



BHARATI VIDYAPEETH DEEMED UNIVERSITY
COLLEGE OF ENGINEERING, PUNE - 43



DEPARTMENT OF COMPUTER ENGINEERING

Lab Manual

Machine Learning

B.Tech. Computer Sem VIII

Name of Faculty: Sheetal S.Patil.

DEPARTMENT OF COMPUTER ENGINEERING

VISION OF THE INSTITUTE

“To be World Class Institute for Social Transformation through Dynamic Education”

MISSION OF THE INSTITUTE

- To provide quality technical education with advanced equipment, qualified faculty members, infrastructure to meet needs of profession and society.
- To provide an environment conducive to innovation, creativity, research and entrepreneurial leadership.
- To practice and promote professional ethics, transparency and accountability for social community, economic and environmental conditions.

VISION OF THE DEPARTMENT

“To pursue and excel in the endeavour for creating globally recognized computer engineers through quality education”.

MISSION OF THE DEPARTMENT

1. To impart fundamental knowledge and strong skills conforming to a dynamic curriculum leading to developing professional competencies and continuing intellectual growth.
2. To develop professional, entrepreneurial and research competencies encompassing continuing intellectual growth.
3. To strive for producing qualified graduates possessing proficient abilities and ethical responsibilities adapting to the demand of working environment.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates upon completion of B. Tech Computer Engineering Programme will be able to:

1. Exhibit Competence and professional skills pertinent to the working environment.
2. Continue professional Development through available resources and avenues for career growth.
3. Act with ethical and societal awareness as expected from competent practicing professionals.

PROGRAM OUTCOMES

1. To apply knowledge of computing and mathematics appropriate to the domain.
2. To logically define, analyse and solve real world problems
3. To apply design principles in developing hardware/software systems of varying complexity that meet the specified needs
4. To interpret and analyse data for providing solutions to complex engineering problems
5. To use and practice engineering and IT tools for professional growth
6. To understand and respond to legal and ethical issues involving the use of technology for societal benefits
7. To develop societal relevant projects using available resources
8. To exhibit professional and ethical responsibilities
9. To work effectively as an individual and a team member within the professional environment
10. To prepare and present technical documents using effective communication skills
11. To demonstrate effective leadership skills throughout the project management life cycle.
12. To understand the significance of lifelong learning for professional development

GENERAL INSTRUCTIONS:

- Equipment in the lab is meant for the use of students. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care.
- Students are required to carry their reference materials, files and records with completed assignment while entering the lab.
- Students are supposed to occupy the systems allotted to them and are not supposed to talk or make noise in the lab.
- All the students should perform the given assignment individually.
- Lab can be used in free time/lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- All the Students are instructed to carry their identity cards when entering the lab.
- Lab files need to be submitted on or before date of submission.
- Students are not supposed to use pen drives, compact drives or any other storage devices in the lab.
- For Laboratory related updates and assignments students should refer to the notice board in the Lab.

COURSE NAME: Machine Learning

WEEKLY PLAN:

Week No.	Practical/Assignment Name	Problem Definition
1	Learning Systems	Introduction to Learning Systems (Structure, Goals, Need, Applications, Examples).
2	Machine Learning techniques.	Introduction to Machine Learning techniques.
3	Machine Learning Models.	Study of various Machine Learning Models.
4	Decision Tree	Study and implement Decision Tree using R Programming.
5	Support Vector Machines	Study and implement Support Vector Machines using R Programming.
6	Linear Regression	What is Regression? Implement Linear Regression using R Programming.
7	Classification and Regression	Examine Classification and Regression. What are the issues regarding classification and regression.
8	Supervised and Unsupervised machine learning	Distinguish Supervised and Unsupervised machine learning.
9	K-Means clustering algorithm.	Study and implement K-Means clustering algorithm.
10	SCIKIT-LEARN, WEKA	Case study on SCIKIT-LEARN, WEKA tool for machine learning.
11	ACCORD, SHOGUN	Case study on ACCORD, SHOGUN tool for machine learning.

EXAMINATION SCHEME

Practical Exam: 25 Marks

Term Work: 25 Marks

Total: 50 Marks

Minimum Marks required: 20 Marks

PROCEDURE OF EVALUATION

Each practical/assignment shall be assessed continuously on the scale of 25 marks. The distribution of marks as follows.

Sr. No	Evaluation Criteria	Marks for each Criteria	Rubrics
1	Timely Submission	07	➤ Punctuality reflects the work ethics. Students should reflect that work ethics by completing the lab assignments and reports in a timely manner without being reminded or warned.
2	Presentation	06	➤ Student are expected to write the technical document (lab report) in their own words. The presentation of the contents in the lab report should be complete, unambiguous, clear, understandable. The report should document approach/algorithm/design and code with proper explanation.
3	Understanding	12	➤ Correctness and Robustness of the code is expected. The Learners should have an in-depth knowledge of the practical assignment performed. The learner should be able to explain methodology used for designing and developing the program/solution. He/she should clearly understand the purpose of the assignment and its outcome.

LABORATORY USAGE

Students use R Studio on computers for executing the lab experiments, document the results and to prepare the technical documents for making the lab reports.

OBJECTIVES:-

To develop ability to use the computational languages necessary for engineering practice.

PRACTICAL PRE-REQUISITE:-

1. Basic knowledge of Artificial intelligence, Discrete Mathematics, Database Management System, Engineering Mathematics.

HARDWARE/ SOFTWARE REQUIREMENTS:-

1. R Studio
2. WEKA/ACCORD etc.

COURSE OUTCOMES:

1. Explain significance of Machine Learning.
2. Distinguish between paradigms of Machine Learning.
3. Illustrate use of algorithms in Supervised Learning and Unsupervised Learning.
4. Build Learning Model.
5. Analyze performance of Supervised and Unsupervised Learning.
6. Tackle real world problems in the domain.

HOW OUTCOMES ARE ASSESSED?

Outcome	Assignment Number	Level	Proficiency evaluated by
Explain significance of Machine Learning	1,2,3,4,5	3,3,3,2,2	Problem definition & document the results.
Distinguish between paradigms of Machine Learning.	1,2,3,4,5,8	3,3,3,2,2,2	Problem definition & Performing Practical and reporting results.
Illustrate use of algorithms in Supervised Learning and Unsupervised Learning.	2,6,7,8	3,3,3,3	Performing experiments and reporting results.
Build Learning Model	2,3	2,3	Performing experiments and reporting results.
Analyze performance of Supervised and Unsupervised Learning	1,2,3,4,5,6,7,8,9	2,3,2,2,2,2,2,2,3,2	Performing experiments and reporting results.
Tackle real world problems in the domain.	4,5,6,9,10,11	3,3,3,3,3	Performing experiments and reporting results.

CONTRIBUTION TO PROGRAM OUTCOMES

Program outcomes	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2
Machine Learning	3		2				3		3	3	2	2	1	
	3	3	3				2	1	2	3	2	2		2
	3	3	2	3			2	2	3	2	1	2		
	3	2	2	3			3	3	3	1	1	3		
		2						2	3	3	3	3		
	3	3		3			2	2	3	3	2	3		3

DESIGN EXPERIENCE GAINED

The students gain design experience by understanding basics of machine learning and implement the algorithms in R Programming.

LEARNING EXPERIENCE GAINED

The students learn both soft skills and technical skills while they are undergoing the practical sessions. The soft skills gained in terms of communication, presentation and behaviour.

While technical skills they gained in terms of programming and efficient algorithm design using data structures.

DESIGN EXPERIENCE GAINED

The students learn algorithms of data analytics and using R convert that algorithm to executable code.

LEARNING EXPERIENCE GAINED

The students learn both soft skills and technical skills while they are undergoing the practical sessions. The soft skills gained in terms of communication, presentation and behavior. While technical skills they gained in terms of programming and efficient algorithm design using concepts of data analytics.

LIST OF PRACTICAL ASSIGNMENTS:

No	Detail of Assignment
1.	Introduction to Learning Systems (Structure, Goals, Need, Applications, Examples).
2.	Introduction to Machine Learning techniques.
3.	Study of various Machine Learning Models.
4.	Study and implement Decision Tree using R Programming.
5.	Study and implement Support Vector Machines using R Programming.
6.	What is Regression? Implement Linear Regression using R Programming.
7.	Examine Classification and Regression. What are the issues regarding classification and regression.
8.	Distinguish Supervised and Unsupervised machine learning.
9.	Study and implement K-Means clustering algorithm.
10.	Case study on SCIKIT-LEARN, WEKA tool for machine learning.
11.	Case study on ACCORD, SHOGUN tool for machine learning.

ASSIGNMENT 1 :

AIM : Introduction to Learning Systems

THEORY :

Learning Systems :

A learning system can be loosely defined as a system which can adapt its behaviour to become more effective at a particular task or set of tasks. Generally learning systems are taught using a set of training examples presented by an expert or teacher, and are called supervised learning systems. In addition there are autonomous or unsupervised learning systems which change their behaviour automatically on the basis of natural clusters within the training data. A learning system consists of two components: an architecture with a set of variable parameters and an algorithm to update these parameters in order to improve the performance of the overall system. The system must therefore have some memory to retain the current values of its parameters. The learning algorithm needs a measure of the accuracy (or error) at the system output to be able to determine how the parameters should be changed to improve the system, this measure is called the performance index.

Learning systems are useful in many fields, one of the major areas is in control and system identification. They can be used to form automatically a model of a system which can be employed in fault diagnosis or to develop a controller in an indirect control scheme. Alternatively the parameters of a controller can be updated automatically to form a direct adaptive control system. They also have applications in cognitive science where they can be used as pattern recognition devices or as classifiers.

Goals of Learning Systems :

1. **Generalization** : A core objective of a learner is to generalize from its experience. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning data set. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a

general model about this space that enables it to produce sufficiently accurate predictions in new cases.

2. **Analysis of the generalized idea** : The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common.
3. **Time complexity and feasibility of learning** : In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time.

Structure of Learning Systems :

Representation : Representation is, what exactly is the classifier? Is it a rule like the ones we saw with the golf example? A simple hypothesis of maybe when it's sunny, we always play golf,? The space of all possible such hypotheses, all possible such rules, is the representation of this classifier, okay, or each one of those is perhaps the representation. Or is it the space of all possible neural networks that map inputs to outputs, right? Or is it a decision tree which we'll talk about in a little bit? Or if it's more numerical data, if you've got a 2D scatter plot and you try to draw a line that precisely separates the data such that all the positive examples are on one side of the line and all the negative examples are on another side of the line, the space of all possible such lines is the representation here.

Evaluation : So, once we have that figured out, how do I judge whether one particular example of this instance of this representation is effective or not? How do I know the good ones from the bad ones? So, for example, with the rules of when you're trying to play golf, I've got a rule that says when it's sunny we play golf and when it's rainy and windy we don't. Those are two different rules. How do I judge which one's better? You have to make some sort of a choice about how you're going to evaluate this thing. And it could be just the number of errors it makes on some given test set. Or you could define some notion of precision and recall. Or, if it's more of a regression, numerical case, you could talk about the absolute error or the squared error or other

variance. Right, but you need to make some sort of a decision here. So now we have a representation, we have a space of possible units, and now we have evaluation method selected where we can take a given instance and determine whether it was any good or not, fine.

Optimization : The third piece here is optimization. So how do you search among this potentially massive space or potentially infinite space of possible hypotheses. Okay, so in pretty much no case is it going to be feasible to enumerate all possible instances and apply your evaluation technique to find the best one. So you're going to have to have some method of searching through this efficiently. Okay. Now what's useful, I think, about this breakdown is that it helps make sense of the various terms and methods you'll hear if you sort of scan the machine learning literature, is that what will be being presented is just an optimization technique that can apply to a variety of different representations and even evaluations. And you can plug in your own evaluation and plug in your own representation and it'll still work. Other times you're talking about a representation that is amenable to a variety of different optimization strategies or particular algorithms.

Need of Learning Systems :

Resurging interest in machine learning is due to the same factors that have made data mining and Bayesian analysis more popular than ever. Things like growing volumes and varieties of available data, computational processing that is cheaper and more powerful, and affordable data storage.

All of these things mean it's possible to quickly and automatically produce models that can analyze bigger, more complex data and deliver faster, more accurate results – even on a very large scale. And by building precise models, an organization has a better chance of identifying profitable opportunities – or avoiding unknown risks.

What's required to create good machine learning systems?

- Data preparation capabilities.
- Algorithms – basic and advanced.
- Automation and iterative processes.

- Scalability.
- Ensemble modelling.

Machine learning—and AI in general—has been around for a while. But it's recently started accelerating at a rate that's surprised a lot of people.

As recently as 2014, most experts thought it would be 10 years before a machine beat the world's best players at Go. DeepMind proved them wrong. It's becoming increasingly apparent that many tasks we once thought would be the domain of humans alone for the foreseeable future—if not forever—will be accomplished by machine learning systems much sooner than expected.

Applications of Learning Systems :

Most industries working with large amounts of data have recognized the value of machine learning technology. By gleaning insights from this data – often in real time – organizations are able to work more efficiently or gain an advantage over competitors.

Financial services

Banks and other businesses in the financial industry use machine learning technology for two key purposes: to identify important insights in data, and prevent fraud. The insights can identify investment opportunities, or help investors know when to trade. Data mining can also identify clients with high-risk profiles, or use cybersurveillance to pinpoint warning signs of fraud.

Government

Government agencies such as public safety and utilities have a particular need for machine learning since they have multiple sources of data that can be mined for insights. Analyzing sensor data, for example, identifies ways to increase efficiency and save money. Machine learning can also help detect fraud and minimize identity theft.

Health care

Machine learning is a fast-growing trend in the health care industry, thanks to the advent of wearable devices and sensors that can use data to assess a patient's health in real time. The technology can also help medical experts analyze data to identify trends or red flags that may lead to improved diagnoses and treatment.

Marketing and sales

Websites recommending items you might like based on previous purchases are using machine learning to analyze your buying history – and promote other items you'd be interested in. This ability to capture data, analyze it and use it to personalize a shopping experience (or implement a marketing campaign) is the future of retail.

