7
7	1	bagging and random forests, bagging, meta learning	W8
8	1	Recapitulation and Discussion of Important Questions	
		Total No of Hours Planned For Unit – III	**8**
UNIT-IV			
1.	1	TREE AND RULE MODELS : Decision trees	W12
2.	1	learning decision trees – ranking and probability estimation trees	W12
3.	1	Regression trees – clustering trees	T2:183,190
4.	1	learning ordered rule lists	T2:191
5.	1	learning unordered rule lists – descriptive rule learning	T2:214,228
6.	1	association rule mining	T2:233
7.	1	first -order rule learning	T2:228
8.	1	Recapitulation and Discussion of Important Questions	
		Total No of Hours Planned For Unit – IV	**8**
UNIT-V			
1.	1	REINFORCEMENT LEARNING :Passive	W9

		reinforcement learning	
2.	1	direct utility estimation – adaptive dynamic programming – temporal	W9
3.	1	- difference learning – active reinforcement learning – exploration	W9
4.	1	– learning an action utility function – Generalization in reinforcement learning	W9
5.	1	policy search – applications in game playing – applications in robot control	W9
6.	1	Recapitulation and Discussion of Important Questions	
7.	1	Discussion of Previous ESE Question Papers	
8.	1	Discussion of Previous ESE Question Papers	
9.	1	Discussion of Previous ESE Question Papers	
		Total No of Hours Planned For Unit – V	**9**
		Total No. of Hours Planned:	**40**

SUGGESTED READINGS

1. Y. S. Abu - Mostafa, M. Magdon-Ismail, and H.-T. Lin, “Learning from Data”, AMLBook

Publishers, 2012.

2. P. Flach, “Machine Learning: The art and science of algorithms that make sense of data”,

Cambridge University Press, 2012.

3. K. P. Murphy, “Machine Learning: A probabilistic perspective”, MIT Press, 2012.

4. C. M. Bishop, “Pattern Recognition and Machine Learning”, Springer, 2007.

5. D. Barber, “Bayesian Reasoning and Machine Learning”, Cambridge University Press, 2012.

WEBSITES:

W1: https://towardsdatascience.com/k-nearest-neighbours-introduction-to-machine-learningalgorithms-18e7ce3d802a

W2:Lhttps://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-

6a6e67336aa1

W3:https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_426

W4:https://en.wikipedia.org/wiki/Silhouette_(clustering)

W5:https://en.wikipedia.org/wiki/K-d_tree

W6:https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

W7:https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-andwhy-use-them-68ec3f9fef5f

W8:https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-formachine-learning/

W9:https://www.techemergence.com/machine-learning-in-gaming-building-ais-to-conquervirtual-worlds/

W10: https://machinelearningmastery.com/linear-regression-for-machine-learning/

W11: https://www.cambridge.org/core/books/machine-learning/distancebased-models/

W12:https://dzone.com/articles/machine-learning-with-decision-trees

W13:http://reinforcementlearning.ai-depot.com/

UNIT-I

SYLLABUS

FOUNDATIONS OF LEARNING : Components of learning – learning models – geometric models – probabilistic models – logic models – grouping and grading – learning versus design – types of learning – supervised – unsupervised – reinforcement – theory of learning – feasibility of learning – error and noise – training versus testing – theory of generalization – generalization bound –bias and variance – learning curve

FOUNDATION OF LEARNING:

What is Machine Learning?

Machine Learning is getting computers to program themselves. If programming is automation, then machine learning is automating the process of automation.

Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming, coined the term **"Machine Learning"**. He defined machine learning as – **"Field of study that gives computers the capability to learn without being explicitly programmed.**

A breakthrough in machine learning would be worth ten Microsofts.

— Bill Gates, Former Chairman, Microsoft

- Traditional Programming: Data and program is run on the computer to produce the output.
- Machine Learning: Data and output is run on the computer to create a program. This program can be used in traditional programming.

Machine learning is like farming or gardening. Seeds is the algorithms, nutrients is the data, the gardner is you and plants is the programs.

"A computer program is said to learn from experience E with some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." -Tom M. Mitchell

Examples of Machine Learning

There are many examples of machine learning. Here are a few examples of classification problems where the goal is to categorize objects into a fixed set of categories.

Face detection: Identify faces in images (or indicate if a face is present).

Email filtering: Classify emails into spam and not-spam.

Medical diagnosis: Diagnose a patient as a sufferer or non-sufferer of some disease.

Weather prediction: Predict, for instance, whether or not it will rain tomorrow.

Applications of Machine Learning

Sample applications of machine learning:

- **Web search**: ranking page based on what you are most likely to click on.
- **Computational** biology: rational design drugs in the computer based on past experiments.

- **Finance**: decide who to send what credit card offers to. Evaluation of risk on credit offers. How to decide where to invest money.
- **E-commerce**: Predicting customer churn. Whether or not a transaction is fraudulent.
- **Space exploration**: space probes and radio astronomy.
- **Robotics**: how to handle uncertainty in new environments. Autonomous. Self-driving car.
- **Information extraction**: Ask questions over databases across the web.
- **Social networks**: Data on relationships and preferences. Machine learning to extract value from data.
- **Debugging**: Use in computer science problems like debugging. Labor intensive process. Could suggest where the bug could be.

Key elements of machine learning

There are tens of thousands of machine learning algorithms and hundreds of new algorithms are developed every year.

Every machine learning algorithm has three components:

- **Representation**: how to represent knowledge. Examples include decision trees, sets of rules, instances, graphical models, neural networks, support vector machines, model ensembles and others.
- **Evaluation**: the way to evaluate candidate programs (hypotheses). Examples include accuracy, prediction and recall, squared error, likelihood, posterior probability, cost, margin, entropy k-L divergence and others.
- **Optimization**: the way candidate programs are generated known as the search process. For example combinatorial optimization, convex optimization, constrained optimization.

All machine learning algorithms are combinations of these three components. A framework for understanding all algorithms.

COMPONENTS OF LEARNING

the components involved in solving a problem using machine learning.

1) Feature Extraction + Domain knowledge

First and foremost we really need to understand what type of data we are dealing with and what eventually we want to get out of it. Essentially we need to understand how and what *features* need to be *extracted* from the data. For instance assume we want to build a software that distinguishes between male and female names. All the names in text can be thought of as our raw data while our features could be number of vowels in the name, length, first & last character, etc of the name.

2) Feature Selection

In many scenarios we end up with a lot of features at our disposal. We might want to *select* a *subset* of those based on the resources and computation power we have. In this step we select a few of those influential features and separate them from the not-so-influential features. There are many ways to do this, information gain, gain ratio, correlation etc.

3) Choice of Algorithm

There are wide range of algorithms from which we can choose based on whether we are trying to do prediction, classification or clustering. We can also choose between linear and non-linear algorithms. Naive Bayes, Support Vector Machines, Decision Trees, k-Means Clustering are some common algorithms used.

4) Training

In this step we tune our algorithm based on the data we already have. This data is called training set as it is used to train our algorithm. This is the part where our machine or software learn and improve with experience.

5) Choice of Metrics/Evaluation Criteria

Here we decide our evaluation criteria for our algorithm. Essentially we come up with metrics to evaluate our results. Commonly used measures of performance are precision, recall, f1-measure, robustness, specificity-sensitivity, error rate etc.

6) Testing

Lastly, we test how our machine learning algorithm performs on an unseen set of test cases. One way to do this, is to partition the data into training and testing set. The training set is used in step

4 while the test set is then used in this step. Techniques such as *cross-validation* and *leave-one-out* can be used to deal with scenarios where we do not have enough data.

The above list of buckets, definitely is not exhaustive and cannot do complete justice to a broad field like Machine Learning. Even then, most of the times a Machine Learning project would involve most of the above mentioned buckets, if not all.

EX:

Each customer record has personal information related to credit, such as annual salary, years in residence, outstanding loans, etc. The record also keeps track of whether approving credit for that customer was a good idea, i.e., did the bank make money on that customer. This data guides the construction of a successful formula for credit approval that can be used on future applicants. Let us give names and symbols to the main components of this learning problem. There is the input x (customer information that is used to make a credit decision) , the unknown target function f: X -- Y (ideal formula for credit approval) , where X is the input space (set of all possible inputs x) , and Y is the output space (set of all possible outputs, in this case just a yes/no decision) . There is a data set D of input-output examples (x1, Y1), · · · , (xN, YN), where Yn = f (xn) for n = 1, ... , N (inputs corresponding to previous customers and the correct credit decision for them in hindsight). The examples are often referred to as data points. Finally, there is the learning algorithm that uses the data set D to pick a formula g: X -- Y that approximates f. The algorithm chooses g from a set of candidate formulas under consideration, which we call the hypothesis set 1-l. For instance, 1-l could be the set of all linear formulas from which the algorithm would choose the best linear fit to the data, as we will introduce later in this section. When a new customer applies for credit, the bank will base its decision on g (the hypothesis that the learning algorithm produced) , not on f (the ideal target function which remains unknown) . The decision will be good only to the extent that g faithfully replicates f. To achieve that, the algorithm chooses g that best matches f on the training examples of previous customers,

with the hope that it will continue to match f on new customers. Whether

or not this hope is justified remains to be seen.

Figure 1.2 illustrates the components of the learning problem.

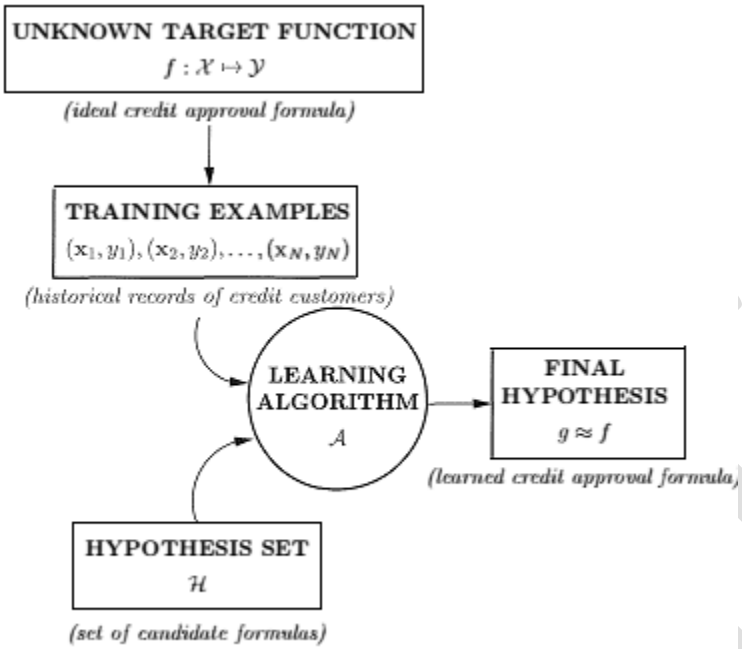


Figure 1.2: Basic setup of the learning problem

LEARNING MODEL

Let us consider the different components of Figure 1.2. Given a specific learn- ing problem, the target function and training examples are dictated by the problem. However, the learning algorithm and hypothesis set are not. These

are solution tools that we get to choose. The hypothesis set and learning algorithm are referred to informally as the learning model.

Here is a simple model. Let X =]R^d be the input space, where JR^d is the d-dimensional Euclidean space, and let Y = { + 1, -1} be the output space, denoting a binary (yes/no) decision. In our credit example, different coor- dinates of the input vector x E JR^d correspond to salary, years in residence, outstanding debt, and the other data fields in a credit application. The bi- nary output y corresponds to approving or denying credit. We specify the hypothesis set 1{ through a functional form that all the hypotheses h E 1{ share. The functional form h(x) that we choose here gives different weights to the different coordinates of x, reflecting their relative importance in the credit decision. The weighted coordinates are then combined to form a 'credit score' and the result is compared to a threshold value. If the applicant passes the threshold, credit is approved; if not, credit is denied

$$\text{Approve credit if} \quad \sum_{i=1}^{d} w_i x_i > \text{threshold},$$

$$\text{Deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}.$$

This formula can be written more compactly as

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right), \qquad (1.1)$$

This model of 1{ is called the perceptron, a name that it got in the context of artificial intelligence. The learning algorithm will search 1{ by looking for weights and bias that perform well on the data set. Some

of the weights $w1, \cdots,$ Wd may end up being negative, corresponding to an adverse effect on credit approval. For instance, the weight of the 'outstanding debt' field should come out negative since more debt is not good for credit. The bias value b may end up being large or small, reflecting how lenient or stringent the bank
should be in extending credit. The optimal choices of weights and bias define the final hypothesis g E 1-l that the algorithm produces.

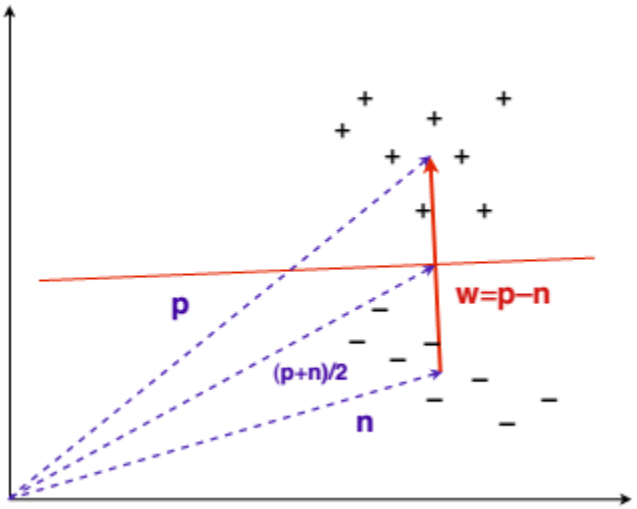
MACHINE LEARNING MODELS:

Machine learning models can be distinguished according to their main intuition:
t *Geometric* models use intuitions from geometry such as separating (hyper-)planes, linear transformations and distance metrics.
t *Probabilistic* models view learning as a process of reducing uncertainty, modelled by means of probability distributions.
t *Logical* models are defined in terms of easily interpretable logical expressions.
Alternatively, they can be characterised by their *modus operandi*:
t *Grouping models* divide the instance space into segments; in each segment a very simple (e.g., constant) model is learned.
t *Grading models* learning a single, global model over the instance space.

1. **Geometric models**

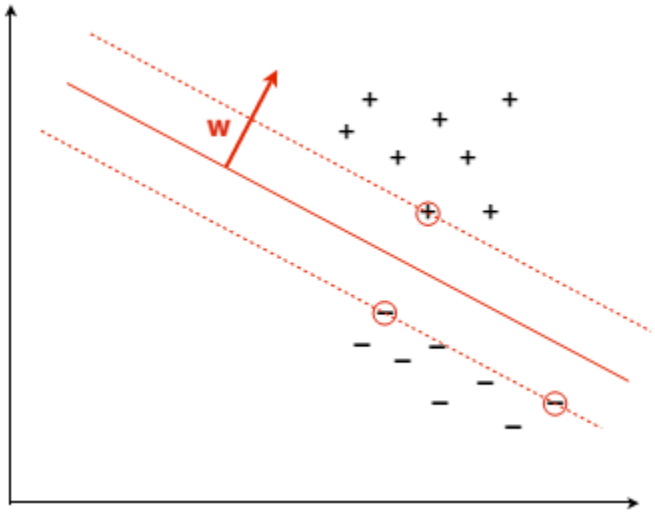
Basic linear classifier

The basic linear classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass. It is described by the equation $\mathbf{w}\cdot\mathbf{x} = t$, with $\mathbf{w} = \mathbf{p}-\mathbf{n}$; the decision threshold can be found by noting that $(\mathbf{p}+\mathbf{n})/2$ is on the decision boundary, and hence $t = (\mathbf{p}-\mathbf{n})\cdot(\mathbf{p}+\mathbf{n})/2 = (||\mathbf{p}||2 - ||\mathbf{n}||2)/2$, where $||\mathbf{x}||$ denotes the length of vector $\mathbf{x}$.



Support vector machine

The decision boundary learned by a support vector machine from the linearly separable data from Figure 1. The decision boundary maximises the margin, which is indicated by the dotted lines. The circled data points are the support vectors.



Probabilistic models

A simple probabilistic model

Assuming that X and Y are the only variables we know and care about, the posterior distribution $P(Y|X)$ helps us to answer many questions of interest. For instance, to classify a new e-mail we determine whether the words 'Booking' and 'lottery' occur in it, look up the corresponding probability $P(Y = \text{spam}|\text{Booking},\text{lottery})$, and predict spam if this probability exceeds 0.5 and ham otherwise.
Such a recipe to predict a value of Y on the basis of the values of X and the posterior distribution $P(Y|X)$ is called a *decision rule*.

Missing values I

Suppose we skimmed an e-mail and noticed that it contains the word 'lottery' but we haven't looked closely enough to determine whether it uses the word 'Booking'. This means that we don't know whether to use the second or the fourth row in Table 1.2 to make a prediction. This is a problem, as we would predict spam if the e-mail contained the word 'Booking' (second row) and ham if it didn't (fourth row). The solution is to average these two rows, using the probability of 'Booking' occurring in any e-mail (spam or not):
$P(Y|\text{lottery}) = P(Y|\text{Booking} = 0,\text{lottery})P(\text{Booking} = 0) + P(Y|\text{Booking} = 1,\text{lottery})P(\text{Booking} = 1)$

Missing values II

For instance, suppose for the sake of argument that one in ten e-mails contain the word 'Booking', then $P(\text{Booking} = 1) = 0.10$ and $P(\text{Booking} = 0) = 0.90$. Using the above formula, we obtain $P(Y = \text{spam}|\text{lottery} = 1) = 0.65 \cdot 0.90 + 0.40 \cdot 0.10 = 0.625$ and $P(Y = \text{ham}|\text{lottery} = 1) = 0.35 \cdot 0.90 + 0.60 \cdot 0.10 = 0.375$. Because the occurrence of 'Booking' in any e-mail is relatively rare, the resulting distribution deviates only a little from the second row in Table 1.2.

Likelihood ratio

As a matter of fact, statisticians work very often with different conditional probabilities, given by the *likelihood function* $P(X|Y)$.

t I like to think of these as thought experiments: if somebody were to send me a spam e-mail, how likely would it be that it contains exactly the words of the e-mail I'm looking at? And how likely if it were a ham e-mail instead?

t What really matters is not the magnitude of these likelihoods, but their ratio: how much more likely is it to observe this combination of words in a spam e-mail than it is in a non-spam e-mail.

t For instance, suppose that for a particular e-mail described by X we have $P(X|Y = \text{spam}) = 3.5 \cdot 10{-}5$ and $P(X|Y = \text{ham}) = 7.4 \cdot 10{-}6$, then observing X in a spam e-mail is nearly five times more likely than it is in a ham e-mail.

t This suggests the following decision rule (maximum a posteriori, MAP): predict spam if the likelihood ratio is larger than 1 and ham otherwise.

Logical models

A feature tree

A feature tree combining two Boolean features. Each internal node or split is labelled with a feature, and each edge emanating from a split is labelled with a feature value. Each leaf therefore corresponds to a unique combination of feature values. Also indicated in each leaf is the class distribution derived from the training set. **(right)** A feature tree partitions the instance space into rectangular regions, one for each leaf. We can clearly see that the majority of ham lives in the lower left-hand corner.

Labelling a feature tree

t The leaves of the tree in Figure 1.4 could be labelled, from left to right, as ham – spam – spam, employing a simple decision rule called *majority class*.

t Alternatively, we could label them with the proportion of spam e-mail

occurring in each leaf: from left to right, 1/3, 2/3, and 4/5. t Or, if our task was a regression task, we could label the leaves with predicted real values or even linear functions of some other, real-valued features.

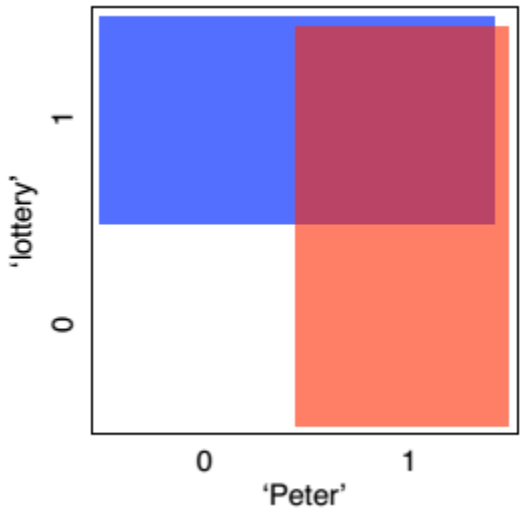
Overlapping rules

Consider the following rules:

·**if** lottery = 1 **then** Class = Y = spam·

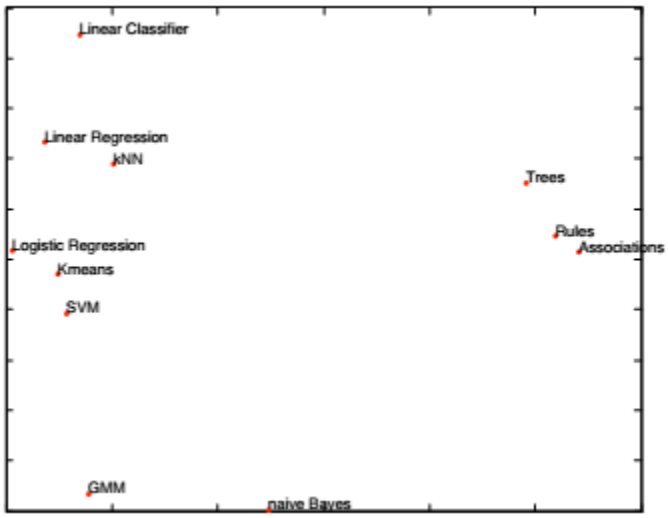
·**if** Peter = 1 **then** Class = Y = ham·

As can be seen in Figure 1.6, these rules overlap for lottery = 1 ∧ Peter = 1, for which they make contradictory predictions. Furthermore, they fail to make any predictions for lottery = 0 ∧ Peter = 0.



The effect of overlapping rules in instance space. The two rules make contradictory predictions in the top right-hand corner, and no prediction at all in the bottom left-hand corner.

Grouping and grading



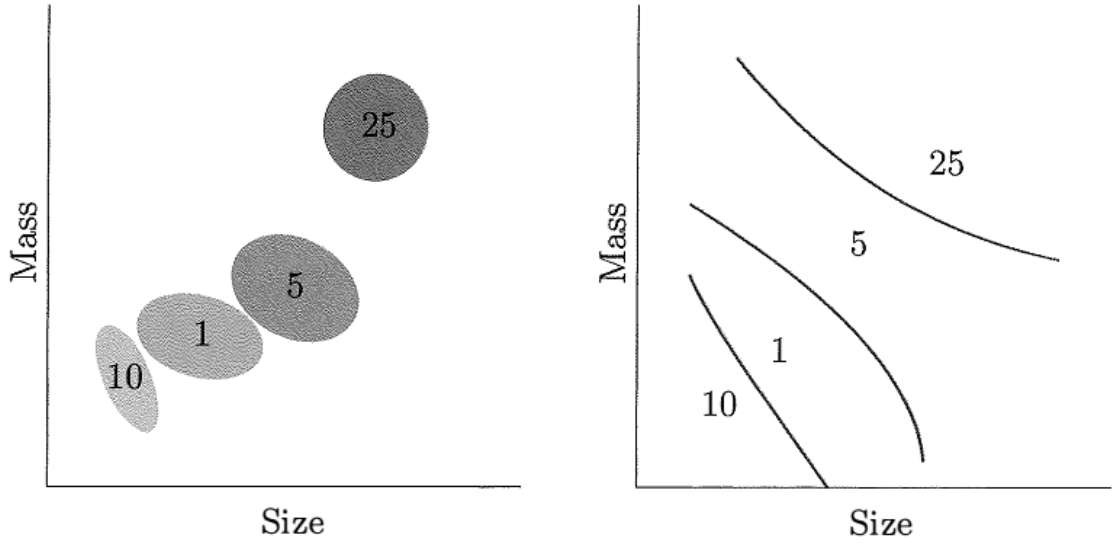
A ‘map’ of some of the models that will be considered in this book. Models that share characteristics are plotted closer together: logical models to the right, geometric models on the top left and probabilistic models on the bottom left. The horizontal dimension roughly ranges from grading models on the left to grouping models on the right.

LEARNING VS DESIGN

So far, we have discussed what learning is. Now, we discuss what it is not. The goal is to distinguish between learning and a related approach that is used for similar problems. While learning is based on data, this other approach does not use data. It is a 'design' approach based on specifications, and is often discussed alongside the learning approach in pattern recognition literature. Consider the problem of recognizing coins of different denominations, which is relevant to vending machines, for example. We want the machine to recognize quarters, dimes, nickels and pennies. We will contrast the 'learning from data' approach and the 'design from specifications' approach for this problem. We assume that each coin will be represented by its size and mass, a two-dimensional input.

In the learning approach, we are given a sample of coins from each of the four denominations and we use these coins as our data set. We treat the size and mass as the input vector, and the denomination as the output. Figure 1.4(a) shows what the data set may look like in the input space. There is some variation of size and mass within each class, but by and large coins of the same denomination cluster together. The learning algorithm searches

for a hypothesis that classifies the data set well. If we want to classify a new coin, the machine measures its size and mass, and then classifies it according to the learned hypothesis in Figure l .4(b) .
In the design approach, we call the United States Mint and ask them about the specifications of different coins. We also ask them about the number



(a) Probabilistic model of data (b) Inferred classifier

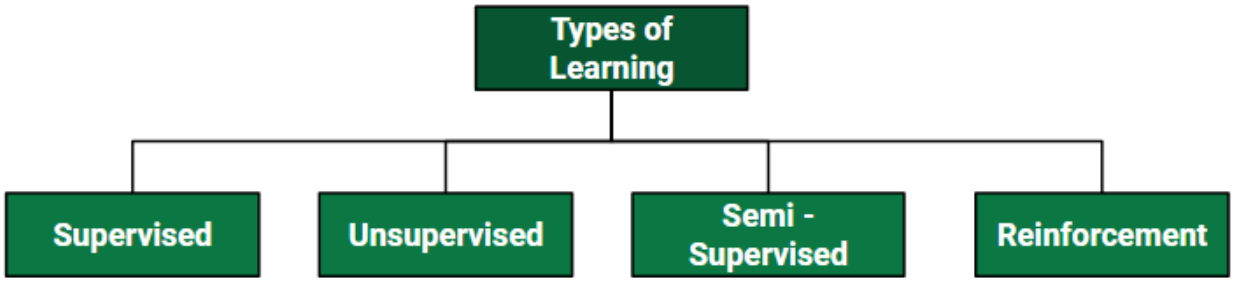
Figure 1 .5: The design approach to coin classification (a) A probabilistic model for the size, mass, and denomination of coins is derived from known specifications. The figure shows the high probability region for each denom ination (1, 5, 10, and 25 cents) according to the model. (b) A classification rule is derived analytically to minimize the probability oferror in classifying a coin based on size and mass. The resulting regions for each denomination are shown

of coins of each denomination in circulation, in order to get an estimate of the relative frequency of each coin. Finally, we make a physical model of the variations in size and mass due to exposure to the elements and due to errors in measurement. We put all of this information together and compute the full joint probability distribution of size, mass, and coin denomination (Figure 1.5(a)) . Once we have that joint distribution, we can construct the optimal decision rule to classify coins based on size and mass (Figure 1 .5(b)) . The rule chooses the denomination that has the highest probability for a given size and mass, thus achieving the smallest possible probability of error.2

The main difference between the learning approach and the design approach is the role that data plays. In the design approach, the problem is well specified and one can analytically derive f without the need to see any data. In the learning approach, the problem is much less specified, and one needs data to pin down what f is.
Both approaches may be viable in some applications, but only the learning approach is possible in many applications where the target function is unknown. We are not trying to compare the utility or the performance of the two approaches. We are just making the point that the design approach is distinct from learning. This book is about learning.

TYPES OF LEARNING

There are four types of machine learning:



- **Supervised learning**: (also called inductive learning) Training data includes desired outputs. This is spam this is not, learning is supervised.
- **Unsupervised learning**: Training data does not include desired outputs. Example is clustering. It is hard to tell what is good learning and what is not.
- **Semi-supervised learning**: Training data includes a few desired outputs.
- **Reinforcement learning**: Rewards from a sequence of actions. AI types like it, it is the most ambitious type of learning.

Supervised learning is the most mature, the most studied and the type of learning used by most machine learning algorithms. Learning with supervision is much easier than learning without supervision.

Inductive Learning is where we are given examples of a function in the form of data (x) and the output of the function ($f(x)$). The goal of inductive learning is to learn the function for new data (x).

- **Classification**: when the function being learned is discrete.
- **Regression**: when the function being learned is continuous.
- **Probability Estimation**: when the output of the function is a probability.

Supervised Learning :

Supervised learning is when the model is getting trained on a labelled dataset. **Labelled** dataset is one which have both input and output parameters. In this type of learning both training and validation datasets are labelled as shown in the figures below.

User ID	Gender	Age	Salary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	1
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	1
15728773	Male	27	58000	1
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	1
15727311	Female	35	65000	0
15570769	Female	26	80000	1
15606274	Female	26	52000	0
15746139	Male	20	86000	1
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1

Figure A: CLASSIFICATION

Temperature	Pressure	Relative Humidity	Wind Direction	Wind Speed
10.69261758	986.882019	54.19337313	195.7150879	3.278597116
13.59184184	987.8729248	48.0648859	189.2951202	2.909167767
17.70494885	988.1119385	39.11965597	192.9273834	2.973036289
20.95430404	987.8500366	30.66273218	202.0752869	2.965289593
22.9278274	987.2833862	26.06723423	210.6589203	2.798230886
24.04233986	986.2907104	23.46918024	221.1188507	2.627005816
24.41475295	985.2338867	22.25082295	233.7911987	2.448749781
23.93361956	984.8914795	22.35178837	244.3504333	2.454271793
22.68800023	984.8461304	23.7538641	253.0864716	2.418341875
20.56425726	984.8380737	27.07867944	264.5071106	2.318677425
17.76400389	985.4262085	33.54900114	280.7827454	2.343950987
11.25680746	988.9386597	53.74139903	68.15406036	1.650191426
14.37810685	989.6819458	40.70884681	72.62069702	1.553469896
18.45114201	990.2960205	30.85038484	71.70604706	1.005017161
22.54895853	989.9562988	22.81738811	44.66042709	0.264133632
24.23155922	988.796875	19.74790765	318.3214111	0.329656571

Figure B: REGRESSION

Both the above figures have labeled data set –

- **Figure A:** It is a dataset of a shopping store which is useful in predicting whether a customer will purchase a particular product under consideration or not based on his/ her gender, age and salary.

  Input : Gender, Age, Salary

  Ouput : Purchased i.e. 0 or 1 ; 1 means yes the customer will purchase and 0 means that customer won't purchase it.

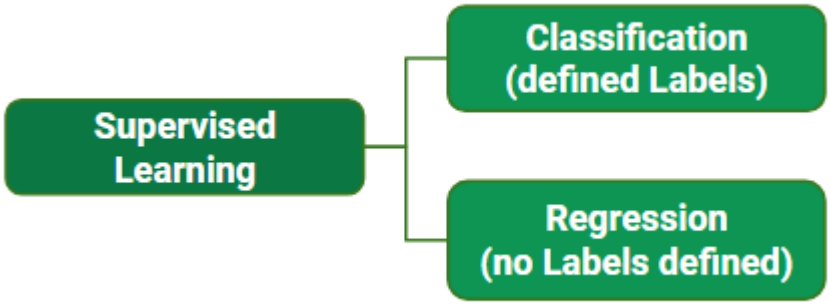
- **Figure B:** It is a Meteorological dataset which serves the purpose of predicting wind speed based on different parameters.

  Input : Dew Point, Temperature, Pressure, Relative Humidity, Wind Direction
  Output : Wind Speed

Training the system:

While training the model, data is usually split in the ratio of 80:20 i.e. 80% as training data and rest as testing data. In training data, we feed input as well as output for 80% data. The model learns from training data only. We use different machine learning algorithms (which we will discuss in detail in next articles) to build our model. By learning, it means that the model will build some logic of its own.

Once the model is ready then it is good to be tested. At the time of testing, input is fed from remaining 20% data which the model has never seen before, the model will predict some value and we will compare it with actual output and calculate the accuracy.



Types of Supervised Learning:

1. **Classification:** It is a Supervised Learning task where output is having defined labels (discrete value). For example in above Figure A, Output – Purchased has defined labels i.e. 0 or 1 ; 1 means the customer will purchase and 0 means that customer won't

purchase. The goal here is to predict discrete values belonging to a particular class and evaluate on the basis of accuracy.

It can be either binary or multi class classification. In **binary** classification, model predicts either 0 or 1 ; yes or no but in case of **multi class** classification, model predicts more than one class.
Example: Gmail classifies mails in more than one classes like social, promotions, updates, forum.

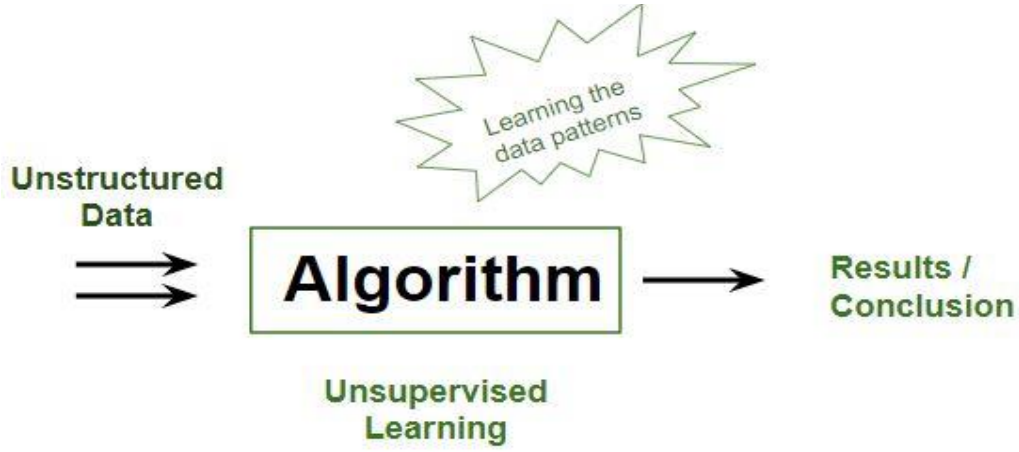
2. **Regression :** It is a Supervised Learning task where output is having continuous value.