Oil and gas

Finding new energy sources. Analyzing minerals in the ground. Predicting refinery sensor failure. Streamlining oil distribution to make it more efficient and cost-effective. The number of machine learning use cases for this industry is vast – and still expanding.

Transportation

Analyzing data to identify patterns and trends is key to the transportation industry, which relies on making routes more efficient and predicting potential problems to increase profitability. The data analysis and modelling aspects of machine learning are important tools to delivery companies, public transportation and other transportation organizations.

Examples :

In 2006, the online movie company Netflix held the first "Netflix Prize" competition to find a program to better predict user preferences and improve the accuracy on its existing Cinematch movie recommendation algorithm by at least 10%. A joint team made up of researchers from AT&T Labs-Research in collaboration with the teams Big Chaos and Pragmatic Theory built an ensemble model to win the Grand Prize in

2009 for \$1 million .Shortly after the prize was awarded, Netflix realized that viewers' ratings were not the best indicators of their viewing patterns ("everything is a recommendation") and they changed their recommendation engine accordingly.

In 2010 The Wall Street Journal wrote about the firm Rebellion Research and their use of Machine Learning to predict the financial crisis.

In 2012, co-founder of Sun Microsystems Vinod Khosla predicted that 80% of medical doctors jobs would be lost in the next two decades to automated machine learning medical diagnostic software.

In 2014, it has been reported that a machine learning algorithm has been applied in Art History to study fine art paintings, and that it may have revealed previously unrecognized influences between artists.

Questions:

1. What is learning system?
2. What is machine learning?
3. What is testing, training, designing etc?
4. Goals, Need, Applications of machine learning.

ASSIGNMENT 2:

AIM: Introduction to Machine Learning Techniques

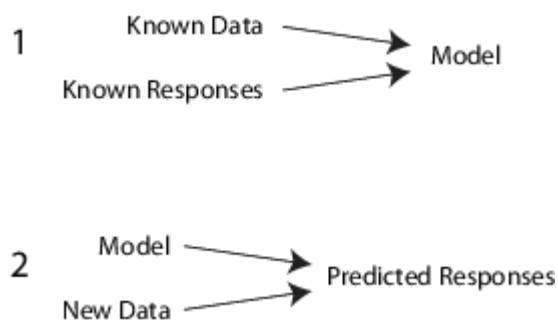
THEORY:

Machine learning – subfield of computer science (more particularly soft computing) that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. In 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed". Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from an example *training set* of input observations in order to make data-driven predictions or decisions expressed as outputs, rather than following strictly static program instructions.

What is Supervised Learning?

The aim of supervised, machine learning is to build a model that makes predictions based on evidence in the presence of uncertainty. As adaptive algorithms identify patterns in data, a computer "learns" from the observations. When exposed to more observations, the computer improves its predictive performance.

Specifically, a supervised learning algorithm takes a known set of input data and known responses to the data (output), and *trains* a model to generate reasonable predictions for the response to new data.



For example, suppose you want to predict whether someone will have a heart attack within a year. You have a set of data on previous patients, including age, weight, height, blood pressure, etc. You know whether the previous patients had heart attacks within a year of their measurements. So, the problem is combining all the existing data into a model that can predict whether a new person will have a heart attack within a year.

You can think of the entire set of input data as a heterogeneous matrix. Rows of the matrix are called *observations*, *examples*, or *instances*, and each contain a set of measurements for a subject (patients in the example). Columns of the matrix are called *predictors*, *attributes*,

or *features*, and each are variables representing a measurement taken on every subject (age, weight, height, etc. in the example). You can think of the response data as a column vector where each row contains the output of the corresponding observation in the input data (whether the patient had a heart attack). To *fit* or *train* a supervised learning model, choose an appropriate algorithm, and then pass the input and response data to it.

Unsupervised Machine Learning

Unsupervised learning is where you only have input data (X) and no corresponding output variables.

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

These are called unsupervised learning because unlike supervised learning above there are no correct answers and there is no teacher. Algorithms are left to their own devices to discover and present the interesting structure in the data.

Unsupervised learning problems can be further grouped into clustering and association problems.

- **Clustering:** A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association:** An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y. Some popular examples of unsupervised learning algorithms are:
 - k-means for clustering problems.
 - Apriori algorithm for association rule learning problems.

Semi-Supervised Machine Learning

Problems where you have a large amount of input data (X) and only some of the data is labeled (Y) are called semi-supervised learning problems.

These problems sit in between both supervised and unsupervised learning.

A good example is a photo archive where only some of the images are labeled, (e.g. dog, cat, person) and the majority are unlabeled.

Many real world machine learning problems fall into this area. This is because it can be expensive or time-consuming to label data as it may require access to domain experts. Whereas unlabeled data is cheap and easy to collect and store.

You can use unsupervised learning techniques to discover and learn the structure in the input variables.

You can also use supervised learning techniques to make best guess predictions for the unlabeled data, feed that data back into the supervised learning algorithm as training data and use the model to make predictions on new unseen data.

List of Common Machine Learning Algorithms

Here is the list of commonly used machine learning algorithms. These algorithms can be applied to almost any data problem:

- Linear Regression
- Logistic Regression
- Decision Tree
- SVM
- Naive Bayes
- KNN
- K-Means
- Random Forest
- Dimensionality Reduction Algorithms
- Gradient Boosting algorithms
 - GBM
 - XGBoost
 - LightGBM
- CatBoost

1. Linear Regression

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation $Y = a * X + b$.

The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade to arrange people in his class by increasing order of weight, without asking them their weights! What do you think the child will do? He / she would likely look (visually analyze) at the height and build of people and

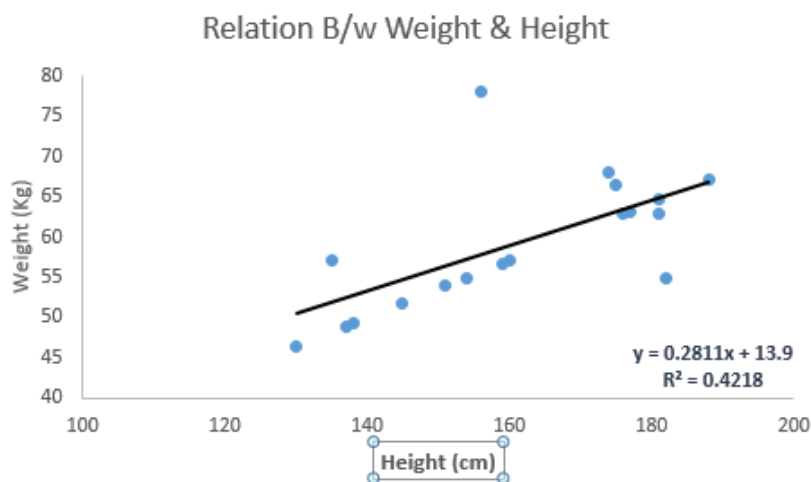
arrange them using a combination of these visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to the weight by a relationship, which looks like the equation above.

In this equation:

- Y – Dependent Variable
- a – Slope
- X – Independent variable
- b – Intercept

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.

Look at the below example. Here we have identified the best fit line having linear equation $y=0.2811x+13.9$. Now using this equation, we can find the weight, knowing the height of a person.



Linear Regression is of mainly two types: Simple Linear Regression and Multiple Linear Regression. Simple Linear Regression is characterized by one independent variable. And, Multiple Linear Regression(as the name suggests) is characterized by multiple (more than 1) independent variables. While finding best fit line, you can fit a polynomial or curvilinear regression. And these are known as polynomial or curvilinear regression.

Python Code

```
#Import Library

#Import other necessary libraries like pandas, numpy...

from sklearn import linear_model

#Load Train and Test datasets

#Identify feature and response variable(s) and values must be numeric and numpy
arrays

x_train=input_variables_values_training_datasets

y_train=target_variables_values_training_datasets

x_test=input_variables_values_test_datasets

# Create linear regression object

linear = linear_model.LinearRegression()

# Train the model using the training sets and check score

linear.fit(x_train, y_train)

linear.score(x_train, y_train)

#Equation coefficient and Intercept

print('Coefficient: \n', linear.coef_)
```



```
print('Intercept: \n', linear.intercept_)
```

```
#Predict Output
```

```
predicted= linear.predict(x_test)
```

R Code

```
#Load Train and Test datasets
```

```
#Identify feature and response variable(s) and values must be numeric and numpy  
arrays
```

```
x_train <- input_variables_values_training_datasets
```

```
y_train <- target_variables_values_training_datasets
```

```
x_test <- input_variables_values_test_datasets
```

```
x <- cbind(x_train,y_train)
```

```
# Train the model using the training sets and check score
```

```
linear <- lm(y_train ~ ., data = x)
```

```
summary(linear)
```

```
#Predict Output
```

```
predicted= predict(linear,x_test)
```

2. Logistic Regression

It is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as **logit regression**. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

Again, let us try and understand this through a simple example.

Let's say your friend gives you a puzzle to solve. There are only 2 outcome scenarios – either you solve it or you don't. Now imagine, that you are being given wide range of puzzles / quizzes in an attempt to understand which subjects you are good at. The outcome to this study would be something like this – if you are given a trigonometry based tenth grade problem, you are 70% likely to solve it. On the other hand, if it is grade fifth history question, the probability of getting an answer is only 30%. This is what Logistic Regression provides you.

Coming to the math, the log odds of the outcome is modelled as a linear combination of the predictor variables.

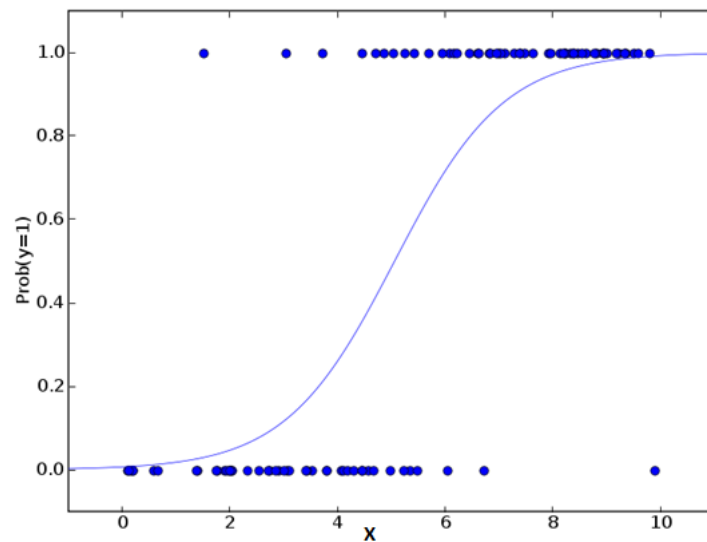
$$\text{odds} = p / (1-p) = \text{probability of event occurrence} / \text{probability of not event occurrence}$$

$$\ln(\text{odds}) = \ln(p/(1-p))$$

$$\text{logit}(p) = \ln(p/(1-p)) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \dots + b_kX_k$$

Above, p is the probability of presence of the characteristic of interest. It chooses parameters that maximize the likelihood of observing the sample values rather than that minimize the sum of squared errors (like in ordinary regression).

Now, you may ask, why take a log? For the sake of simplicity, let's just say that this is one of the best mathematical way to replicate a step function. I can go in more details, but that will beat the purpose of this article.



Python Code

```
#Import Library

from sklearn.linear_model import LogisticRegression

#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset

# Create logistic regression object

model = LogisticRegression()

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Equation coefficient and Intercept

print('Coefficient: \n', model.coef_)
```

```
print('Intercept: \n', model.intercept_)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
x <- cbind(x_train,y_train)

# Train the model using the training sets and check score

logistic <- glm(y_train ~ ., data = x,family='binomial')

summary(logistic)

#Predict Output

predicted= predict(logistic,x_test)
```

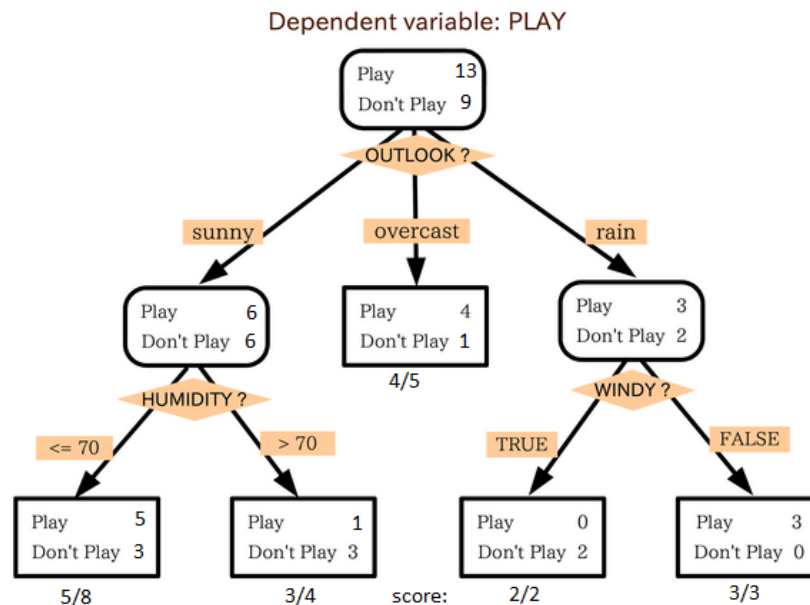
3. Decision Tree

Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression).

Methods like decision trees, random forest, gradient boosting are being popularly used in all kinds of data science problems.

It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous

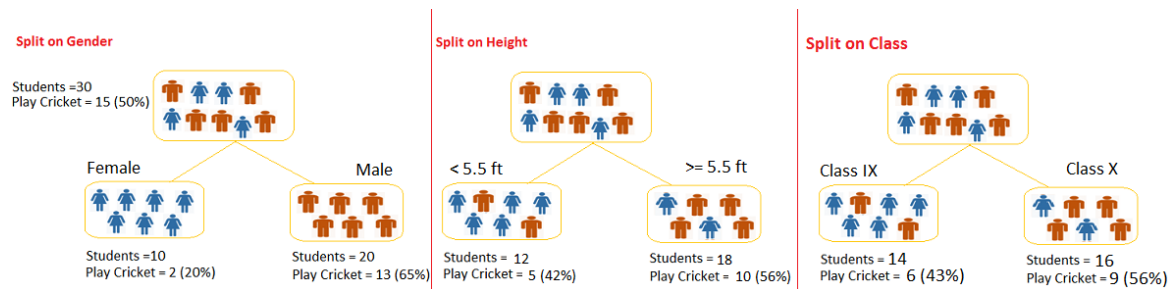
dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible.



In the image above, you can see that population is classified into four different groups based on multiple attributes to identify 'if they will play or not'. To split the population into different heterogeneous groups, it uses various techniques like Gini, Information Gain, Chi-square, entropy.

Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class(IX/ X) and Height (5 to 6 ft). 15 out of these 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.

This is where decision tree helps, it will segregate the students based on all values of three variable and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other). In the snapshot below, you can see that variable Gender is able to identify best homogeneous sets compared to the other two variables.



As mentioned above, decision tree identifies the most significant variable and its value that gives best homogeneous sets of population. Now the question which arises is, how does it identify the variable and the split? To do this, decision tree uses various algorithms, which we will shall discuss in the following section.

Types of Decision Trees

Types of decision tree is based on the type of target variable we have. It can be of two types:

1. **Categorical Variable Decision Tree:** Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example:- In above scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.
2. **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Python Code

```
#Import Library

#Import other necessary libraries like pandas, numpy...

from sklearn import tree

#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset
```



```
# Create tree object

model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here
you can change the algorithm as gini or entropy (information gain) by default it
is gini

# model = tree.DecisionTreeRegressor() for regression

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
library(rpart)

x <- cbind(x_train,y_train)

# grow tree

fit <- rpart(y_train ~ ., data = x,method="class")

summary(fit)

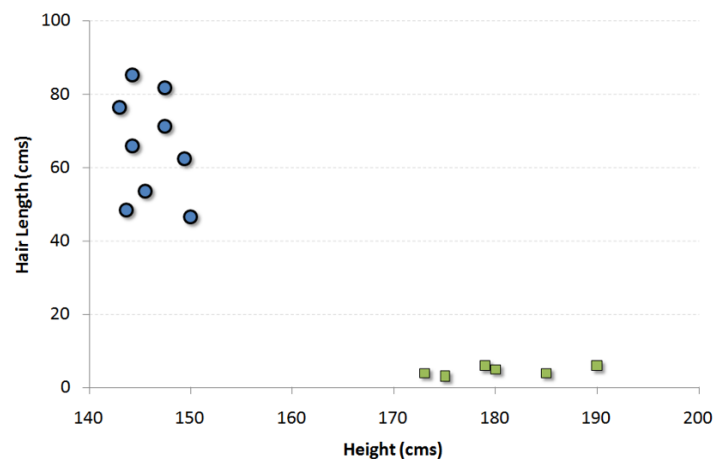
#Predict Output
```

```
predicted= predict(fit,x_test)
```

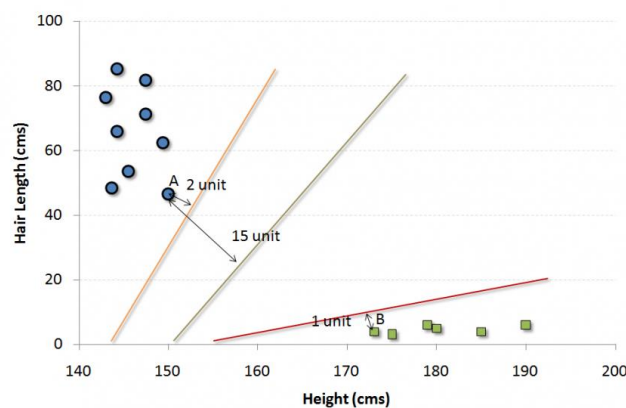
4. SVM (Support Vector Machine)

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

For example, if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**)



Now, we will find some *line* that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.



In the example shown above, the line which splits the data into two differently classified groups is the *black* line, since the two closest points are the farthest apart from the line. This line is our classifier. Then, depending on where the testing data lands on either side of the line, that's what class we can classify the new data as.

Think of this algorithm as playing JezzBall in n-dimensional space. The tweaks in the game are:

- You can draw lines / planes at any angles (rather than just horizontal or vertical as in classic game)
- The objective of the game is to segregate balls of different colors in different rooms.
- And the balls are not moving.

Python Code

```
#Import Library

from sklearn import svm

#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset

# Create SVM classification object

model = svm.svc() # there is various option associated with it, this is simple for
classification. You can refer link, for more detail.

# Train the model using the training sets and check score

model.fit(X, y)
```

```
model.score(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
library(e1071)

x <- cbind(x_train,y_train)

# Fitting model

fit <-svm(y_train ~ ., data = x)

summary(fit)

#Predict Output

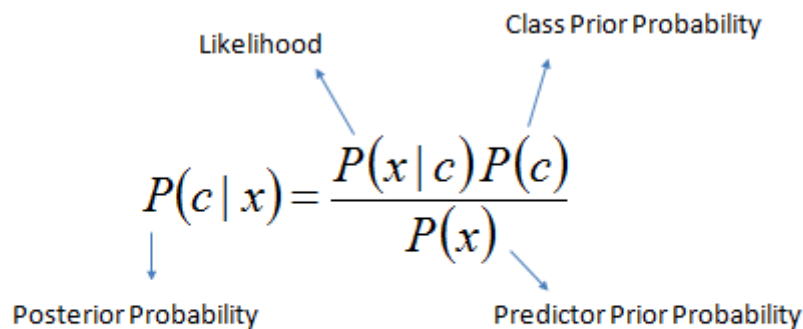
predicted= predict(fit,x_test)
```

5. Naive Bayes

It is a classification technique based on Bayes' theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

Naive Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability $P(c|x)$ from $P(c)$, $P(x)$ and $P(x|c)$. Look at the equation below:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$


$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Here,

- $P(c|x)$ is the posterior probability of *class (target)* given *predictor (attribute)*.
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

Example: Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play'. Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set to frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
All	5	9
	=5/14	=9/14
	0.36	0.64

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Problem: Players will play if weather is sunny, is this statement is correct?

We can solve it using above discussed method, so $P(\text{Yes} | \text{Sunny}) = P(\text{Sunny} | \text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$

Here we have $P(\text{Sunny} | \text{Yes}) = 3/9 = 0.33$, $P(\text{Sunny}) = 5/14 = 0.36$, $P(\text{Yes}) = 9/14 = 0.64$

Now, $P(\text{Yes} | \text{Sunny}) = 0.33 * 0.64 / 0.36 = 0.60$, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

Python Code

```
#Import Library

from sklearn.naive_bayes import GaussianNB

#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset
```



```
# Create SVM classification object model = GaussianNB() # there is other
distribution for multinomial classes like Bernoulli Naive Bayes, Refer link

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
library(e1071)

x <- cbind(x_train,y_train)

# Fitting model

fit <-naiveBayes(y_train ~ ., data = x)

summary(fit)

#Predict Output

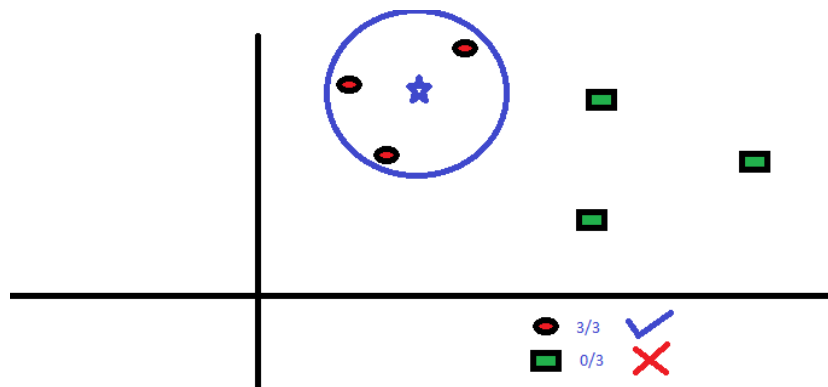
predicted= predict(fit,x_test)
```

6. KNN (K- Nearest Neighbors)

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The

case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If $K = 1$, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing KNN modeling.



KNN can easily be mapped to our real lives. If you want to learn about a person, of whom you have no information, you might like to find out about his close friends and the circles he moves in and gain access to his/her information!

Things to consider before selecting KNN:

- KNN is computationally expensive
- Variables should be normalized else higher range variables can bias it
- Works on pre-processing stage more before going for KNN like outlier, noise removal

Python Code

```
#Import Library

from sklearn.neighbors import KNeighborsClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset

# Create KNeighbors classifier object model
```

```
KNeighborsClassifier(n_neighbors=6) # default value for n_neighbors is 5

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
library(knn)

x <- cbind(x_train,y_train)

# Fitting model

fit <-knn(y_train ~ ., data = x,k=5)

summary(fit)

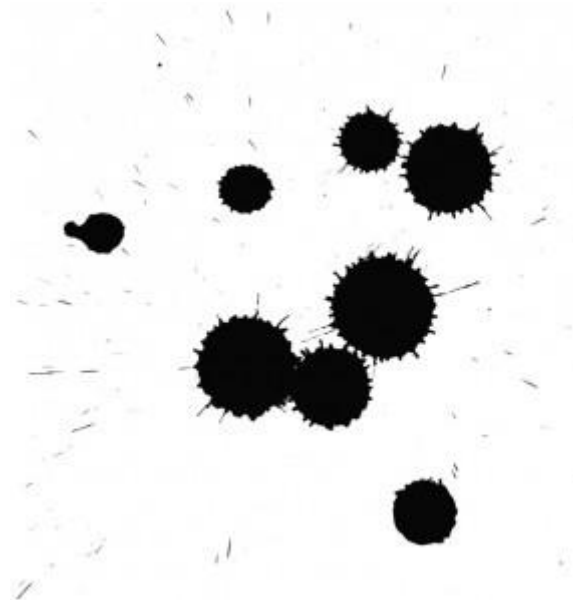
#Predict Output

predicted= predict(fit,x_test)
```

7. K-Means

It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups.

Remember figuring out shapes from ink blots? k means is somewhat similar this activity. You look at the shape and spread to decipher how many different clusters / population are present!



How K-means forms cluster:

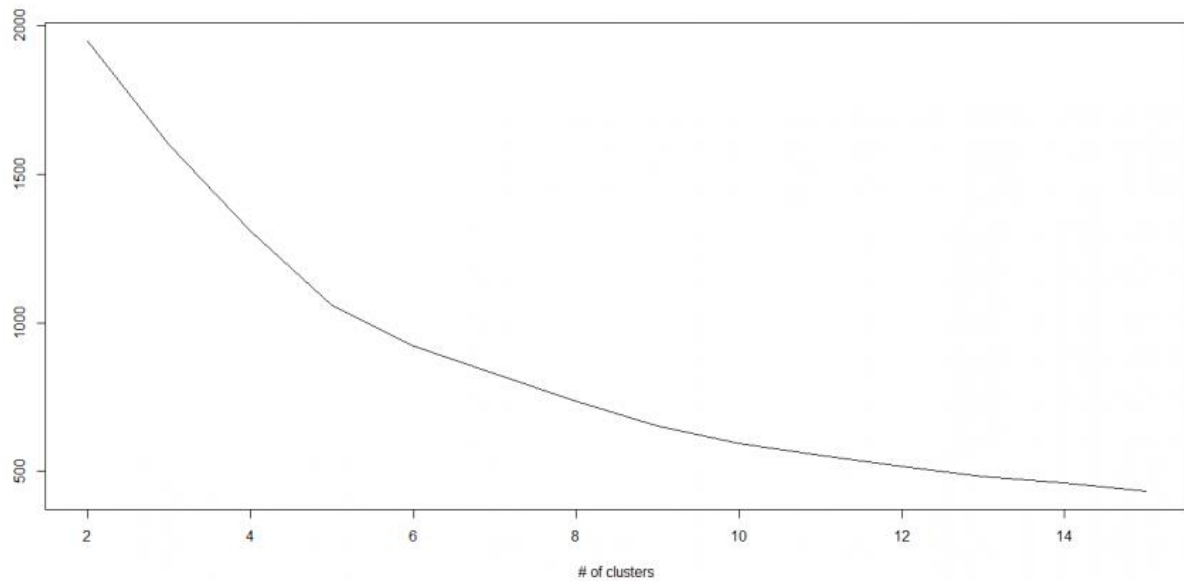
1. K-means picks k number of points for each cluster known as centroids.
2. Each data point forms a cluster with the closest centroids i.e. k clusters.
3. Finds the centroid of each cluster based on existing cluster members. Here we have new centroids.
4. As we have new centroids, repeat step 2 and 3. Find the closest distance for each data point from new centroids and get associated with new k-clusters. Repeat this process until convergence occurs i.e. centroids does not change.

How to determine value of K:

In K-means, we have clusters and each cluster has its own centroid. Sum of square of difference between centroid and the data points within a cluster constitutes within sum of square value for that cluster. Also, when the sum of square values for all the clusters are added, it becomes total within sum of square value for the cluster solution.

We know that as the number of cluster increases, this value keeps on decreasing but if you plot the result you may see that the sum of squared distance decreases sharply up

to some value of k, and then much more slowly after that. Here, we can find the optimum number of cluster.



Python Code

```
#Import Library

from sklearn.cluster import KMeans

#Assumed you have, X (attributes) for training data set and x_test(attributes) of
test_dataset

# Create KNeighbors classifier object model

k_means = KMeans(n_clusters=3, random_state=0)

# Train the model using the training sets and check score

model.fit(X)

#Predict Output
```

```
predicted= model.predict(x_test)
```

R Code

```
library(cluster)
```

```
fit <- kmeans(X, 3) # 5 cluster solution
```

8. Random Forest

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is planted & grown as follows:

1. If the number of cases in the training set is N, then sample of N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.
2. If there are M input variables, a number $m \ll M$ is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

Python

```
#Import Library
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
#Assumed you have, X (predictor) and Y (target) for training data set and
```

```
x_test(predictor) of test_dataset
```

```
# Create Random Forest object
```

```
model= RandomForestClassifier()

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
library(randomForest)

x <- cbind(x_train,y_train)

# Fitting model

fit <- randomForest(Species ~ ., x,ntree=500)

summary(fit)

#Predict Output

predicted= predict(fit,x_test)
```

9. Dimensionality Reduction Algorithms

In the last 4-5 years, there has been an exponential increase in data capturing at every possible stages. Corporates/ Government Agencies/ Research organisations are not only coming with new sources but also they are capturing data in great detail.