   Example in above Figure B, Output – Wind Speed is not having any discrete value but is continuous in the particular range. The goal here is to predict a value as much closer to actual output value as our model can and then evaluation is done by calculating error value. The smaller the error the greater the accuracy of our regression model.

Example of Supervised Learning Algorithms:

- Linear Regression
- Nearest Neighbor
- Guassian Naive Bayes
- Decision Trees
- Support Vector Machine (SVM)
- Random Forest

Unsupervised Learning:



It's a type of learning where we don't give target to our model while training i.e. training model has only input parameter values. The model by itself has to find which way it can learn.

Data-set in Figure A is mall data that contains information of its clients that subscribe to them.

Once subscribed they are provided a membership card and so the mall has complete information about customer and his/her every purchase.

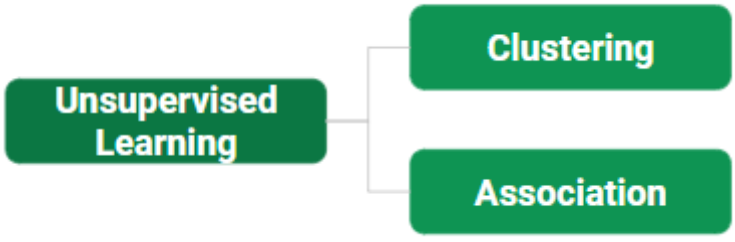
Now using this data and unsupervised learning techniques, mall can easily group clients based on the parameters we are feeding in.

CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13
16	Male	22	20	79
17	Female	35	21	35

Figure A

Training data we are feeding is –

- **Unstructured data:** May contain noisy(meaningless) data, missing values or unknown data
- **Unlabeled data:** Data only contains value for input parameters, there is no targeted value (output). It is easy to collect as compared to labeled one in supervised approach.



Types of Unsupervised Learning:

- **Clustering:** Broadly this technique is applied to group data based on different patterns, our machine model finds. For example in above figure we are not given output parameter

value, so this technique will be used to group clients based on the input parameters provided by our data.

- **Association:** This technique is a rule based ML technique which finds out some very useful relations between parameters of a large data set. For e.g. shopping stores use algorithms based on this technique to find out relationship between sale of one product w.r.t to others sale based on customer behavior. Once trained well, such models can be used to increase their sales by planning different offers.

Some algorithms:

- K-Means Clustering
- DBSCAN – Density-Based Spatial Clustering of Applications with Noise
- BIRCH – Balanced Iterative Reducing and Clustering using Hierarchies
- Hierarchical Clustering

Semi-supervised Learning

Its working lies between Supervised and Unsupervised techniques. We use these techniques when we are dealing with a data which is a little bit labeled and rest large portion of it is unlabeled.

We can use unsupervised technique to predict labels and then feed these labels to supervised techniques.

This technique is mostly applicable in case of image data-sets where usually all images are not labeled.

Reinforcement Learning:

In this technique, model keeps on increasing its performance using a Reward Feedback to learn the behavior or pattern. These algorithms are specific to a particular problem e.g. Google Self Driving car, AlphaGo where a bot competes with human and even itself to getting better and better performer of Go Game. Each time we feed in data, they learn and add the data to its knowledge that is training data. So, more it learns the better it get trained and hence experienced.

- Agents observe input.
- Agent performs an action by making some decisions.
- After its performance, agent receives reward and accordingly reinforce and the model stores in state-action pair of information.

Some algorithms:

- Temporal Difference (TD)
- Q-Learning
- Deep Adversarial Networks

THEORY OF LEARNING

Differences between Supervised Learning and Unsupervised Learning

(http://www.differencebetween.net/technology/differences-between-supervised-learning-and-unsupervised-learning/)

1. **Input Data in Supervised Learning and Unsupervised Learning**

The primary difference between supervised learning and unsupervised learning is the data used in either method of machine learning. It is worth noting that both methods of machine learning require data, which they will analyze to produce certain functions or data groups. However, the input data used in supervised learning is well known and is labeled. This means that the machine is only tasked with the role of determining the hidden patterns from already labeled data. However, the data used in unsupervised learning is not known nor labeled. It is the work of the machine to categorize and label the raw data before determining the hidden patterns and functions of the input data.

2. **Computational Complexity in Supervised Learning and Unsupervised Learning**

Machine learning is a complex affair and any person involved must be prepared for the task ahead. One of the stand out differences between supervised learning and unsupervised learning is computational complexity. Supervised learning is said to be a complex method of learning while unsupervised method of learning is less complex. One of the reason that makes supervised learning affair is the fact that one has to understand and label the inputs while in unsupervised learning, one is not required to understand and label the inputs. This explains why many people have been preferring unsupervised learning as compared to the supervised method of machine learning.

3. **Accuracy of the Results of Supervised Learning and Unsupervised Learning**

The other prevailing difference between supervised learning and unsupervised learning is the accuracy of the results produced after every cycle of machine analysis. All the results generated from supervised method of machine learning are more accurate and reliable as compared to the

results generated from the unsupervised method of machine learning. One of the factor that explains why supervised method of machine learning produces accurate and reliable results is because the input data is well known and labeled which means that the machine will only analyze the hidden patterns. This is unlike unsupervised method of learning where the machine has to define and label the input data before determining the hidden patterns and functions.

4. Number of Classes in Supervised Learning and Unsupervised Learning

It is also worth noting that there is a significant difference when it comes to the number of classes. It is worth noting that all the classes used in supervised learning are known which means that also the answers in the analysis are likely to be known. The only goal of supervised learning is therefore to determine the unknown cluster. However, there is no prior knowledge in unsupervised method of machine learning. In addition, the numbers of classes are not known which clearly means that no information is known and the results generated after the analysis cannot be ascertained. Moreover, the people involved in unsupervised method of learning are not aware of any information concerning the raw data and the expected results.

5. Real Time Learning in Supervised Learning and Unsupervised Learning

Among other differences, there exist the time after which each method of learning takes place. It is important to highlight that supervised method of learning takes place off-line while unsupervised method of learning takes place in real time. People involved in preparation and labeling of the input data do so off-line while the analysis of the hidden pattern is done online which denies the people involved in machine learning an opportunity to interact with the machine as it analyzes the discrete data. However, unsupervised method of machine learning takes place in real time such that all the input data is analyzed and labeled in the presence of learners which helps them to understand different methods of learning and classification of raw data. Real time data analysis remains to be the most significant merit of unsupervised method of learning.

TABLE SHOWING DIFFERENCES BETWEEN SUPERVISED LEARNING AND UNSUPERVISED LEARNING: COMPARISON CHART

	Supervised Learning	**Unsupervised Learning**
Input Data	Uses Known and Labeled Input Data	Uses Unknown Input Data
Computational Complexity	Very Complex in Computation	Less Computational Complexity
Real Time	Uses off-line analysis	Uses Real Time Analysis of Data
Number of Classes	Number of Classes is Known	Number of Classes is not Known
Accuracy of Results	Accurate and Reliable Results	Moderate Accurate and Reliable Results

Summary of Supervised Learning and Unsupervised Learning

- Data mining is becoming an essential aspect in the current business world due to increased raw data that organizations need to analyze and process so that they can make sound and reliable decisions.
- This explains why the need for machine learning is growing and thus requiring people with sufficient knowledge of both supervised machine learning and unsupervised machine learning.
- It is worth understanding that each method of learning offers its own advantages and disadvantages. This means that one has to be conversant with both methods of machine learning before determine which method one will use to analyze data.

FEASIBILITY OF LEARNING:

Consider a bin with red and green marbles P[picking a red marble] = μ P[picking a green marble] = 1 − μ The value of μ is unknown to us How to infer μ? Pick N marbles independently ν: the fraction of red marbles

Do you know μ? No Sample can be mostly green while bin is mostly red Can you say something about μ? Yes ν is "probably" close to μ

In big sample (large N), ν (sample mean) is probably close to μ: $P[|\nu - \mu| >] \leq 2e^{-2}\ 2N$ This is called Hoeffding's inequality The statement "μ = ν" is probably approximately correct (PAC)

$P[|\nu - \mu| >] \leq 2e^{-22N}$

Valid for all N and > 0 Does not depend on μ (no need to know μ) Larger sample size N or looser gap ⇒ higher probability for $\mu \approx \nu$

How to connect this to learning? Each marble (uncolored) is a data point $x \in X$ red ball: h(x)6=f (x) (h is correct) green ball: h(x)=f (x) (h is wrong)

ERROR AND NOISE

Error and Noise

We close this chapter by revisiting two notions in the learning problem in order to bring them closer to the real world. The first notion is what approximation means when we say that our hypothesis approximates the target function well. The second notion is about the nature of the target function. In many situations, there is noise that makes the output of f not uniquely determined by the input. What are the ramifications of having such a 'noisy' target on the learning problem

ErrorMeasures

Learning is not expected to replicate the target function perfectly. The final hypothesis g is only an approximation of f. To quantify how well g approximates f, we need to define an error measure3 that quantifies how far we are from the target.

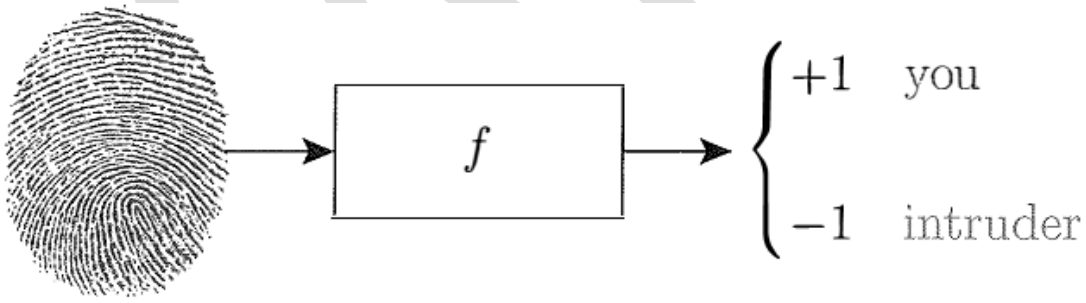
The choice of an error measure affects the outcome of the learning process. Different error measures may lead to different choices of the final hypothesis, even if the target and the data are the same, since the value of a particular error measure may be small while the value of another error measure in the same situation is large. Therefore, which error measure we use has consequences

for what we learn. What are the criteria for choosing one error measure over another? We address this question here.
First, let's formalize this notion a bit. An error measure quantifies how well each hypothesis h in the model approximates the target function f,

Error=E(h,f) .

While E(h, f) is based on the entirety of h and f, it is almost universally defined based on the errors on individual input points x. If we define a pointwise error measure e(h(x), f(x)) , the overall error will be the average value of this pointwise error. So far, we have been working with the classification error e(h(x) , f(x)) = [h(x) f- J(x)] .
In an ideal world, E(h, J) should be user-specified. The same learning task in different contexts may warrant the use of different error measures. One may view E(h, J) as the 'cost' of using h when you should use f. This cost depends on what h is used for, and cannot be dictated just by our learning techniques.
Here is a case in point.

Example 1.1 (Fingerprint verification). Consider the problem of verifying that a fingerprint belongs to a particular person. What is the appropriate error measure?



The target function takes as input a fingerprint, and returns + 1 if it belongs to the right person, and -1 if it belongs to an intruder.

There are two types of error that our hypothesis h can make here. If the correct person is rejected (h = -1 but f = +1), it is called false reject, and if an incorrect person is accepted (h = +1 but f = -1), it is called false accept.

h +1
-1
+1
no error
false reject
f
-1
false accept
no error

How should the error measure be defined in this problem? If the right person is accepted or an intruder is rejected, the error is clearly zero. We need to specify the error values for a false accept and for a false reject. The right values depend on the application.
Consider two potential clients of this fingerprint system. One is a supermarket who will use it at the checkout counter to verify that you are a member of a discount program. The other is the CIA who will use it at the entrance to a secure facility to verify that you are authorized to enter that facility.
For the supermarket, a false reject is costly because if a customer gets wrongly rejected, she may be discouraged from patronizing the supermarket in the future. All future revenue from this annoyed customer is lost. On the other hand, the cost of a false accept is minor. You just gave away a discount to someone who didn't deserve it, and that person left their fingerprint in your system they must be bold indeed.
For the CIA, a false accept is a disaster. An unauthorized person will gain access to a highly sensitive facility. This should be reflected in a much higher cost for the false accept. False rejects, on the other hand, can be tolerated since authorized persons are employees (rather than customers as with the supermarket) . The inconvenience of retrying when rejected is just part of the job, and they must deal with it.
The costs of the different types of errors can be tabulated in a matrix. For our examples, the matrices might look like:

f f
+1 -1 +1 -1
h +1 0 1 h +1 0 1000

-1 10 0 -1 1 0
Supermarket CIA

These matrices should be used to weight the different types of errors when we compute the total error. When the learning algorithm minimizes a costweighted error measure, it automatically takes into consideration the utility of the hypothesis that it will produce. In the supermarket and CIA scenarios, this could lead to two completely different final hypotheses. D
The moral of this example is that the choice of the error measure depends on how the system is going to be used, rather than on any inherent criterion

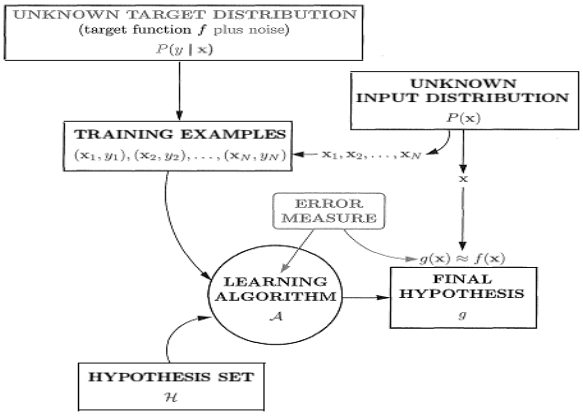


Figure 1.11: The general (supervised) learning problem

that we can independently determine during the learning process. However, this ideal choice may not be possible in practice for two reasons. One is that the user may not provide an error specification, which is not uncommon. The other is that the weighted cost may be a difficult objective function for optimizers to work with. Therefore, we often look for other ways to define the error measure, sometimes with purely practical or analytic considerations in mind. We have already seen an example of this with the simple binary error used in this chapter, and we will see other error measures in later chapters.
1.4.2NoisyTargets
In many practical applications, the data we learn from are not generated by a deterministic target function. Instead, they are generated in a noisy way such that the output is not uniquely determined by the input. For instance, in the credit-card example we presented in Section 1.1, two customers may have identical salaries, outstanding loans, etc., but end up with different credit behavior. Therefore, the credit 'function' is not really a deterministic function,

but a noisy one.
This situation can be readily modeled within the same framework that we have. Instead of y = f(x) , we can take the output y to be a random variable that is affected by, rather than determined by, the input x. Formally, we have a target distribution P(y I x) instead of a target function y = f(x) . A data point (x, y) is now generated by the joint distribution P(x, y) = P(x)P(y I x) . One can think of a noisy target as a deterministic target plus added noise. If y is real-valued for example, one can take the expected value of y given x to be the deterministic f(x) , and consider y - f(x) as pure noise that is added to f.
This view suggests that a deterministic target function can be considered a special case of a noisy target, just with zero noise. Indeed, we can formally express any function f as a distribution P(y I x) by choosing P(y I x) to be zero for all y except y = f(x) . Therefore, there is no loss of generality if we consider the target to be a distribution rather than a function. Figure 1 . 1 1 modifies the previous Figures 1 .2 and 1 . 9 to illustrate the general learning problem, covering both deterministic and noisy targets.

TRAINING VERSUS TESTING

Before the final exam, a professor may hand out some practice problems and solutions to the class. Although these problems are not the exact ones that will appear on the exam, studying them will help you do better. They are the 'training set' in your learning.
If the professor's goal is to help you do better in the exam, why not give out the exam problems themselves? Well, nice try @. Doing well in the exam is not the goal in and of itself. The goal is for you to learn the course material. The exam is merely a way to gauge how well you have learned the material. If the exam problems are known ahead of time, your performance on them will no longer accurately gauge how well you have learned.
The same distinction between training and testing happens in learning from data. In this chapter, we will develop a mathematical theory that characterizes this distinction. We will also discuss the conceptual and practical implications of the contrast between training and testing.

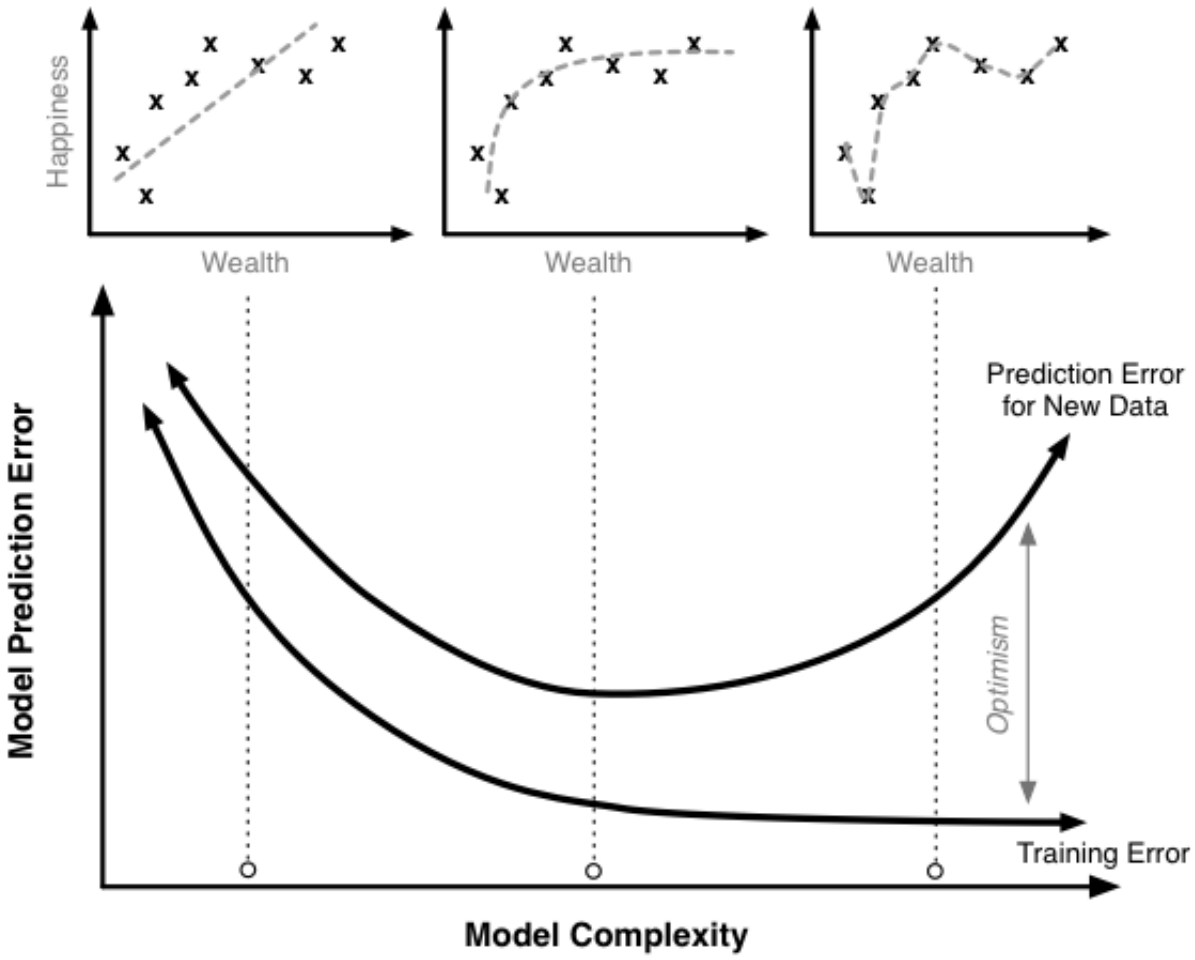
THEORY OF GENERALIZATION:

A machine learning algorithm is used to fit a model to data. Training the model is kind of like infancy for humans... examples are presented to the model and the model tweaks its internal

parameters to better understand the data. Once training is over, the model is unleashed upon new data and then uses what it has learned to explain that data.

Now here's where problems can emerge. If you overtrain the model on the training data, then it will be able to identify all the relevant information in the training data, but will fail miserably when presented with the new data. We then say that the model is incapable of *generalizing*, or that it is *overfitting* the training data.

Here's a great example of the phenomenon: modelling happiness as a function of wealth.



the top three diagrams we have data (x's) and models (dashed curves). From left to right the models have been trained longer and longer on the training data. The training error curve in the bottom box shows that the training error gets better and better as we train longer (increasing

model complexity). You may think, great! If we train longer we'll get better! Well, yes, but only better at describing the training data. The top right box shows a very complex model that hits all the data points. This model does great on the training data, but when presented with new data (examine the Prediction error curve in the bottom box) then it does abysmally.

So the lesson here is this. To create good predictive models in machine learning that are capable of generalizing, one needs to know when to stop training the model so that it doesn't overfit. Just like a parent, sometimes a programmer needs to know when to kick their kid out of the house and have them go into the real world. Otherwise they'll know everything about their parents' basement and yet be incapable of doing anything productive outside that small enclosure.

In supervised learning applications in machine learning and statistical learning theory, **generalization error** (also known as the **out-of-sample error**[1]) is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data. Because learning algorithms are evaluated on finite samples, the evaluation of a learning algorithm may be sensitive to sampling error. As a result, measurements of prediction error on the current data may not provide much information about predictive ability on new data. Generalization error can be minimized by avoiding overfitting in the learning algorithm. The performance of a machine learning algorithm is measured by plots of the generalization error values through the learning process, which are called learning curves.

The concepts of generalization error and overfitting are closely related. Overfitting occurs when the learned function {\displaystyle f_{S}} f_S becomes sensitive to the noise in the sample. As a result, the function will perform well on the training set but not perform well on other data from the joint probability distribution of {\displaystyle x} x and {\displaystyle y} y. Thus, the more overfitting occurs, the larger the generalization error.

The amount of overfitting can be tested using cross-validation methods, that split the sample into simulated training samples and testing samples. The model is then trained on a training sample and evaluated on the testing sample. The testing sample is previously unseen by the algorithm and so represents a random sample from the joint probability distribution of {\displaystyle x} x and {\displaystyle y} y. This test sample allows us to approximate the expected error and as a result approximate a particular form of the generalization error.

Many algorithms exist to prevent overfitting. The minimization algorithm can penalize more complex functions (known as Tikhonov regularization), or the hypothesis space can be

constrained, either explicitly in the form of the functions or by adding constraints to the minimization function (Ivanov regularization).

The approach to finding a function that does not overfit is at odds with the goal of finding a function that is sufficiently complex to capture the particular characteristics of the data. This is known as the bias–variance tradeoff. Keeping a function simple to avoid overfitting may introduce a bias in the resulting predictions, while allowing it to be more complex leads to overfitting and a higher variance in the predictions. It is impossible to minimize both simultaneously.

GENERALIZATION BOUND

Minimizing the empirical risk (or the training error) is not in itself a solution to the learning problem, it could only be considered a solution if we can guarantee that the difference between the training error and the generalization error (which is also called the **generalization gap**) is small enough. We formalized such requirement using the probability:

$P[\sup h \in H | R(h) - Remp(h)| > \epsilon]$

That is if this probability is small, we can guarantee that the difference between the errors is not much, and hence the learning problem can be solved.

In this part we'll start investigating that probability at depth and see if it indeed can be small, but before starting you should note that I skipped a lot of the mathematical proofs here. You'll often see phrases like "It can be proved that …", "One can prove …", "It can be shown that …", … etc without giving the actual proof. This is to make the post easier to read and to focus all the effort on the conceptual understanding of the subject.

Hoeffding's inequality

The law of large numbers is like someone pointing the directions to you when you're lost, they tell you that by following that road you'll eventually reach your destination, but they provide no

information about how fast you're gonna reach your destination, what is the most convenient vehicle, should you walk or take a cab, and so on.

To our destination of ensuring that the training and generalization errors do not differ much, we need to know more info about the how the road down the law of large numbers look like. These info are provided by what we call the **concentration inequalities**. This is a set of inequalities that quantifies how much random variables (or function of them) deviate from their expected values (or, also, functions of them). One inequality of those is **Heoffding's inequality**:

If x1,x2,…,xmx1,x2,…,xm are mm i.i.d. samples of a random variable XXdistributed by PP, and a≤xi≤ba≤xi≤b for every ii, then for a small positive non-zero value ϵϵ:

P[|||EX∼P[X]−1m∑i=0mxi|||>ϵ]≤2exp(−2mϵ2(b−a)2)P[|EX∼P[X]−1m∑i=0mxi|>ϵ]≤2exp(−2mϵ2(b−a)2)

You probably see why we specifically chose Heoffding's inequality from among the others. We can naturally apply this inequality to our generalization probability, assuming that our errors are bounded between 0 and 1 (which is a reasonable assumption, as we can get that using a 0/1 loss function or by squashing any other loss between 0 and 1) and get for a **single** hypothesis hh:

P[|R(h)−Remp(h)|>ϵ]≤2exp(−2mϵ2)P[|R(h)−Remp(h)|>ϵ]≤2exp(−2mϵ2)

This means that the probability of the difference between the training and the generalization errors exceeding ϵϵ exponentially decays as the dataset size goes larger. This should align well with our practical experience that the bigger the dataset gets, the better the results become.

If you noticed, all our analysis up till now was focusing on a **single** hypothesis hh. But the learning problem doesn't know that single hypothesis beforehand, it needs to pick one out of an entire hypothesis space HH, so we need a generalization bound that reflects the challenge of choosing the right hypothesis.

Generalization Bound: 1st Attempt

In order for the entire hypothesis space to have a generalization gap bigger than $\epsilon\epsilon$, at least one of its hypothesis: h1h1 **or** h2h2 **or** h3h3 **or** … etc should have. This can be expressed formally by stating that:

$$P[\sup_{h\in H}|R(h)-Remp(h)|>\epsilon]=P[\bigcup_{h\in H}|R(h)-Remp(h)|>\epsilon]P[\sup_{h\in H}|R(h)-Remp(h)|>\epsilon]=P[\bigcup_{h\in H}|R(h)-Remp(h)|>\epsilon]$$

Where $\cup\cup$ denotes the union of the events, which also corresponds to the logical **OR** operator. Using the union bound inequality, we get:

$$P[\sup_{h\in H}|R(h)-Remp(h)|>\epsilon]\leq\sum_{h\in H}P[|R(h)-Remp(h)|>\epsilon]P[\sup_{h\in H}|R(h)-Remp(h)|>\epsilon]\leq\sum_{h\in H}P[|R(h)-Remp(h)|>\epsilon]$$

We exactly know the bound on the probability under the summation from our analysis using the Heoffding's inequality, so we end up with:

$$P[\sup_{h\in H}|R(h)-Remp(h)|>\epsilon]\leq 2|H|\exp(-2m\epsilon 2)P[\sup_{h\in H}|R(h)-Remp(h)|>\epsilon]\leq 2|H|\exp(-2m\epsilon 2)$$

Where $|H||H|$ is the size of the hypothesis space. By denoting the right hand side of the above inequality by $\delta\delta$, we can say that with a confidence $1-\delta 1-\delta$:

$$|R(h)-Remp|\leq\epsilon\Rightarrow R(h)\leq Remp(h)+\epsilon|R(h)-Remp|\leq\epsilon\Rightarrow R(h)\leq Remp(h)+\epsilon$$

And with some basic algebra, we can express $\epsilon\epsilon$ in terms of $\delta\delta$ and get:

$$R(h)\leq Remp(h)+\ln|H|+\ln 2\delta 2m----------\sqrt{R(h)\leq Remp(h)+\ln|H|+\ln 2\delta 2m}$$

This is our first generalization bound, it states that the generalization error is bounded by the training error plus a function of the hypothesis space size and the dataset size. We can also see that the the bigger the hypothesis space gets, the bigger the generalization error becomes. This explains why the memorization hypothesis form last time, which theoretically has $|H|=\infty|H|=\infty$, fails miserably as a solution to the learning problem despite having $Remp=0Remp=0$; because for the memorization hypothesis hmemhmem:

$$R(hmem)\leq 0+\infty\leq\infty R(hmem)\leq 0+\infty\leq\infty$$

But wait a second! For a linear hypothesis of the form $h(x)=wx+bh(x)=wx+b$, we also have $|H|=\infty|H|=\infty$ as there is infinitely many lines that can be drawn. So the generalization error of the linear hypothesis space should be unbounded just as the memorization hypothesis! If that's true, why does perceptrons, logistic regression, support vector machines and essentially any ML model that uses a linear hypothesis work?

Our theoretical result was able to account for some phenomena (the memorization hypothesis, and any finite hypothesis space) but not for others (the linear hypothesis, or other infinite hypothesis spaces that empirically work). This means that there's still something missing from our theoretical model, and it's time for us to revise our steps. A good starting point is from the source of the problem itself, which is the infinity in $|H||H|$.

Notice that the term |H||H| resulted from our use of the union bound. The basic idea of the union bound is that it bounds the probability by the worst case possible, which is when all the events under union are mutually independent. This bound gets more tight as the events under consideration get less dependent. In our case, for the bound to be tight and reasonable, we need the following to be true:

BIAS AND VARIANCE:

In supervised machine learning an algorithm learns a model from training data.

The goal of any supervised machine learning algorithm is to best estimate the mapping function (f) for the output variable (Y) given the input data (X). The mapping function is often called the target function because it is the function that a given supervised machine learning algorithm aims to approximate.

The prediction error for any machine learning algorithm can be broken down into three parts:

- Bias Error
- Variance Error
- Irreducible Error

The irreducible error cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

In this post, we will focus on the two parts we can influence with our machine learning algorithms. The bias error and the variance error.

Bias Error

Bias are the simplifying assumptions made by a model to make the target function easier to learn.

Generally, parametric algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

- **Low Bias**: Suggests less assumptions about the form of the target function.
- **High-Bias**: Suggests more assumptions about the form of the target function.

Examples of low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Examples of high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Variance Error

Variance is the amount that the estimate of the target function will change if different training data was used.

The target function is estimated from the training data by a machine learning algorithm, so we should expect the algorithm to have some variance. Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables.

Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. This means that the specifics of the training have influences the number and types of parameters used to characterize the mapping function.

- **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
- **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.

Generally, nonparametric machine learning algorithms that have a lot of flexibility have a high variance. For example, decision trees have a high variance, that is even higher if the trees are not pruned before use.

Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Bias-Variance Trade-Off

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

You can see a general trend in the examples above:

- Parametric or linear machine learning algorithms often have a high bias but a low variance.
- Non-parametric or non-linear machine learning algorithms often have a low bias but a high variance.

The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

Below are two examples of configuring the bias-variance trade-off for specific algorithms:

- The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model.
- The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning.

- Increasing the bias will decrease the variance.
- Increasing the variance will decrease the bias.

There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem

In reality, we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function. Nevertheless, as a framework, bias and variance provide the tools to understand the behavior of machine learning algorithms in the pursuit of predictive performance.

LEARNING CURVE:

Let's say we have some data and split it into a training set and validation set. We take one single instance (that's right, one!) from the training set and use it to estimate a model. Then we measure the model's error on the validation set and on that single training instance. The error on the training instance will be 0, since it's quite easy to perfectly fit a single data point. The error on the validation set, however, will be very large. That's because the model is built around a single instance, and it almost certainly won't be able to generalize accurately on data that hasn't seen before.

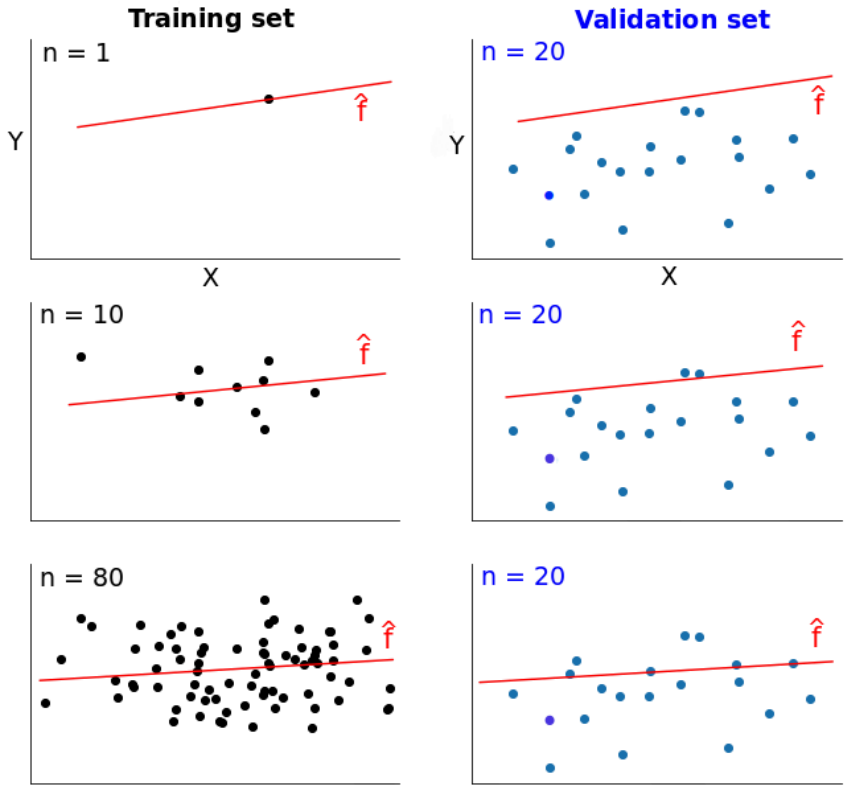
Now let's say that instead of one training instance, we take ten and repeat the error measurements. Then we take fifty, one hundred, five hundred, until we use our entire training set. The error scores will vary more or less as we change the training set.