For example: E-commerce companies are capturing more details about customer like their demographics, web crawling history, what they like or dislike, purchase history, feedback and many others to give them personalized attention more than your nearest grocery shopkeeper.

As a data scientist, the data we are offered also consist of many features, this sounds good for building good robust model but there is a challenge. How'd you identify highly significant variable(s) out 1000 or 2000? In such cases, dimensionality reduction algorithm helps us along with various other algorithms like Decision Tree, Random Forest, PCA, Factor Analysis, Identify based on correlation matrix, missing value ratio and others.

Python Code

```
#Import Library

from sklearn import decomposition

#Assumed you have training and test data set as train and test

# Create PCA object pca= decomposition.PCA(n_components=k) #default value of k
=min(n_sample, n_features)

# For Factor analysis

#fa= decomposition.FactorAnalysis()

# Reduced the dimension of training dataset using PCA

train_reduced = pca.fit_transform(train)

#Reduced the dimension of test dataset

test_reduced = pca.transform(test)
```


#For more detail on this, please refer [this link](#).

R Code

```
library(stats)

pca <- princomp(train, cor = TRUE)

train_reduced <- predict(pca,train)

test_reduced <- predict(pca,test)
```

10. Gradient Boosting Algorithms

10.1. GBM

GBM is a boosting algorithm used when we deal with plenty of data to make a prediction with high prediction power. Boosting is actually an ensemble of learning algorithms which combines the prediction of several base estimators in order to improve robustness over a single estimator. It combines multiple weak or average predictors to build a strong predictor. These boosting algorithms always work well in data science competitions like Kaggle, AV Hackathon, CrowdAnalytix.

More: [Know about Boosting algorithms in detail](#)

Python Code

```
#Import Library

from sklearn.ensemble import GradientBoostingClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset
```

```

# Create Gradient Boosting Classifier object

model= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
max_depth=1, random_state=0)

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)

```

R Code

```

library(caret)

x <- cbind(x_train,y_train)

# Fitting model

fitControl <- trainControl( method = "repeatedcv", number = 4, repeats = 4)

fit <- train(y ~ ., data = x, method = "gbm", trControl = fitControl,verbose =
FALSE)

predicted= predict(fit,x_test,type= "prob")[,2]

```

Gradient Boosting Classifier and Random Forest are two different boosting tree classifier and often people ask about the [difference between these two algorithms](#).

10.2. XGBoost

Another classic gradient boosting algorithm that's known to be the decisive choice between winning and losing in some Kaggle competitions.

The XGBoost has an immensely high predictive power which makes it the best choice for accuracy in events as it possesses both linear model and the tree learning algorithm, making the algorithm almost 10x faster than existing gradient booster techniques.

The support includes various objective functions, including regression, classification and ranking.

One of the most interesting things about the XGBoost is that it is also called a regularized boosting technique. This helps to reduce overfit modelling and has a massive support for a range of languages such as Scala, Java, R, Python, Julia and C++.

Supports distributed and widespread training on many machines that encompass GCE, AWS, Azure and Yarn clusters. XGBoost can also be integrated with Spark, Flink and other cloud dataflow systems with a built in cross validation at each iteration of the boosting process.

```
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

X = dataset[:,0:10]

Y = dataset[:,10:]

seed = 1
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,  
random_state=seed)  
  
model = XGBClassifier()  
  
model.fit(X_train, y_train)  
  
#Make predictions for test data  
  
y_pred = model.predict(X_test)
```

R Code:

```
require(caret)  
  
x <- cbind(x_train,y_train)  
  
# Fitting model  
  
TrainControl <- trainControl( method = "repeatedcv", number = 10, repeats = 4)  
  
model<- train(y ~ ., data = x, method = "xgbLinear", trControl =  
TrainControl,verbose = FALSE)  
  
OR  
  
model<- train(y ~ ., data = x, method = "xgbTree", trControl =  
TrainControl,verbose = FALSE)  
  
predicted <- predict(model, x_test)
```

10.3. LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Parallel and GPU learning supported
- Capable of handling large-scale data

The framework is a fast and high-performance gradient boosting one based on decision tree algorithms, used for ranking, classification and many other machine learning tasks. It was developed under the Distributed Machine Learning Toolkit Project of Microsoft.

Since the LightGBM is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.

Python Code:

```
data = np.random.rand(500, 10) # 500 entities, each contains 10 features

label = np.random.randint(2, size=500) # binary target

train_data = lgb.Dataset(data, label=label)

test_data = train_data.create_valid('test.svm')

param = {'num_leaves':31, 'num_trees':100, 'objective':'binary'}

param['metric'] = 'auc'

num_round = 10
```

```
bst = lgb.train(param, train_data, num_round, valid_sets=[test_data])
```

```
bst.save_model('model.txt')
```

```
# 7 entities, each contains 10 features
```

```
data = np.random.rand(7, 10)
```

```
ypred = bst.predict(data)
```

R Code:

```
library(RLightGBM)
```

```
data(example.binary)
```

```
#Parameters
```

```
num_iterations <- 100
```

```
config <- list(objective = "binary", metric="binary_logloss, auc", learning_rate = 0.1, num_leaves = 63,  
tree_learner = "serial", feature_fraction = 0.8, bagging_freq = 5, bagging_fraction = 0.8, min_data_in_leaf = 50,  
min_sum_hessian_in_leaf = 5.0)
```

```
#Create data handle and booster
```

```
handle.data <- lgbm.data.create(x)
```

```
lgbm.data.setField(handle.data, "label", y)
```

```
handle.booster <- lgbm.booster.create(handle.data, lapply(config, as.character))
```

```
#Train for num_iterations iterations and eval every 5 steps

lgbm.booster.train(handle.booster, num_iterations, 5)

#Predict

pred <- lgbm.booster.predict(handle.booster, x.test)

#Test accuracy

sum(y.test == (y.pred > 0.5)) / length(y.test)

#Save model (can be loaded again via lgbm.booster.load(filename))

lgbm.booster.save(handle.booster, filename = "/tmp/model.txt")
```

If you're familiar with the Caret package in R, this is another way of implementing the LightGBM.

```
require(caret)

require(RLightGBM)

data(iris)

model <- caretModel.LGBM()

fit <- train(Species ~ ., data = iris, method=model, verbosity = 0)

print(fit)

y.pred <- predict(fit, iris[,1:4])
```

```
library(Matrix)

model.sparse <- caretModel.LGBM.sparse()

#Generate a sparse matrix

mat <- Matrix(as.matrix(iris[,1:4]), sparse = T)

fit <- train(data.frame(idx = 1:nrow(iris)), iris$Species, method = model.sparse, matrix = mat, verbosity = 0)

print(fit)
```

10.4. Catboost

CatBoost is a recently open-sourced machine learning algorithm from Yandex. It can easily integrate with deep learning frameworks like Google's TensorFlow and Apple's Core ML.

The best part about CatBoost is that it does not require extensive data training like other ML models, and can work on a variety of data formats; not undermining how robust it can be.

Make sure you handle missing data well before you proceed with the implementation.

Catboost can automatically deal with categorical variables without showing the type conversion error, which helps you to focus on tuning your model better rather than sorting out trivial errors.

Python Code:

```
import pandas as pd

import numpy as np
```



```

from catboost import CatBoostRegressor

#Read training and testing files

train = pd.read_csv("train.csv")

test = pd.read_csv("test.csv")

#Imputing missing values for both train and test

train.fillna(-999, inplace=True)

test.fillna(-999,inplace=True)

#Creating a training set for modeling and validation set to check model performance

X = train.drop(['Item_Outlet_Sales'], axis=1)

y = train.Item_Outlet_Sales

from sklearn.model_selection import train_test_split

X_train, X_validation, y_train, y_validation = train_test_split(X, y, train_size=0.7,
random_state=1234)

categorical_features_indices = np.where(X.dtypes != np.float)[0]

#importing library and building model

from catboost import CatBoostRegressor
model=CatBoostRegressor(iterations=50, depth=3,
learning_rate=0.1, loss_function='RMSE')

```

```
model.fit(X_train, y_train, cat_features=categorical_features_indices, eval_set=(X_validation,
y_validation), plot=True)
```

```
submission = pd.DataFrame()
```

```
submission['Item_Identifier'] = test['Item_Identifier']
```

```
submission['Outlet_Identifier'] = test['Outlet_Identifier']
```

```
submission['Item_Outlet_Sales'] = model.predict(test)
```

R Code:

```
set.seed(1)
```

```
require(titanic)
```

```
require(caret)
```

```
require(catboost)
```

```
tt <- titanic::titanic_train[complete.cases(titanic::titanic_train),]
```

```
data <- as.data.frame(as.matrix(tt), stringsAsFactors = TRUE)
```

```
drop_columns = c("PassengerId", "Survived", "Name", "Ticket", "Cabin")
```

```
x <- data[,!(names(data) %in% drop_columns)] y <- data[,c("Survived")]
```

```
fit_control <- trainControl(method = "cv", number = 4, classProbs = TRUE)
```

```
grid <- expand.grid(depth = c(4, 6, 8), learning_rate = 0.1, iterations = 100, l2_leaf_reg = 1e-3,
  rsm = 0.95, border_count = 64)

report <- train(x, as.factor(make.names(y)), method = catboost.caret, verbose = TRUE,
preProc = NULL, tuneGrid = grid, trControl = fit_control)

print(report)

importance <- varImp(report, scale = FALSE)

print(importance)
```

Questions:

1. What are different machine learning techniques?
2. What is supervised and unsupervised machine learning?
3. Difference between supervised and unsupervised machine learning?

ASSIGNMENT 3:

AIM : Study of various Machine Learning Models

THEORY :

In machine learning and statistics, **classification** is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. An example would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.

In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.

Often, the individual observations are analyzed into a set of quantifiable properties, known variously as explanatory variables or *features*. These properties may variously be categorical (e.g. "A", "B", "AB" or "O", for blood type), ordinal (e.g. "large", "medium" or "small"), integer-valued (e.g. the number of occurrences of a particular word in an email) or real-valued (e.g. a measurement of blood pressure). Other classifiers work by comparing observations to previous observations by means of a similarity or distance function.

An algorithm that implements classification, especially in a concrete implementation, is known as a **classifier**. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category.

Terminology across fields is quite varied. In statistics, where classification is often done with logistic regression or a similar procedure, the properties of observations are

termed explanatory variables (or independent variables, regressors, etc.), and the categories to be predicted are known as outcomes, which are considered to be possible values of the dependent variable. In machine learning, the observations are often known as *instances*, the explanatory variables are termed *features* (grouped into a feature vector), and the possible categories to be predicted are *classes*. Other fields may use different terminology: e.g. in community ecology, the term "classification" normally refers to cluster analysis, i.e. a type of unsupervised learning, rather than the supervised learning described in this article.

Classification and clustering are examples of the more general problem of pattern recognition, which is the assignment of some sort of output value to a given input value. Other examples are regression, which assigns a real-valued output to each input; sequence labeling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence); parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence; etc.

A common subclass of classification is probabilistic classification. Algorithms of this nature use statistical inference to find the best class for a given instance. Unlike other algorithms, which simply output a "best" class, probabilistic algorithms output a probability of the instance being a member of each of the possible classes. The best class is normally then selected as the one with the highest probability. However, such an algorithm has numerous advantages over non-probabilistic classifiers:

- It can output a confidence value associated with its choice (in general, a classifier that can do this is known as a *confidence-weighted classifier*).
- Correspondingly, it can *abstain* when its confidence of choosing any particular output is too low.
- Because of the probabilities which are generated, probabilistic classifiers can be more effectively incorporated into larger machine-learning tasks, in a way that partially or completely avoids the problem of *error propagation*.

Types of MACHINE LEARNING Models

Machine Learning can be built using three types of models: binary classification, multiclass classification, and regression. The type of model you should choose depends on the type of target that you want to predict.

Binary Classification Model

Binary or binomial classification is the task of classifying the elements of a given set into two groups (predicting which group each one belongs to) on the basis of a classification rule. Contexts requiring a decision as to whether or not an item has some qualitative property, some specified characteristic, or some typical binary classification include:

- Medical testing to determine if a patient has certain disease or not – the classification property is the presence of the disease.
- A "pass or fail" test method or quality control in factories, i.e. deciding if a specification has or has not been met – a Go/no go classification.
- Information retrieval, namely deciding whether a page or an article should be in the result set of a search or not – the classification property is the relevance of the article, or the usefulness to the user.

Binary classification is dichotomization applied to practical purposes, and in many practical binary classification problems, the two groups are not symmetric – rather than overall accuracy, the relative proportion of different types of errors is of interest. For example, in medical testing, a false positive (detecting a disease when it is not present) is considered differently from a false negative (not detecting a disease when it is present).

Machine Learning models for binary classification problems predict a binary outcome (one of two possible classes). Example : to train binary classification models, Amazon MACHINE LEARNING uses the industry-standard learning algorithm known as logistic regression.

Examples of Binary Classification Problems

- "Is this email spam or not spam?"
- "Will the customer buy this product?"
- "Is this product a book or a farm animal?"
- "Is this review written by a customer or a robot?"

Multiclass Classification Model

In machine learning, **multiclass** or **multinomial classification** is the problem of classifying instances into one of three or more classes. (Classifying instances into one of the two classes is called binary classification.)

While some classification algorithms naturally permit the use of more than two classes, others are by nature binary algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies.

Multiclass classification should not be confused with multi-label classification, where multiple labels are to be predicted for each instance.

MACHINE LEARNING models for multiclass classification problems allow you to generate predictions for multiple classes (predict one of more than two outcomes). Example, for training multiclass models, Amazon MACHINE LEARNING uses the industry-standard learning algorithm known as multinomial logistic regression.

Examples of Multiclass Problems

- "Is this product a book, movie, or clothing?"
- "Is this movie a romantic comedy, documentary, or thriller?"
- "Which category of products is most interesting to this customer?"