We thus have two error scores to monitor: one for the validation set, and one for the training sets. If we plot the evolution of the two error scores as training sets change, we end up with two curves. These are called *learning curves*.

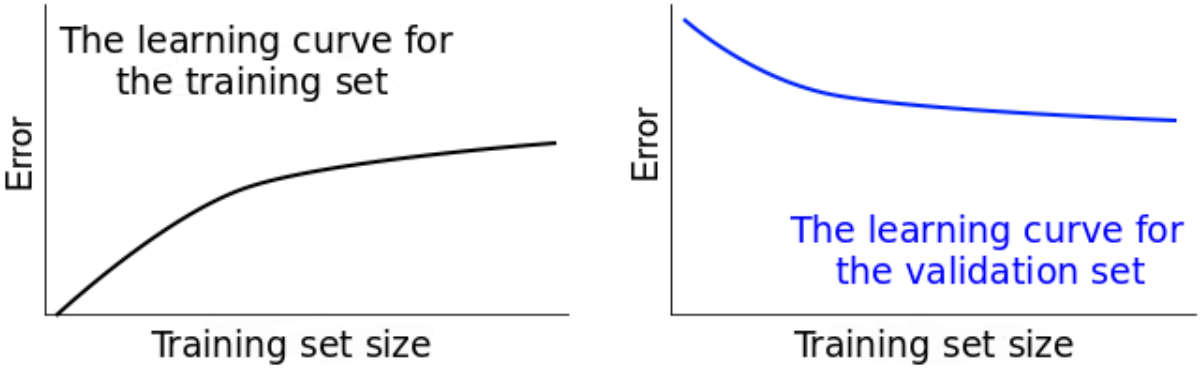
In a nutshell, a learning curve shows how error changes as the training set size increases. The diagram below should help you visualize the process described so far. On the training set column you can see that we constantly increase the size of the training sets. This causes a slight change in our models ^ff^.

In the first row, where n = 1 (*n* is the number of training instances), the model fits perfectly that single training data point. However, the very same model fits really bad a validation set of 20 different data points. So the model's error is 0 on the training set, but much higher on the validation set.

As we increase the training set size, the model cannot fit perfectly anymore the training set. So the training error becomes larger. However, the model is trained on more data, so it manages to fit better the validation set. Thus, the validation error decreases. To remind you, the validation set stays the same across all three cases.



If we plotted the error scores for each training size, we'd get two learning curves looking similarly to these:



The learning curves plotted above are idealized for teaching purposes. In practice, however, they usually look significantly different. So let's move the discussion in a practical setting by using some real-world data.

We'll try to build regression models that predict the hourly electrical energy output of a power plant. The data we use come from Turkish researchers Pınar Tüfekci and Heysem Kaya, and can be downloaded from here. As the data is stored in a .xlsx file, we use pandas' read_excel() function to read it in:
The PE column is the target variable, and it describes the net hourly electrical energy output. All the other variables are potential features, and the values for each are actually hourly averages (not net values, like for PE).
The electricity is generated by gas turbines, steam turbines, and heat recovery steam generators. According to the documentation of the data set, the vacuum level has an effect on steam turbines, while the other three variables affect the gas turbines. Consequently, we'll use all of the feature columns in our regression models.

At this step we'd normally put aside a test set, explore the training data thoroughly, remove any outliers, measure correlations, etc. For teaching purposes, however, we'll assume that's already done and jump straight to generate some learning curves. Before we start that, it's worth noticing that there are no missing values. Also, the numbers are unscaled, but we'll avoid using models that have problems with unscaled data.

Deciding upon the training set sizes

Let's first decide what training set sizes we want to use for generating the learning curves.

The minimum value is 1. The maximum is given by the number of instances in the training set. Our training set has 9568 instances, so the maximum value is 9568.

However, we haven't yet put aside a validation set. We'll do that using an 80:20 ratio, ending up with a training set of 7654 instances (80%), and a validation set of 1914 instances (20%). Given that our training set will have 7654 instances, the maximum value we can use to generate our learning curves is 7654.

POSSIBLE QUESTIONS

6 MARKS

1. Explain Learning models in detail.
2. Describe the types of learning in detail.
3. Write a note on (i) error and noise (ii) training versus testing
4. Explain the theory of generalization in detail.

10 MARKS

1. Explain the types of leaning in detail with suitable example.

2. Write a brief note on Models in learning.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of CS,CA & IT

III B.Sc(CS) (BATCH 2016-2019)

Machine Learning

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS **ONE MARK QUESTIONS**

S.NO	QUESTIONS	OPTION1	OPTION2	OPTION3	OPTION4	ANSWER
1	_____is a field of computer science that deals with system programming to learn and improve with experience.	Artifical Intelligence	Machine learning	Data Mining	Model Selection.	Machine learning
2	The process of choosing models among diverse mathematical models, which are used to define the same data set is known as _____	Model Selection.	Data Mining	Artifical Intelligence	Machine learning	Model Selection.

3	Machine learning is_____	The autonomous acquisition of knowledge through the use of computer programs	The autonomous acquisition of knowledge through the use of manual programs	The selective acquisition of knowledge through the use of computer programs	The selective acquisition of knowledge through the use of manual programs	The autonomous acquisition of knowledge through the use of computer programs
4	Who provided a formal definition of machine learning?	Tom A. Mitchell	Tom M. Mitchell	Tom K. Mitchell	Tom J. Mitchell	Tom M. Mitchell
5	Machine learning is divided into how many categories?	2	3	4	5	2
6	Inputs are divided into how many in Classifications?	2	3	4	5	2
7	What happens in clustering?	An exponential of the inputs is found	Inputs are multiplied	Inputs are divided into groups	Sets are multiplied	Inputs are divided into groups

8	Computers are best at learning	facts	concepts	procedures	principles	facts
9	Data used to build a data mining model.	validation data	training data	test data	hidden data	training data
10	Supervised learning and unsupervised clustering both require at least one	hidden attribute.	output attribute	input attribute	categorical attribute.	hidden attribute.
11	Supervised learning differs from unsupervised clustering in that supervised learning requires	at least one input attribute.	input attributes to be categorical	at least one output attribute	ouput attriubutes to be categorical	input attributes to be categorical
12	Data used to optimize the parameter settings of a supervised learner model.	training	test	verification	validation	validation

13	The average squared difference between classifier predicted output and actual output.	mean squared error	root mean squared error	mean absolute error	mean relative error	mean squared error
14	The process of forming general concept definitions from examples of concepts to be learned.	deduction	abduction	induction	conjunction	induction
15	Data mining is best described as the process of	identifying patterns in data.	deducing relationships in data.	representing data.	simulating trends in data.	identifying patterns in data.
16	Computers are best at learning	facts.	concepts.	procedures.	principles.	concepts.
17	Like the probabilistic view, the _____ view allows us to associate a probability of membership with each classification.	exemplar	deductive	classical	inductive	exemplar

18	Data used to build a data mining model.	validation data	training data	test data	hidden data	training data
19	Supervised learning and unsupervised clustering both require at least one	hidden attribute.	output attribute.	input attribute.	categorical attribute.	input attribute.
20	Supervised learning differs from unsupervised clustering in that supervised learning requires	at least one input attribute.	input attributes to be categorical.	at least one output attribute.	ouput attriubutes to be categorical.	at least one output attribute.
21	Database query is used to uncover this type of knowledge.	deep	hidden	shallow	multidimensional	shallow
22	A statement to be tested.	theory	procedure	principle	hypothesis	hypothesis

23	A person trained to interact with a human expert in order to capture their knowledge.	knowledge programmer	knowledge developer	knowledge engineer	knowledge extractor	knowledge engineer
24	Which of the following is not a characteristic of a data warehouse?	contains historical data	designed for decision support	stores data in normalized tables	promotes data redundancy	stores data in normalized tables
25	A structure designed to store data for decision support.	operational database	flat file	decision tree	data warehouse	data warehouse
26	A nearest neighbor approach is best used	with large-sized datasets.	when irrelevant attributes have been removed from the data.	when a generalized model of the data is desireable.	when an explanation of what has been found is of primary importance.	when irrelevant attributes have been removed from the data.
27	If a customer is spending more than expected, the customer's intrinsic value is _____ their actual value.	greater than	less than	less than or equal to	equal to	less than

28	_____ can be any unprocessed fact, value, text, sound or picture that is not being interpreted and analyze	data	knowledge	information	machine	data
29	____ has been interpreted and manipulated and has now some meaningful inferences for the users.	data	knowledge	information	machine	data
30	_____ is the combination of inferred information and learning.	data	knowledge	information	machine	knowledge
31	_____ is the part of data we use to train our model.	training data	testing data	validation data	knowledge	training data
32	How we split data in Machine Learning?	3	5	6	8	3

33	_____ means scale of data	volume	value	velocity	veracity	volume
34	Which defined correctness in data?	volume	value	velocity	veracity	veracity
35	Supervised Learning is also called as _____.	Inductive Learning	semi-supervised	regression	labeled	Inductive Learning
36	Which dataset is one which has both input and output parameters.	labeled	unlabelled	function	model	labeled
37	What is another name for meaningless data?	unstructured data	structured data	labeled data	value	unstructured data

38	The data which contains only an input parameters.	unstructured data	structured data	unlabeled data	value	unlabeled data
39	_____ is a classification algorithm for binary and multi class classification problems.	naïve bayes	bayes stephen	bias	flemming bayes	naïve bayes
40	_____is a sub-field of mathematics concerned with vectors, matrices, and linear transforms.	linear algebra	linear graphs	linear arrays	linear matrix	linear algebra
41	_____ is a method of teaching and learning in a logical manner.	Machine learning	PAC Learning	Artifical Intelligence	Sequence learning	Sequence learning
42	_____is about identifying group membership while regression technique involves predicting a response	Classification	Association	Regression models	Clustering	Classification

43	Which training data includes a few desired outputs?	Inductive Learning	semi-supervised	regression	labeled	semi-supervised
44	The way candidate programs are generated known as the _____ process.	search	hypotheses	evaluation	knowledge	search
45	Which is called as idiot?	navie	navie bayes	bias	flemming bayes	navie bayes
46	The field of study that gives computers the capability to learn without being explicitly programmed is known as _____.	machine learning	data learning	testing learning	type learning.	machine learning
47	_____ defined labels.	classification	regression	supervised learning	data warehouse	classification

48	Predictive models having target attribute having discrete values can be termed as _____	Regression models	Classification models	supervised learning	data warehouse	Classification models
49	When was the name coined?	1987	1959	1978	1990	1959
50	_____is based on an assumption that all of the features in the data set are important, equal and independent.	navie	navie bayes	bias	flemming bayes	navie bayes
51	_____ is a process or a study whether it closely relates to design, development of the algorithms that provide an ability to the machines to capacity to learn.	Model Selection.	Data Mining	Artifical Intelligence	Machine learning	Machine learning
52	_____technique is a rule based ML technique which finds out some very useful relations between parameters of a large data set.	Classification models	Association	Regression models	data warehouse	Association

53	_____ technique is mostly applicable in case of image data-sets where usually all images are not labeled.	Inductive Learning	semi-supervised	regression	labeled	semi-supervised
54	_____model keeps on increasing its performance using a Reward Feedback to learn the behavior or pattern.	supervised	semi-supervised	reinforcement	unsuervised	reinforcement
55	Which is example of supervised learning algorithm?	K-Means Clustering	Decision Trees	Temporal Difference (TD)	Q-Learning	Decision Trees
56	With Bayes classifier, missing data items are	treated as equal compares.	treated as unequal compares.	replaced with a default value.	ignored.	treated as unequal compares.
57	This unsupervised clustering algorithm terminates when mean values computed for the current iteration of the algorithm are identical to the computed mean values for the previous iteration.	agglomerative clustering	conceptual clustering	K-Means clustering	expectation maximization	K-Means clustering

58	Machine learning techniques differ from statistical techniques in that machine learning methods	typically assume an underlying distribution for the data.	are better able to deal with missing and noisy data.	are not able to explain their behavior.	have trouble with large-sized datasets.	are better able to deal with missing and noisy data.
59	The_____ involves the process of learning by examples, where a system, from a set of observed instances tries to induce a general rule.	semi-supervised	regression	Inductive machine learning	Artificial Intelligence	Inductive machine learning
60	_____ techniques allow learning a function or predictor from a set of observed data that can make predictions about unseen or future data.	Statistical learning techniques allow learning a function or predictor from a set of observed	Artificial Intelligence	inductive machine learning	machine learning	Statistical learning techniques allow learning a function or predictor from a set of observed data that can make predictions about unseen or

CLASS: I M.Sc CS

COURSE CODE: 19CSP205A UNIT II: LINEAR MODELS BATCH: 2019-2021

UNIT-II

SYLLABUS

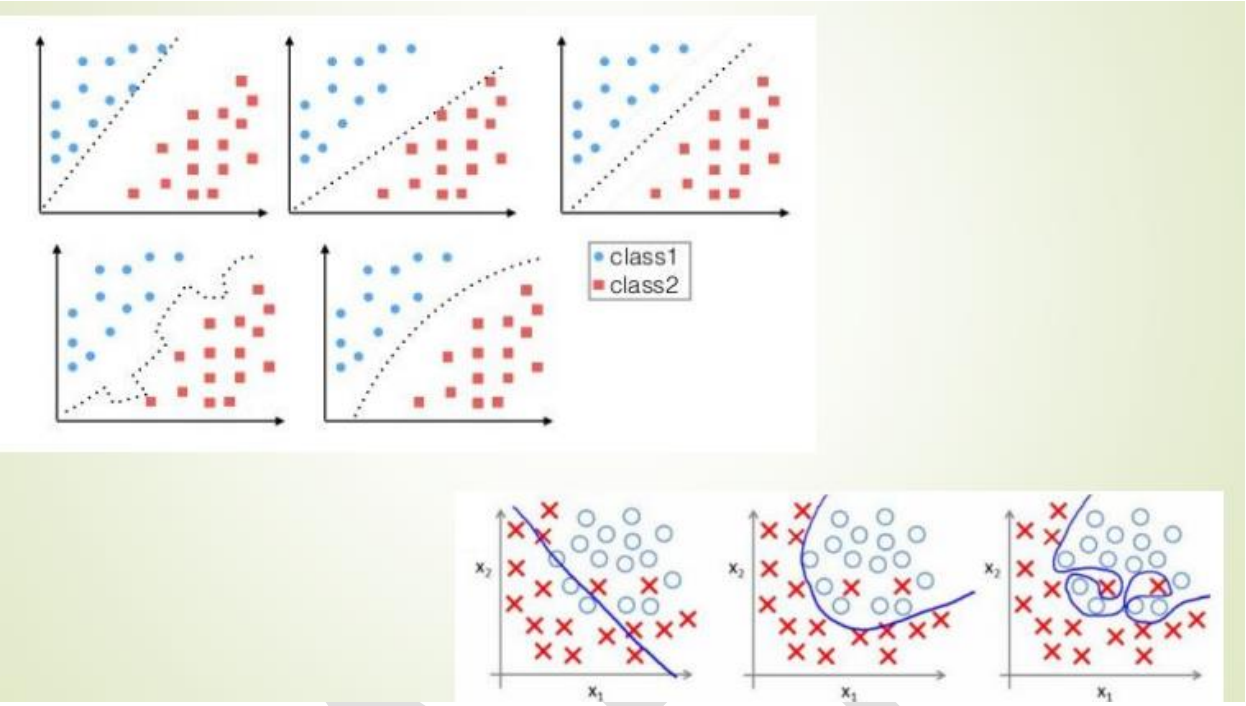
LINEAR MODELS : Linear classification – univariate linear regression – multivariate linear regression – regularized regression – Logistic regression – perceptrons – multilayer neural networks – learning neural networks structures – support vector machines – soft margin SVM – generalization and over fitting – regularization – validation

LINEAR CLASSIFICATION:

In the field of machine learning, the goal of statistical classification is to use an object's characteristics to identify which class (or group) it belongs to. A **linear classifier** achieves this by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector. Such classifiers work well for practical problems such as document classification, and more generally for problems with many variables (features), reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use.[1]

Classification:

- Use an object characteristics to identify which class/category it belongs to
- Example:
- a new email is ‘spam’ or ‘non-spam’
- A patient diagnosed with a disease or not
- Classification is an example of pattern recognition



Linear classification

 A classification algorithm (Classifier) that makes its classification based on a linear predictor function combining a set of weights with the feature vector

 Decision boundaries is flat

 Line, plane, ….

 May involve non-linear operations

UNIVARIATE LINEAR FUNCTION:

Univariate linear regression focuses on determining relationship between one independent (explanatory variable) variable and one dependent variable. Regression comes handy mainly in situation where the relationship between two features is not obvious to the naked eye. For example, it could be used to study how the terrorist attacks frequency affects the economic growth of countries around the world or the role of unemployment in a country in the bankruptcy of the government.

Simple linear regression

Given a dataset of variables (xi,yi) where xi is the explanatory variable and yi is the dependent variable that varies as xi does, the simplest model that could be applied for the relation between two of them is a linear one. Simple linear regression model is as follows:

$yi=\alpha+\beta*xi+\epsilon i$

ϵi is the random component of the regression handling the residue, i.e. the lag between the estimation and actual value of the dependent parameter. If Y is the estimation value of the dependent variable, it is determined by two parameters:

1. The core parameter term $\alpha+\beta*xi$ which is not random in nature. α is known as the constant term or the intercept (also is the measure of the y-intercept value of regression line). β is the coefficient term or slope of the intercept line.
2. Above explained random component, ϵi.

A cosmetic company wants to know its customer base. They have a intuition that the money spent by a person on cosmetics yearly is related to his/her annual income. Given **N** data entries collected from a survey.

Help the cosmetic company build a linear regression model over the parameters. Report α & βvalues for it.

Input constraints

$N\geq 2$ $xi\geq 0$, $yi\geq 0$

Input format

First line: N

Next N lines have x,y values (x: annual income, y: money spent in cosmetics (in dollars))

x1 y1

x2 y2

....

xN yN

Output format

α β

MULTIVARIATE LINEAR REGRESSION:

Generally one dependent variable depends on multiple factors. For example, the rent of a house depends on many factors like the neighborhood it is in, size of it, no.of rooms, attached facilities, distance of nearest station from it, distance of nearest shopping area from it, etc. How do we deal with such scenarios? Let's jump into **multivariate linear regression** and figure this out.

This is quite similar to the simple linear regression model we have discussed previously, but with multiple independent variables contributing to the dependent variable and hence multiple coefficients to determine and complex computation due to the added variables. Jumping straight into the equation of multivariate linear regression,

$$Yi=\alpha+\beta 1xi(1)+\beta 2xi(2)+....+\beta nxi(n)$$

Yi is the estimate of ith component of dependent variable y, where we have **n** independent variables and xij denotes the ith component of the jth independent variable/feature. Similarly cost function is as follows,

$$E(\alpha,\beta 1,\beta 2,...,\beta n)=12m\sum i=1m(yi-Yi)$$

where we have **m** data points in training data and y is the observed data of dependent variable. As per the formulation of the equation or the cost function, it is pretty straight forward generalization of simple linear regression. But computing the parameters is the matter of interest here.

Alice and Bob are planing to buy a new home. Nancy has mentioned a rate for the home they like, but they want a method to verify it. All they have got is the data set of size **M** house price and 3 feature counts (no.of bedrooms, size of home in sq mtrs and age of the home). Bob has decided to use his statistical data processing experience from his previous job and do a multivariate linear regression. If **B**,**S**,**A** are the parameter values of the house they like and **P** is price mentioned by Nancy, help Bob decide if they are being cheated or not. (If the price mentioned by Nancy - expected price ≥ 2000 dollars then they are being cheated.)

Input constraints

P,B,S,A≥0, M≥2

Input format

Line *1*: M value

Next M lines contain 4 values separated by spaces

Line *2* to m+1: Bi Si Ai Pi

Line m+2: *B S A* (features of their future home)

Line m+3: *P* (price mentioned by Nancy)

Output format

P_{exp} *C*

P_{exp} is expected price of their home and *C* is a binary value (0: being cheated, 1: not cheated)

REGULARIZED REGRESSION:

This is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.

A simple relation for linear regression looks like this. Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X).

$Y \approx \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta pXp$

The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

Now, this will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

Ridge Regression

Above image shows ridge regression, where the RSS is modified by adding the shrinkage quantity. Now, the coefficients are estimated by minimizing this function. Here, λ is the tuning parameter that decides how much we want to penalize the flexibility of our model. The increase in flexibility of a model is represented by increase in its coefficients, and if we want to minimize the above function, then these coefficients need to be small. This is how the Ridge regression technique prevents coefficients from rising too high. Also, notice that we shrink the estimated association of each variable with the response, except the intercept β0, This intercept is a measure of the mean value of the response when xi1 = xi2 = …= xip = 0.

When λ = 0, the penalty term has no effect, and the estimates produced by ridge regression will be equal to least squares. However, as λ→∞, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero. As can be seen, selecting a good value of λ is critical. Cross validation comes in handy for this purpose. The coefficient estimates produced by this method are also known as the L2 norm.

The coefficients that are produced by the standard least squares method are scale equivariant, i.e. if we multiply each input by c then the orresponding coefficients are scaled by a factor of 1/c. Therefore, regardless of how the predictor is scaled, the multiplication of predictor and coefficient(Xjβj) remains the same. However, this is not the case with ridge regression, and therefore, we need to standardize the predictors or bring the predictors to the same scale before performing ridge regression. The formula used to do this is given below.

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2}}.$$

LOGISTIC REGRESSION:

Logistic regression is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary / categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to alogit function.

Logistic regression was developed by statistician David Cox in 1958. This binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). It allows one to say that the presence of a risk factor increases the probability of a given outcome by a specific percentage.

Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Application of Logistic Regression :

It’s being used in Healthcare , Social Sciences & various ML for advanced research & analytics.

Example:

TRISS : Trauma & Injury Severity Score, which is widely used to predict mortality in injured patients, was originally developed by Boyd et al. using logistic regression. Many other medical scales used to assess severity of a patient have been developed using logistic regression.

Logistic Regression Function:

Most often, we would want to predict our outcomes as YES/NO (1/0).

For example:

Is your favorite football team going to win the match today?—yes/no (0/1)

Does a student pass in exam?—yes/no (0/1)

The logistic function is given by:

f(x) = L/(1+e ^-k(x-x0))

where

L – Curve's maximum value

k – Steepness of the curve

x0 – x value of Sigmoid's midpoint

A standard logistic function is called sigmoid function (k=1,x0=0,L=1)

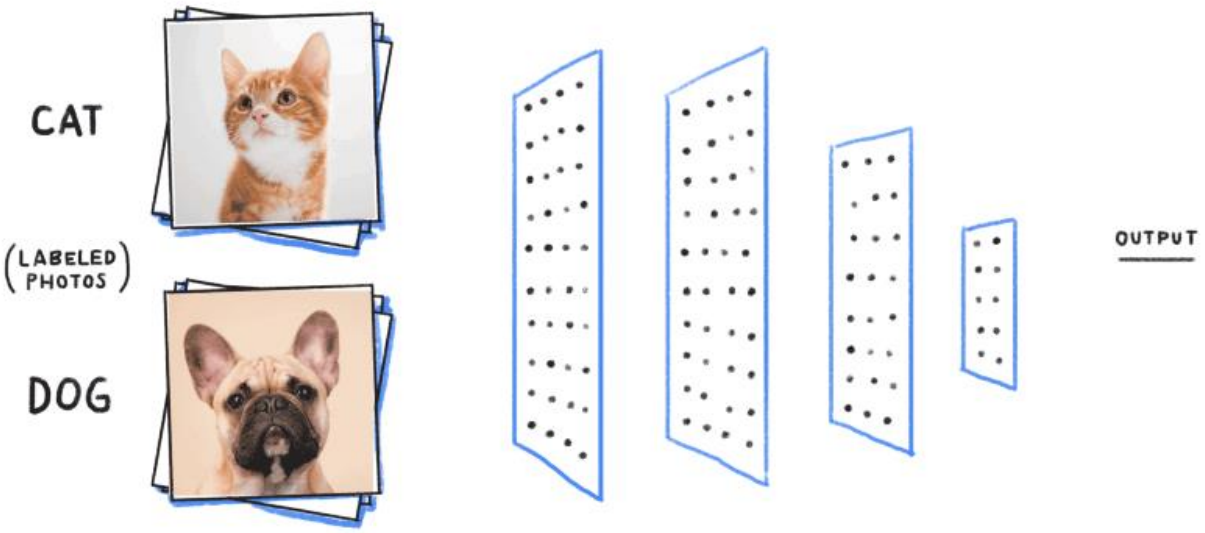
S(x) = 1/1+ (e ^- x)

PERCEPTRON:

Perceptron is a single layer neural network *and a multi-layer perceptron is called Neural Networks.*

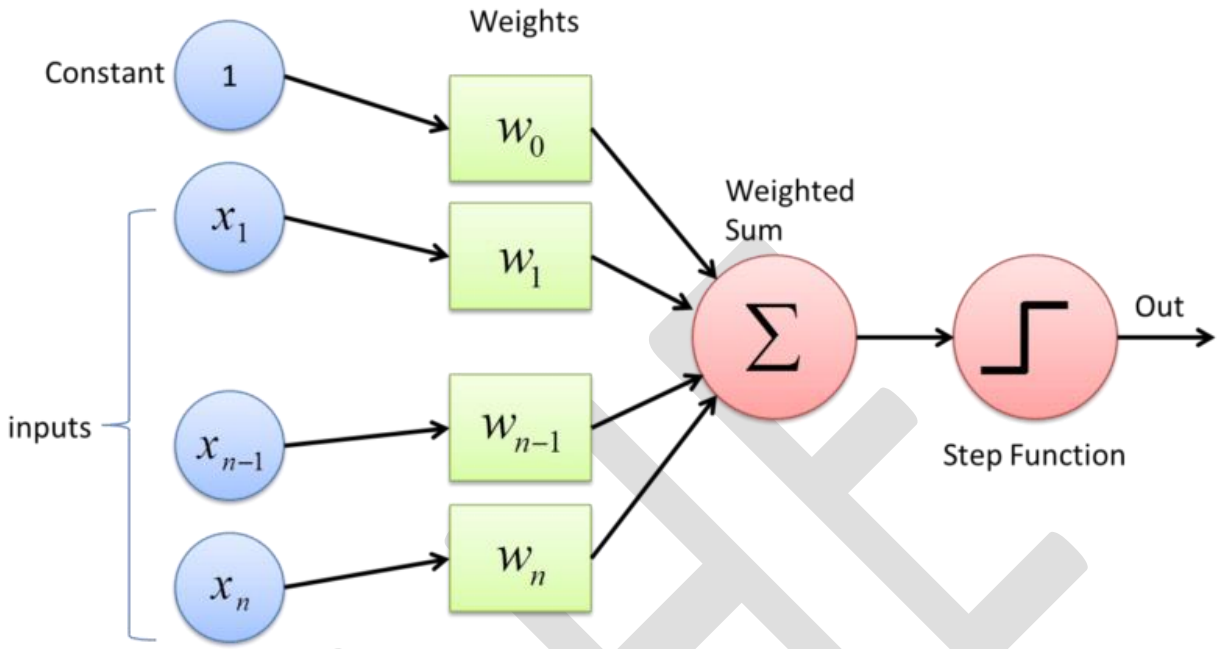
Perceptron is a linear classifier (binary). Also, it is used in supervised learning. It helps to classify the given input data. But how the heck it works ?

A normal neural network looks like this as we all know



The perceptron consists of 4 parts .

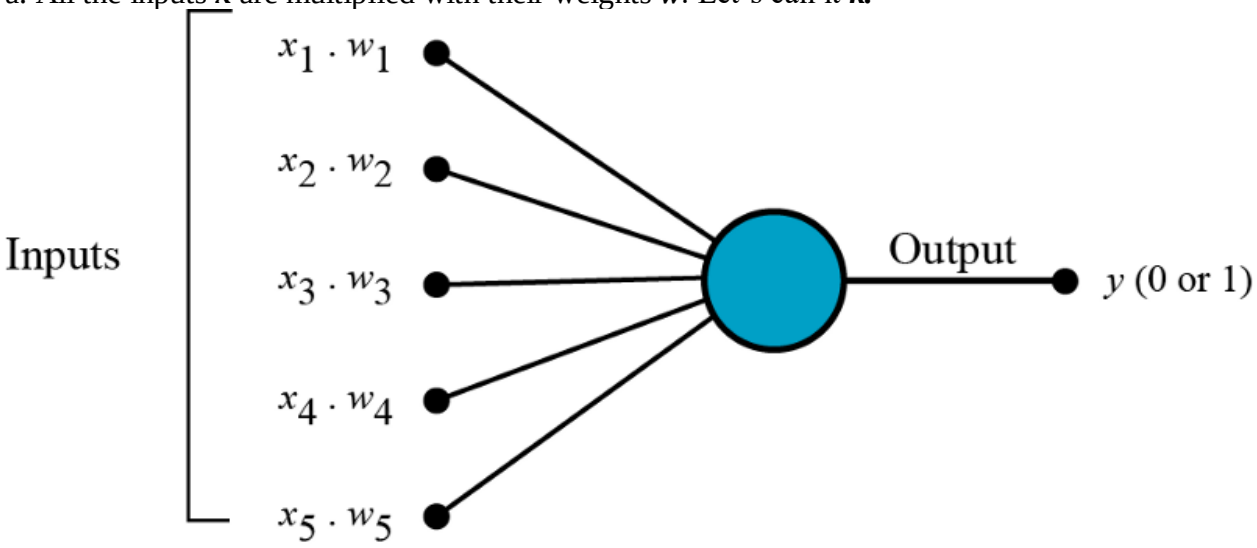
1. Input values or One input layer
2. Weights and Bias
3. Net sum
4. Activation Function



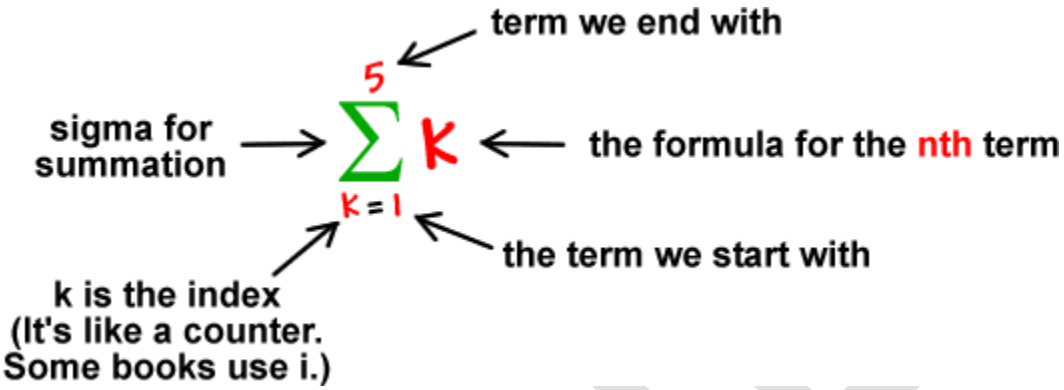
But how does it work?

The perceptron works on these simple steps

a. All the inputs ***x*** are multiplied with their weights ***w***. Let's call it ***k.***

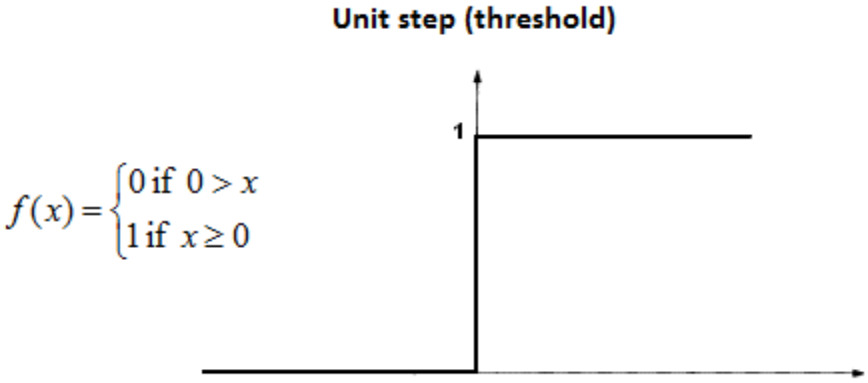


b. ***Add*** all the multiplied values and call them ***Weighted Sum.***



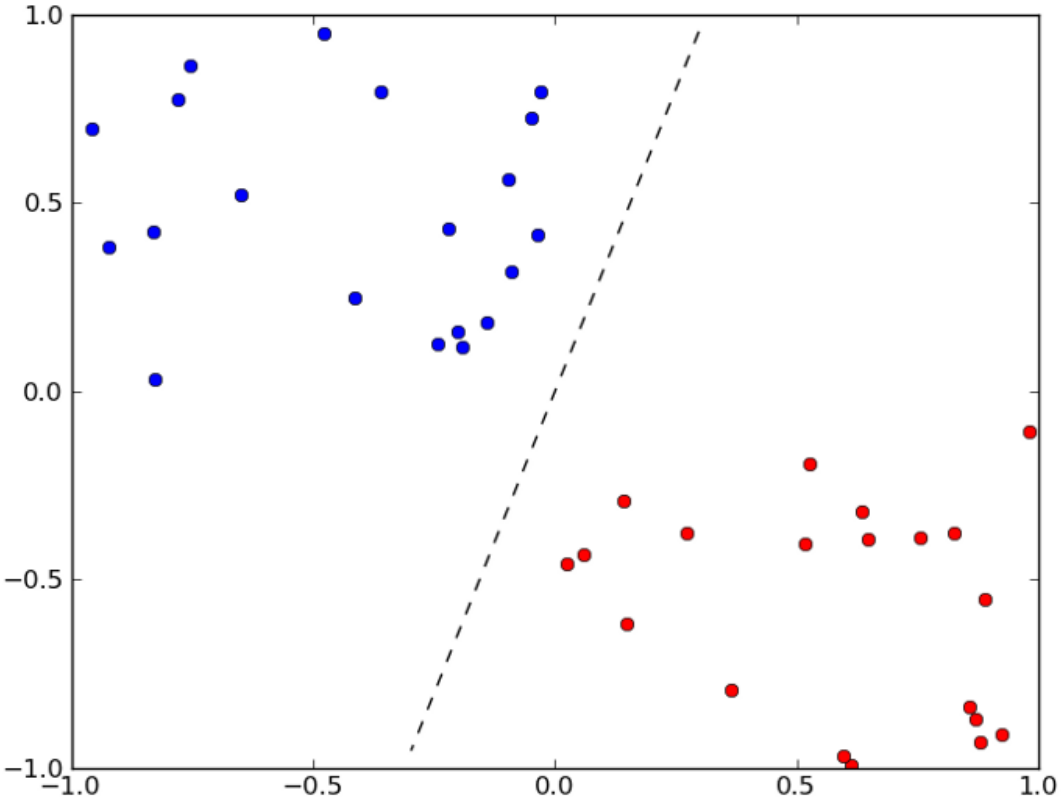
c. Apply that weighted sum to the correct ***Activation Function***.

For Example : Unit Step Activation Function.



Where we use Perceptron?

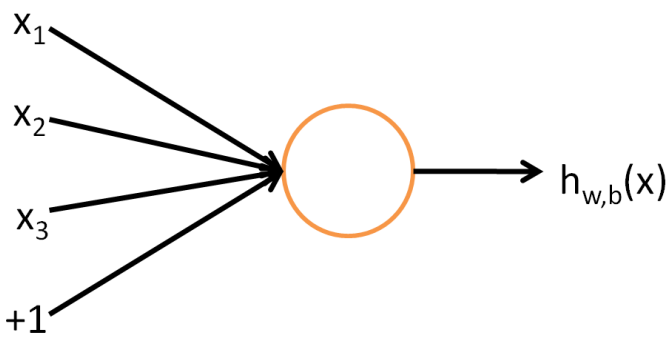
Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.



MULTILAYER NEURAL NETWORKS:

Consider a supervised learning problem where we have access to labeled training examples (x(i),y(i))(x(i),y(i)). Neural networks give a way of defining a complex, non-linear form of hypotheses hW,b(x)hW,b(x), with parameters W,bW,b that we can fit to our data.

To describe neural networks, we will begin by describing the simplest possible neural network, one which comprises a single "neuron." We will use the following diagram to denote a single neuron:



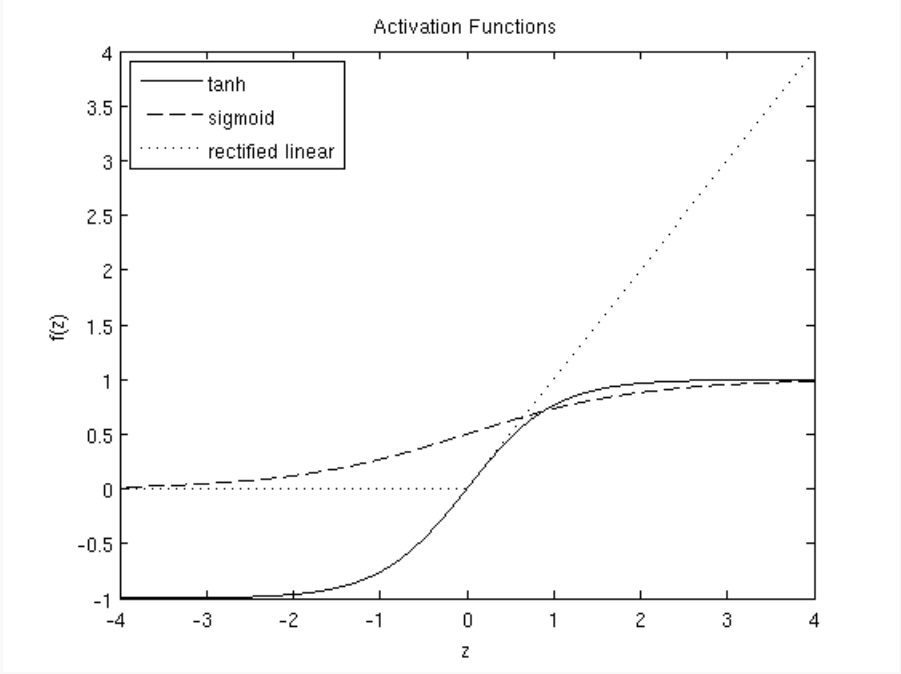
This "neuron" is a computational unit that takes as input x1,x2,x3x1,x2,x3 (and a +1 intercept term), and outputs hW,b(x)=f(WTx)=f(∑3i=1Wixi+b)hW,b(x)=f(WTx)=f(∑i=13Wixi+b), where f:R↦Rf:ℜ↦ℜ is called the **activation function**. In these notes, we will choose f(·)f(·) to be the sigmoid function:

F(z)=1/1+exp(-z)

Thus, our single neuron corresponds exactly to the input-output mapping defined by logistic regression.

Recent research has found a different activation function, the rectified linear function, often works better in practice for deep neural networks. This activation function is different from sigmoid and tanhtanhbecause it is not bounded or continuously differentiable. The rectified linear activation function is given by,
f(z)=max(0,x).f(z)=max(0,x).

Here are plots of the sigmoid, tanhtanh and rectified linear functions:

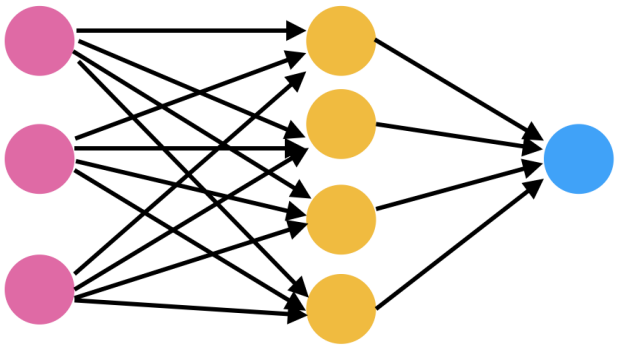


The tanh(z)tanh(z) function is a rescaled version of the sigmoid, and its output range is [−1,1][−1,1] instead of [0,1][0,1]. The rectified linear function is piece-wise linear and saturates at exactly 0 whenever the input z is less than 0.

Note that unlike some other venues we are not using the convention here of x0=1x0=1. Instead, the intercept term is handled separately by the parameter b.

LEARNING NEURAL NETWORK STRUCTURES

To understand how neural networks function, it is helpful to know the different parts of a network. Suppose we have the following network:



- The left pink circles represent what we call the network's **input layer** where each circle represents an **input neuron**.
- The rightmost blue layer is known as the **output layer** and contains the **output neuron(s)**.
- The middle layer is called the **hidden layer**, or simply, not an input or output. The example above has only one single layer made of 4 neurons, but networks can have multiple hidden layers.
- Each circle represents a **neuron**. We can think of each neuron as a function that takes the output of all neurons in the previous layer and spits out a number between 0 and 1.
- Synapse

Basic steps:

1. Our input neurons represent an input based on the information we are trying to classify
2. Each number in the input neurons is given a weight at each synapse
3. At each neuron in the next layer, we add the outputs of all synapses coming to that neuron along with a bias and apply an activation function (commonly a sigmoid function) to the weighted sum (this makes the number something between 0 and 1)

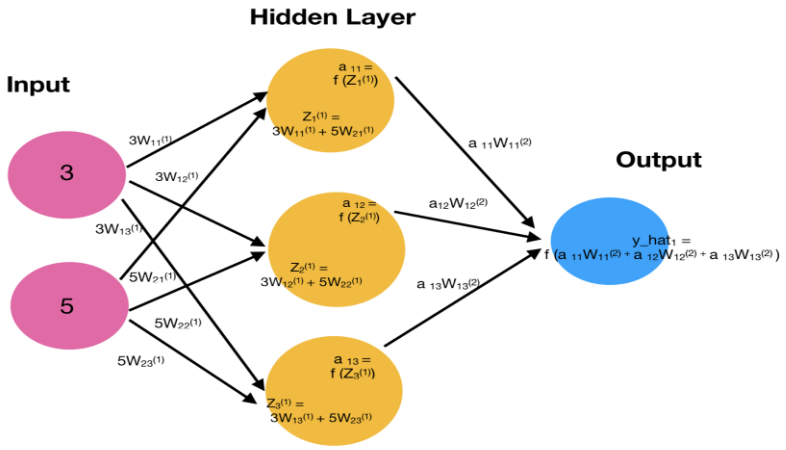
   The output of that function will be treated as the input for the next synapse layer

   Continue until you reach the output

   Let's say the input is the hours you slept and hours you studied; and you are trying to figure out what you will score on a test as a result. Some observations can be seen below:

X hours of sleep, hours spent studying	Y score on test
3 , 5	75
5 , 1	82
10 , 2	93
3 , 3	?

To be able to predict a score based on hours slept and hours spent studying, we need to train a model. Let's first just look at how these inputs would be processed through a neural network. Let's look at only the first input value (3,5). A representation of this process can be seen in the diagram below.



The input layer has two input neurons based on the hours of sleep and hours spent studying. We have a single output neuron for the score on the test. The number of layers and number of neurons in the hidden layer(s) is a bit arbitrary and we usually experiment to decide what works best for our particular model. It is important to note, however, that once this is chosen, it does not change throughout the model. It is considered one of our hyper parameters. The "learning" part of a neural net, and the parameters we train, are the various weights applied at each synapse.

1. **Our input neurons represent an input based on the information we are trying to predict/classify**

Here we put 3 in one input neuron and 5 in the other to represent hours of sleep and hours spent studying; these make up a single input observation.

2. Each number in the input neurons is given a weight at each synapse

The number of hours of sleep and the number of hours spent studying are weighted differently at each synapse

At each neuron in the next layer, we add the outputs of all synapses coming to that neuron and a bias and apply an activation function to the weighted sum (this makes the number something between 0 and 1)

The results of each synapse are added together to give us Zi(1) which is then passed into an activation function to give us a1i (this will be our activation number with a value between 0 and 1)

4. The output of that function will be treated as the input for the next synapse layer

The result of this function is treated as the new input and weighted

Continue until you reach the output

Since this example has only one hidden layer, the results of these synapses are added together and passed into a sigmoid activation function to give us our prediction for the first observation

The diagram above shows us the process for a single input. Doing this one at a time for each input pair is a bit cumbersome and would result in a very slow and inefficient code. Instead, we commonly process all inputs at one time using matrices.

To see how this works, we should first realize that we can represent the step from the input neurons to hidden layer neurons for all inputs as follows:

$$\begin{bmatrix} 3 & 5 \\ 5 & 1 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{bmatrix} = \begin{bmatrix} 3W_{11}^{(1)} + 5W_{21}^{(1)} & 3W_{12}^{(1)} + 5W_{22}^{(1)} & 3W_{13}^{(1)} + 5W_{23}^{(1)} \\ 5W_{11}^{(1)} + 1W_{21}^{(1)} & 5W_{12}^{(1)} + 1W_{22}^{(1)} & 5W_{13}^{(1)} + 1W_{23}^{(1)} \\ 10W_{11}^{(1)} + 2W_{21}^{(1)} & 10W_{12}^{(1)} + 2W_{22}^{(1)} & 10W_{13}^{(1)} + 2W_{23}^{(1)} \end{bmatrix}$$

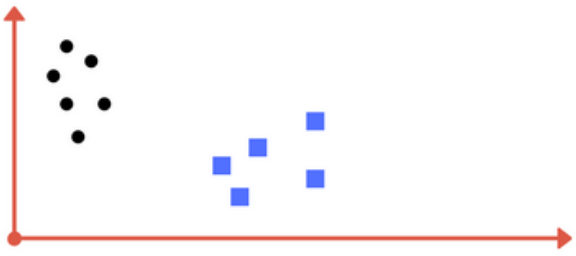
$X \qquad W^{(1)} \qquad Z^1$

What we have in the red circle is the same thing we get in the diagram for Z1(1) Z2(1) Z3(1). As we can see, with matrix multiplication, we can quickly run through the process. All we need to do is write our results from the neurons in one layer as a column matrix and organize our weights as a row matrix. By matrix multiplication, we can get the weighted sum for each input observation. Building from this, the steps are quite similar as before but deal with all inputs at once.

SUPPORT VECTOR MACHINE:

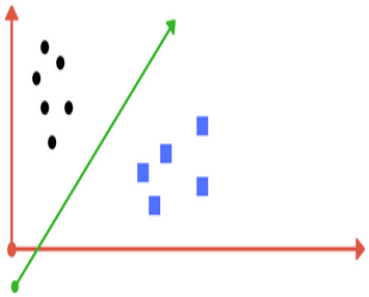
*A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line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

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

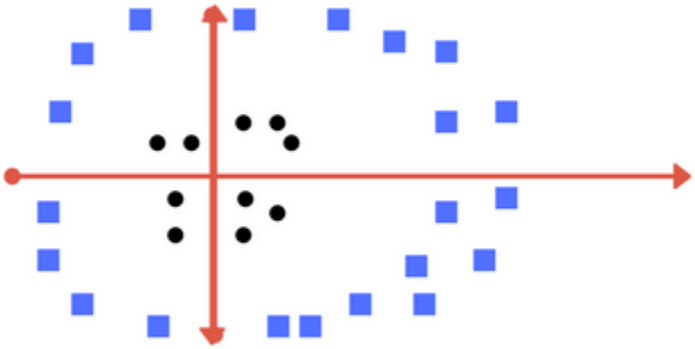


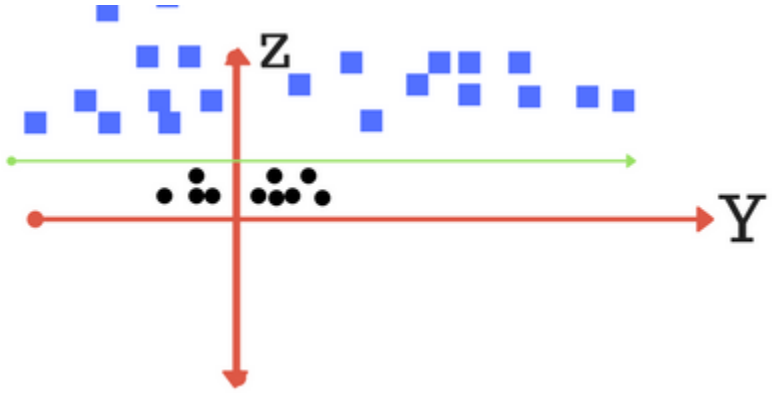
You might have come up with something similar to following image (*image B*). It fairly separates the two classes. Any point that is left of line falls into black circle class and on right falls into blue square class. ***Separation of classes. That's what SVM does.*** It finds out a line/ hyper-

plane (in multidimensional space that separate outs classes). Shortly, we shall discuss why I wrote multidimensional space.

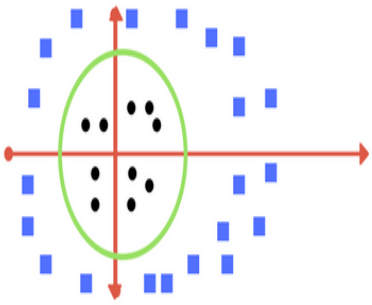


Lets assume value of points on z plane, $w = x^2 + y^2$. In this case we can manipulate it as distance of point from z-origin.





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



SOFT MARGIN SVM:

real data is messy and cannot be separated perfectly with a hyperplane.

The constraint of maximizing the margin of the line that separates the classes must be relaxed. This is often called the soft margin classifier. This change allows some points in the training data to violate the separating line.

An additional set of coefficients are introduced that give the margin wiggle room in each dimension. These coefficients are sometimes called slack variables. This increases the complexity of the model as there are more parameters for the model to fit to the data to provide this complexity.

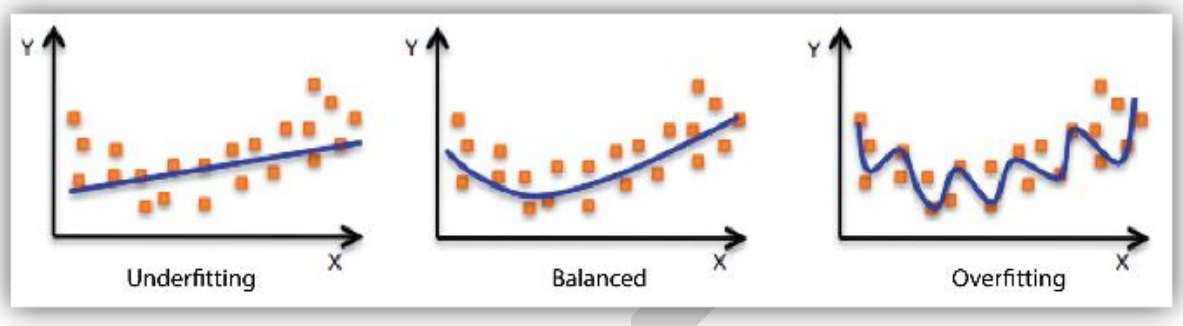
A tuning parameter is introduced called simply C that defines the magnitude of the wiggle allowed across all dimensions. The C parameters defines the amount of violation of the margin allowed. A C=0 is no violation and we are back to the inflexible Maximal-Margin Classifier described above. The larger the value of C the more violations of the hyperplane are permitted.

During the learning of the hyperplane from data, all training instances that lie within the distance of the margin will affect the placement of the hyperplane and are referred to as support vectors. And as C affects the number of instances that are allowed to fall within the margin, C influences the number of support vectors used by the model.

- The smaller the value of C, the more sensitive the algorithm is to the training data (higher variance and lower bias).
- The larger the value of C, the less sensitive the algorithm is to the training data (lower variance and higher bias).

GENERALIZATION AND OVERFITTING

Generalization is a term used to describe a model's ability to react to new data. That is, after being trained on a training set, a model can digest new data and make accurate predictions. A model's ability to generalize is central to the success of a model. If a model has been trained too well on training data, it will be unable to generalize. It will make inaccurate predictions when given new data, making the model useless even though it is able to make accurate predictions for the training data. This is called overfitting. The inverse is also true. Underfitting happens when a model has not been trained enough on the data. In the case of underfitting, it makes the model just as useless and it is not capable of making accurate predictions, even with the training data.



The figure demonstrates the three concepts discussed above. On the left, the blue line represents a model that is underfitting. The model notes that there is some trend in the data, but it is not specific enough to capture relevant information. It is unable to make accurate predictions for training or new data. In the middle, the blue line represents a model that is balanced. This model notes there is a trend in the data, and accurately models it. This middle model will be able to generalize successfully. On the right, the blue line represents a model that is overfitting. The model notes a trend in the data, and accurately models the training data, but it is too specific. It will fail to make accurate predictions with new data because it learned the training data too well. Next week, I will discuss the specifics of gathering data, formatting data to fit the needs of a model, and different ways to represent data. Data is what a machine learning model uses to make predictions for new situations. It is great to have a model, but without data for the model to interact with, the predictions the model make will be useless.

REGULARIZATION:

This is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, ***this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.***

A simple relation for linear regression looks like this. Here Y represents the learned relation and *β represents the coefficient estimates for different variables or predictors(X).*

$$Y \approx \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta pXp$$

The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

Now, this will adjust the coefficients based on your training data. *If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.*

Ridge Regression

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

Above image shows ridge regression, where the ***RSS is modified by adding the shrinkage quantity.*** Now, the coefficients are estimated by minimizing this function. Here, ***λ is the tuning parameter that decides how much we want to penalize the flexibility of our model.*** The increase in flexibility of a model is represented by increase in its coefficients, and if we want to minimize the above function, then these coefficients need to be small. This is how the Ridge regression technique prevents coefficients from rising too high. Also, notice that we shrink the estimated association of each variable with the response, except the intercept β0, This intercept is a measure of the mean value of the response when xi1 = xi2 = …= xip = 0.

When λ = 0, the penalty term has no effect, and the estimates produced by ridge regression will be equal to least squares. However, ***as λ→∞, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero***. As can be seen, selecting a good value of λ is critical. Cross validation comes in handy for this purpose. The coefficient estimates produced by this method are ***also known as the L2 norm***.

The coefficients that are produced by the standard least squares method are scale equivariant, i.e. if we multiply each input by c then the corresponding coefficients are scaled by a factor of 1/c. Therefore, regardless of how the predictor is scaled, the multiplication of predictor and coefficient(Xjβj) remains the same. ***However, this is not the case with ridge***

VALIDATION:

Validation Dataset*: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.*

The validation set is used to evaluate a given model, but this is for frequent evaluation. We as machine learning engineers use this data to fine-tune the model hyperparameters. Hence the model occasionally *sees* this data, but never does it "*Learn*" from this. We
se the validation set results and update higher level hyperparameters. So the validation set in a way affects a model, but indirectly.

Validation techniques in machine learning are used to get the error rate of the ML model, which can be considered as close to the true error rate of the population. If the data volume is large enough to be representative of the population, you may not need the validation techniques. However, in real-world scenarios, we work with samples of data that may not be a true representative of the population. This is where validation techniques come into the picture.

different validation techniques:

- Resubstitution
- Hold-out
- K-fold cross-validation
- LOOCV
- Random subsampling
- Bootstrapping

Machine Learning Validation Techniques

Resubstitution

If all the data is used for training the model and the error rate is evaluated based on outcome vs. actual value from the same training data set, this error is called the *resubstitution error*. This technique is called the resubstitution validation technique.

Holdout

To avoid the resubstitution error, the data is split into two different datasets labeled as a training and a testing dataset. This can be a 60/40 or 70/30 or 80/20 split. This technique is called the hold-out validation technique. In this case, there is a likelihood that uneven distribution of different classes of data is found in training and test dataset. To fix this, the training and test dataset is created with equal distribution of different classes of data. This process is called stratification.

K-Fold Cross-Validation

In this technique, k-1 folds are used for training and the remaining one is used for testing as shown in the picture given below.

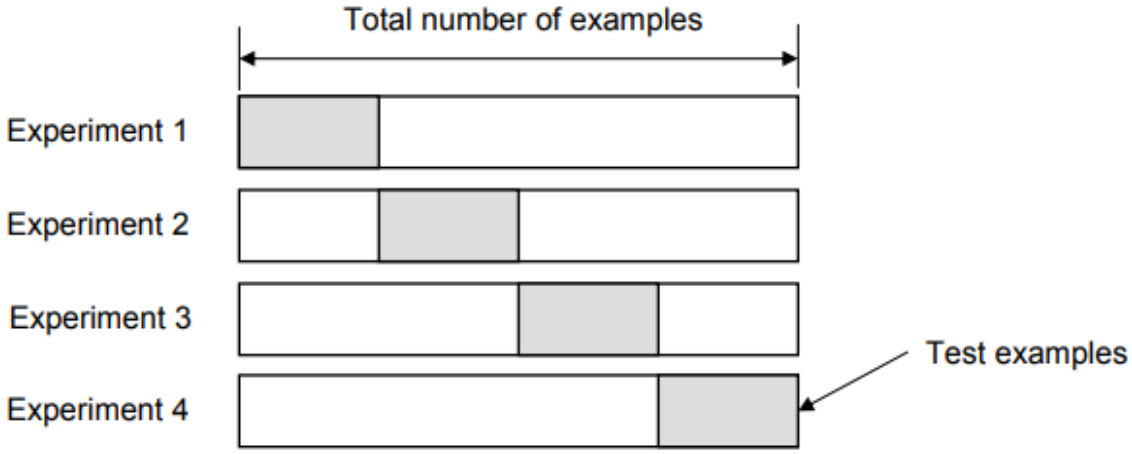


Figure 1: K-fold cross-validation

The advantage is that entire data is used for training and testing. The error rate of the model is average of the error rate of each iteration. This technique can also be called a form the repeated hold-out method. The error rate could be improved by using stratification technique.

Leave-One-Out Cross-Validation (LOOCV)

In this technique, all of the data except one record is used for training and one record is used for testing. This process is repeated for N times if there are N records. The advantage is that entire data is used for training and testing. The error rate of the model is average of the error rate of each iteration. The following diagram represents the LOOCV validation technique.

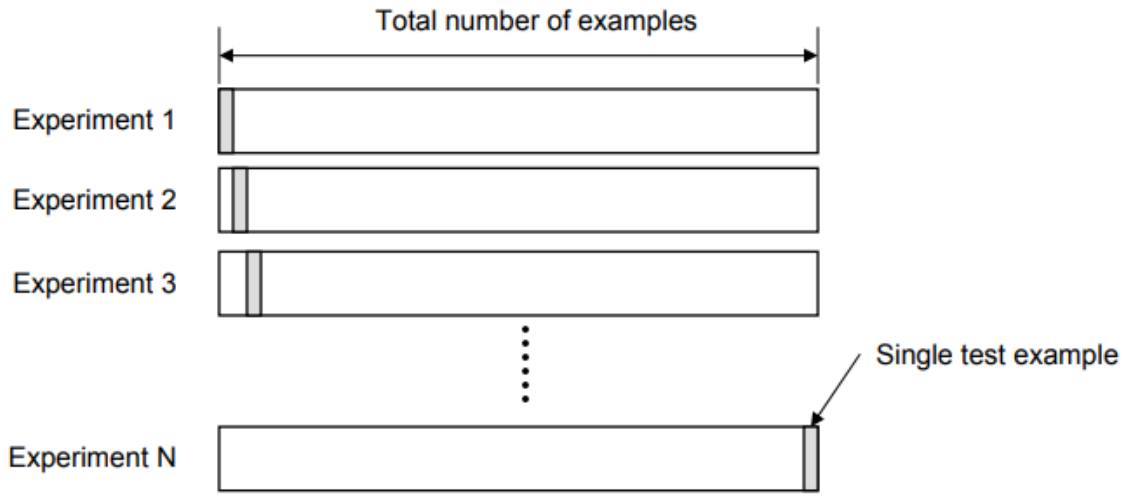


Figure 2: LOOCV validation technique

Random Subsampling

In this technique, multiple sets of data are randomly chosen from the dataset and combined to form a test dataset. The remaining data forms the training dataset. The following diagram represents the random subsampling validation technique. The error rate of the model is the average of the error rate of each iteration.

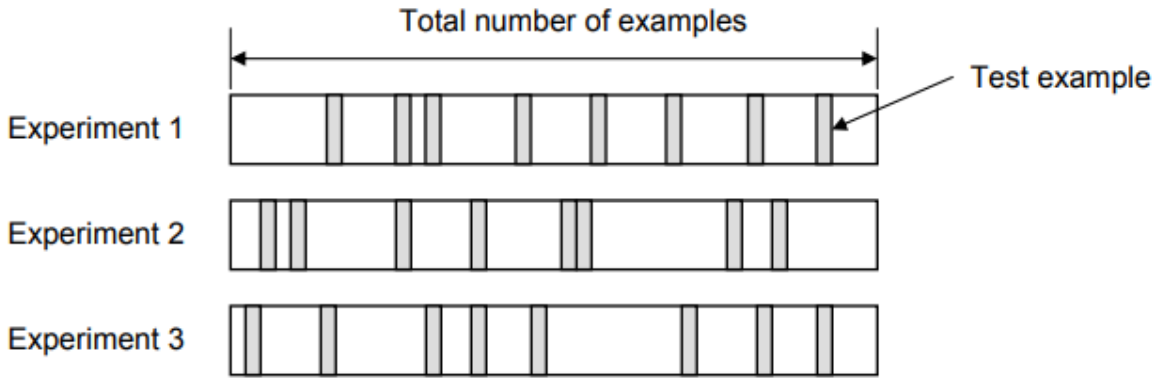
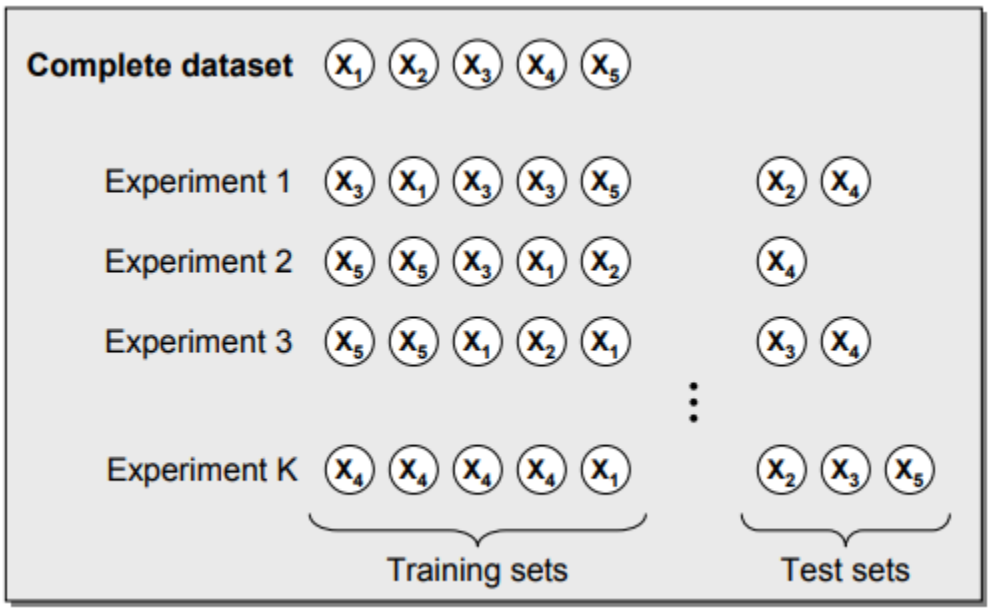


Figure 3: Random subsampling validation technique

Bootstrapping

In this technique, the training dataset is randomly selected with replacement. The remaining examples that were not selected for training are used for testing. Unlike K-fold cross-validation, the value is likely to change from fold-to-fold. The error rate of the model is average of the error rate of each iteration. The following diagram represents the same.



CLASS: I M.Sc CS

COURSE CODE: 19CSP205A UNIT II: LINEAR MODELS BATCH: 2019-2021

POSSIBLE QUESTIONS

PART-B

1. Explain about univariate linear regression in detail.
2. Describe about multivariate linear regression in detail.
3. Write a note on (i) multilayer neural network (ii) perceptrons

PART-C

1. Briefly explain about learning neural network structures.
2. Explain about Validation in detail.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of CS,CA & IT

III B.Sc(CS) (BATCH 2016-2019)

Machine Learning

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS ONE MARK QUESTIONS

QUESTIONS	OPT1	OPT2	OPT3	OPT4	ANSWER
MATLAB stands for _____	Maths Laboratory	Matrix Laboratory	Mathematical Lab	Maths Lab	Matrix Laboratory
MATLAB was developed by _____	MathsWorks	Intel	Microsoft	IBM	MathsWorks
In MATLAB the matrix is defined as an _____	vector	scalar	array	integer	array
_____ acts as an outstanding tool for visulaizing technical data	C	C++	Java	MATLAB	MATLAB
In command window the _____ are entered	datas	values	commands	fiels	commands

_____ window displays plots and graphs	command	Edit	Figure	Command history	Figure
A variable can be deleted from the workspace with the _____ command	delete	remove	clear	omit	clear
The _____ command will display a list of possible help topics in the command window	help	helper	lookfor	order	help
The _____ operator swaps the row and columns of any array that is given	transpose	concatenates	colon	semicolon	transpose
The _____ function can be used ti create an all zero array	ones	zero	eye	randn	zero
The _____ function can be used to generate arrays containing all ones	ones	zero	eye	randn	ones

The _____ function accepts an array arugment and displayus the value of athe array in the command window	disp	format	special	fprintf	disp
The MATLAB command to make a plot is	figure	fit	plot	pplot	plot
The command to add text to the x axis of a plot is	xtitle	label,x	xlabel	xtext	xlabel
The basic building block in MATLAB is _____	matrix	vector	scalar	functions	matrix
The _____ command clears the screen	clc	clr	cls	cle	clc
When the _____ function is executed, MATLAB opens the Figure window and displays the plot in that window	edit	figure	plotting	plot	plot

_____ function is used to find the minimum of given numbers	min	max	medium	poor	min
_____ function is used to find the maximum of given numbers	poor	min	max	medium	max
The _____ command can be used to save a plot as a graphical image by specifying appropriate options and a filename	plot	print	draw	multiple	print
The _____ gives the transpose of x	x'	x''	x'''	x	x'
_____ are operations performed between arrays on an element by element basis	matrix operations	array operations	vector operations	arthimetic operations	array operations
In _____ the number of rows and columns in both arrays must be the same	matrix operations	array operations	vector operations	arthimetic operations	array operations

The term _____ is used to describe an array with only one dimension	array	vector	matrix	scale	vector
The term _____ is used to describe an array with two or more dimensions	array	vector	matrix	scale	matrix
_____ window displays plots and graphs	command	Edit	Figure	Command history	Figure
In MATLAB, the process of replacing loops by vectorized statements is known as _____	scalarization	vectorization	looping	branching	vectorization

UNIT-III

SYLLABUS

DISTANCE-BASED MODELS : Nearest neighbor models – K-means – clustering around medoids – silhouttes – hierarchical clustering – k- d trees – locality sensitive hashing – non - parametric regression – ensemble learning – bagging and random forests – boosting – meta learning

NEAREST NEIGHBOUR MODEL:

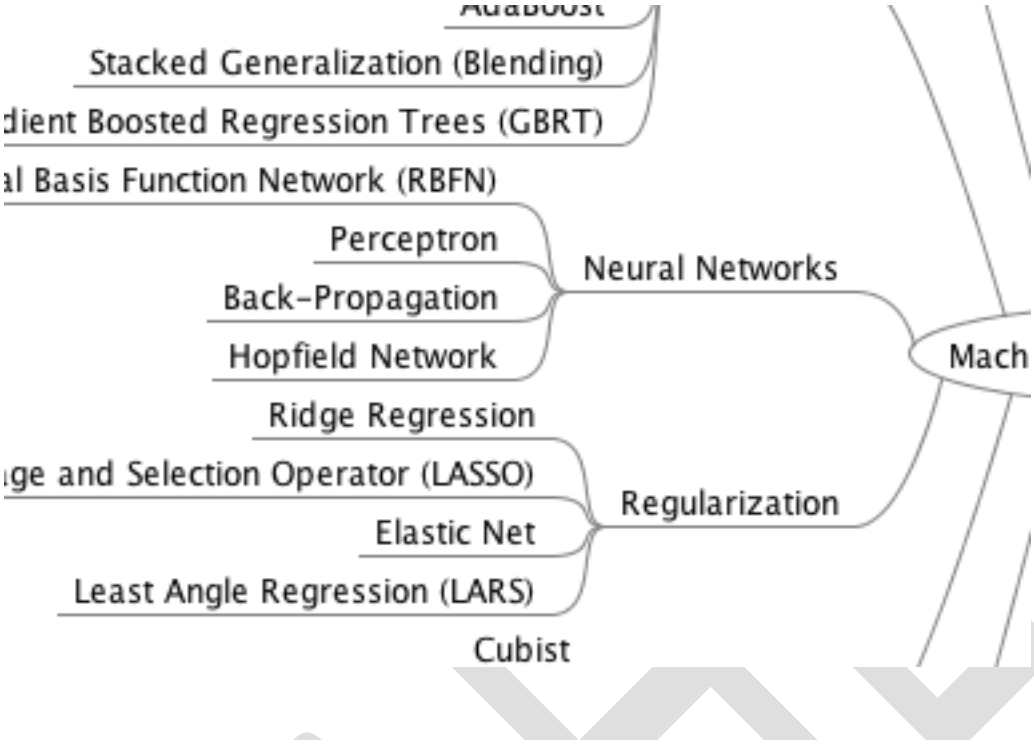
The model representation for KNN is the entire training dataset.