Regression Model

In statistical modelling, **regression analysis** is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modelling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors').

More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Most commonly, regression analysis estimates the conditional expectation of the dependent variable given the independent variables – that is, the average value of the dependent variable when the independent variables are fixed. Less commonly, the focus is on a quantile, or other location parameter of the conditional distribution of the dependent variable given the independent variables. In all cases, a function of the independent variables called the **regression function** is to be estimated. In regression analysis, it is also of interest to characterize the variation of the dependent variable around the prediction of the regression function using a probability distribution. A related but distinct approach is necessary condition analysis(NCA), which estimates the maximum (rather than average) value of the dependent variable for a given value of the independent variable (ceiling line rather than central line) in order to identify what value of the independent variable is necessary but not sufficient for a given value of the dependent variable.

Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Regression analysis is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships

MACHINE LEARNING models for regression problems predict a numeric value. For training regression models, Amazon MACHINE LEARNING uses the industry-standard learning algorithm known as linear regression.

Examples of Regression Problems

- "What will the temperature be in Seattle tomorrow?"
- "For this product, how many units will sell?"
- "What price will this house sell for?"

Questions:

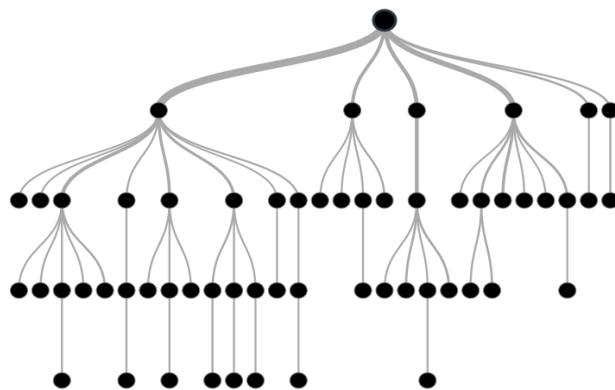
1. Enlist various machine learning models.
2. What is Linear based model, Algebraic, probabilistic models ?

ASSIGNMENT 4 :

AIM : Study and implement Decision Tree using R Programming

THEORY :

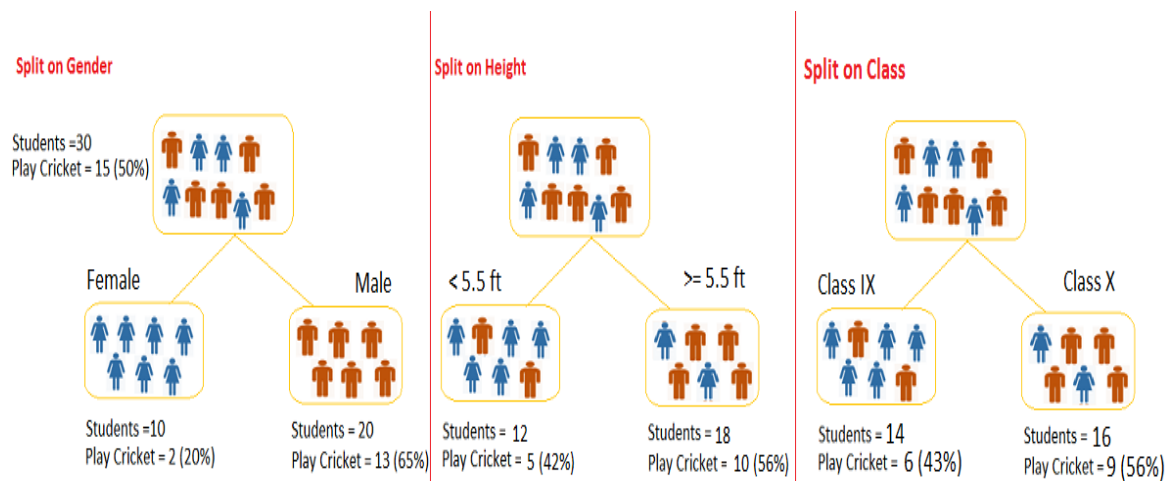
Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.



Example:-

Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class(IX/ X) and Height (5 to 6 ft). 15 out of these 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.

This is where decision tree helps, it will segregate the students based on all values of three variable and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other). In the snapshot below, you can see that variable Gender is able to identify best homogeneous sets compared to the other two variables.



As mentioned above, decision tree identifies the most significant variable and its value that gives best homogeneous sets of population. Now the question which arises is, how does it identify the variable and the split? To do this, decision tree uses various algorithms, which we will shall discuss in the following section.

Types of Decision Trees

Types of decision tree is based on the type of target variable we have. It can be of two types:

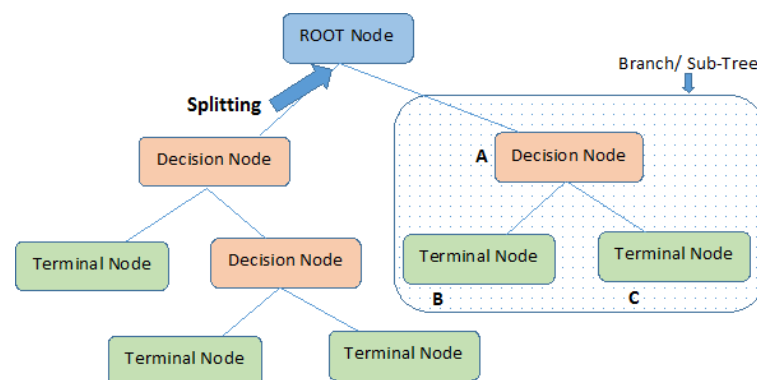
1. **Categorical Variable Decision Tree:** Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example:- In above scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.
2. **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Example:- Let’s say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that income of customer is a significant variable but insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product and various other variables. In this case, we are predicting values for continuous variable.

Important Terminology related to Decision Trees

Let's look at the basic terminology used with Decision trees:

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.



5. **Note:-** A is parent node of B and C.

6. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
7. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
8. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

These are the terms commonly used for decision trees. As we know that every algorithm has advantages and disadvantages, below are the important factors which one should know.

Advantages

1. **Easy to Understand:** Decision tree output is very easy to understand even for people from non-analytical background. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive and users can easily relate their hypothesis.

2. **Useful in Data exploration:** Decision tree is one of the fastest way to identify most significant variables and relation between two or more variables. With the help of decision trees, we can create new variables / features that has better power to predict target variable. You can refer article (Trick to enhance power of regression model) for one such trick. It can also be used in data exploration stage. For example, we are working on a problem where we have information available in hundreds of variables, there decision tree will help to identify most significant variable.
3. **Less data cleaning required:** It requires less data cleaning compared to some other modelling techniques. It is not influenced by outliers and missing values to a fair degree.
4. **Data type is not a constraint:** It can handle both numerical and categorical variables.
5. **Non Parametric Method:** Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

Disadvantages

1. **Over fitting:** Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning (discussed in detailed below).
2. **Not fit for continuous variables:** While working with continuous numerical variables, decision tree loses information when it categorizes variables in different categories.

IMPLEMENTATION OF DECISION TREES IN R :

For R users, there are multiple packages available to implement decision tree such as **ctree**, **rpart**, **tree** etc.

```
> library(rpart)
```

We will use the **ctree()** function to create the decision tree and see its graph.

```
# Load the party package. It will automatically load other dependent packages.  
>library(party)
```

```

# Create the input data frame.
>input.dat <- readingSkills[c(1:105),]

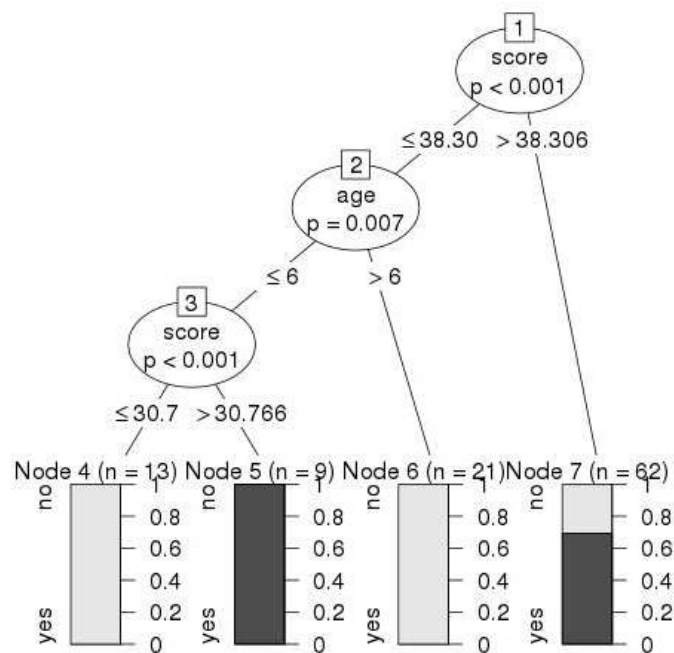
# Give the chart file a name.
>png(file = "decision_tree.png")

# Create the tree.
> output.tree <- ctree(
  nativeSpeaker ~ age + shoeSize + score,
  data = input.dat)

# Plot the tree.
>plot(output.tree)

```

OUTPUT :



Questions:

1. What is Decision Tree?
2. How to implement decision tree using R Programming?

ASSIGNMENT 5:

AIM: Study and implement Support Vector Machines using R Programming.

THEORY:

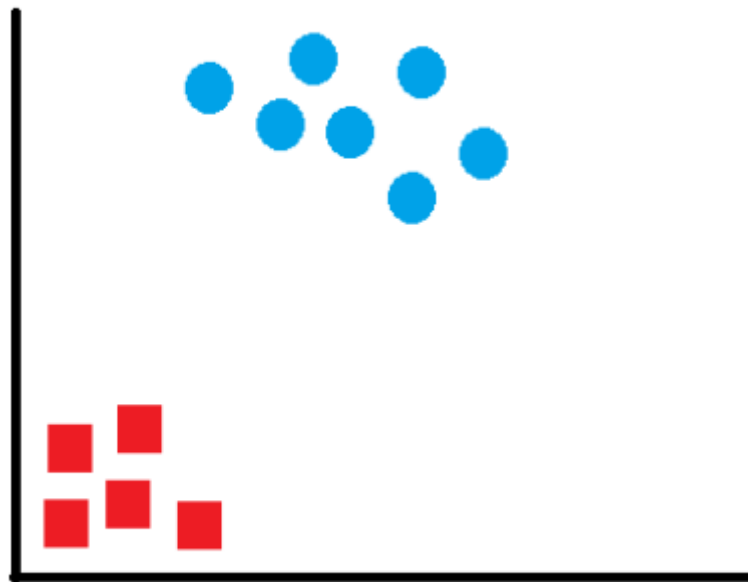
In machine learning, **support vector machines** (SVMs, also **support vector networks**) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data are not labelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The clustering algorithm which provides an improvement to the support vector machines is called **support vector clustering** and is often used in industrial applications either when data are not labelled or when only some data are labelled as a pre-processing for a classification pass.

Support Vector Machines (SVM) is a data classification method that separates data using hyper planes. The concept of SVM is very intuitive and easily understandable. If we have labelled data, SVM can be used to generate multiple separating hyper planes such that the data space is divided into segments and each segment contains only one kind of data. SVM technique is generally useful for data which has non-regularity which means, data whose distribution is unknown.

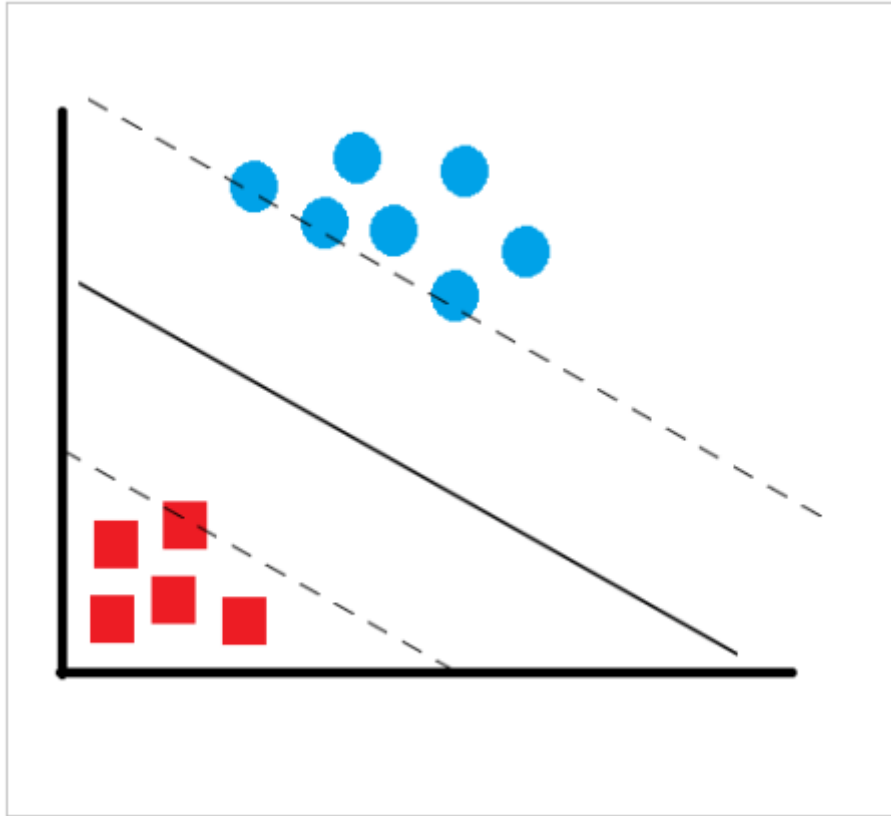
Let's take a simple example to understand how SVM works. Say you have only two kinds of values and we can represent them as in the figure:



This data is simple to classify and one can see that the data is clearly separated into two segments. Any line that separates the red and blue items can be used to classify the above data. Had this data been multi-dimensional data, any plane can separate and successfully classify the data. However, we want to find the “most optimal” solution. What will then be the characteristic of this most optimal line? We have to remember that this is just the training data and we can have more data points which can lie anywhere in the subspace. If our line is too close to any of the data points, noisy test data is more likely to get classified in a wrong segment. We have to choose the line

which lies between these groups and is at the farthest distance from each of the segments.