It is as simple as that.

KNN has no model other than storing the entire dataset, so there is no learning required.

Efficient implementations can store the data using complex data structures like k-d trees to make look-up and matching of new patterns during prediction efficient.
Because the entire training dataset is stored, you may want to think carefully about the consistency of your training data.

It might be a good idea to curate it, update it often as new data becomes available and remove erroneous and outlier data.



Making Predictions with KNN

KNN makes predictions using the training dataset directly.

Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification this might be the mode (or most common) class value.

To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.
Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j.

EuclideanDistance(x, xi) = sqrt(sum((xj – xij)^2))

Other popular distance measures include:

- **Hamming Distance**: Calculate the distance between binary vectors.
- **Manhattan Distance**: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance.
- **Minkowski Distance**: Generalization of Euclidean and Manhattan distance.

There are many other distance measures that can be used, such as Tanimoto, Jaccard,Mahalanobis and cosine distance. You can choose the best distance metric based on the properties of your data. If you are unsure, you can experiment with different distance metrics and different values of K together and see which mix results in the most accurate models.
Euclidean is a good distance measure to use if the input variables are similar in type (e.g. all measured widths and heights). Manhattan distance is a good measure to use if the input variables are not similar in type (such as age, gender, height, etc.).

The value for K can be found by algorithm tuning. It is a good idea to try many different values for K (e.g. values from 1 to 21) and see what works best for your problem.

The computational complexity of KNN increases with the size of the training dataset. For very large training sets, KNN can be made stochastic by taking a sample from the training dataset from which to calculate the K-most similar instances.

KNN has been around for a long time and has been very well studied. As such, different disciplines have different names for it, for example:

- **Instance-Based Learning**: The raw training instances are used to make predictions. As such KNN is often referred to as instance-based learning or a case-based learning (where each training instance is a case from the problem domain).
- **Lazy Learning**: No learning of the model is required and all of the work happens at the time a prediction is requested. As such, KNN is often referred to as a lazy learning algorithm.
- **Non-Parametric**: KNN makes no assumptions about the functional form of the problem being solved. As such KNN is referred to as a non-parametric machine learning algorithm.

KNN can be used for regression and classification problems.

KNN for Regression

When KNN is used for regression problems the prediction is based on the mean or the median of the K-most similar instances.

KNN for Classification

When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.

Class probabilities can be calculated as the normalized frequency of samples that belong to each class in the set of K most similar instances for a new data instance. For example, in a binary classification problem (class is 0 or 1):

p(class=0) = count(class=0) / (count(class=0)+count(class=1))

If you are using K and you have an even number of classes (e.g. 2) it is a good idea to choose a K value with an odd number to avoid a tie. And the inverse, use an even number for K when you have an odd number of classes.

Ties can be broken consistently by expanding K by 1 and looking at the class of the next most similar instance in the training dataset.

Curse of Dimensionality

KNN works well with a small number of input variables (p), but struggles when the number of inputs is very large.

Each input variable can be considered a dimension of a p-dimensional input space. For example, if you had two input variables x1 and x2, the input space would be 2-dimensional.

As the number of dimensions increases the volume of the input space increases at an exponential rate.

In high dimensions, points that may be similar may have very large distances. All points will be far away from each other and our intuition for distances in simple 2 and 3-dimensional spaces breaks down. This might feel unintuitive at first, but this general problem is called the "Curse of Dimensionality".

Best Prepare Data for KNN

- **Rescale Data**: KNN performs much better if all of the data has the same scale. Normalizing your data to the range [0, 1] is a good idea. It may also be a good idea to standardize your data if it has a Gaussian distribution.
- **Address Missing Data**: Missing data will mean that the distance between samples can not be calculated. These samples could be excluded or the missing values could be imputed.
- **Lower Dimensionality**: KNN is suited for lower dimensional data. You can try it on high dimensional data (hundreds or thousands of input variables) but be aware that it may not perform as well as other techniques. KNN can benefit from feature selection that reduces the dimensionality of the input feature space.

K-MEANS

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.

Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

AndreyBu, who has more than 5 years of machine learning experience and currently teaches people his skills, says that "the objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (*k*) of clusters in a dataset."

A cluster refers to a collection of data points aggregated together because of certain similarities.

You'll define a target number *k*, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.

Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.

How the K-means algorithm works

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids

It halts creating and optimizing clusters when either:

- The centroids have stabilized—there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

K-means algorithm example problem

Let's see the steps on how the K-means machine learning algorithm works using the Python programming language.

We'll use the Scikit-learn library and some random data to illustrate a K-means clustering simple explanation.

Step 1: Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
%matplotlib inline
```

As you can see from the above code, we'll import the following libraries in our project:

- Pandas for reading and writing spreadsheets
- Numpy for carrying out efficient computations
- Matplotlib for visualization of data

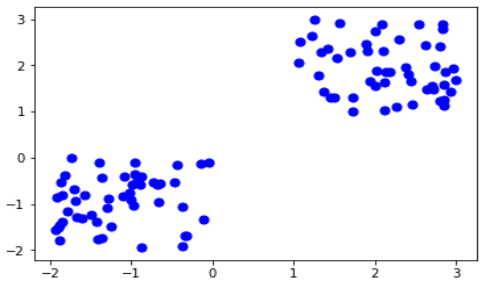
Step 2: Generate random data

Here is the code for generating some random data in a two-dimensional space:

```
X= -2 * np.random.rand(100,2)
X1 = 1 + 2 * np.random.rand(50,2)
X[50:100, :] = X1
plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = 'b')
plt.show()
```

A total of 100 data points has been generated and divided into two groups, of 50 points each.

Here is how the data is displayed on a two-dimensional space:



Step 3: Use Scikit-Learn

We'll use some of the available functions in the Scikit-learn library to process the randomly generated data.

Here is the code:

```
from sklearn.cluster import KMeans
Kmean = KMeans(n_clusters=2)
Kmean.fit(X)
```

Here is the output of the K-means parameters we get if we run the code:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300
 n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
 random_state=None, tol=0.0001, verbose=0)
```

Step 4: Finding the centroid

Here is the code for finding the center of the clusters:

```
Kmean.cluster_centers_
```

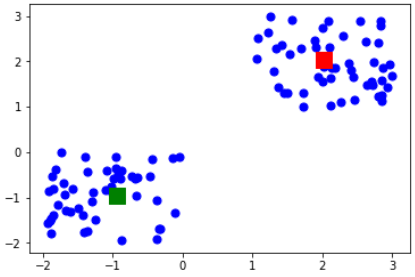
Here is the result of the value of the centroids:

```
array([[-0.94665068, -0.97138368],
 [ 2.01559419, 2.02597093]])
```

Let's display the cluster centroids (using green and red color).

```
plt.scatter(X[ : , 0], X[ : , 1], s =50, c='b')
plt.scatter(-0.94665068, -0.97138368, s=200, c='g', marker='s')
plt.scatter(2.01559419, 2.02597093, s=200, c='r', marker='s')
plt.show()
```

Here is the output:



Step 5: Testing the algorithm

Here is the code for getting the labels property of the K-means clustering example dataset; that is, how the data points are categorized into the two clusters.

```
Kmean.labels_
```

Here is the result of running the above K-means algorithm code:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

As you can see above, 50 data points belong to the **0** cluster while the rest belong to the **1** cluster.

For example, let's use the code below for predicting the cluster of a data point:

```
sample_test=np.array([-3.0,-3.0])
second_test=sample_test.reshape(1, -1)
Kmean.predict(second_test)
```

Here is the result:

```
array([0])
```

CLUSTERING AROUND MEDOIDS

The ***k*-medoids algorithm** is a clustering algorithm related to the *k*-means algorithm and the medoidshift algorithm. Both the *k*-means and *k*-medoids algorithms are partitional (breaking the dataset up into groups) and both attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the *k*-means algorithm, *k*-medoids chooses datapoints as centers (medoids or exemplars) and works with a generalization of the Manhattan Norm to define distance between datapoints instead of l2.

k-medoid is a classical partitioning technique of clustering, which clusters the data set of *n* objects into *k* clusters, with the number *k* of clusters assumed known *a priori*. If unknown, *k*can be determined with methods such as silhouette.

It is more robust to noise and outliers as compared to *k*-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

A medoid can be defined as the object of a cluster whose average dissimilarity to all the objects in the cluster is minimal. i.e. it is a most centrally located point in the cluster.

The most common realisation of *k*-medoid clustering is the **Partitioning Around Medoids (PAM)** algorithm.

PAM uses a greedy search which may not find the optimum solution, but it is faster than exhaustive search. It works as follows:

1. Initialize: select *k* of the *n* data points as the medoids
2. Associate each data point to the closest medoid.
3. While the cost of the configuration decreases:
    1. For each medoid *m*, for each non-medoid data point *o*:
        1. Swap *m* and *o*, associate each data point to the closest medoid, recompute the cost (sum of distances of points to their medoid)
        2. If the total cost of the configuration increased in the previous step, undo the swap

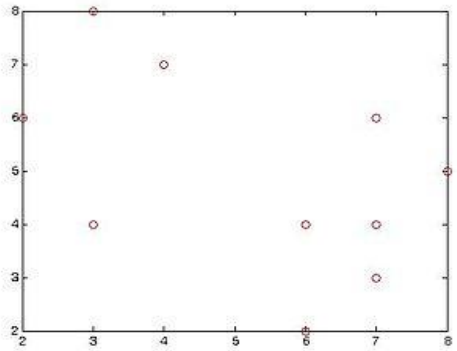
Algorithms other than PAM have also been suggested in the literature, including the following Voronoi iteration method:

1. Select initial medoids
2. Iterate while the cost decreases:
    1. In each cluster, make the point that minimizes the sum of distances within the cluster the medoid.
    2. Reassign each point to the cluster defined by the closest medoid determined in the previous step.

Demonstration of PAM

Cluster the following data set of ten objects into two clusters i.e. $k = 2$.

Consider a data set of ten objects as follows :



3.
4. Figure 1.1 – distribution of the data

X_1	2	6
X_2	3	4
X_3	3	8
X_4	4	7
X_5	6	2
X_6	6	4
X_7	7	3
X_8	7	4
X_9	8	5
X_{10}	7	6

SILHOUTTES

Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The technique provides a succinct graphical representation of how well each object lies within its cluster.

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched

to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

The silhouette can be calculated with any distance metric, such as the Euclidean distance or the Manhattan distance.

It is a way to measure how close each point in a cluster is to the points in its neighboring clusters. Its a neat way to find out the optimum value for k during k-means clustering. Silhouette values lies in the range of [-1, 1]. A value of +1 indicates that the sample is far away from its neighboring cluster and very close to the cluster its assigned. Similarly, value of -1 indicates that the point is close to its neighboring cluster than to the cluster its assigned. And, a value of 0 means its at the boundary of the distance between the two cluster. Value of +1 is idea and -1 is least preferred. Hence, higher the value better is the cluster configuration.

Mathematically: Lets define Silhouette for each of the sample in the data set.
For an example (i) in the data, lets define a(i) to be the mean distance of point (i) to all the other points in the cluster its assigned (A). We can interpret a(i) as how well the point is assigned to the cluster. Smaller the value better the assignment.

Similarly, lets define b(i) to be the mean distance of point(i) to other points to its closet neighboring cluster (B). The cluster (B) is the cluster to which point (i) is not assigned to but its distance is closest amongst all other cluster.

Thus, the silhouette s(i) can be calculated as

$$s(i) = \frac{b(i) - a(i)}{\max(b(i), a(i))}$$

We can easily say that s(i) lies in the range of [-1,1].

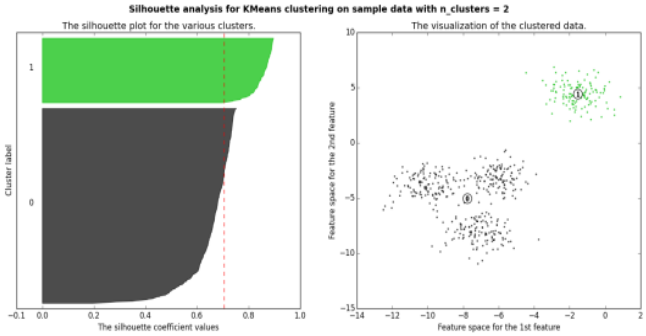
For s(i) to be close to 1, a(i) has be be very small as compared to b(i), i.e. a(i) <<< b(i). This happens when a(i) is very close to its assigned cluster. A large value of b(i) implies its extremely far from its next closest cluster. Hence, s(i) == 1 indicates that the data set (i) is well matched in the cluster assignment.

The above definition talks about Silhouette score for each data item. Although, its nice for visual analysis, it doesn't quickly tell the SA score for the entire cluster.

Mean Silhouette score: Mean score can be simply calculated by taking the mean of silhouette score of all the examples in the data set. This gives us one value representing the Silhouette score of the entire cluster.

Advantages of using S.A: The best advantage of using S.A. score for finding the best number of cluster is that you use it for un-labelled data set. This is usually the case when running k-means. Hence, I prefer this over other k-means scores like V-measure, Adjusted rank Index, V-score, Homogeneity etc

Example:

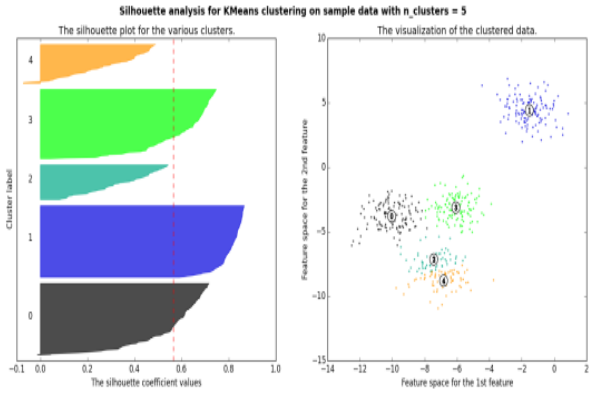


Left pic: depicts a sorted list of SA cluster of each point in a given cluster. The black region is the plot of S score for examples belonging to cluster 0, whereas green plot is the S score for examples belonging to cluster 1.
The red doted line is the mean S. score for the cluster in consideration. The value is roughly around 0.7

For this to be a good value for number of cluster, one should consider the following points

- Firstly, The mean value should be as close to 1 as possible
- Secondly, The plot of each cluster should be above the mean value as much as possible. Any plot region below the mean value is not desirable.
- Lastly, the width of the plot should be as uniform as possible.

Right pic is the visualization of the cluster assignment.
From the example above, the following SA score graph is not desirable as few of the cluster have samples less than the mean SS score. Also, they are not uniformly distributed in width.



That's all for the post. In my upcoming post I will go through a real life example of how to use Silhouette analysis for selecting the number of cluster for k-means clustering.

HIERARCHICAL CLUSTERING

Unlike K-mean clustering *Hierarchical* clustering starts by assigning all data points as their own cluster. As the name suggests it builds the hierarchy and in the next step, it combines the two nearest data point and merges it together to one cluster.

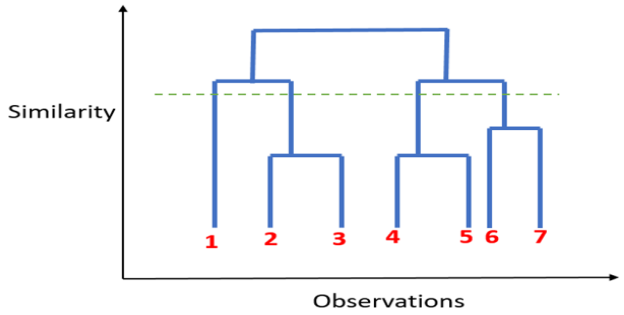
1. Assign each data point to its own cluster.

2. Find closest pair of cluster using euclidean distance and merge them in to single cluster.

3. Calculate distance between two nearest clusters and combine until all items are clustered in to a single cluster.

In this technique, you can decide the optimal number of clusters by noticing which vertical lines can be cut by horizontal line without intersecting a cluster and covers the maximum distance.

- Standardizing variables so that all are on the same scale. It is important when calculating distances.
  • Treat data for outliers before forming clusters as it can influence the distance between the data points.



Two important things that you should know about hierarchical clustering are:

- This algorithm has been implemented above using bottom up approach. It is also possible to follow top-down approach starting with all data points assigned in the same cluster and recursively performing splits till each data point is assigned a separate cluster.
- The decision of merging two clusters is taken on the basis of closeness of these clusters. There are multiple metrics for deciding the closeness of two clusters :
  - Euclidean distance: $||a-b||_2 = \sqrt{(\Sigma(a_i-b_i))}$
  - Squared Euclidean distance: $||a-b||_2^2 = \Sigma((a_i-b_i)^2)$
  - Manhattan distance: $||a-b||_1 = \Sigma|a_i-b_i|$
  - Maximum distance:$||a-b||_{INFINITY} = max_i|a_i-b_i|$
  - Mahalanobis distance: $\sqrt{((a-b)^T S^{-1} (-b))}$ {where, s : covariance matrix}

Difference between K Means and Hierarchical clustering

- Hierarchical clustering can't handle big data well but K Means clustering can. This is because the time complexity of K Means is linear i.e. O(n) while that of hierarchical clustering is quadratic i.e. $O(n^2)$.
- In K Means clustering, since we start with random choice of clusters, the results produced by running the algorithm multiple times might differ. While results are reproducible in Hierarchical clustering.
- K Means is found to work well when the shape of the clusters is hyper spherical (like circle in 2D, sphere in 3D).
- K Means clustering requires prior knowledge of K i.e. no. of clusters you want to divide your data into. But, you can stop at whatever number of clusters you find appropriate in hierarchical clustering by interpreting the dendrogram

K-D TREES

A K-D Tree(also called as K-Dimensional Tree) is a binary search tree where data in each node is a K-Dimensional point in space. In short, it is a space partitioning(details below) data structure for organizing points in a K-Dimensional space.

A non-leaf node in K-D tree divides the space into two parts, called as half-spaces.

Points to the left of this space are represented by the left subtree of that node and points to the right of the space are represented by the right subtree. We will soon be explaining the concept on how the space is divided and tree is formed.

For the sake of simplicity, let us understand a 2-D Tree with an example.

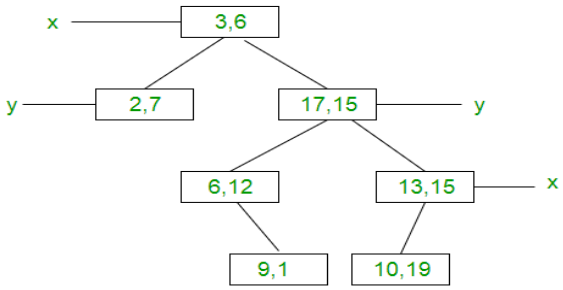
The root would have an x-aligned plane, the root's children would both have y-aligned planes, the root's grandchildren would all have x-aligned planes, and the root's great-grandchildren would all have y-aligned planes and so on.

Generalization:
Let us number the planes as 0, 1, 2, …(K – 1). From the above example, it is quite clear that a point (node) at depth D will have A aligned plane where A is calculated as:
A = D mod K

How to determine if a point will lie in the left subtree or in right subtree?
If the root node is aligned in planeA, then the left subtree will contain all points whose coordinates in that plane are smaller than that of root node. Similarly, the right subtree will contain all points whose coordinates in that plane are greater-equal to that of root node.

Creation of a 2-D Tree:
Consider following points in a 2-D plane:
(3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19)

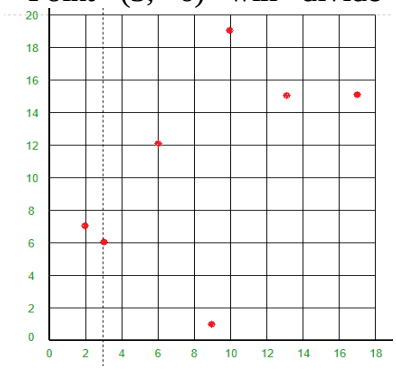
1. Insert (3, 6): Since tree is empty, make it the root node.
2. Insert (17, 15): Compare it with root node point. Since root node is X-aligned, the X-coordinate value will be compared to determine if it lies in the rightsubtree or in the right subtree. This point will be Y-aligned.
3. Insert (13, 15): X-value of this point is greater than X-value of point in root node. So, this will lie in the right subtree of (3, 6). Again Compare Y-value of this point with the Y-value of point (17, 15) (Why?). Since, they are equal, this point will lie in the right subtree of (17, 15). This point will be X-aligned.
4. Insert (6, 12): X-value of this point is greater than X-value of point in root node. So, this will lie in the right subtree of (3, 6). Again Compare Y-value of this point with the Y-value of point (17, 15) (Why?). Since, 12 < 15, this point will lie in the left subtree of (17, 15). This point will be X-aligned.
5. Insert (9, 1):Similarly, this point will lie in the right of (6, 12).
6. Insert (2, 7):Similarly, this point will lie in the left of (3, 6).
7. Insert (10, 19): Similarly, this point will lie in the left of (13, 15).



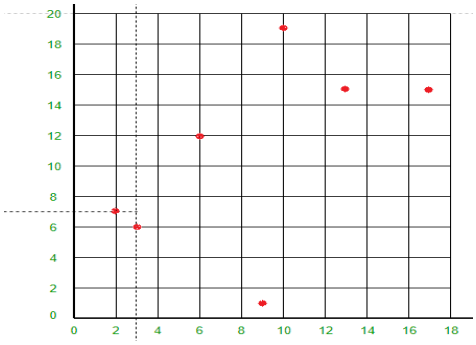
How is space partitioned?

All 7 points will be plotted in the X-Y plane as follows:

1. Point (3, 6) will divide the space into two parts: Draw line X = 3.

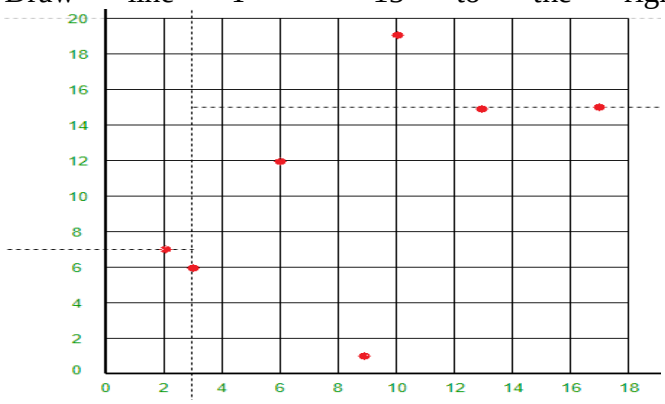


2. Point (2, 7) will divide the space to the left of line X = 3 into two parts horizontally.
   Draw line Y = 7 to the left of line X = 3.

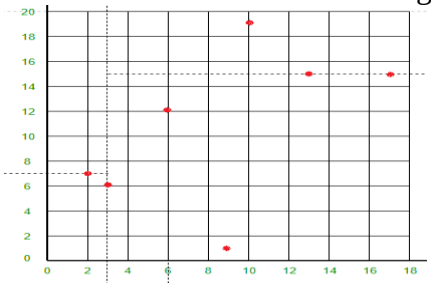


3. Point (17, 15) will divide the space to the right of line X = 3 into two parts horizontally.

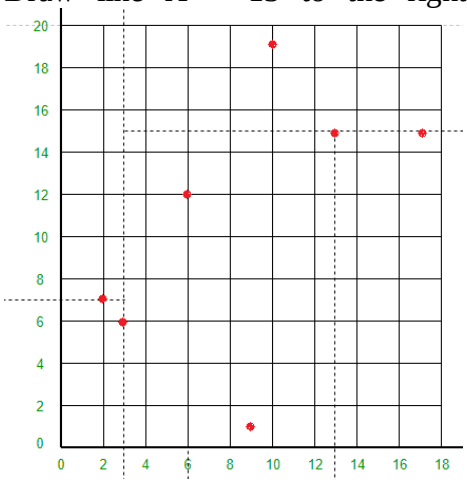
Draw line Y = 15 to the right of line X = 3.



- Point (6, 12) will divide the space below line Y = 15 and to the right of line X = 3 into two parts.
Draw line X = 6 to the right of line X = 3 and below line Y = 15.

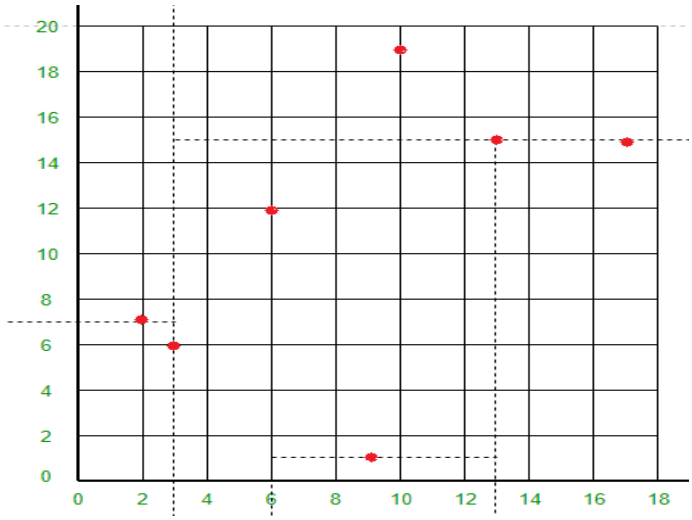


- Point (13, 15) will divide the space below line Y = 15 and to the right of line X = 6 into two parts.
Draw line X = 13 to the right of line X = 6 and below line Y = 15.



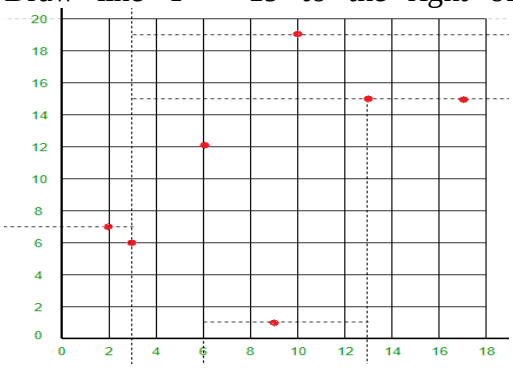
- Point (9, 1) will divide the space between lines X = 3, X = 6 and Y = 15 into two parts.

Draw line Y = 1 between lines X = 3 and X = 6.



- Point (10, 19) will divide the space to the right of line X = 3 and above line Y = 15 into two parts.

Draw line Y = 19 to the right of line X = 3 and above line Y = 15.



Following is C++ implementation of KD Tree basic operations like search, insert and delete.

filter_none
edit
play_arrow
brightness_4

```
// A C++ program to demonstrate operations of KD tree
#include<bits/stdc++.h>
using namespace std;

const int k = 2;

// A structure to represent node of kd tree
struct Node
{
```

```
    int point[k]; // To store k dimensional point
    Node *left, *right;
};

// A method to create a node of K D tree
struct Node* newNode(int arr[])
{
    struct Node* temp = new Node;

    for (int i=0; i<k; i++)
       temp->point[i] = arr[i];

    temp->left = temp->right = NULL;
    return temp;
}

// Inserts a new node and returns root of modified tree
// The parameter depth is used to decide axis of comparison
Node *insertRec(Node *root, int point[], unsigned depth)
{
    // Tree is empty?
    if (root == NULL)
       return newNode(point);

    // Calculate current dimension (cd) of comparison
    unsigned cd = depth % k;

    // Compare the new point with root on current dimension 'cd'
    // and decide the left or right subtree
    if (point[cd] < (root->point[cd]))
        root->left  = insertRec(root->left, point, depth + 1);
    else
        root->right = insertRec(root->right, point, depth + 1);

    return root;
}

// Function to insert a new point with given point in
// KD Tree and return new root. It mainly uses above recursive
// function "insertRec()"
Node* insert(Node *root, int point[])
{
    return insertRec(root, point, 0);
}
```

```
// A utility method to determine if two Points are same
// in K Dimensional space
bool arePointsSame(int point1[], int point2[])
{
    // Compare individual pointinate values
    for (int i = 0; i < k; ++i)
        if (point1[i] != point2[i])
            return false;

    return true;
}

// Searches a Point represented by "point[]" in the K D tree.
// The parameter depth is used to determine current axis.
bool searchRec(Node* root, int point[], unsigned depth)
{
    // Base cases
    if (root == NULL)
        return false;
    if (arePointsSame(root->point, point))
        return true;

    // Current dimension is computed using current depth and total
    // dimensions (k)
    unsigned cd = depth % k;

    // Compare point with root with respect to cd (Current dimension)
    if (point[cd] < root->point[cd])
        return searchRec(root->left, point, depth + 1);

    return searchRec(root->right, point, depth + 1);
}

// Searches a Point in the K D tree. It mainly uses
// searchRec()
bool search(Node* root, int point[])
{
    // Pass current depth as 0
    return searchRec(root, point, 0);
}

// Driver program to test above functions
int main()
```

```
{
   struct Node *root = NULL;
   int points[][k] = {{3, 6}, {17, 15}, {13, 15}, {6, 12},
                      {9, 1}, {2, 7}, {10, 19}};

int n = sizeof(points)/sizeof(points[0]);

   for (int i=0; i<n; i++)
     root = insert(root, points[i]);

   int point1[] = {10, 19};
   (search(root, point1))? cout << "Found\n": cout << "Not Found\n";

   int point2[] = {12, 19};
   (search(root, point2))? cout << "Found\n": cout << "Not Found\n";

   return 0;
}
```

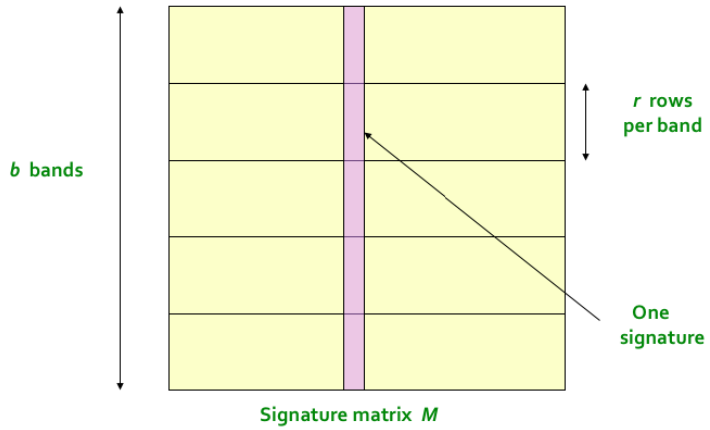
LOCALITY SENSITIVE HASHING

The general idea of LSH is to find a algorithm such that if we input signatures of 2 documents, it tells us that those 2 documents form a candidate pair or not i.e. their similarity is greater than a threshold **t**. Remember that we are taking similarity of signatures as a proxy for Jaccard similarity between the original documents.

Specifically for min-hash signature matrix:

- Hash columns of signature matrix **M** using several hash functions
- If 2 documents hash into same bucket for **at least one** of the hash function we can take the 2 documents as a candidate pair
- Now the question is how to create different hash functions. For this we do band partition.

- **Band partition**



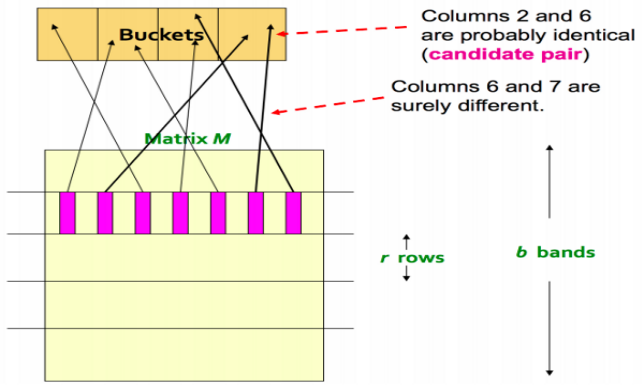
Here is the algorithm:

- Divide the signature matrix into *b bands*, each band having *r rows*
- For each band, hash its portion of each column to a hash table with *k buckets*
- Candidate column pairs are those that hash to the same bucket for *at least 1 band*
- Tune b and r to catch most similar pairs but few non similar pairs

There are few considerations here. Ideally for each band we want to take k to be equal to all possible combinations of values that a column can take within a band. This will be equivalent to identity matching. But in this way k will be a huge number which is computationally infeasible. For ex: If for a band we have 5 rows in it. Now if the elements in the signature are 32 bit integers then k in this case will be $(2^{32})^5 \sim 1.4615016e+48$. You can see what's the problem here. Normally k is taken around 1 million.