To solve this classifier line, we draw the line as $y=ax+b$ and make it equidistant from the respective data points closest to the line. So we want to maximize the margin (m).



We know that all points on the line $ax+b=0$ will satisfy the equation. If we draw two parallel lines $-ax+b=-1$ for one segment and $ax+b=1$ for the other segment such that these lines pass through a data point in the segment which is nearest to our line then the distance between these two lines will be our margin. Hence, our margin will be $m=2/\|a\|$. Looked at another way, if we have a training dataset $(x_1, x_2, x_3 \dots x_n)$ and want to produce an outcome y such that y is either -1 or 1 (depending on which segment the data point belongs to), then our classifier should correctly classify data points in the form of $y=ax+b$. This also means that $y(ax+b)>1$ for all data points. Since we have to maximize the margin, we have to solve this problem with the constraint of maximizing $2/\|a\|$ or, minimizing $\|a\|^2/2$ which is basically the dual form of the constraint. Solving this can be easy or complex depending upon the dimensionality of data. However, we can do this quickly with R and try out with some sample dataset.

IMPLEMENTATION OF SVM MODEL IN R :

To use SVM in R, I just created a random data with two features x and y in excel. I took all the values of x as just a sequence from 1 to 20 and the corresponding values of y as derived using the formula $y(t)=y(t-1) + r(-1:9)$ where $r(a,b)$ generates a random integer between a and b. I took $y(1)$ as 3. The following code in R illustrates a set of sample generated values:

```
x=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)
```

```
y=c(3,4,5,4,8,10,10,11,14,20,23,24,32,34,35,37,42,48,53,60)
```

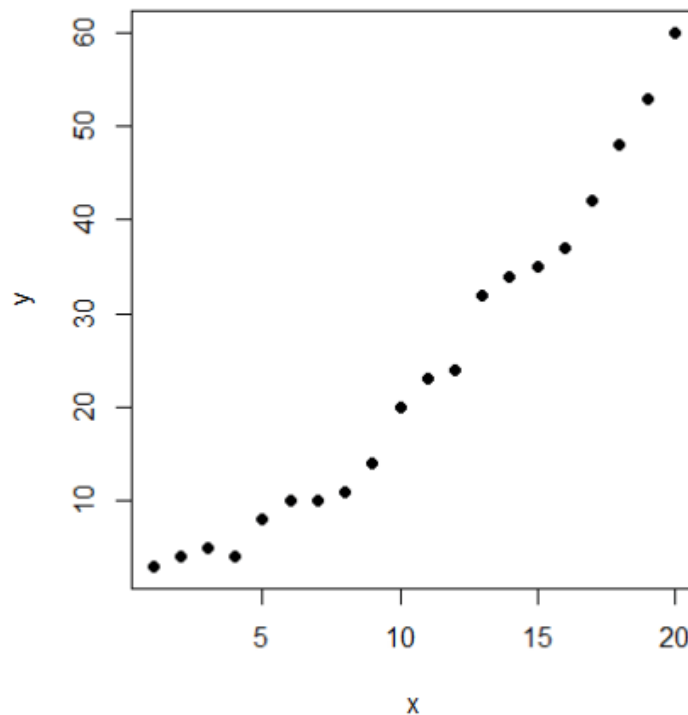
```
#Create a data frame of the data
```

```
train=data.frame(x,y)
```

Let's see how our data looks like. For this we use the plot function

```
#Plot the dataset
```

```
plot(train,pch=16)
```



To use SVM in R, we have a package `e1071`. The package is not preinstalled, hence one needs to run the line `install.packages("e1071")` to install the package and then import the package contents using the `library` command. The syntax of `svm` package is quite similar to linear regression. We use `svm` function here.

```
library(e1071)
```

```
#Fit a model. The function syntax is very similar to lm function
```

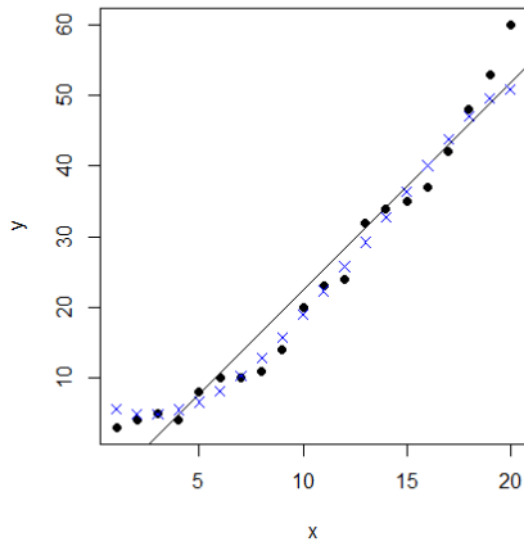
```
model_svm <- svm(y ~ x , train)
```

```
#Use the predictions on the data
```

```
pred <- predict(model_svm, train)
```

```
#Plot the predictions and the plot to see our model fit
```

```
points(train$x, pred, col = "blue", pch=4)
```



Questions:

1. What is Support Vector Machine?
2. How to implement it using R Programming?

ASSIGNMENT 6:

AIM: What is Regression? Implement Linear Regression using R Programming.

THEORY:

Regression : In statistical modelling, regression analysis is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modelling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors'). More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Most commonly, regression analysis estimates the conditional expectation of the dependent variable given the independent variables – that is, the average value of the dependent variable when the independent variables are fixed. Less commonly, the focus is on a quantile, or other location parameter of the conditional distribution of the dependent variable given the independent variables. In all cases, a function of the independent variables called the regression function is to be estimated. In regression analysis, it is also of interest to characterize the variation of the dependent variable around the prediction of the regression function using a probability distribution. A related but distinct approach is necessary condition analysis (NCA), which estimates the maximum (rather than average) value of the dependent variable for a given value of the independent variable (ceiling line rather than central line) in order to identify what value of the independent variable is necessary but not sufficient for a given value of the dependent variable.

Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Regression analysis is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships. In restricted circumstances, regression analysis can be used to infer causal relationships between the independent

and dependent variables. However this can lead to illusions or false relationships, so caution is advisable; for example, correlation does not prove causation.

Many techniques for carrying out regression analysis have been developed. Familiar methods such as linear regression and ordinary least squares regression are parametric, in that the regression function is defined in terms of a finite number of unknown parameters that are estimated from the data. Nonparametric regression refers to techniques that allow the regression function to lie in a specified set of functions, which may be infinite-dimensional.

The performance of regression analysis methods in practice depends on the form of the data generating process, and how it relates to the regression approach being used. Since the true form of the data-generating process is generally not known, regression analysis often depends to some extent on making assumptions about this process. These assumptions are sometimes testable if a sufficient quantity of data is available. Regression models for prediction are often useful even when the assumptions are moderately violated, although they may not perform optimally. However, in many applications, especially with small effects or questions of causality based on observational data, regression methods can give misleading results.

In a narrower sense, regression may refer specifically to the estimation of continuous response (dependent) variables, as opposed to the discrete response variables used in classification. The case of a continuous dependent variable may be more specifically referred to as metric regression to distinguish it from related problems.

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is –

$$y = ax + b$$

Following is the description of the parameters used –

- y is the response variable.
- x is the predictor variable.
- a and b are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is –

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the `lm()` functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction.

Also called residuals.

- To predict the weight of new persons, use the `predict()` function in R.

Input Data

Below is the sample data representing the observations –

Values of height

151, 174, 138, 186, 128, 136, 179, 163, 152, 131

Values of weight.

63, 81, 56, 91, 47, 57, 76, 72, 62, 48

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for lm() function in linear regression is –

```
lm(formula,data)
```

Following is the description of the parameters used –

- formula is a symbol presenting the relation between x and y.
- data is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
```

```
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
```

```
# Apply the lm() function.
```

```
relation <- lm(y~x)
```

```
print(relation)
```

When we execute the above code, it produces the following result –

Call:

```
lm(formula = y ~ x)
```

Coefficients:

(Intercept)	x
-38.4551	0.6746

IMPLEMENTATION OF LINEAR REGRESSION IN R :

Visualize the Regression Graphically

Create the predictor and response variable.

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
```

```
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
```

```
relation <- lm(y~x)
```

Give the chart file a name.

```
png(file = "linearregression.png")
```

Plot the chart.

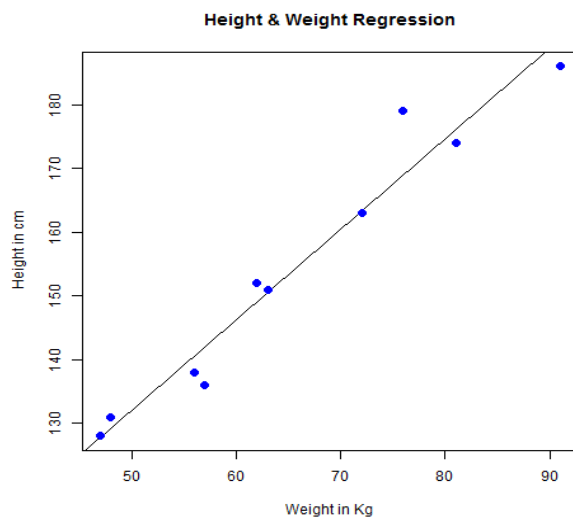
```
plot(y,x,col = "blue",main = "Height & Weight Regression",
```

```
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")
```

Save the file.

```
dev.off()
```

When we execute the above code, it produces the following result –



Questions:

1. What is Regression?
2. What is Linear Regression? How to implement it?

ASSIGNMENT 7:

AIM: Examine Classification and Regression .What are the issues regarding classification and regression.

THEORY:

Regression Trees vs Classification Trees

Often when a machine learning task is presented to you the first thing you will do it's to get to know whether the learning task is Classification or regression problem so that next you can pick the algorithm. These are simple concepts to understand, watch out.

Classification

In classification problems we are trying to predict a discrete number of values.

The labels(y) generally comes in categorical form and represents a finite number of classes. Consider the tasks bellow:

1. Given set of input features predict whether a Breast Cancer is Benign or Malignant.
2. Given an image correctly classify as containing Cats or Dogs.
3. From a given email predict whether it's spam email or not.

Types of classification

Binary classification—when there is only two classes to predict, usually 1 or 0 values.

Multi-Class Classification—When there are more than two class labels to predict we call multi-classification task. E.g. predicting 3 types of iris species, image classification problems where there are more than thousands classes(cat, dog, fish, car,...).

Algorithms for classification

- Decision Trees

- Logistic Regression
- Naive Bayes
- K Nearest Neighbors
- Linear SVC (Support vector Classifier) etc.

Regression Problems

In regression problems we trying to predict continuous valued output, take this example. Given a size of the house predict the price(real value).

Regression Algorithms

- Linear Regression
- Regression Trees(e.g. Random Forest)
- Support Vector Regression (SVR) etc.

Classification VS Regression

Classification: Discrete valued Y (e.g. 1,2,3 and 4)

Regression: Continuous Values Y (e.g. 222.6, 300, 568,...)

Whenever you find machine learning problem first define whether you are dealing with a classification or regression problem and you can get to know that analyzing the target variable (Y), note that here the input X can of any kind (continues or discrete) that doesn't count to define the problem. After defining the problem and getting to know the data it's much easier to chose or try out some algorithms.

We all know that the terminal nodes (or leaves) lies at the bottom of the decision tree. This means that decision trees are typically drawn upside down such that leaves are the the bottom & roots are the tops (shown below).



Both the trees work almost similar to each other, let's look at the primary differences & similarity between classification and regression trees:

1. Regression trees are used when dependent variable is continuous. Classification trees are used when dependent variable is categorical.
2. In case of regression tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mean value.
3. In case of classification tree, the value (class) obtained by terminal node in the training data is the mode of observations falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mode value.
4. Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions. For the sake of simplicity, you can think of these regions as high dimensional boxes or boxes.
5. Both the trees follow a top-down greedy approach known as recursive binary splitting. We call it as 'top-down' because it begins from the top of tree when all the observations are available in a single region and successively splits the predictor space into two new branches down the tree. It is known as 'greedy' because, the algorithm cares (looks for best variable available) about only the current split, and not about future splits which will lead to a better tree.
6. This splitting process is continued until a user defined stopping criteria is reached. For example: we can tell the the algorithm to stop once the number of observations per node becomes less than 50.
7. In both the cases, the splitting process results in fully grown trees until the stopping criteria is reached. But, the fully grown tree is likely to overfit data, leading

to poor accuracy on unseen data. This bring 'pruning'. Pruning is one of the technique used tackle overfitting. We'll learn more about it in following section.

Questions:

1. What is Classification and Regression?
2. What are the issues regarding classification and regression.

ASSIGNMENT 8:

AIM: Distinguish Supervised and Unsupervised machine learning.

THEORY:

In the world of data science supervised, and unsupervised learning algorithms were the famous words, we could hear more frequently these while we were talking with the people who are working in data science field. Furthermore, the key differences between these two learning algorithms are the must learn concepts for differentiating the real world problems.

Supervised Learning Wiki Definition

Supervised learning is a [data mining](#) task of inferring a function from **labeled training data**. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and the desired output value (also called the **supervisory signal**).

A **supervised learning algorithm** analyzes the training data and produces an inferred function, which can used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for **unseen instances**. This requires the learning algorithm to generalize from the training data to unseen situations in a “reasonable” way.

Unsupervised Learning Wiki Definition

In data mining or even in data science world, the problem of an unsupervised learning task is trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution.

Supervised and unsupervised learning with a real-life example



- Suppose you had a basket and filled it with different kinds of fruits.
- Your task is to arrange them into groups.
- For understanding let me explain the names of the fruits in our basket.
- We have four types of fruits. They are



APPLE



BANANA



GRAPE



CHERRIES

Supervised Learning:

- You already learn from your previous work about the physical characters of fruits
- So arranging the same type of fruits at one place is easy now
- In data mining terminology the earlier work is called as **training the data**
- You already learn the things from your train data. This is because of **response variable**
- Response variable means just a **decision variable**
- You can observe response variable below (**FRUIT NAME**)

No.	SIZE	COLOR	SHAPE	FRUIT NAME
1	Big	Red	Rounded shape with depression at the top	Apple
2	Small	Red	Heart-shaped to nearly globular	Cherry
3	Big	Green	Long curving cylinder	Banana
4	Small	Green	Round to oval,Bunch shape Cylindrical	Grape

- Suppose you have taken a new fruit from the basket then you will see the size, color, and shape of that particular fruit.
- If size is Big, color is Red, the shape is rounded shape with a depression at the top, you will confirm the fruit name as apple and you will put in apple group.
- Likewise for other fruits also.
- The job of grouping fruits was done and the happy ending.
- You can observe in the table that a column was labeled as “**FRUIT NAME**“. This is called as a response variable.
- If you learn the thing before from training data and then applying that knowledge to the test data(for new fruit), This type of learning is called as **Supervised Learning**.