The idea is that if 2 documents are similar then they will will appear as candidate pair in at least one of the bands.

Choice of b & r



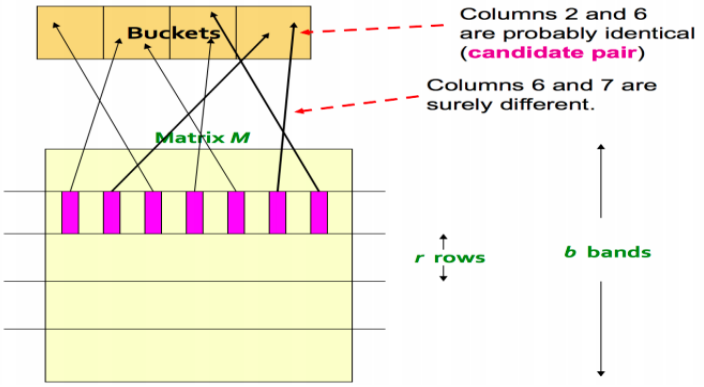
If we take b large i.e more number of hash functions, then we reduce r as b*r is a constant (number of rows in signature matrix). Intuitively it means that we're *increasing the probability of finding a candidate pair*. This case is equivalent to taking a small **t (similarity threshold)**

Let's say your signature matrix has 100 rows. Consider 2 cases:

b1 = 10 → r = 10

b2 = 20 → r = 5

In 2nd case, there is higher chance for 2 documents to appear in same bucket at least once as they have *more opportunities (20 vs 10)* and *fewer elements of the signature are getting compared (5 vs 10).*



NON PARAMETRIC ALGORITHM

Algorithms that do not make strong assumptions about the form of the mapping function are called nonparametric machine learning algorithms. By not making assumptions, they are free to learn any functional form from the training data.

Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don't want to worry too much about choosing just the right features.

Nonparametric methods seek to best fit the training data in constructing the mapping function, whilst maintaining some ability to generalize to unseen data. As such, they are able to fit a large number of functional forms.

An easy to understand nonparametric model is the k-nearest neighbors algorithm that makes predictions based on the k most similar training patterns for a new data instance. The method does not assume anything about the form of the mapping function other than patterns that are close are likely have a similar output variable.

Some more examples of popular nonparametric machine learning algorithms are:

- k-Nearest Neighbors
- Decision Trees like CART and C4.5
- Support Vector Machines

Benefits of Nonparametric Machine Learning Algorithms:

- **Flexibility**: Capable of fitting a large number of functional forms.
- **Power**: No assumptions (or weak assumptions) about the underlying function.
- **Performance**: Can result in higher performance models for prediction.

Limitations of Nonparametric Machine Learning Algorithms:

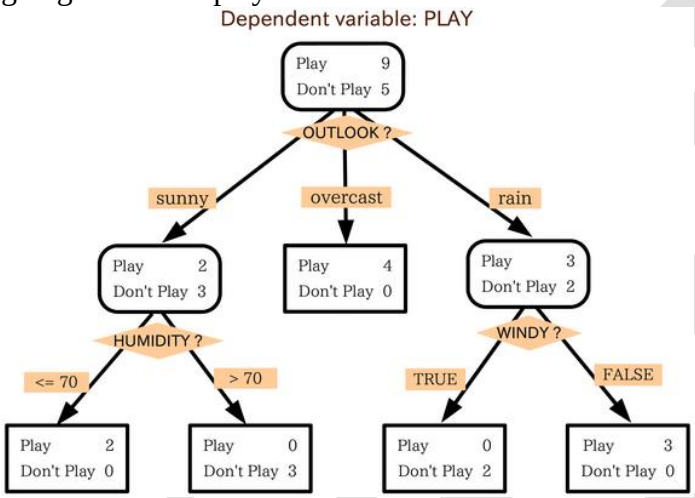
- **More data**: Require a lot more training data to estimate the mapping function.
- **Slower**: A lot slower to train as they often have far more parameters to train.
- **Overfitting**: More of a risk to overfit the training data and it is harder to explain why specific predictions are made.

ENSEMBLE LEARNING:

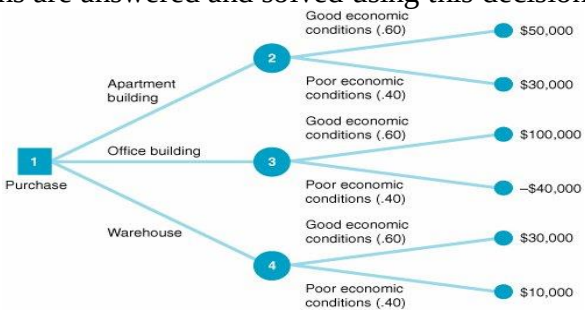
Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. To better understand this definition lets take a step back into ultimate goal of machine learning and model building. This is going to make more sense as I dive into specific examples and why Ensemble methods are used.

I will largely utilize Decision Trees to outline the definition and practicality of Ensemble Methods (however it is important to note that Ensemble Methods do not only pertain to Decision Trees).

A Decision Tree determines the predictive value based on series of questions and conditions. For instance, this simple Decision Tree determining on whether an individual should play outside or not. The tree takes several weather factors into account, and given each factor either makes a decision or asks another question. In this example, every time it is overcast, we will play outside. However, if it is raining, we must ask if it is windy or not? If windy, we will not play. But given no wind, tie those shoelaces tight because were going outside to play.



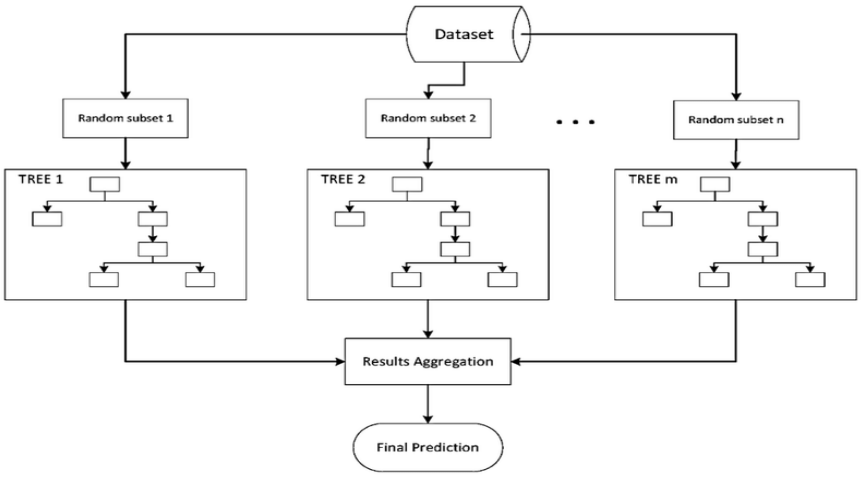
Decision Trees can also solve quantitative problems as well with the same format. In the Tree to the left, we want to know wether or not to invest in a commercial real estate property. Is it an office building? A Warehouse? An Apartment building? Good economic conditions? Poor Economic Conditions? How much will an investment return? These questions are answered and solved using this decision tree.



When making Decision Trees, there are several factors we must take into consideration: On what features do we make our decisions on? What is the threshold for classifying each question into a yes or no answer? In the first Decision Tree, what if we wanted to ask ourselves if we had friends to play with or not. If we have friends, we will play every time. If not, we might continue to ask ourselves questions about the weather. By adding an additional question, we hope to greater define the Yes and No classes.

Types of Ensemble Methods

1. ***BAGG***ing, or ***B***ootstrap ***AGG***regating. **BAGG**ing gets its name because it combines ***B***ootstrapping and ***Agg***regation to form one ensemble model. Given a sample of data, multiple bootstrapped subsamples are pulled. A Decision Tree is formed on each of the bootstrapped subsamples. After each subsample Decision Tree has been formed, an algorithm is used to aggregate over the Decision Trees to form the most efficient predictor. The image below will help explain:



Random Forest Models. Random Forest Models can be thought of as **BAGG**ing, with a slight tweak. When deciding where to split and how to make decisions, BAGGed Decision Trees have the full disposal of features to choose from. Therefore, although the bootstrapped samples may be slightly different, the data is largely going to break off at the same features throughout each model. In contrary, Random Forest models decide where to split based on a random selection of features. Rather than splitting at similar features at each node throughout, Random Forest models implement a level of differentiation because each tree will split based on different features. This level of differentiation provides a greater ensemble to aggregate over, ergo producing a more accurate predictor.

BAGGING AND RANDOM FORESTS

Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method.

An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.

Bootstrap Aggregation is a general procedure that can be used to reduce the variance for those algorithm that have high variance. An algorithm that has high variance are decision trees, like classification and regression trees (CART).

Decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g. a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different.

Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

Let's assume we have a sample dataset of 1000 instances (x) and we are using the CART algorithm. Bagging of the CART algorithm would work as follows.

1. Create many (e.g. 100) random sub-samples of our dataset with replacement.
2. Train a CART model on each sample.
3. Given a new dataset, calculate the average prediction from each model.

For example, if we had 5 bagged decision trees that made the following class predictions for a in input sample: blue, blue, red, blue and red, we would take the most frequent class and predict blue.

When bagging with decision trees, we are less concerned about individual trees overfitting the training data. For this reason and for efficiency, the individual decision trees are grown deep (e.g. few training samples at each leaf-node of the tree) and the trees are not pruned. These trees will have both high variance and low bias. These are important characterize of sub-models when combining predictions using bagging.

The only parameters when bagging decision trees is the number of samples and hence the number of trees to include. This can be chosen by increasing the number of trees on run after run until the accuracy begins to stop showing improvement (e.g. on a cross validation test harness). Very large numbers of models may take a long time to prepare, but will not overfit the training data.

Just like the decision trees themselves, Bagging can be used for classification and regression problems.

Random Forest

Random Forests are an improvement over bagged decision trees.

A problem with decision trees like CART is that they are greedy. They choose which variable to split on using a greedy algorithm that minimizes error. As such, even with Bagging, the decision trees can have a lot of structural similarities and in turn have high correlation in their predictions.

Combining predictions from multiple models in ensembles works better if the predictions from the sub-models are uncorrelated or at best weakly correlated.

Random forest changes the algorithm for the way that the sub-trees are learned so that the resulting predictions from all of the subtrees have less correlation.

It is a simple tweak. In CART, when selecting a split point, the learning algorithm is allowed to look through all variables and all variable values in order to select the most optimal split-point. The random forest algorithm changes this procedure so that the learning algorithm is limited to a random sample of features of which to search.

The number of features that can be searched at each split point (m) must be specified as a parameter to the algorithm. You can try different values and tune it using cross validation.

- For classification a good default is: m = sqrt(p)
- For regression a good default is: m = p/3

Where m is the number of randomly selected features that can be searched at a split point and p is the number of input variables. For example, if a dataset had 25 input variables for a classification problem, then:

- m = sqrt(25)
- m = 5

Estimated Performance

For each bootstrap sample taken from the training data, there will be samples left behind that were not included. These samples are called Out-Of-Bag samples or OOB.

The performance of each model on its left out samples when averaged can provide an estimated accuracy of the bagged models. This estimated performance is often called the OOB estimate of performance.

These performance measures are reliable test error estimate and correlate well with cross validation estimates.

Variable Importance

As the Bagged decision trees are constructed, we can calculate how much the error function drops for a variable at each split point.

In regression problems this may be the drop in sum squared error and in classification this might be the Gini score.

These drops in error can be averaged across all decision trees and output to provide an estimate of the importance of each input variable. The greater the drop when the variable was chosen, the greater the importance.

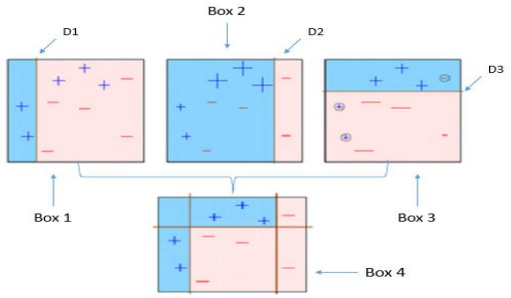
These outputs can help identify subsets of input variables that may be most or least relevant to the problem and suggest at possible feature selection experiments you could perform where some features are removed from the dataset.

BOOSTING:

Boosting is an ensemble strategy thats consecutively builds on weak learners in order to generate one final strong learner. A weak learner is a model that may not be very accurate or may not take many predictors into account. By building a weak model, making conclusions about the various feature importances and parameters, and then using those conclusions to build a new, stronger model, Boosting can effectively convert weak learners into a strong learner. Boosting can be used for *BOTH* classification and regression problems.

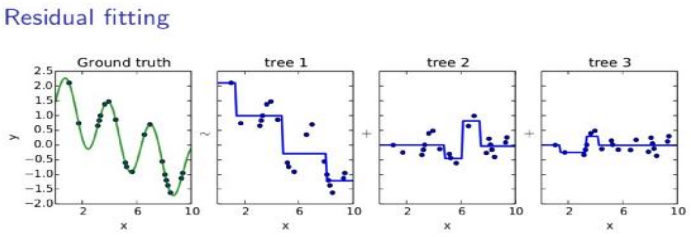
Types of Boosting

1. **AdaBoost** (**Ada**ptive **Boost**ing): AdaBoost uses decision stumps as weak learners. A Decision Stump is a Decision Tree model that only splits off at one level, ergo the final prediction is based off of only one feature. When AdaBoost makes its first Decision Stump, all observations are weighted evenly. In an attempt to correct previous error, when moving to the second Decision Stump, the observations that were classified incorrectly now carry more weight than the observations that were correctly classified. AdaBoost continues this strategy until the best classification model is built. Take the diagram below for example.



After model D1 is made, the observations are split at one divider, separating the blue(+) from the red(—). There are three misclassified (+) that are not in the 'blue' classification. These misclassified (+) now carry more weight than any other observation when making D2. Consequently, D2 adjusts its 'blue' classification to incorporate all (+) . The consecutive models continue to adjust for the error faced with the previous model, until the most accurate predictor is built.

2. **Gradient Boosting:** Similar to all boosting methods, Gradient Boosting looks to consecutively reduce error with each consecutive model, until one final model is produced. Given a set of data observations, Gradient Boosting fits a simple, weak learner to predict outcomes. Then, from this weak model, a loss function is plotted. There are various loss functions we can use in machine learning, but the ultimate goal of each loss function is to reduce error. Next, the two plots—the original data plot and the loss function—are combined to make a stronger predictor. The sum of our predictors gets stronger and stronger after each step. This process is repeated until a final predictor is built. Take the diagram below. The "Ground Truth" plots a set of data, with a line running through each of the points.



"Tree 1" is a best-fit-line of the data. "Tree 2" is a curve that plots the errors from the graph in "Tree 1". These errors are based on how "Tree 1" misrepresented the original plot ("Ground truth" in this case). Finally, "Tree 3" is a combination of 'Tree 1' and 'Tree 2'. This is the cycle of one weak learner in **Gradient Boosting.** By combining weak learner

after weak learner, our final model is able to account for a lot of the error from the original model and reduces this error over time.

Gradient Boosting gets its name from **Gradient** Descent. Given the predetermined loss function, Gradient Descent is utilized to find the parameters which minimize this loss function. Initially, gradient descent uses some parameters to looks at each point along the loss function, and find the negative derivative of that point. As gradient descent continues along the loss function, it continuously tunes the parameters until the minimum point is found. The goal is to find the optimal parameters which have the biggest decrease on the loss function. This is how Gradient Boosting attempts to minimize error. By sequentially minimizing our loss function (meaning we are sequentially minimizing the amount of error with each weak learner), our model gets stronger and stronger until a final predictor is found.

META LEARNING:

A simpler definition can be found in the works of John Biggs (1985) in which he defined meta-learning as "being aware of and taking control of one's own learning". Those definitions are accurate from the cognitive science standpoint but they seemed a bit hard to adapt to the work of artificial intelligence(AI).

As humans, we are able to acquire multiple tasks simultaneously with minimum information. We can recognize a new type of object by seeing a single picture of it or we can learn complex, multi-task activities such as driving or piloting an airplane at once. While AI agents can master really complex tasks, they require massive amounts of training on any atomic subtasks and they remained incredibly bad at multi-tasking. So the path to knowledge versatility requires AI agents to "learn how to learn" or, to used a more obnoxious term, to meta-learn J.

Types of Meta-Learning Models

Humans learn following different methodologies tailored to specific circumstances. In the same way, not all meta-learning models follow the same techniques. Some meta-learning models are focused on optimizing neural network structures while others (like Reptile) focused more on finding the right datasets to train specific models. A recent research paper from UC Berkeley AI Lab does a comprehensive job enumerating the different types of meta-learning. Here are some of my favorites:

Few Shots Meta-Learning

The idea of few shots meta-learning is to create deep neural networks that can learn from minimalistic datasets mimicking, for instance, how babies can learn to identify objects by seeing only a picture or two. The ideas of few shots meta-learning have inspired the

creation of techniques such as memory augmented neural networks or one-shot generative models.

Optimizer Meta-Learning

Optimizer meta-learning models are focused on learning how to optimize a neural network to better accomplish a task. Those models typically include a neural networks that applies different optimizations to the hyperparameters of another neural network in order to improve a target task. A great example of optimizer meta-learning are models that focused on improving gradient descent techniques like the one published in this research.

Metric Meta-Learning

The objectives of metric meta-learning is to determine a metric space in which learning is particularly efficient. This approach can be seen as a subset of few shots meta-learning in which we used a learned metric space to evaluate the quality of learning with a few examples. This research paper shows how to apply metric meta-learning to classification problems.

Recurrent Model Meta-Learning

This type of meta-learning model is tailored to recurrent neural networks(RNNs) such as Long-Short-Term-Memory(LSTM). In this architecture, the meta-learner algorithm will train a RNN model will process a dataset sequentially and then process new inputs from the task.

POSSIBLE QUESTIONS

PART-B

1. Write a note on (i) silhoutes (ii) hierarchical clustering.
2. Describe about nearest neighbor model with example.
3. Write a note on (i) k-d trees (ii) boosting.

PART-C

1. Write in detail about bagging and random forests with example.
2. Explain about K-means model in detail.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of CS,CA & IT

III B.Sc(CS) (BATCH 2016-2019)

Machine Learning

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

SN(	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	ANSWER
1	What's the point of the added bias term in models that use linear combinations of inputs?	Due to the bias-variance tradeoff, it helps reduce overfitting	There is no point, it's just convention at this point.	In logistic/linear regression, it helps shape the curvature of the model, and centers it at the	Aids in a model's ability to fit the data (e.g affine transformations)	Aids in a model's ability to fit the data (e.g affine transformations)
2	Which of the following is false regarding linear regression?	Scaling the features (not bias) by a constant has NO effect on an unregularized	Mean centering the features (not bias) has NO effect on an unregularized model's prediction	Standardization on the data has an effect on the unregularized model's prediction	Scaling the features (not bias) by a constant has NO effect on a regularized model's prediction	Scaling the features (not bias) by a constant has NO effect on a regularized model's prediction
3	Which is true regarding fitting a linear regression model when we're dealing with millions of samples?	We can utilize the normal equations since matrix inversions are computationally	Gradient descent is a more viable solution as opposed to solving the normal equations	Numerical computer scientist implement standard matrix inversion to	We can utilize the normal equations since vector inversions are computationally easy	Gradient descent is a more viable solution as opposed to solving the normal equations

4	How are logistic regression and linear regression related?	The logistic regression model utilizes a polynomial/higher-order model. This gives it the	Logistic regression inputs the linear regression model through a logarithmic	Since the logit function ranges from 0 to 1, just like the linear regression model, the	The logit function ranges from negative infinity to positive infinity. Equating it to the linear regression model and inverting it	**The logit function ranges from negative infinity to positive infinity. Equating it to the linear regression model and**
5	Logistic regression is often used when:	the dependent variable is categorical.	the underlying distribution for p(y|x) is Gaussian.	the dependent variable is continuous.	we need a closed form analytical solution.	**the dependent variable is categorical.**
6	Is the logistic regression model a Generalized Linear Model (GLM)? What is a possible reason?	No, it is not possible since the sigmoid shape of the function is highly non-	Yes, it's a GLM but only by convention. It's properties don't match with the current definition	Yes, it's a GLM since it's parameterized by a linear combination of its features, and	No, it's not a GLM since it belongs to the 'exponential' family of distributions and not the 'linear' family of distributions.	**Yes, it's a GLM since it's parameterized by a linear combination of its features, and shares a linear relationship with**
7	Which of the following is false regarding logistic regression?	We can solve the maximum log-likelihood problem via an analytical closed form solution,	We can solve the maximum log-likelihood problem via gradient descent.	We can solve the maximum log-likelihood problem via the Raphson-Newton Method	The conditional probability is modeled as a Bernoulli distribution.	**We can solve the maximum log-likelihood problem via an analytical closed form solution, though it is a bit slow.**
8	A regression model in which more than one independent variable is used to predict the dependent variable is called	a simple linear regression model	a multiple regression models	an independent model	none of the above	an independent model

9	A term used to describe the case when the independent variables in a multiple regression model are correlated is	regression	correlation	multicollinearity	none of the above	multicollinearity
10	A multiple regression model has	only one independent variable	more than one dependent variable	more than one independent variable	none of the above	more than one dependent variable
11	_____ is a systematic method for adding and removing terms from a multilinear model based on their statistical significance in a regression.	more than one dependent variable	more than one dependent variable	Stepwise regression	none of the above	Stepwise regression
12	_____ is used when _____	The entropy function	The squared error	Linear regression	Multinomial logistic regression	Multinomial logistic regression
13	most commonly used algorithm for solving all classification problems. It is also one of the first methods people get their hands dirty on.	Linear regression	Logistic Regression	Multinomial logistic regression	Polynomial Regression	Logistic Regression

14	_____ is a is a *universal* approximator so it can implement linear regression algorithm.	Neural network	Linear regression	Logistic Regression	Multinomial logistic regression	Neural network
15	Which of the following methods do we use to best fit the data in Logistic Regression?	Least Square Error	Maximum Likelihood	Jaccard distance	Both A and B	Maximum Likelihood
16	Which of the following evaluation metrics can not be applied in case of logistic regression output to compare with target?	AUC-ROC	Accuracy	Logloss	Mean-Squared-Error	Mean-Squared-Error
17	to analyze the performance of Logistic Regression is AIC, which is similar to R-Squared in Linear Regression. Which of the following is true about AIC?	We prefer a model with minimum AIC value	We prefer a model with maximum AIC value	Both but depend on the situation	None of these	We prefer a model with minimum AIC value
18	Which of the following algorithms do we use for Variable Selection?	LASSO	Ridge	Both	None of these	LASSO

19	Which of the following option is true?	Regression errors values has to be normally distributed but in case of Logistic Regression it is	Regression errors values has to be normally distributed but in case of Linear Regression it is	Regression and Logistic Regression error values have to be normally distributed	Both Linear Regression and Logistic Regression error values have not to be normally distributed	Linear Regression errors values has to be normally distributed but in case of Logistic Regression it is not the case
20	_____ is used to estimate / predict the discrete valued output such as success or failure, 0 or 1 etc.	The entropy function	Linear regression	Logistic regression	Polynomial Regression	Logistic regression
21	How many types in logistic regression	4	3	2	4	3
22	_____ is used to estimate the likelihood of outcome dependent variable instead of actual value as like linear regression model	The entropy function	Linear regression	Logistic regression	Polynomial Regression	Logistic regression
23	Logistic regression is used to predict _____ valued output?	Continuous	Categorical	Discrete	Ordinal	**Discrete**

24	How much marks a strudent can get in a competitive exam based on hours of study can be solved using _____ regression model	Multi-linear	Linear	Logistic	Ploynomial	
25	Logistic regression is _____ when the observed outcome of dependent variable can have only two values such as 0 and 1 or success and failure.	Binomial	Multinomial	Ordinal	Discreste	**Binomial**
26	Whether a strudent will pass or fail in the competitive exam based on hours of study can be solved using _____ regression model.	Multi-linear	Logistic	Linear	Polynomial	**-Logistic**
27	_____ regression can be termed as a special case of _____ regression when the outcome variable is categorical.	Logistic,Linear	Linear	Linear	Logistic	**Logistic,Linear**
28	In logistic regression	the goal is to predict _____	Actual value of outcome dependent variable	Odds of outcome dependent variable	agd	**Odds of outcome dependent variable**

29	Which of the following can be used to evaluate the performance of logistic regression model?	Adjusted R-Squared	AIC	Entropy	ROC	**AIC**
30	Which of the following is link function in logistic regression	Identity	Logit	Error		**Logit**
31	Logistic regression is _____ when the observed outcome of dependent variable can have multiple possible types	Binomial	Multinomial	Ordinal	Cardinal	**Multinomial**
32	In logistic regression	following technique is used to measure the goodness of the fit.	Sum of squares calculations	Deviance calculations	The Entropy function	**Deviance calculations**
33	Which of the following can be used to evaluate the performance of logistic regression model?.AIC	Null and Residual Deviance	Both of the above	Null Deviance only	Null Deviance Only	**Both of the above**

34	Given two model with different AIC value	which one would be preferred model?	One with higher AIC value	One with lower AIC value	Dependent variable equalling a given case	**One with lower AIC value**
35	Deviance is a measure of difference between a _____ model and the _____ model.	saturated,fitted	fitted	Fitted,saturated	saturated	**Fitted,saturated**
36	Logistic regression is _____ when the observed outcome of dependent variable are ordered.	Binomial	Multinomial	Ordinal	sum of squares calculations	**Ordinal**
37	Logit transformation is log of _____.	Odds of the event happening for different levels of each independent variable	Ratio of odds of the event happening for different levels of each independent variable	Deviance,sum of squares calculations	Dependent variable equalling a given case	**Ratio of odds of the event happening for different levels of each independent variable**
38	Logistic function is _____.	Dependent variable equalling a given case	Probability that dependent variable equals a case	Deviance,sum of squares calculations	Exponential function of the linear regression function	**Probability that dependent variable equals a case**

39	Deviance is is a function of _____.	Exponential function of likelihood ratio	Logrithmic function of likelihood ratio	Exponential function of the linear regression function	Maximum likelihood estimation method	**Logrithmic function of likelihood ratio**
40	The odds of the dependent variable equaling a case (given some linear combination x of the predictors) is equivalent to _____	Log function of the linear regression expression	Exponential function of the linear regression function	sum of squares calculations	Maximum likelihood estimation method	**Exponential function of the linear regression function**
41	Regression coefficients in logistic regression are estimated using _____.	Ordinary least squares method	Maximum likelihood estimation method	Dependent variable equalling a given case	sum of squares calculations	**Maximum likelihood estimation method**
42	_____ is analogous to _____ in linear regression.	Sum of squares calculations,deviance	deviance	Deviance,sum of squares calculations	sum of squares calculations	**Deviance,sum of squares calculations**
43	Deviance can be shown to follow _____.	t-distribution	F-distribution	Chi-square distribution	None of the above	**Chi-square distribution**

44	_____ value of deviance represents the better fit of model.	Higher	Lower	Smaller	Very Lower	**Lower**
45	If the model deviance is significantly _____ than the null deviance then one can conclude that the predictor or set of predictors significantly improved model fit.	Smaller	Larger	Moderate	Higher	**Smaller**
46	Which of the following is analogous to R-Squared for logistic regression.	Likelihood ration R-squared	McFadden R-squared	Cox and Snell R-Squared	All of the above	**All of the above**
47	Estimation in logistic regression chooses the parameters that _____ the likelihood of observing the sample values.	Minimizes	Maximizes	Higher	Lower	**Maximizes**
48	Which of the following tests can be used to assess whether the logistic regression model is well calibrated.	Hosmer-Lemeshow test	ROC Curve	Both of the above	Regression Operating Characteristic	**Hosmer-Lemeshow test**

49	ROC related with ROC curve stands for _____.	Regression Optimization Characteristic	Regression Operating Characteristic	Receiver Operating Characteristic	Regression Operating Characteristic	**Receiver Operating Characteristic**
50	Which of the following is used to identify the best threshold for separating positive and negative classes.	Hosmer-Lemeshow test	ROC Curve	Both of the above	AIC	**ROC Curve**
51	ROC curve is a plot of _____ vs _____.	Sensitivity,1-specificity	1-specificity	1-specificity	Sensitivity	**Sensitivity,1-specificity**
52	_____ the value of AUC,better is the prediction power of the model.	Moderate	Lower	Higher	Very Higher	**Higher**

UNIT-IV

SYLLABUS

TREE AND RULE MODELS : Decision trees – learning decision trees – ranking and probability estimation trees –Regression trees – clustering trees – learning ordered rule lists – learning unordered rule lists – descriptive rule learning – association rule mining – first -order rule learning

TREE AND RULE MODELS

DECISION TREES

A decision tree is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers the to the question; and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface.

Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example. Each node in the tree acts as a test case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new nodes.

Let's illustrate this with help of an example. Let's assume we want to play badminton on a particular day — say Saturday — how will you decide whether to play or not. Let's say you go out and check if it's hot or cold, check the speed of the wind and humidity, how the weather is, i.e. is it sunny, cloudy, or rainy. You take all these factors into account to decide if you want to play or not.

So, you calculate all these factors for the last ten days and form a lookup table like the one below.

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No

Day	Weather	Temperature	Humidity	Wind	Play?
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

Now, you may use this table to decide whether to play or not. But, what if the weather pattern on Saturday does not match with any of rows in the table? This may be a problem. A decision tree would be a great way to represent data like this because it takes into account all the possible paths that can lead to the final decision by following a tree-like structure.

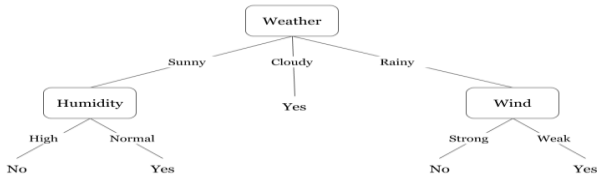


Fig 1. A decision tree for the concept Play Badminton

Fig 1. illustrates a learned decision tree. We can see that each node represents an attribute or feature and the branch from each node represents the outcome of that node. Finally, its the leaves of the tree where the final decision is made. If features are continuous, internal nodes can test the value of a feature against a threshold (see Fig. 2).

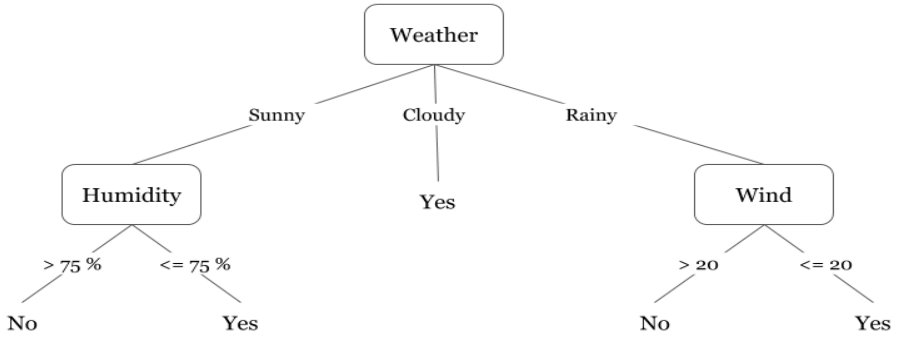


Fig 2. A decision tree for the concept Play Badminton (when attributes are continuous)

A general algorithm for a decision tree can be described as follows:

1. Pick the best attribute/feature. The best attribute is one which best splits or separates the data.
2. Ask the relevant question.
3. Follow the answer path.
4. Go to step 1 until you arrive to the answer.

The best split is one which separates two different labels into two sets.

Expressiveness of decision trees

Decision trees can represent any boolean function of the input attributes. Let's use decision trees to perform the function of three boolean gates AND, OR and XOR.

Boolean Function: AND

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T

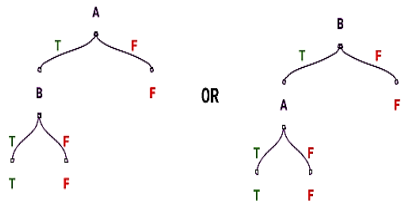


Fig 3. Decision tree for an AND operation.

In Fig 3., we can see that there are two candidate concepts for producing the decision tree that performs the AND operation. Similarly, we can also produce a decision tree that performs the boolean OR operation.

Boolean Function: OR

A	B	A OR B
F	F	F
F	T	T
T	F	T
T	T	T

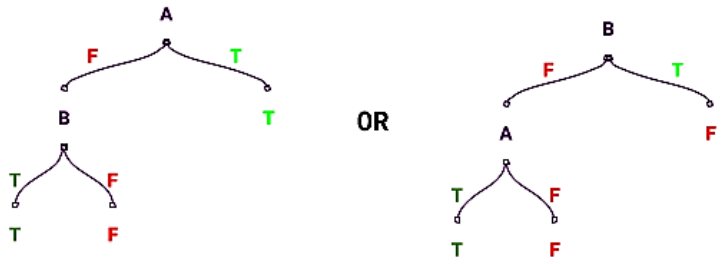


Fig 4. Decision tree for an OR operation

Boolean Function: XOR

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F

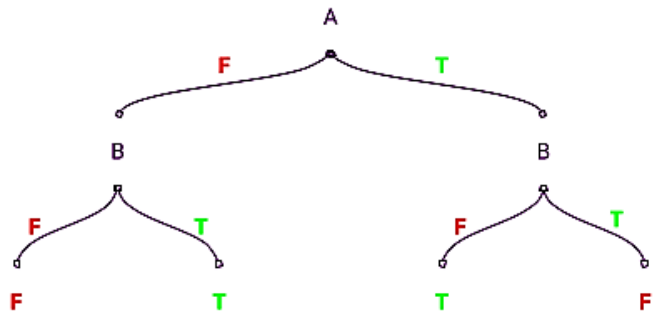


Fig 5. Decision tree for an XOR operation.