Supervised Learning Algorithms:

All classification and regression algorithms come under supervised learning.

- Logistic Regression
- Decision trees
- Support vector machine (SVM)

- k-Nearest Neighbors
- Naive Bayes
- Random forest
- Linear regression
- polynomial regression
- SVM for regression

Unsupervised Learning:

- Suppose you have a basket and it is filled with some different types of fruits and your task is to arrange them as groups.
- This time, you don't know anything about the fruits, honestly saying this is the first time you have seen them. You have no clue about those.
- So, how will you arrange them?
- What will you do first???
- You will take a fruit and you will arrange them by considering the physical character of that particular fruit.
- Suppose you have considered color.
 - Then you will arrange them on considering base condition as **color**.
 - Then the groups will be something like this.
 - RED COLOR GROUP: apples & cherry fruits.
 - GREEN COLOR GROUP: bananas & grapes.
- So now you will take another physical character such as **size**.
 - RED COLOR AND BIG SIZE: apple.
 - RED COLOR AND SMALL SIZE: cherry fruits.
 - GREEN COLOR AND BIG SIZE: bananas.
 - GREEN COLOR AND SMALL SIZE: grapes.
- The job has done, the happy ending.
- Here you did not learn anything before ,means no train data and no response variable.
- In data mining or machine learning, this kind of learning is known as *unsupervised learning*.

Unsupervised learning algorithms:

All clustering algorithms come under unsupervised learning algorithms.

- K – means clustering
- Hierarchical clustering
- Hidden Markov models

Questions:

1. Differentiate between supervised and unsupervised learning algorithms.

ASSIGNMENT 9:

AIM: Study and implement K-Means clustering algorithm.

THEORY:

***k*-means clustering** is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. *k*-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms. Additionally, they both use cluster centers to model the data; however, *k*-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the *k*-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with *k*-means because of the k in the name. One can apply the 1-nearest neighbor classifier on the cluster centers obtained by *k*-means to classify new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

Introduction to K-means Clustering

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K . The algorithm works iteratively to assign each data point to one of

K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K -means clustering algorithm are:

1. The centroids of the K clusters, which can be used to label new data
2. Labels for the training data (each data point is assigned to a single cluster)

Rather than defining groups before looking at the data, clustering allows you to find and analyze the groups that have formed organically. The "Choosing K " section below describes how the number of groups can be determined.

Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

This introduction to the K -means clustering algorithm covers:

- Common business cases where K -means is used
- The steps involved in running the algorithm
- A Python example using delivery fleet data

Finally, this algorithm aims at minimizing an objective function known as squared error function given by:

$$J(V) = \sum_{i=1}^c \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2$$

where,

$\|x_i - v_j\|$ is the Euclidean distance between x_i and v_j .

c_i is the number of data points in i^{th} cluster.

c is the number of cluster centers.

Algorithmic steps for k-means clustering

Let $X = \{x_1, x_2, x_3, \dots, x_n\}$ be the set of data points and $V = \{v_1, v_2, \dots, v_c\}$ be the set of centers.

- 1) Randomly select 'c' cluster centers.
- 2) Calculate the distance between each data point and cluster centers.
- 3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers..
- 4) Recalculate the new cluster center using:

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i$$

where, ' c_i ' represents the number of data points in i^{th} cluster.

- 5) Recalculate the distance between each data point and new obtained cluster centers.
- 6) If no data point was reassigned then stop, otherwise repeat from step 3).

Advantages

- 1) Fast, robust and easier to understand.
- 2) Relatively efficient: $O(tknd)$, where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, k, t, d \ll n.
- 3) Gives best result when data set are distinct or well separated from each other.

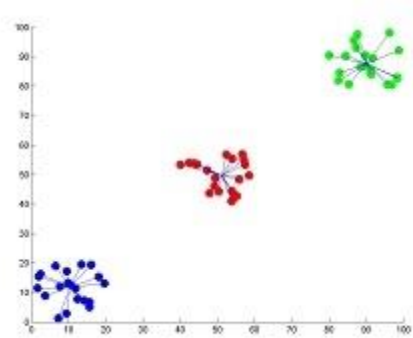


Fig I: Showing the result of k-means for ' N ' = 60 and ' c ' = 3

Note: For more detailed figure for k-means algorithm please refer to k-means figure sub page.

Disadvantages

- 1) The learning algorithm requires apriori specification of the number of cluster centers.
- 2) The use of Exclusive Assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.
- 3) The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).
- 4) Euclidean distance measures can unequally weight underlying factors.
- 5) The learning algorithm provides the local optima of the squared error function.
- 6) Randomly choosing of the cluster center cannot lead us to the fruitful result. Pl. refer Fig.
- 7) Applicable only when mean is defined i.e. fails for categorical data.
- 8) Unable to handle noisy data and outliers.
- 9) Algorithm fails for non-linear data set.

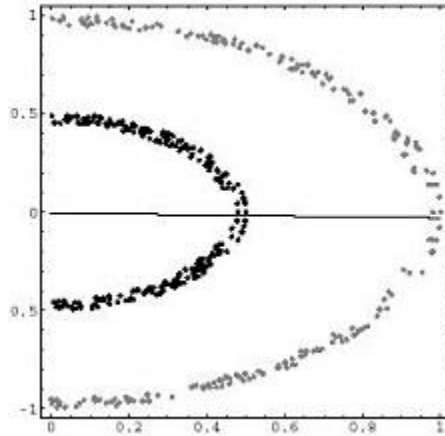


Fig II: Showing the non-linear data set where k-means algorithm fails

APPLICATIONS :

Business Uses

The *K*-means clustering algorithm is used to find groups which have not been explicitly labeled in the data. This can be used to confirm business assumptions about what types of groups exist or to identify unknown groups in complex data sets. Once the algorithm has been run and the groups are defined, any new data can be easily assigned to the correct group.

This is a versatile algorithm that can be used for any type of grouping. Some examples of use cases are:

- Behavioral segmentation:
 - Segment by purchase history
 - Segment by activities on application, website, or platform
 - Define personas based on interests
 - Create profiles based on activity monitoring
- Inventory categorization:
 - Group inventory by sales activity
 - Group inventory by manufacturing metrics
- Sorting sensor measurements:
 - Detect activity types in motion sensors
 - Group images

IMPLEMENTATION OF K MEANS CLUSTERING IN R :

In the Iris Dataset, after a little bit of exploration, it's evident that Petal.Length and Petal.Width were similar among the same species but varied considerably between different species

```
library(datasets)
```

```
library(ggplot2)
```

```
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom_point()
```

```
set.seed(20)
```

```
irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
```

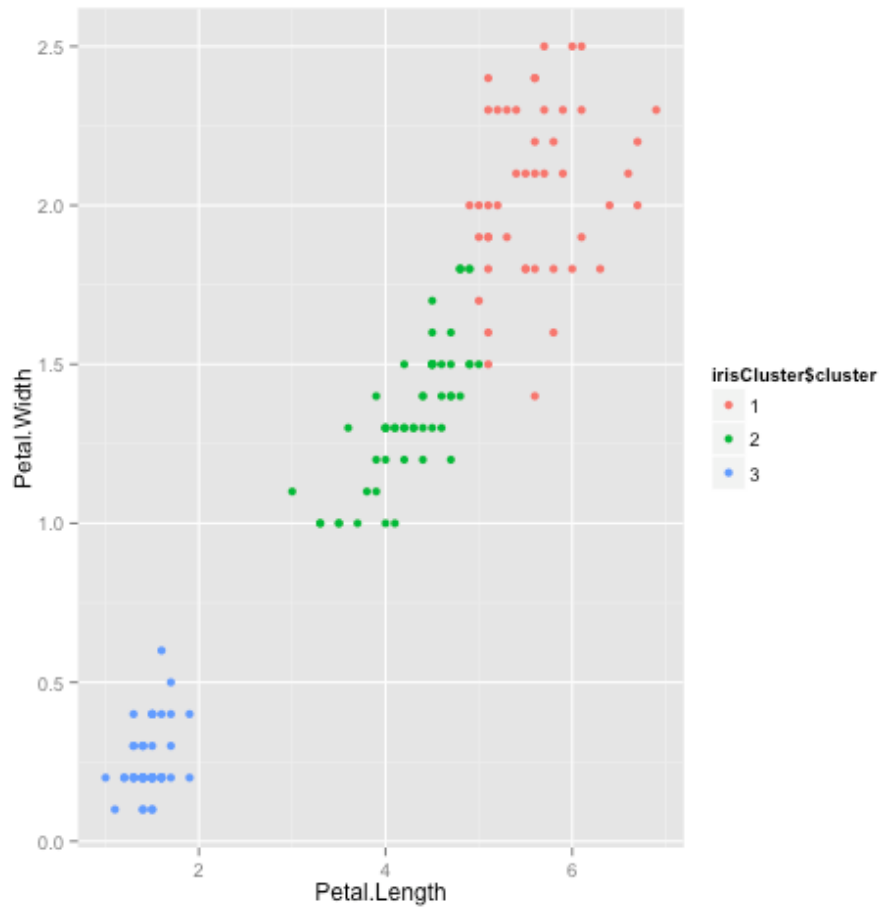
```
irisCluster
```

```
table(irisCluster$cluster, iris$Species)
```

```
irisCluster$cluster <- as.factor(irisCluster$cluster)
```

```
ggplot(iris, aes(Petal.Length, Petal.Width, color = iris$cluster)) + geom_point()
```


Result plot :



Questions:

1. What is Clustering?
2. What is K-Means Clustering?

ASSIGNMENT 10:

AIM: Case study on SCIKIT-LEARN, WEKA tool for machine learning..

THEORY:

Case Study Of SCIKIT Learn : EVERNOTE Application in Windows

The new Evernote Food includes a “My Cookbook” section that collects any recipes you’ve clipped or typed into Evernote in one place. Some users put all of their recipes into a specific notebook or assign them a specific tag, while others don’t follow any particular convention. We wanted “My Cookbook” to *just work* for all users. In other words, no matter how you choose to group or categorize your notes, Evernote should be able to automatically identify which ones are recipes, pluck them out, and show them to you at the right time.

In this post, I’d like to give you a behind-the-scenes look at recipe classification, the new note classification pipeline that makes it possible, and what it means for users, partners, and third-party developers.

Recipe Classification

Separating recipes from non-recipes is a special case of what’s known as a *supervised learning* problem. It’s “supervised” because the classifier has to be fed examples of recipes and non-recipes, usually prepared manually, before it can learn how to automatically tell them apart. Most of us already encounter supervised learning daily whenever we check our email: a spam classifier, trained on numberless examples of canny solicitations from Nigerian princes, keeps the junk out of our inboxes.

For finding recipes in your Evernote account, we encountered a number of specific challenges. Good food has universal appeal, and we wanted to launch “My Cookbook” in many different languages to start: English, Chinese, Japanese, Korean, Spanish, French, Italian, German, Portuguese, and Russian. Not only does the cuisine in the

places where these languages are spoken vary a great deal, but so does what counts as a recipe. For example, most Chinese recipes omit specific measurements and quantities — instead of a teaspoon of sugar or half a cup of flour, a Chinese recipe may simply instruct you to use “the right amount”. (I always imagine the recipe author delivering this phrase with a knowing wink.)

Then there’s the matter of actually gathering representative training data and preparing it for use. Here, we faced an additional challenge: how to get data that was representative of real-world notes. One way would be to peek at users’ private notes until we had the data we needed, but respect for user privacy is paramount at Evernote, and so we never considered this approach. In the end, we went about gathering the recipe and non-recipe data in two different ways. For recipes, we enlisted speakers of supported languages to clip recipes in their native language, targeting a variety of different online recipe sources. For non-recipes, we started with a random sample of Evernote notes published to public notebooks, then enlisted native speakers to pluck out any recipes that happened to make it in.

The Note Classification Flow

Now, a bit on what’s happening under the hood. Building a classifier is typically an iterative process of exploring the data, selecting the features (the attributes of the data believed to be predictive in some way), training the models, and finally evaluating them. For many of these tasks, we relied on the excellent scikit-learn package for Python. Currently, we use a variant of Naive Bayes classification to do our prediction. We picked Naive Bayes because it’s simple, fast, and performs quite well on this kind of classification problem despite a few limitations arising from its simplicity. A trained Naive Bayes model is basically a giant table in which each row consists of a single feature (e.g., the phrase “teaspoon of sugar”), followed by a list of float values representing how likely we are to encounter that feature in a note belonging to each output class (e.g., “recipe” or “non-recipe”). To classify a note, we extract its feature values, look them up in the table, and keep sums of the corresponding probabilities. What we’re left with is an overall measure of how likely the note is to belong to either output class.

Once we train the models, we load them via home-grown Java code that runs in the Evernote service backend. During any `createNote()` or `updateNote()` API call, the note passes through a gauntlet of different classifiers before being persisted. For recipe classification, we use a two-step approach. First, we attempt to automatically guess the note's language from its content. An Evernote user can have notes in many different languages (indeed, any one note can contain text in many different languages). Once we determine a note's predominant language, we pass it to a language-specific recipe model, trained to capture the quirks and idiosyncrasies of recipes in that particular language/locale. Finally, if the model believes the note to be a recipe, we save this classification as part of the note metadata. This process takes only a tiny fraction of a second to complete, and there should be no delay perceptible to the end user.

Where to Find Classifications

Like everything else in an Evernote account, all classifications assigned to a note are accessible via the Evernote API. Note classifications are stored in a note's `NoteAttributes.classifications` field, a map of string keys to string values. A key in this map represents the type of classification. For now, the only classification used by Evernote apps is the recipe classification (identified by the key "recipe") but the data model is extensible to allow us to classify all sorts of notes.