Let's produce a decision tree performing XOR functionality using 3 attributes:

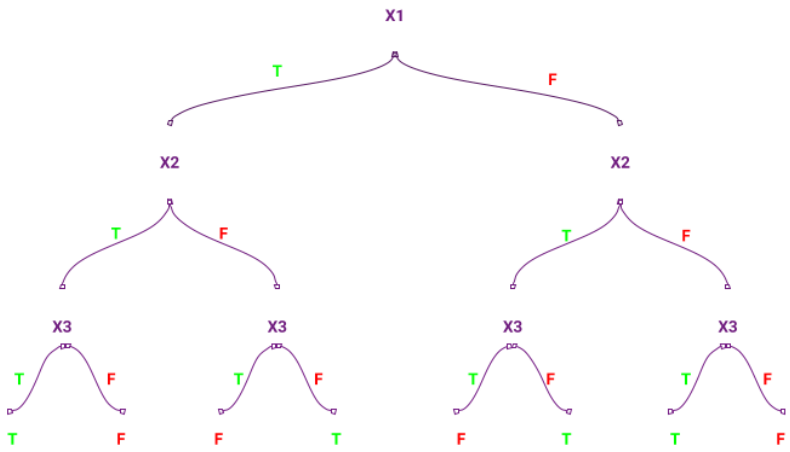


Fig 6. Decision tree for an XOR operation involving three operands

In the decision tree, shown above (Fig 6.), for three attributes there are 7 nodes in the tree, i.e., for $n = 3$, number of nodes = 2^3-1. Similarly, if we have n attributes, there are 2^n nodes (approx.) in the decision tree. So, the tree requires exponential number of nodes in the worst case.

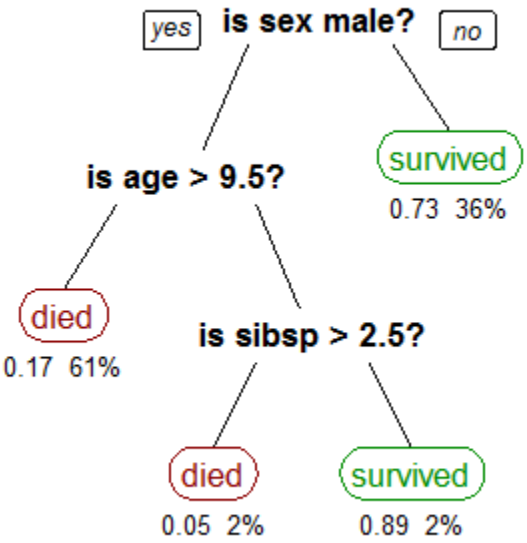
We can represent boolean operations using decision trees. But, what other kind of functions can we represent and if we search over the various possible decision trees to find the right one, how many decision trees do we have to worry about. Let's answer this question by finding out the possible number of decision trees we can generate given N different attributes (assuming the attributes are boolean). Since a truth table can be transformed into a decision tree, we will form a truth table of N attributes as input.

X1	X2	X3	....	XN	OUTPUT
T	T	T	...	T	
T	T	T	...	F	
...	...	...	...	...	
...	...	...	...	...	
...	...	...	...	...	
F	F	F	...	F	

The above truth table has 2^n rows (i.e. the number of nodes in the decision tree), which represents the possible combinations of the input attributes, and since each node can a hold a binary value, the number of ways to fill the values in the decision tree is ${2^{2^n}}$. Thus, the space of decision trees, i.e, the hypothesis space of the decision tree is very expressive because there are a lot of different functions it can represent. But, it also means one needs to have a clever way to search the best tree among them.

LEARNING DECISION TREES

A decision tree is drawn upside down with its root at the top. In the image on the left, the bold text in black represents a condition/**internal node**, based on which the tree splits into branches/ **edges**. The end of the branch that doesn't split anymore is the decision/**leaf**, in this case, whether the passenger died or survived, represented as red and green text respectively.



Although, a real dataset will have a lot more features and this will just be a branch in a much bigger tree, but you can't ignore the simplicity of this algorithm. The **feature importance is clear** and relations can be viewed easily. This methodology is more commonly known as **learning decision tree from data** and above tree is called **Classification tree** as the target is to classify passenger as survived or died. **Regression trees** are represented in the same manner, just they predict continuous values like price of a house. In general, Decision Tree algorithms are referred to as CART or Classification and Regression Trees.

Growing a tree involves deciding on **which features to choose** and **what conditions to use** for splitting, along with knowing when to stop. As a tree generally grows arbitrarily, **you will need to trim it down** for it to look beautiful. Lets start with a common technique used for splitting.

Recursive Binary Splitting

n this procedure all the features are considered and different split points are tried and tested using a cost function. The split with the best cost (or lowest cost) is selected.

Consider the earlier example of tree learned from titanic dataset. In the first split or the root, all attributes/features are considered and the training data is divided into groups based on this split. We have 3 features, so will have 3 candidate splits. Now we will **calculate how much accuracy each split will cost us, using a function**. **The split that costs least is chosen**, which in our example is sex of the passenger. This **algorithm is recursive in nature** as the groups formed can be sub-divided using same strategy. Due to this procedure, this algorithm is also known as the **greedy algorithm**, as we have an

excessive desire of lowering the cost. **This makes the root node as best predictor/classifier.**

Cost of a split

Lets take a closer look at **cost functions used for classification and regression**. In both cases the cost functions try to **find most homogeneous branches, or branches having groups with similar responses**. This makes sense we can be more sure that a test data input will follow a certain path.
Regression : sum(y—prediction)²
Lets say, we are predicting the price of houses. Now the decision tree will start splitting by considering each feature in training data. The mean of responses of the training data inputs of particular group is considered as prediction for that group. The above function is applied to all data points and cost is calculated for all candidate splits. Again the split with lowest cost is chosen. Another cost function involves reduction of standard deviation, more about it can be found here.
Classification : G = sum(pk * (1—pk))

A Gini score gives an idea of how good a split is by how mixed the response classes are in the groups created by the split. Here, pk is proportion of same class inputs present in a particular group. A perfect class purity occurs when a group contains all inputs from the same class, in which case pk is either 1 or 0 and G = 0, where as a node having a 50–50 split of classes in a group has the worst purity, so for a binary classification it will have pk = 0.5 and G = 0.5.

When to stop splitting?

You might ask **when to stop growing a tree?** As a problem usually has a large set of features, it results in large number of split, which in turn gives a huge tree. Such trees are complex and can lead to overfitting. So, we need to know when to stop? One way of doing this is to **set a minimum number of training inputs to use on each leaf.** For example we can use a minimum of 10 passengers to reach a decision(died or survived), and ignore any leaf that takes less than 10 passengers. Another way is to set **maximum depth** of your model. **Maximum depth refers to the the length of the longest path from a root to a leaf.**

Pruning

The performance of a tree can be further increased by **pruning**. It involves**removing the branches that make use of features having low importance**. This way, we reduce the complexity of tree, and thus increasing its predictive power by reducing overfitting.

Pruning can start at either root or the leaves. The simplest method of pruning starts at leaves and removes each node with most popular class in that leaf, this change is kept if it doesn't deteriorate accuracy. Its also called **reduced error pruning**. More sophisticated pruning methods can be used such as **cost complexity pruning** where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree. This is also known as **weakest link pruning.**

Advantages of CART

- Simple to understand, interpret, visualize.
- Decision trees implicitly perform variable screening or feature selection.
- Can handle both numerical and categorical data. Can also handle multi-output problems.
- Decision trees require relatively little effort from users for data preparation.
- Nonlinear relationships between parameters do not affect tree performance.

Disadvantages of CART

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging and **boosting**.
- Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

RANKING AND PROBABILITY ESTIMATION TREES

Decision trees divide the instance space into segments, by learning ordering on those segments the decision trees can be turned into rankers. Thanks to access to class distribution in each leaf the optimal orderdering for the training data can be obtained from empirical probabilities $\dot{p}$ (of positive class). The ranking obtained from the empirical probabilities in the leaves of a decision tree yields a convex ROC curve on the training data.

Consider the tree in Figure 5.4 (left). Each node is labelled with the numbers of positive and negative examples covered by it: so, for instance, the root of the tree is labelled with the overall class distribution (50 positives and 100 negatives), resulting in the trivial ranking [50+, 100−]. The corresponding one-segment coverage curve is the ascending diagonal (Figure 5.4 (right)). t Adding split (1) refines this ranking into [30+, 35−][20+, 65−], resulting in a two-segment curve. t Adding splits (2) and (3) again breaks up the segment corresponding to the parent into two segments corresponding to the children. t However, the ranking produced by the full tree – [15+, 3−][29+, 10−][5+, 62−][1+, 25−] – is different from the left-to-right ordering of its leaves, hence we need to reorder the segments of the coverage curve, leading to the top-most, solid curve. This reordering always leads to a convex coverage curve

(left) Abstract representation of a tree with numbers of positive and negative examples covered in each node. Binary splits are added to the tree in the order indicated. (right) Adding a split to the tree will add new segments to the coverage curve as indicated by the arrows. After a split is added the segments may need reordering, and so only the solid lines represent actual coverage curves.

Graphical depiction of all possible labellings and all possible rankings that can be obtained with the four-leaf decision tree. There are 2 4 = 16 possible leaf labellings; e.g., '+ − +−' denotes labelling the first and third leaf from the left as + and the second and fourth leaf as −. There are 4! = 24 possible blue-violet-red-orange paths through these points which start in − − −− and switch each leaf to + in some order; these represent all possible four-segment coverage curves or rankings.

REGRESSION TREES

Decision trees where the target variable can take continuous values (typically real numbers) are called **regression trees**.

In simple linear regression, a real-valued dependent variable Y is modeled as a linear function of a real-valued independent variable X plus noise: $Y = \beta 0 + \beta 1X + \square$ In multiple regression, we let there be multiple independent variables X1, X2, . . . Xp ≡ X, $Y = \beta 0 + \beta\ T\ X + \square$ This is all very well so long as the independent variables each has a separate, strictly additive effect on Y , regardless of what the other variables are doing. It's possible to incorporate some kinds of interaction, $Y = \beta 0 + \beta\ T\ X + \gamma XXT + \square$ but the number of parameters is clearly getting very large very fast with even two-way interactions among the independent variables, and stronger nonlinearities are going to be trouble... Linear regression is a global model, where there is a single predictive formula holding over the entire data-space. When the data has lots of features which interact in complicated, nonlinear ways, assembling a single global model can be very difficult, and hopelessly confusing when you do succeed. An alternative approach to nonlinear regression is to sub-divide, or partition, the space into smaller regions, where the interactions are more manageable. We then partition the sub-divisions again — this is called recursive partitioning — until finally we get to chunks of the space which are so tame that we can fit simple models to them. The global model thus has two parts: one is just the recursive partition, the other is a simple model for each cell of the partition.

Prediction trees use the tree to represent the recursive partition. Each of the terminal nodes, or leaves, of the tree represents a cell of the partition, and has attached to it a simple model which applies in that cell only. A point x belongs to a leaf if x falls in the corresponding cell of the partition. To figure out which cell we are in, we start at the root node of the tree, and ask a sequence of questions about the features. The interior nodes are labeled with questions, and the edges or branches between them labeled by the answers. Which question we ask next depends on the answers to previous questions. In the classic version, each question refers to only a single attribute, and has a yes or no answer, e.g., "Is Horsepower > 50?" or "Is GraduateStudent == FALSE?" Notice that the variables do not all have to be of the same type; some can be continuous, some can be discrete but ordered, some can be categorical, etc.

You could do more-than-binary questions, but that can always be accommodated as a larger binary tree. Somewhat more useful would be questions which involve two or more variables, but we'll see a way to fake that in the lecture on multiple trees. That's the recursive partition part; what about the simple local models? For classic regression trees, the model in each cell is just a constant estimate of Y . That is, suppose the points (xi , yi),(x2, y2), . . .(xc, yc) are all the samples belonging to the leaf-node l.

Then our model for l is just ˆy = 1 c Pc i=1 yi , the sample mean of the dependent variable in that cell. This is a piecewise-constant model.1 There are several advantages to this: • Making predictions is fast (no complicated calculations, just looking up constants in the tree) • It's easy to understand what variables are important in making the prediction (look at the tree) • If some data is missing, we might not be able to go all the way down the tree to a leaf, but we can still make a prediction by averaging all the leaves in the sub-tree we do reach • The model gives a jagged response, so it can work when the true regression surface is not smooth. If it is smooth, though, the piecewise-constant surface can approximate it arbitrarily closely (with enough leaves).

CLUSTERING TREES

Let Dis: $X \times X \rightarrow R$ be an abstract function that measures dissimularity of any two instances $x, x_0 \in X$, such that the higher $Dis(x, x_0)$ is, the less similar x and x_0 are. The cluster dissimilarity of a set of instances D is: $Dis(D) = 1 |D| 2 X x \in D X x_0 \in D Dis(x, x_0)$

Assessing the nine transactions on the online auction site from Example 5.4, using some additional features such as reserve price and number of bids, you come up with the following dissimilarity matrix:

0	1	1	6	13	10	3	13	3	12
1	1	0	1	1	1	3	0	4	0
6	1	0	2	1	1	2	2	1	0
1	3	1	2	0	0	4	0	4	0
1	0	1	1	0	0	3	0	2	0
3	3	1	4	3	0	4	1	3	0
1	3	0	2	0	0	4	0	4	0
3	4	2	4	2	1	4	0	4	0
1	2	0	1	0	0	3	0	4	0

This shows, for instance, that the first transaction is very different from the other eight. The average pairwise dissimilarity over all nine transactions is 2.94.

Using the same features from Example 5.4, the three possible splits are (now with transaction number rather than price):

Model = [A100,B3,E112,M102,T202] [3, 6, 8][1][9][5][2, 4, 7]

Condition = [excellent,good,fair] [1, 6][3, 4, 5, 8][2, 7, 9]

Leslie = [yes,no] [2, 5, 8][1, 3, 4, 6, 7, 9]

The cluster dissimilarity among transactions 3, 6 and 8 is

1

3

2 (0+1+2+1+0+1+2+1+0) = 0.89; and among transactions 2, 4 and 7 it is

1

3

2 (0+1+0+1+0+0+0+0+0) = 0.22. The other three children of the first split contain only a single element and so have zero cluster dissimilarity. The weighted average cluster dissimilarity of the split is then $3/9 \cdot 0.89+1/9 \cdot 0+1/9 \cdot 0+1/9 \cdot 0+3/9 \cdot 0.22 = 0.37$. For the second split, similar calculations result in a split dissimilarity of $2/9 \cdot 1.5+4/9 \cdot 1.19+3/9 \cdot 0 = 0.86$, and the third split yields $3/9 \cdot 1.56+6/9 \cdot 3.56 = 2.89$. The Model feature thus captures most of the given dissimilarities, while the Leslie feature is virtually unrelated

ORDERED RULE LIST:

- Rules are rank ordered according to their priority
  - An ordered rule set is known as a decision list
- When a test record is presented to the classifier
  - It is assigned to the class label of the highest ranked rule it has triggered
  - If none of the rules fired, it is assigned to the default class
- R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds
- R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes
- R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals
- R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles
- R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
turtle	cold	no	no	sometimes	?

Rule Ordering Schemes

- Rule-based ordering
  - Individual rules are ranked based on their quality
- Class-based ordering
  - Rules that belong to the same class appear together

Rule-based Ordering

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

(Refund=No, Marital Status={Married}) ==> No

Class-based Ordering

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

(Refund=No, Marital Status={Married}) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

Direct Method:

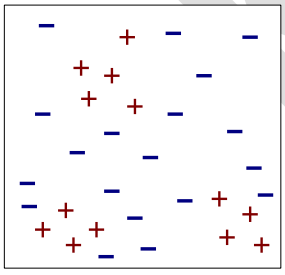
- Extract rules directly from data
- e.g.: RIPPER, CN2, Holte's 1R

Indirect Method:

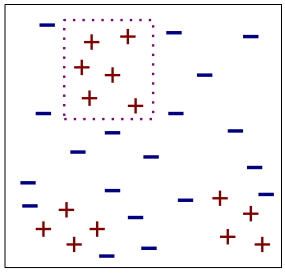
- Extract rules from other classification models (e.g. decision trees, etc).
- e.g: C4.5 rules

Direct Method: Sequential Covering

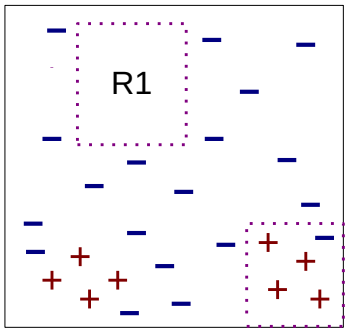
1. Start from an empty rule
2. Grow a rule using the Learn-One-Rule function
3. Remove training records covered by the rule
4. Repeat Step (2) and (3) until stopping criterion is met



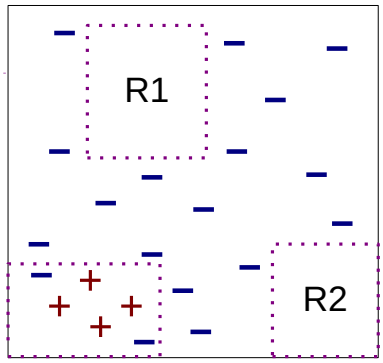
(i) Original Data



(ii) Step 1



(iii) Step 2



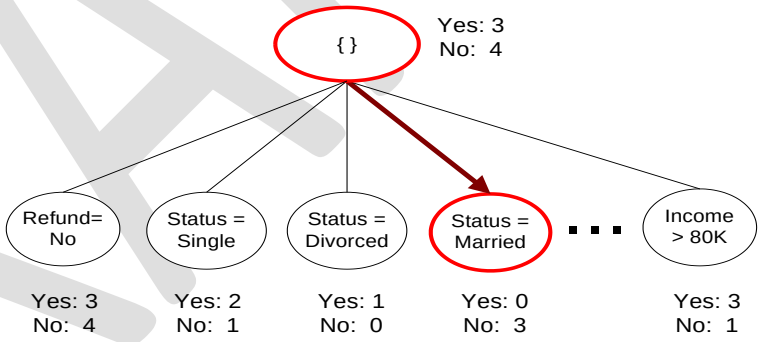
(iv) Step 3

Aspects of Sequential Covering

- Rule Growing
  - Instance Elimination
- Rule Evaluation
- Stopping Criterion

Rule Pruning

Two common strategies



(a) General-to-specific

UNORDERED RULE LIST

An unordered set of rules
→ three situations:
• Matching to rules indicating the same class.
• Multiple matching to rules from different classes.
• No matching to any rule.
• An example:
• e1={(Age=m), (Job=p),(Period=6),(Income=3000),(Purpose=K)}

• rule 3: if (Period
∈[3.5,12.5)) then (Dec=d) [2]
• Exact matching to rule 3.
→ Class (Dec=d)
• e2={(Age=m), (Job=p),(Period=2),(Income=2600),(Purpose=M)}
• No matching!

LERS classification strategy (Grzymala 94)
• Multiple matching
• Two factors: Strength
(
R) – number of learning examples
correctly classified by R and final class Support
(
Yi):
• Partial matching
• Matching factor MF
(
R) and
• e2={(Age=m), (Job=p), (Period=2),(Income=2600),(Purpose=M)}
• Partial matching to rules 2 , 4 and 5 for all with MF = 0.5
• Support(r) = 0.5·2 =1 ; Support(d) = 0.5·2+0.5·2=2
• Alternative approaches – e.g. nearest rules (Stefanowski 95)
• Instead of MF use a kind of normalized distance
x to conditions of r

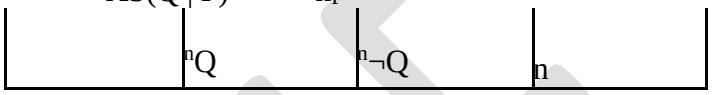
DESCRIPTIVE RULE LEARNING

In a descriptive perspective
• To present, analyse the relationships between values of attributes, to explain and understand classification patterns
• In a prediction/classification perspective,
• To predict value of decision class for new (unseen) object)
Evaluating single rules
• rule r (if P then Q) derived from DT, examples U.
rule r (if P then Q) derived from DT, examples U.

	Q	¬Q	
P	^{n}PQ	$^{n}P\neg Q$	n_P
¬P	$^{n}\neg PQ$	$^{n}\neg P\neg Q$	$^{n}\neg P$

- Hilderman R.J., Hamilton H.J, Knowledge Discovery and Measures of Interest. Kluwer, 2002.

- Support of rule r $G(P \wedge Q) = \frac{n_{PQ}}{n}$ Coverage $AS(P \mid Q) = \frac{n_{PQ}}{n_Q}$
- Confidence of rule r $AS(Q \mid P) = \frac{n_{PQ}}{n_P}$ and others …



- Reviews of measures, e.g.
- Yao Y.Y, Zhong N., An analysis of quantitative measures associated with rules, In: Proc. the 3rd Pacific-Asia Conf. on Knowledge Discovery and Data Mining, LNAI 1574, Springer, 1999, pp. 479-488.

Aggregated measures

Based on previous measures:

Significance of a rule (propozycja Yao i Liu)

$S(Q \mid P) = AS(Q \mid P) \cdot IND(Q, P)$

Klosgen's measure of interest

$K(Q \mid P) = G(P)\alpha \cdot (AS(Q \mid P) - G(Q))$

Michalski's weighted sum

$WSC(Q \mid P) = w1 \cdot AS(Q \mid P) + w2 \cdot AS(P \mid Q)$

The relative risk (Ali, Srikant): $r(Q \mid P) = \frac{AS(Q \mid P)}{AS(Q \mid \neg P)}$

Descriptive requirements to single rules

- In descriptive perspective users may prefer to discover rules which should be:
- strong / general – high enough rule coverage AS(P|Q) or support.
- accurate – sufficient accuracy AS(Q|P).
- simple (e.g. which are in a limited number and have short condition parts).
- Number of rules should not be too high.
- Covering algorithms biased towards minimum set of rules
- containing only a limited part of potentially `interesting' rules.

- We need another kind of rule induction algorithms!

Explore algorithm (Stefanowski, Vanderpooten)

- Another aim of rule induction
- to extract from data set inducing all rules that satisfy some user's requirements connected with his interest (regarding, e.g. the strength of the rule, level of confidence, length, sometimes also emphasis on the syntax of rules).
- Special technique of exploration the space of possible rules:
- Progressively generation rules of increasing size using in the most efficient way some 'good' pruning and stopping condition that reject unnecessary candidates for rules.

- Similar to adaptations of Apriori principle for looking frequent itemsets [AIS94]; Brute [Etzioni]

ASSOCIATION RULE MINING

Association Rule Mining, as the name suggests, association rules are simple If/Then statements that help discover relationships between seemingly independent relational databases or other data repositories.

Most machine learning algorithms work with numeric datasets and hence tend to be mathematical. However, association rule mining is suitable for non-numeric, categorical data and requires just a little bit more than simple counting.

Association rule mining is a procedure which aims to observe frequently occurring patterns, correlations, or associations from datasets found in various kinds of databases such as relational databases, transactional databases, and other forms of repositories.

An association rule has two parts:

- an antecedent (if) and
- a consequent (then).

An antecedent is something that's found in data, and a consequent is an item that is found in combination with the antecedent. Have a look at this rule for instance:

"If a customer buys bread, he's 70% likely of buying milk."

In the above association rule, bread is the antecedent and milk is the consequent. Simply put, it can be understood as a retail store's association rule to target their customers better. If the above rule is a result of thorough analysis of some data sets, it can be used to not only improve customer service but also improve the company's revenue.

Association rules are created by thoroughly analyzing data and looking for frequent if/then patterns. Then, depending on the following two parameters, the important relationships are observed:

1. **Support**: Support indicates how frequently the if/then relationship appears in the database.
2. **Confidence**: Confidence tells about the number of times these relationships have been found to be true.

So, in a given transaction with multiple items, Association Rule Mining primarily tries to find the rules that govern how or why such products/items are often bought together. For example, peanut butter and jelly are frequently purchased together because a lot of people like to make PB&J sandwiches.

Association Rule Mining is sometimes referred to as "Market Basket Analysis", as it was the first application area of association mining. The aim is to discover associations of items occurring together more often than you'd expect from randomly sampling all the possibilities. The classic anecdote of Beer and Diaper will help in understanding this better.

The story goes like this: young American men who go to the stores on Fridays to buy diapers have a predisposition to grab a bottle of beer too. However unrelated and vague that may sound to us laymen, association rule mining shows us how and why!

Let's do a little analytics ourselves, shall we?

Suppose an X store's retail transactions database includes the following data:

- Total number of transactions: 600,000
- Transactions containing diapers: 7,500 (1.25 percent)
- Transactions containing beer: 60,000 (10 percent)
- Transactions containing both beer and diapers: 6,000 (1.0 percent)

From the above figures, we can conclude that if there was no relation between beer and diapers (that is, they were statistically independent), then we would have got only 10% of diaper purchasers to buy beer too.

However, as surprising as it may seem, the figures tell us that **80% (=6000/7500) of the people who buy diapers also buy beer**.

This is a significant jump of 8 over what was the expected probability. This factor of increase is known as Lift – which is the ratio of the observed frequency of co-occurrence of our items and the expected frequency.

Let's look at some areas where Association Rule Mining has helped quite a lot:

1. **Market Basket Analysis:**

This is the most typical example of association mining. Data is collected using barcode scanners in most supermarkets. This database, known as the "market basket" database, consists of a large number of records on past transactions. A single record lists all the items bought by a customer in one sale. Knowing which groups are inclined towards which set of items gives these shops the freedom to adjust the store layout and the store catalogue to place the optimally concerning one another.

2. Medical Diagnosis:

Association rules in medical diagnosis can be useful for assisting physicians for curing patients. Diagnosis is not an easy process and has a scope of errors which may result in

unreliable end-results. Using relational association rule mining, we can identify the probability of the occurrence of an illness concerning various factors and symptoms. Further, using learning techniques, this interface can be extended by adding new symptoms and defining relationships between the new signs and the corresponding diseases.

3. Census Data:

Every government has tonnes of census data. This data can be used to plan efficient public services(education, health, transport) as well as help public businesses (for setting up new factories, shopping malls, and even marketing particular products). This application of association rule mining and data mining has immense potential in supporting sound public policy and bringing forth an efficient functioning of a democratic society.

4. Protein Sequence:

Proteins are sequences made up of twenty types of amino acids. Each protein bears a unique 3D structure which depends on the sequence of these amino acids. A slight change in the sequence can cause a change in structure which might change the functioning of the protein. This dependency of the protein functioning on its amino acid sequence has been a subject of great research. Earlier it was thought that these sequences are random, but now it's believed that they aren't. Nitin Gupta, Nitin Mangal, Kamal Tiwari, and Pabitra Mitra have deciphered the nature of associations between different amino acids that are present in a protein. Knowledge and understanding of these association rules will come in extremely helpful during the synthesis of artificial proteins.

FIRST ORDER RULE LEARNING

In machine learning, first-order inductive learner (FOIL) is a rule-based learning algorithm.

Developed in 1990 by Ross Quinlan,[1] FOIL learns function-free Horn clauses, a subset of first-order predicate calculus. Given positive and negative examples of some concept and a set of background-knowledge predicates, FOIL inductively generates a logical concept definition or rule for the concept. The induced rule must not involve any constants (color(X,red) becomes color(X,Y), red(Y)) or function symbols, but may allow negated predicates; recursive concepts are also learnable.

Like the ID3 algorithm, FOIL hill climbs using a metric based on information theory to construct a rule that covers the data. Unlike ID3, however, FOIL uses a separate-and-conquermethod rather than divide-and-conquer, focusing on creating one rule at a time and collecting uncovered examples for the next iteration of the algorithm

The FOIL algorithm is as follows:

Input List of examples
Output Rule in first-order predicate logic
FOIL(Examples)

Let **Pos** be the positive examples
Let **Pred** be the predicate to be learned
Until **Pos** is empty do:

Let **Neg** be the negative examples
Set **Body** to empty
Call LearnClauseBody
Add **Pred** ← **Body** to the rule
Remove from **Pos** all examples which satisfy **Body**
Procedure LearnClauseBody

Until **Neg** is empty do:

Choose a literal **L**
Conjoin **L** to **Body**
Remove from **Neg** examples that do not satisfy **L**

The **FOCL** algorithm[3] (First Order Combined Learner) extends FOIL in a variety of ways, which affect how FOCL selects literals to test while extending a clause under construction. Constraints on the search space are allowed, as are predicates that are defined on a rule rather than on a set of examples (called intensional predicates); most importantly a potentially incorrect hypothesis is allowed as an initial approximation to the predicate to be learned. The main goal of FOCL is to incorporate the methods of explanation-based learning (EBL) into the empirical methods of FOIL.

Even when no additional knowledge is provided to FOCL over FOIL, however, it utilizes an iterative widening search strategy similar to depth-first search: first FOCL attempts to learn a clause by introducing no free variables. If this fails (no positive gain), one additional free variable per failure is allowed until the number of free variables exceeds the maximum used for any predicate.

Unlike FOIL, which does not put typing constraints on its variables, FOCL uses typing as an inexpensive way of incorporating a simple form of background knowledge. For example, a predicate livesAt(X,Y) may have types livesAt(person, location). Additional predicates may need to be introduced, though – without types, nextDoor(X,Y) could determine whether person X and person Y live next door to each other, or whether two locations are next door to each other. With types, two different predicates nextDoor(person, person) and nextDoor(location, location) would need to exist for this functionality to be maintained. However, this typing mechanism eliminates the need for predicates such as isPerson(X) or isLocation(Y), and need not consider livesAt(A,B) when A and B are defined to be person variables, reducing the search space. Additionally, typing can improve the accuracy of the resulting rule by eliminating from consideration impossible literals such as livesAt(A,B) which may nevertheless appear to have a high information gain.

Rather than implementing trivial predicates such as equals(X,X) or between(X,X,Y), FOCL introduces implicit constraints on variables, further reducing search space. Some predicates must have all variables unique, others must have commutativity (adjacent(X,Y) is equivalent to adjacent(Y,X)), still others may require that a particular variable be present in the current clause, and many other potential constraints.

Operational rules

Operational rules are those rules which are defined extensionally, or as a list of tuples for which a predicate is true. FOIL allows only operational rules; FOCL extends its knowledge base to allow combinations of rules called non-operational rules as well as partially defined or incorrect rules for robustness. Allowing for partial definitions reduces the amount of work needed as the algorithm need not generate these partial definitions for itself, and the incorrect rules do not add significantly to the work needed since they are discarded if they are not judged to provide positive information gain. Non-operational rules are advantageous as the individual rules which they combine may not provide information gain on their own, but are useful when taken in conjunction. If a literal with the most information gain in an iteration of FOCL is non-operational, it is operationalized and its definition is added to the clause under construction.

Inputs Literal to be operationalized, List of positive examples, List of negative examples

Output Literal in operational form

Operationalize(Literal, Positive examples, Negative examples)

If **Literal** is operational

Return **Literal**

Initialize **OperationalLiterals** to the empty set

For each clause in the definition of **Literal**

Compute information gain of the clause over Positive examples and Negative examples

For the clause with the maximum gain

For each literal **L** in the clause

Add Operationalize(**L**, Positive examples, Negative examples) to **OperationalLiterals**

An operational rule might be the literal lessThan(X,Y); a non-operational rule might be between(X,Y,Z) ← lessThan(X,Y), lessThan(Y,Z).

Initial rules

The addition of non-operational rules to the knowledge base increases the size of the space which FOCL must search. Rather than simply providing the algorithm with a target concept (e.g. grandfather(X,Y)), the algorithm takes as input a set of non-operational rules which it tests for correctness and operationalizes for its learned concept. A correct target concept will clearly improve computational time and accuracy, but even an incorrect concept will give the algorithm a basis from which to work and improve accuracy and time.

POSSIBLE QUESTIONS

PART-B (6 MARKS)

1. Explain about decision tree with neat diagram.
2. Write a note on clustering tree and regression tree.
3. Describe descriptive rule learning in detail.
4. Write a note on first order rule learning.

PART-C(10 MARKS)

1.Write in detail about Association rule mining with neat diagram.
2. Describe about decision tree with suitable diagram

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of CS,CA & IT

III B.Sc(CS) (BATCH 2016-2019)

Machine Learning

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS **ONE MARK QUESTIONS**

QUESTIONS	OPTION1	OPTION2	OPTION3	OPTION4	ANSWER
_____ is a is a *universal* approximator so it can implement linear regression algorithm.	Neural network	Linear regression	Logistic Regression	Multinomial logistic regression	Neural network
Which of the following methods do we use to best fit the data in Logistic Regression?	Least Square Error	Maximum Likelihood	Jaccard distance	Both A and B	Maximum Likelihood

A _____ can only contain objects of the same class.	list	vector	data frames	factor	vector
>m <- matrix(nrow = 2, ncol = 3) >m > attributes(m)	2 3	3 2	dim	NA	dim
_____ function to find the data type of the variable	datatype()	class()	type()	cls()	class()
The _____ Function get the current working directory	get()	getwd()	getw()	wd()	getwd()
Which function is used to transpose data frame?	t()	ti()	transpose()	trans()	t()

Vectors come in two parts:_____ and _____.	**atomic vectors and matrix**	**atomic vectos and array**	**atomic vectors and list**	**atomic vectors and functions**	**atomic vectors and list**
Which of the following is a base package for R language ?	**util**	**lang**	**tools**	**math**	**tools**
Which of the following is used for reading in saved workspaces ?	**unserialize**	**load**	**get**	**read**	**load**
The _____ function can be useful for reading in lines of webpages	**Load()**	**readLines()**	**read()**	**readpage()**	**readLines()**
Text files can be read line by line using the _____function.	**Load()**	**readpage()**	**read()**	**readLines()**	**readLines()**

The _____ package is recently developed by Hadley Wickham to deal with reading in large flat files quickly.	**readr**	**dplyr**	**read**	**dr**	**readr**
The _____ and _____ functions are useful because the resulting textual format is editable, and in the case of corruption, potentially recoverable.	**dump() and dget()**	**dump() and dput()**	**dget() and dput()**	**dump() and dp()**	**dump() and dput()**
_____ opens a connection to a file	**file**	**gzfile**	**bzfile**	**url**	**file**
_____ opens a connection to a file compressed with gzip	**file**	**gzfile**	**bzfile**	**url**	**gzfile**
_____ opens a connection to a file compressed with bzip2	**file**	**gzfile**	**bzfile**	**url**	**bzfile**

_____ opens a connection to a webpage	**file**	**gzfile**	**bzfile**	**url**	**url**
The ____ function has a number of arguments that are common to many other connection	**f()**	**close()**	**file()**	**open()**	**file()**
_____ open file in read only mode	**“r”**	**“a”**	**“w”**	**"ab"**	**“r”**
_____ open a file for writing (and initializing a new file)	**“r”**	**“a”**	**“w”**	**"ab"**	**“w”**
_____ open a file for appending	**“r”**	**“a”**	**“w”**	**"ab"**	**“a”**

The_____ operator can be used to extract multiple elements of a vector by passing the operator an integer sequence	$	[	[[	((	[
We can also create an empty list of a prespecified length with the_____function	create()	file()	vector()	list()	vector()
Which of the following is example of vecto	x+y	x-y	x*y	x/y	x-y
In simulating linear model can also simula	generalized model	generalized linea	linear model	ungeneralized line	generalized linear mode
Simulating _____ numbers is useful bu	arbitrary	sample	random	sequence	random

The function call stack is the _____ c	arbitrary	sample	random	sequence	sequence
A nearest neighbor approach is best used	**with large-sized datasets.**	**when irrelevant attributes have been removed from the data.**	**when a generalized model of the data is desireable.**	**when an explanation of what has been found is of primary importance.**	**when irrelevant attributes have been removed from the data.**
If a customer is spending more than expected, the customer's intrinsic value is _____ their actual value.	**greater than**	**less than**	**less than or equal to**	**equal to**	**less than**
_____ can be any unprocessed fact, value, text, sound or picture that is not being interpreted and analyze	**data**	**knowledge**	**information**	**machine**	**data**
Database query is used to uncover this type of knowledge.	**deep**	**hidden**	**shallow**	**multidimensional**	**shallow**

A statement to be tested.	**theory**	**procedure**	**principle**	**hypothesis**	**hypothesis**
A person trained to interact with a human expert in order to capture their knowledge.	**knowledge programmer**	**knowledge developer**	**knowledge engineer**	**knowledge extractor**	**knowledge engineer**
Logistic regression is _____ when the observed outcome of dependent variable are ordered.	**Binomial**	**Multinomial**	**Ordinal**	**sum of squares calculations**	**Ordinal**
Logit transformation is log of _____.	**Odds of the event happening for different levels of each independent variable**	**Ratio of odds of the event happening for different levels of each independent**	**Deviance,sum of squares calculations**	**Dependent variable equalling a given case**	**Ratio of odds of the event happening for different levels of each independent variable**
Logistic regression is _____ when the observed outcome of dependent variable can have multiple possible types	**Binomial**	**Multinomial**	**Ordinal**	**Cardinal**	**Multinomial**

Regression coefficients in logistic regression are estimated using _____.	Ordinary least squares method	Maximum likelihood estimation method	Dependent variable equalling a given case	sum of squares calculations	Maximum likelihood estimation method
_____ acts as an outstanding tool for visulaizing technical data	C	C++	Java	MATLAB	MATLAB
The _____ command will display a list of possible help topics in the command window	help	helper	lookfor	order	help
The _____ function accepts an array arugment and displayus the value of athe array in the command window	disp	format	special	fprintf	disp
_____ are operations performed between arrays on an element by element basis	matrix operations	array operations	vector operations	arthimetic operations	array operations

UNIT-V

SYLLABUS

REINFORCEMENT LEARNING : Passive reinforcement learning – direct utility estimation – adaptive dynamic programming – temporal - difference learning – active reinforcement learning – exploration – learning an action utility function – Generalization in reinforcement learning – policy search – applications in game playing – applications in robot control

REINFORCEMENT LEARNING

In many domains it is difficult to give a precise evaluation function, which would allow an agent to evaluate the effectiveness or correctness of her actions, except after she has reached a terminal state. In the terminal state the agent by default obtains an objective evaluation of her actions which is termed a reward, or a reinforcement. It is convenient to state the agent's task so that it would have to act in such a way to maximize this reward or reinforcement. This is called reinforcement learning.

In a general case the agent may not have full information about her environment, as well as a precise, or any, description of her actions. The

situation of this agent is similar to one of the possible statement of a full task of artificial intelligence — the agent is placed in an environment she does not know, and cannot act in it. She has to learn to act in this environment effectively, to maximize some criterion, available as reinforcements.

We shall consider a probabilistic model of the agent's action outcomes. In fact, we shall assume that we are dealing with an unknown Markov decision problem (MDP).

PASSIVE REINFORCEMENT LEARNING

Let us first consider passive reinforcement learning, where we assume that the agent's policy $\pi(s)$ is fixed. Agent is therefore bound to do what the policy dictates, although the outcomes of her actions are probabilistic. The agent may watch what is happening, so she knows what states she is reaching and what rewards she gets there.

The agent's jobs is to learn the utilities of the states $U^{\pi}(s)$, computed according to the equation:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

For the 4x3 example environment used here for illustration we will assume

$\gamma = 1$.

DIRECT UTILITY DETERMINATION

The objective of the agent is to compute the state utilities $U^{\pi}(s)$ generated by the current policy $\pi(s)$. The state utilities are defined as expected values of the sums of the (discounted) reinforcements received by the agent, who started from a given state, and is acting according to her policy:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

The agent may treat the trials as a source of training data to learn the state utilities by computing the reward-to-go for each state. At the end of each trial the agent takes the reward obtained in that state, and then, backtracking along its path, computes the reward-to-go for subsequent states, by adding up the rewards obtained there.

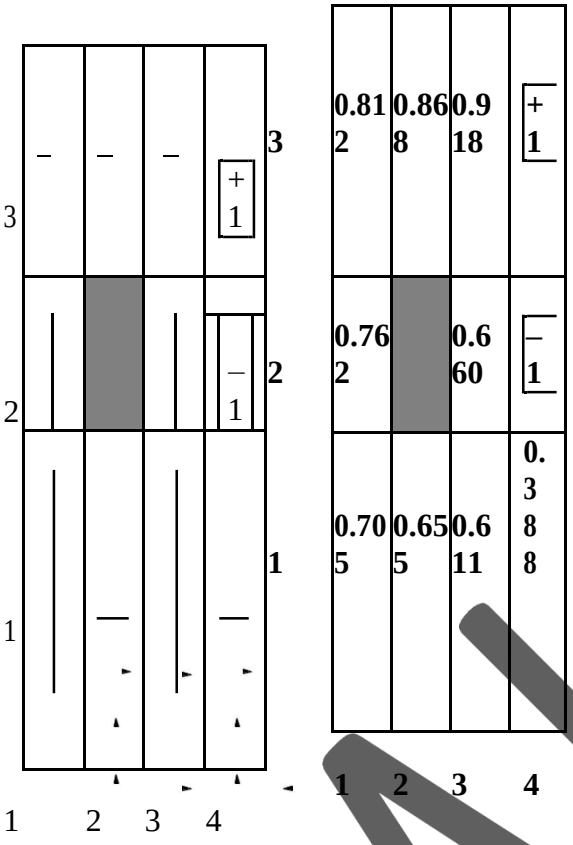
For example, for the trial:

$(1, 1)_{-0.04} \rightarrow (1, 2)_{-0.04} \rightarrow (1, 3)_{-0.04} \rightarrow (1, 2)_{-0.04} \rightarrow (1, 3)_{-0.04} \rightarrow (2, 3)_{-0.04} \rightarrow (3, 3)_{-0.04} \rightarrow (4, 3)_{+1}$

we get $R_{tg}(4, 3) = 1$, $R_{tg}(3, 3) = 0.96$, $R_{tg}(2, 3) = 0.92$, $R_{tg}(1, 3) = 0.88$, $R_{tg}(1, 2) = 0.84$, $R_{tg}(1, 3) = 0.80$, $R_{tg}(1, 2) = 0.76$, $R_{tg}(1, 1) = 0.72$

Trials

Recall the 4 × 3 environment we studied earlier:



The agent executes the trials where she performs actions according to the policy, until reaching a terminal state, and receives percepts indicating both the current state and the reinforcement. Example trials:

$(1, 1)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(2, 3)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(4, 3)_{+1}$

$(1, 1)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(2, 3)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(3, 2)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(4, 3)_{+1}$

$(1, 1)_{-0.04}$ ❀ $(2, 1)_{-0.04}$ ❀ $(3, 1)_{-0.04}$ ❀ $(3, 2)_{-0.04}$ ❀ $(4, 2)_{-1}$

By averaging over a large number of samples she can determine the subsequent approximations of the expected value of state utilities, which converge in the infinity to the real expected values. This way the reinforcement learning task is reduced to a simple inductive learning.

This approach works, but is not very efficient, since it requires a large number of trials. The problem is, that the algorithm, by using simple averaging, ignores important information contained in the trials, namely, that state utilities in neighboring states are related.

$(1, 1)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(2, 3)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(4, 3)_{+1}$
$(1, 1)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(2, 3)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(3, 2)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(4, 3)_{+1}$

$(1, 1)_{-0.04}$ ❀ $(2, 1)_{-0.04}$ ❀ $(3, 1)_{-0.04}$ ❀ $(3, 2)_{-0.04}$ ❀ $(4, 2)_{-1}$

For example, in the second trial of the previous example, the algorithm evaluates the utility of state (3,2) as the reward-to-go from this trial, but ignores the fact, that the successor state in this state is (3,3), which has a high, and already known, utility. The Bellman equation relates the utilities of subsequent state, but this approach cannot take advantage of this

ADAPTIVE DYNAMIC PROGRAMMING

The adaptive dynamic programming (ADP) is a process similar to the dynamic programming, combined with learning the model of the environment, ie. the state transition probability distribution and the reward function. It works by counting the transitions from the state-action pairs to the next states. The trials provide the training data of such transitions. The agent can compute the probability of the transitions as frequencies of their occurrences in the trials.

For example, in the presented trials, the action → (Right) was executed three times in the state (1,3). Two of these times the successor state was (2,3), so the agent should compute
$P((2, 3)|(1, 3), \text{Right}) = \frac{2}{3}$.

$(1, 1)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(2, 3)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(4, 3)_{+1}$
$(1, 1)_{-0.04}$ ❀ $(1, 2)_{-0.04}$ ❀ $(1, 3)_{-0.04}$ ❀ $(2, 3)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(3, 2)_{-0.04}$ ❀ $(3, 3)_{-0.04}$ ❀ $(4, 3)_{+1}$

$(1, 1)_{-0.04}$ ❀ $(2, 1)_{-0.04}$ ❀ $(3, 1)_{-0.04}$ ❀ $(3, 2)_{-0.04}$ ❀ $(4, 2)_{-1}$

After executing every single action the agent updates the state utilities by solving the (simplified) Bellman equation using one of the appropriate methods. The equation is simplified because we only know the distribution of the effects of actions belonging to the policy, and we cannot compute the best action for each state. Since we are computing the U^{π} we take just these actions

Adaptive dynamic programming — the algorithm

```
function PASSIVE-ADP-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
              mdp, an MDP with model P, rewards R, discount γ
              U, a table of utilities, initially empty
              N_sa, a table of frequencies for state–action pairs, initially zero
              N_s'|sa, a table of outcome frequencies given state–action pairs, initially zero
              s, a, the previous state and action, initially null

  if s' is new then U[s'] ← r'; R[s'] ← r'
  if s is not null then
      increment N_sa[s, a] and N_s'|sa[s', s, a]
      for each t such that N_s'|sa[t, s, a] is nonzero do
          P(t | s, a) ← N_s'|sa[t, s, a] / N_sa[s, a]
  U ← POLICY-EVALUATION(π, U, mdp)
  if s'.TERMINAL? then s, a ← null else s, a ← s', π[s']
  return a
```

Adaptive dynamic programming — efficiency

The ADP algorithm updates the utility values as best as it is possible, and in this respect it is a standard that the other algorithms can be compared to. The policy computation procedure, which solves a system of linear equations, can be intractable for problems with large state spaces (eg. for the backgammon game we get 10^{50} equations with 10^{50} unknowns).

TEMPORAL DIffERENCE LEARNING

Instead of solving the full equation system for each trial, it is possible to update the utilities using the currently observed reinforcements. Such algorithm is called the temporal difference (TD) method:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

In this case the single utility value is updated from a single observed state transition, instead of the expected value of all transitions. This is why we take this correction — the difference

between the utility of the move and the utility of the state — reduced by a factor $\alpha < 1$. This introduces small corrections after each move. The correction converges to zero when the state utility becomes equal to the discounted utility of a move.

Note that this method does not require having a model of the environment

P (s′|s, a), nor does it compute one.
Algorithm

function PASSIVE-TD-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    U, a table of utilities, initially empty
    N_s, a table of frequencies for states, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if s' is new **then** $U[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$
  if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

Convergence of the temporal difference method

There is a close relationship and similarity between the ADP and TD algorithms. While the latter makes only local updates in utility values, their average values converge to the same values as for the ADP algorithm.

In case of learning with many transition examples, the frequencies of state occurrences agrees with their probability distributions, and it can be proved that the utilities converge to the correct values.

For this to occur the learning parameter α should decrease with the number of processed trials. More precisely, the subsequent values of this parameter should satisfy the requirements:

$$\sum_{n=1}^{\infty} \alpha(n) = \infty$$

and also:

$$\sum_{n=1}^{\infty} \alpha^2(n) < \infty$$

ACTIVE REINFORCEMENT LEARNING

What should do an agent, which does not have a policy, or who would like to find an optimal one?

First she should compute the complete transition model for all the actions. The ADP algorithm for a passive agent can be used for that. After that, the optimal policy, satisfying the Bellman equation, can be determined like for a regular Markov decision process:

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} P(s'|s, a) U(s')$$

The agent can use the value iteration or the policy iteration algorithm. Then, having determined the optimal policy she could simply execute this policy.

EXPLORATION

Unfortunately, if the agent does not learn the correct environment model from the initial series of trials, and computes the optimal policy for the model it learned, then any subsequent trials will be generated according to the policy possibly suboptimal for the given environment.

This is the compromise between the exploitation of the knowledge already possessed, and the exploration of the environment. The agent should not too early accept the learned environment model, and the policy computed for it. She should try different possibilities.

Moreover, she should try multiple times all the actions in all the states, if she wants to avoid the possible situation, that an unlucky series of trials prevents her from discovering some particularly advantageous choices of actions.

However, eventually she needs to start acting according to the optimal policy, to tune it to its specific patterns

The exploration policy

In order to combined the effective world exploration with the exploitation of the possessed knowledge, the agent must have some exploration policy. It is necessary to ensure that the agent would be able to learn all her available actions to the degree permitting her to compute the globally optimal policy.

A simple exploration policy would be to execute some random actions in all states some fraction of the time, and follow the optimal policy the rest of the time.

This approach works, but is slow to converge. It would be better to prefer the exploration of relatively unexplored state-action pairs, while at the same time avoiding the pairs already better known and believed to be of low utility.

The exploration function

A reasonable exploration policy can be achieved by introducing the optimistic approximation of utilities $U^+(s)$:

$$U(s) \leftarrow R(s) + \gamma \max_a f(\sum_{s'} P(s'|s,a) U(s'), N(a,s))$$

where N (a, s) is the number of previous choices of action a in state s, and

f (u, n) is the exploration function, trading off the greed (large values of u) against curiosity (small values of n).

Obviously the f function should be increasing with respect to u and decreasing with respect to n. A simple definition of f can be:

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

where the R^+ denotes the optimistic estimate of the best reward possible to obtain in any state, and the N_e is the minimal number of times that the agent will want to try each state-action pair.

LEARNING AN ACTION UTILITY FUNCTION

Q-learning

An alternative to temporal difference learning is the Q-learning method, which learns an action-value representation in the form of the function Q(a, s), which expresses the value of the action a in state s, and is related to the utilities with the formula:

$$U(s) = \max_a Q(a, s)$$

The goal values of Q satisfy the equation:

$$Q(a, s) = R(S) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(a', s')$$

The above formula can be utilized in the iterative process for updating the value of Q. However, this requires to simultaneously learn the Q values, and the model, as the probability distribution P , which is present in the formula.
The fact, that the update formula for U^+ in the right hand side also uses U^+ is important. Since the states and actions surrounding the starting state will be executed multiple times, if the agent used non-optimistic utilities for updating, she might get discouraged from such states, and start to avoid "setting out". Using U^+ instead means, that the optimistic values generated around the newly explored regions will be propagated back, and the actions leading to unknown regions will be favored.

Active temporal difference learning

The method of temporal differences can also be applied to active learning. The agent might not have a fixed policy, and still calculate the state utilities using the same update formula as in the passive case:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

Given the computed utilities the agent can determine the optimal actions for each state using the utilities of successor states. It can be proved, that the active TD agent will eventually arrive at the same resulting utility values as the active ADP agent.

Q-learning — updating by temporal differences

It is also possible to apply local updating of the Q function, which is a variant of the temporal difference learning. It is given by the following updating formula, computed whenever action a is executed in state s leading to the result state s′:

$$Q(a, s) \leftarrow Q(a, s) + \alpha(R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s))$$

Q-learning with temporal difference updating converges to the result much slower than the ADP algorithm since it does not enforce computing the full consistency of the model, which it does not have.

The full Q-learning algorithm with exploration

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

In general an active Q-learning agent requires exploration, just as it is in the case with the ADP method. That is why the algorithm uses the exploration function

and the action frequency table N . With a simpler exploration function (eg. executing some fraction of random moves) the N table might not be necessary.
Q-learning is a more flexible algorithm, as it permits the agent to learn the optimal behavior even if she currently executes a policy different from the learned patterns. On the other hand, SARSA is more realistic, since, for example, if the agent was unable to fully control her policy, then it would be better for her to learn the patterns corresponding to what will in fact happen with her, than to learn the best possible patterns.

Both Q-learning and SARSA are capable of learning the optimal policy for the 4x3 example environment, albeit more slowly than the ADP (in terms of the number of iterations). This is because the local updates they make do not enforce the full consistency of the Q function.

Comparing these two methods we might take a broader look and ask, whether it is better to learn the model of the environment and the utility function, or to directly determine the mapping from states to actions, without paying attention to the environment model.

This is one of the fundamental questions of how the artificial intelligence should be constructed. For many years of its initial development, the paradigm of the knowledge-based systems dominated, postulating to build declarative models. The fact that recently model-free approaches such as the Q-learning emerge suggests, that perhaps this was not necessary. However, the model-based methods are still better for some complex tasks, so this issue remains open.

GENERALIZATION IN REINFORCEMENT LEARNING

The above reinforcement learning algorithms assume an explicit representation of the U (s) or Q(s) function, such as, eg. table representation. This may be practical only up to some size of the problem.

For example, for problems with a very large number of states (eg. $\gg 10^{20}$ for games such as chess or backgammon), is is hard to envision executing a sufficient number of trials to visit each

state frequently enough. It is necessary to use some generalization method, which would permit to determine an effective policy based on a small part of the state space explored.

SARSA — State-Action-Reward-State-Action

There exists a variant of the Q-learning algorithm with a temporal difference update method called SARSA (State-Action-Reward-State-Action):

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma Q(s', a') - Q(s, a))$$

SARSA updates take into account five elements: s, a, r, s', a'. While the Q-learning algorithm adjustments are based on the best action selected for the state reached by the action a, SARSA considers which action was in fact selected. Therefore, eg. for a greedy agent executing only exploitation, these two approaches would give the same result.

However, in case of exploration the difference is significant. Q-learning is an off-policy learning algorithm, computing the best possible Q values, regardless of where the current policy takes us. On the other hand, SARSA is an on-policy algorithm, which is more appropriate for the agent acting according to the policy.

Function approximation

One of such methods is the function approximation, based on the representation of the function under examination (eg. U) in some nontabular form, like a finite formula. Similarly as was the case with the heuristic functions, some linear combination of some features (also called the state attributes) can be used:

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + ... + \theta_n f_n(s)$$

The reinforcement learning algorithm would learn the vector of coefficients

$\theta =< \theta_1, \theta_2, ..., \theta_n >$ so that the evaluation function $\hat{U}_\theta$ would closely enough approximate the state utility function.

This approach is called a function approximation because there is no proof that the real evaluation function can be expressed by this kind of a formula. However, while it seems doubtful that, for example, the optimal policy for chess can be expressed by a function with just a few coefficients, it is entirely possible that a good level of playing can be achieved this way.
The main idea of this approach is not an approximation using fewer coefficients a function, which in fact requires many more of them, but the generalization. This is, we want to generate a policy valid for all the states based on the analysis of a small fraction of them.

For examples, in the experiments with the game backgammon, it was possible to train a player to a level of play comparable to human players based on examining one in 10^{12} states.

Obviously, the success in reinforcement learning in such cases depends on the correct selection of the approximation function. If no combination of the selected features can give a good strategy for a game, then no method of learning the coefficients will lead to one. On the other hand, selecting a very elaborate function, with a large number of features and coefficients, increases the chance for a success, but at the expense of a slower convergence and, consequently, the learning process.

Function parameter correction

In order to facilitate on-line learning, some way of updating the parameters based on the reinforcements obtained after each trial (or each step) is needed.

For example, if $u_j(s)$ is the reward-to-go for state s in j-th trial, then the utility function approximation error can be computed as:

$$E_j = \frac{(\hat{U}_\theta(s) - u_j(s))^2}{2}$$

The rate of change of this error with respect to the parameter θ_i can be written as $\partial E_j/\partial\theta_i$, so in order to adjust this parameter toward decreasing the error, the proper adjustment formula is:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha(u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

The above formula is known as the Widrow-Hoff or the delta rule.

An example

As an example, for the 4x3 environment the state utility function could be approximated using a linear combination of the coordinates:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

According to the delta rule the corrections will be given by:

$$\theta_0 \leftarrow \theta_0 + \alpha(u_j(s) - \hat{}$$

$$\hat{U}_\theta(s))$$

$$\leftarrow \theta_1 + \alpha(u_j(s) - \theta_1 \hat{U}_\theta(s))x$$

$$\leftarrow \theta_2 + \alpha(u_j(s) - \theta_2 \hat{U}_\theta(s))y$$

Assuming for an example $\theta =< \theta_0, \theta_1, \theta_2 >=< 0.5, 0.2, 0.1 >$ we get the

initial approximation $\hat{U}_\theta(1, 1) = 0.8$. If, after executing a trial, we computed eg. $u_j(1, 1) = 0.72$, then all the coefficients $\theta_0, \theta_1, \theta_2$ would be reduced by

0.08α, which in turn would decrease the error for the state (1,1). Obviously, all

the values of $\hat{U}_\theta(s)$ would then change, which is the idea of generalization.

APPLICATIONS IN GAME PLAYING AND ROBOT CONTROL

There are tremendous work on applying RL in Robotics. Readers are referred to [10] for a survey of RL in Robotics. In particular, [11] trained a robot to learn policies to map raw video images to robot's actions. The RGB images were fed to a CNN and outputs were the motor torques. The RL component was the guided policy search to generate training data that came from its own state distribution.

Web System Configuration

There are more than 100 configurable parameters in a web system and the process of tuning the parameters requires a skilled operator and numerous trail-and-error tests. The paper "A Reinforcement Learning Approach to Online Web System Auto-configuration" [5] showed the first attempt in the domain on how to do autonomic reconfiguration of parameters in multi-tier web systems in VM-based dynamic environments.

The reconfiguration process can be formulated as a finite MDP. The state space was the system configuration, action space was {increase, decrease, keep} for each parameter, and reward was defined as the difference between the given targeted response time and measured response time. The authors used the model-free Q-learning algorithm to do the task.

Although the authors used some other technique like policy initialization to remedy the large state space and computational complexity of the problem instead of the potential combinations of RL and neural network, it is believed that the pioneering work has paved the way for future research in this area.

General Game Playing. The annual AAAI GGP Competition [5] defines a general game player as a system that can understand the rules of any n-player game given in the general Game Description Language (GDL) and is able to play those games effectively. The functionality is illustrated in Fig. 1: A player must first accept any communicated game description (START message). After a given time period, and without human intervention, the player then makes his

opening move, accepts legal moves by other players (PLAY message) and continues to play until the game terminates.

Since the first AAAI competition in 2005, General Game Playing has evolved into a thriving AI research area. Established methods include Monte Carlo tree search [2], the automatic generation of evaluation functions [4] and knowledge acquisition [7]. Several complete general game-playing systems also have been described [12, 2]. Robotics. Building a robot that can recognise and manipulate objects in a physical environment is a complex and challenging problem. While basic robot kinematics is well understood, the ability to recognise and manipulate arbitrary objects in a complex environment remains an active area of research [9]. However, dealing with a predetermined set of rigid objects, such as common household items or game boards and pieces, is more amenable to known techniques and technologies [10, 6]. Consequently,

the research in domestic robotics provides an excellent platform for the development of general game playing robots that can play a wide variety of games.

Game descriptions must satisfy certain basic requirements to ensure that the game is effectively playable; for example, players should always have at least one legal move in nonterminal positions [5]. In bringing gameplay from mere virtual into physical environments, general game-playing robots add a new class of desirable properties that concern the manifestation of the game rules in the real world. Notably, a good GDL description requires all moves deemed legal by the rules to be actually physically possible.

In this section we develop a framework for mathematically describing the application of

game descriptions to physical environments that allows us to formalise such properties.

Environment Models. Consider the robotic game environment shown in Fig. 2(a). It features a 4 × 4 chess-like board with an additional row of 4 marked positions on the right. Tin cans are the only type of objects and can be moved between the marked positions (but cannot be stacked). To formally analyse the use of physical environments like this to play a game, we model them by state machines $(\Sigma, S, s0, \delta)$, where

– Σ denotes the actions that can be performed in the environment;
– S are the states the environment can be in, including the special failure $\in S$;
– $s0 \in S$ is the starting state;
– $\delta : S \times \Sigma \rightarrow S$ is the transition function.

Example. The following is a model for our cans-on-a-chessboard environment.

– We want to allow players two actions: doing nothing and moving a can. Formally, Σ = {noop} $\cup$ {move(u, v, x, y): u, x $\in$ {a, b, c, d, x}; v, y $\in$ {1, 2, 3, 4}} .
– Any location can either be empty or house a can. Hence, the environment can be in

any of 2 20 different states plus failure . Fig. 2(b) illustrates two example states.

– Any configurations of cans can be set up as the starting state s0.
– For the transition function δ , action noop has no effect, hence $\delta(s, noop) = s$.

Action move(u, v, x, y) maps any state with a can at (u, v) and no can at (x, y) to the same state except that now there is a can at (x, y) and none at (u, v). If no can is at (u, v) or there already is one at (x, y) , then $\delta(s, move(u, v, x, y)) =$ failure.

E.g., $s2 = \delta(move(d, 4, b, 2), \delta(move(x, 2, a, 1), s1))$ where s1 and s2 respectively denote the top and bottom states depicted in Fig. 2(b).

Projecting Games onto Physical Environments.

When we use a physical environment to play a game, the real objects become representatives of entities in the abstract game. A pawn in chess, for example, is typically manifested by an actual wooden piece of a certain shape and colour. But any other physical object, including a tin can, can serve the same purpose. Conversely, any game environment like our 4 × 4 board with cans can be interpreted in countless ways as physical manifestation of a game. Thus our example board can not only be used for all kinds of mini chess-like games (cf. top of
Fig. 2(c)) but also to play, say, the single-player 8-PUZZLE (cf. bottom of Fig. 2(c)). For this the cans represent numbered tiles that need to be brought in the right order.
The manifestation of a game in a physical environment can be mathematically captured by projecting the positions from the abstract game onto actual states of the game environment. general game player as the "brain" and any suitable robotic system as the "body."
The Robot Controller serves as the low-level executor of the robot. The required functionality is, (1) to process sensor data in order to recognise moves by detecting changes in the environment, and (2) to command the manipulator to execute any instance of the defined actions in the environment. Performing a single action may require a complex series of operations; for example, the sequence of operations required to move a can from one location to another. The robot controller needs to monitor the execution of such an action and, if possible, recover from failures whenever they occur.
The Game Controller provides the link between the low-level executor and the high-level decision making. It accepts any new GDL description sent through the interface and transmits these game rules to the general game-playing system in the form of a standard START message. During gameplay, the game controller passes on any move decided upon by the high-level controller to the low-level controller, and it converts opponent moves reported by the robot controller into standard PLAY messages for the general game player. The latter involves validating the legality of all opponent moves.

POSSIBLE QUESTIONS
PART-B (6 MARKS)

1. Explain in detail about passive reinforcement learning.
2. Describe about active reinforcement learning in detail
3. Write a short note on direct utility estimation.
4. Explain about adaptive dynamic programming and difference learning.
5. Describe about applications in game playing n detail.
6. Explain about applications in robot control.

PART-C (10 MARKS)

1. Write in detail about applications in robot control and game playing.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of CS,CA & IT

III B.Sc(CS) (BATCH 2016-2019)

Machine Learning

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS	ONE MARK QUESTIONS			
Questions	opt1	opt2	opt3	opt4
When connect is interrupted by a caught signal that is not restarted,we must call_____to wait for the connection to complete.	wait	select	write	READS
For the process calling WAIT,the_____gives us control over which process to wait and whether or not to block	waitpid	wait	select	none
If the client needs perform too many writes to the server before reading error from readline_____signal is sent to process.	SIGPIPE	SIG_IGN	SIGKILL	VC
When a process writes to a socket that has received an RST,_____signal is sent to the process.	EPIPE	SIGPIPE	SIGSTOP	none
When a unix system is shutdown the _____process normally sent the SIGTERM signal to call the process.	INIT	SIGKILL	select	Slash
_____is an optional flag when set,assistance call interrupt by the signal will be automatically restarted by the kernel.	SA_INTERRUPT	SIGALRM	SIG_RESTART	none
Posix allows us to specify a set of signal that will be _____when signal handler is called	delivered	blocked	started	none

Any signal that is blocked_____be delivered to the process.	blocked	can be	may be	Slash
compliment of SA_RESTART flag	SIGALRM	SA_START	blocked	none
If SA_INTERRUPT is defined,we set it if the signal being caught is	SIGALRM	SIGACTION	SIGSTOP	none
If a process terminate and the process has children in the zombie state,the parent process up all the zombie children is set to_____	0	1	-1	none
Whenever we fork the children,we must_____for them to parent them from becoming zombies.	LISTEN	WAIT	WRITE	none
We terminate the client by typing_____	EOF	Slash	/n	none
We used to term slow system call to discrete_____	return	terminate	accept	none
Function that we cannot restart ourself is ?	wait	connect	readline	write
We call the_____function to handle nthe terminate child.	wait	waitpid	connect	none
Serverhost crashed and there were no response atall to theclients datasegment,the error is	EHOSTUREACH	ENETUNREACH	ETIMEOUT	none
The server host was unreachable and respond with the ICMP ddestination unreachable message error is	ETIMEOUT	ECHOSTUNREACH	ECONNRESET	none
Our client is blocked in the call to read line when the__is received.	RST	CLR	ACK	none
Berkely delivered implementation retransmit the data segment_____times and waiting for around_____minute.	10,11	12,9	11,12	none
When a client is handling multiple descriptors, _____ is used	I/O Multiplexing	Signal Handling	I/O Demultiplexing	I/O Messaging

When a client is handling multiple socket at the same time, _____ is used	Signal Handling	I/O Multiplexing	I/O Demultiplexing	I/O Messaging
To handle both listening socket & its connected socket, a TCP server uses _____ .	I/O Demultiplexing	I/O Messaging	Signal Handling	I/O Multiplexing
If a server handles both TCP and UDP, _____ is used.	I/O Demultiplexing	I/O Messaging	I/O Multiplexing	Signal Handling
Capability of handling one or more I/O conditions is called _____	I/O Multiplexing	Signal Handling	I/O Demultiplexing	I/O Messaging
If a server handles multiple services, _____ is used.	Signal Handling	I/O Multiplexing	I/O Demultiplexing	I/O Messaging
If a server handles multiple protocols, _____ is used.	I/O Demultiplexing	I/O Messaging	Signal Handling	I/O Multiplexing
fcntl stands for what?	Signal Handling	function control	file corrupt	function corrupt
Which of the following is the IPv4 Socket option?	Signal Handling	ICMP6_FILTER	IPV6_HOPOPTS	TCP_KEEPALIVE
Which of the following is the IPv6 Socket option?	IP_RECVDSTADDR	ICMP6_FILTER	IPV6_HOPOPTS	TCP_KEEPALIVE
Which of the following is the TCP Socket option?	IP_RECVDSTADDR	ICMP6_FILTER	IPV6_HOPOPTS	TCP_KEEPALIVE
Every UDP socket has a _____ .	send buffer	TCP_KEEPALIVE	send buffer & receive buffer	None of the above
Every TCP socket has a _____ .	send buffer	receive buffer	send buffer & receive buffer	None of the above

Answers
select
waitpad
SIGPIPE
SIGPIPE
INIT
started
blocked

blocked
blocked
SIGALRM
1
WAIT
EOF
accept
connect
wait
ETIMEOUT
ECHOSTUNREACH
RST
12,9
I/O Multiplexing

Signal Handling
I/O Multiplexing
I/O Multiplexing
I/O Multiplexing
Signal Handling
I/O Multiplexing
Signal Handling
Signal Handling
IPV6_HOPOPTS
TCP_KEEPALIVE
TCP_KEEPALIVE
send buffer & receive buffer