The classification value for recipes is one of the constants that begin with `CLASSIFICATION_RECIPE_`. For example, `CLASSIFICATION_RECIPE_SERVICE_RECIPE` means "the Evernote service believes this note to be a recipe". The constants that begin with `CLASSIFICATION_RECIPE_USER_` signify that the end user, as opposed to the Evernote service, designated a note as having a particular classification. If the classifier sees a note with this type of user-assigned classification, it will never attempt to overwrite it. This allows a user to take control when we misclassify a note. For now, Evernote Food allows you to tell us that a note shown in "My Cookbook" isn't a recipe. In the future, we'll give you ways to mark a note as a recipe if our classifier misses it.

The classifications map is extensible, and all classifications are public, so any third-party app can examine whether we believe a note to be a recipe. To find all of the

notes with a recipe classification in a given user's account, simply use the search API to search for "any: classifications_recipe:001 classifications_recipe:002".

Case Study WEKA TOOL for DATA MINING :

Introduction :

Weka (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

Weka is free software available under the GNU General Public License.

WEKA 3 (Last Update: 2014-09-23), the latest version of WEKA has been optimized to handle BIG Data.

It contains a collection of visualization tools and algorithms for data analysis and predictive modeling,together with graphical user interfaces for easy access to this functionality. The original non-Java version of Weka was a TCL/TK front-end to (mostly third-party) modeling algorithms implemented in other programming languages, plus data preprocessing utilities in C, and a Makefile-based system for running machine learning experiments. This original version was primarily designed as a tool for analyzing data from agricultural domains, but the more recent fully Java-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research.

Advantages of Weka include:

- free availability under the GNU General Public License
- portability, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform
- a comprehensive collection of data preprocessing and modeling techniques

- ease of use due to its graphical user interfaces

Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. All of Weka's techniques are predicated on the assumption that the data is available as a single flat file or relation, where each data point is described by a fixed number of attributes (normally, numeric or nominal attributes, but some other attribute types are also supported). Weka provides access to SQL databases using Java Database Connectivity and can process the result returned by a database query. It is not capable of multi-relational data mining, but there is separate software for converting a collection of linked database tables into a single table that is suitable for processing using Weka. Another important area that is currently not covered by the algorithms included in the Weka distribution is sequence modeling.

- **User interfaces**

Weka's main user interface is the Explorer, but essentially the same functionality can be accessed through the component-based *Knowledge Flow* interface and from the command line. There is also the *Experimenter*, which allows the systematic comparison of the predictive performance of Weka's machine learning algorithms on a collection of datasets.

The *Explorer* interface features several panels providing access to the main components of the workbench:

- The *Preprocess* panel has facilities for importing data from a database, a CSV file, etc., and for preprocessing this data using a so-called *filtering* algorithm. These filters can be used to transform the data (e.g., turning numeric attributes into discrete ones) and make it possible to delete instances and attributes according to specific criteria.
- The *Classify* panel enables the user to apply classification and regression algorithms (indiscriminately called *classifiers* in Weka) to the resulting dataset, to estimate the accuracy of the resulting predictive model, and to visualize erroneous predictions, ROC curves, etc., or the model itself (if the model is amenable to visualization like, e.g., a decision tree).

- The *Associate* panel provides access to association rule learners that attempt to identify all important interrelationships between attributes in the data.
- The *Cluster* panel gives access to the clustering techniques in Weka, e.g., the simple k-means algorithm. There is also an implementation of the expectation maximization algorithm for learning a mixture of normal distributions.
- The *Select attributes* panel provides algorithms for identifying the most predictive attributes in a dataset.
- The *Visualize* panel shows a scatter plot matrix, where individual scatter plots can be selected and enlarged, and analyzed further using various selection operators.
- **Package Manager**

In the latest update, Weka 3.7.2, a package manager was added to allow the easier installation of extension packages. This change makes it easier for other Developers to contribute extensions to Weka and to maintain the software, as this modular architecture allows independent updates of the Weka core and individual extensions. This makes addition of new features quick and easy.

- **Applications**

Weka was originally developed for the purpose of pr

ocessing agricultural data, motivated by the import

ance of this application area in New Zealand. However, the machine learning methods and data engineering capability it embodies have grown so quickly, and so radically, that the workbench is now commonly used in all forms of data mining applications—from bioinformatics to competition datasets issued by major conferences such as Knowledge Discovery in Databases.

They worked on:

-predicting the internal bruising sustained by different varieties of apple as they make their way through a packing-house on a conveyor belt;

-predicting, in real time, the quality of a mushroom from a photograph in order to provide automatic grading;

-classifying kiwifruit vines into twelve classes, based on visible-NIR spectra, in order to determine which of twelve pre-harvest fruit management treatments has been applied to the vines;

Weka has been used extensively in the field of bioinformatics. Published studies include automated protein annotation , probe selection for gene expression arrays , plant genotype discrimination , and classifying gene expression profiles and extracting rules from them.

Text mining is another major field of application, and the workbench has been used to automatically extract key phrases from text , and for document categorization, and word sense disambiguation .

There are many projects that extend or wrap WEKA in some fashion. There are 46 such projects listed on the Related Projects web page of the WEKA site³.

Questions:

1. How WEKA tool is useful in machine learning?
2. How SCIKIT-LEARN is useful in machine learning?

ASSIGNMENT 11:

AIM: Case study on ACCORD, SHOGUN tool for machine learning.

THEORY:

Case Study of ACCORD Tool for .NET Developers :

The Accord.NET Framework is both a C# machine learning framework and a complete framework for building computer vision, computer audition, signal processing and statistical applications. Sample applications provide a fast start to get up and running quickly, and an extensive documentation helps fill in the details.

For the best experience, please [download the framework through NuGet](Getting started).

Getting started

The easiest way to get started is through NuGet. Search for "Accord.NET" in the package manager, then chose to install the modules you are more likely interested. Because not all users will want to use, for example, audio or video processing capabilities in their projects, those have been written as separate modules.

Choose which module your are most interested:

PM> Install-Package Accord.MachineLearning

PM> Install-Package Accord.Imaging

PM> Install-Package Accord.Neuro

then, see the getting started guide to get up and running with the framework or download one sample application to start your project.

Why .NET?

If you are a Windows developer, using .NET is something you do without thinking. Indeed, a vast majority of Windows business applications written in the last 15 years use managed code—most of it written in C#. Although it is difficult to categorize millions of software developers, it is fair to say that .NET developers often come from nontraditional backgrounds. Perhaps a developer came to .NET from a BCSC degree but it is equally likely s/he started writing VBA scripts in Excel, moving up to Access applications, and then into VB.NET/C# applications. Therefore, most .NET developers are likely to be familiar with C#/VB.NET and write in an imperative and perhaps OO style.

One real advantage to writing and then deploying your machine learning code in .NET is that you can get everything with one stop shopping. I know several large companies who write their models in R and then have another team rewrite them in Python or C++ to deploy them. Also, they might write their model in Python and then rewrite it in C# to deploy on Windows devices. Clearly, if you could write and deploy in one language stack, there is a tremendous opportunity for efficiency and speed to market.

Loading data

If you plan to load data into your application (which is most likely the case), then you should consider adding a reference to the Accord.IO library into your project. This library provides data readers for formats like Excel, comma-separated values, matlab matrix files, LibSVM and LibLinear's data formats, and others. If you are interested in loading sounds into your application, consider adding a reference to the Accord.Audio.Formats library.

Tables

Excel and Excel compatible files (such as .csv)

To import a table from Excel or .CSV files, you can use the ExcelReader class class. To load the contents of "Sheet1" located in "worksheet.xlsx", use:

```
DataTable table = new ExcelReader("worksheet.xlsx").GetWorksheet("Sheet1");
```

Matrices

The framework can load any MATLAB-compatible .mat file using the MatReader class. It can also parse matrices written in MATLAB, Octave, Mathematica and C#/NET formats directly from text. Examples are shown below.

```
// From free-text format
```

```
double[,] a = Matrix.Parse(@"1 2  
  
3 4");
```

Images

Images can be loaded in the standard .NET Framework way. However, one might be interested into converting images from matrices and vice-versa; in this case, the classes in the Accord.Imaging.Converters namespace.

Otherwise, if you would like to apply image processing filters to standard images such as Lena Söderberg's picture, you can use the TestImages dataset.

Sounds

Sounds can be loaded from files or recorded on-the-fly using a capture device. For examples on how to record audio on-the-fly, please refer to the audio recording, beat detection and FFT sample applications.

If you are looking into a quick way to load audio samples into your application, please refer to the Signal.FromFile method.

If you would like to use an audio database in your machine learning applications, please see the FreeSpokenDigitsDataset documentation page.

Video

Video capturing is done using AForge.NET.

Learning from input and output pairs

The framework adopts an interface similar to Python's Scikit-learn package. If you would like to learn a new classifier or regression model that is able to map a set of given inputs to a set of corresponding outputs, you can first identify the algorithm that you would like to use (SVMs are a good initial choice), create it:

```
// As an example, we will try to learn a decision machine
```

```
// that can replicate the "exclusive-or" logical function:
```

```
double[][] inputs =  
  
{  
  
    new double[] { 0, 0 }, // the XOR function takes two booleans  
  
    new double[] { 0, 1 }, // and computes their exclusive or: the  
  
    new double[] { 1, 0 }, // output is true only if the two booleans  
  
    new double[] { 1, 1 } // are different  
  
};
```

```
int[] xor = // this is the output of the xor function
```

```
{  
  
    0, // 0 xor 0 = 0 (inputs are equal)  
  
    1, // 0 xor 1 = 1 (inputs are different)  
  
    1, // 1 xor 0 = 1 (inputs are different)
```

```

    0, // 1 xor 1 = 0 (inputs are equal)

};

// Now, we can create the sequential minimal optimization teacher

var learn = new SequentialMinimalOptimization<Gaussian>()

{

    UseComplexityHeuristic = true,

    UseKernelEstimation = true

};

```

And then you will be able to call the universal `.Learn()` method which is common for all learning algorithms:

```

// And then we can obtain a trained SVM by calling its Learn method

SupportVectorMachine<Gaussian> svm = learn.Learn(inputs, xor);

```

The `.Learn()` method will use the learning algorithm you have chosen to create a new machine learning model. In the case of `SequentialMinimalOptimization`, this will end up creating a new `SupportVectorMachine` object. The nice thing about the framework is that you do not even need to know about how a SVM works to be able to use it (although this would be highly advisable in case you would like to use it for something other than a toy example). All classification models in the framework implement the `.Decide()` method, which can be used to obtain class predictions for any new data you would like to present the model to:

```

// Finally, we can obtain the decisions predicted by the machine:

bool[] prediction = svm.Decide(inputs);

```

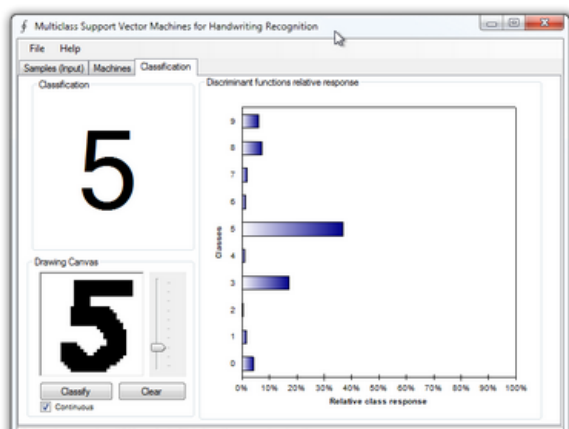
Different models can predict different kinds of data. If you need to predict bool[] vectors, then Support Vector Machines should be your first choice. If you need to predict int[] class labels, then you are invited to take a look at multi-class and multi-label Support Vector Machines.

If you would like easier examples that you could just download to your computer, press F5 and see some classification problems being solved in action, please refer to the Classification (SVMs), Classification (Naive Bayes), Classification (Decision Trees)[<https://github.com/accord-net/framework/wiki/Sample-applications#classification-decision-trees>], or Handwriting Recognition with SVMs sample applications to get up and running in no time.

Handwriting (Multi-class SVM)

- Download the [application](#)
- Browse the [source code](#)

This sample application shows how to teach [Multi-class Support Vector Machines](#) using [Sequential Minimal Optimization](#) to recognize handwritten digits from the UCI's Optdigits dataset.



Example Application using ACCORD tool.

CASE STUDY OF SHOGUN TOOL :

Shogun is a free, open source machine learning software library written in C++. It offers numerous algorithms and data structures for machine learning problems. It offers interfaces for Octave, Python, R, Java, Lua, Ruby and C# using SWIG.

It is licensed under the terms of the GNU General Public License version 3 or later.

The Shogun Machine learning toolbox offers a wide range of efficient and unified Machine Learning methods.

Shogun is accessible

- Supports many languages (Python, Octave, R, Java/Scala, Lua, C#, Ruby, etc) and platforms (Linux/Unix, MacOS and Windows) and integrates with their scientific computing environments.
- Try Shogun in the cloud from your browser.

Shogun is state-of-the-art

- Efficient implementation (from standard to cutting edge algorithms), modern software architecture in C++.
- Easy combination of multiple data representations, algorithm classes and general purpose tools for rapid prototyping of data pipelines.

Shogun is open source

- Free software, community-based development and machine learning education.
- GPLv3 license and working towards BSD compatibility.

The focus of *Shogun* is on kernel machines such as support vector machines for regression and classification problems. *Shogun* also offers a full implementation of Hidden Markov models. The core of *Shogun* is written in C++ and offers interfaces for MATLAB, Octave, Python, R, Java, Lua, Ruby and C#. *Shogun* has been under active development since 1999. Today there is a vibrant user community all over the world

using *Shogun* as a base for research and education, and contributing to the core package.

Supported algorithms

Currently *Shogun* supports the following algorithms:

- Support vector machines
- Dimensionality reduction algorithms, such as PCA, Kernel PCA, Locally Linear Embedding, Hessian Locally Linear Embedding, Local Tangent Space Alignment, Linear Local Tangent Space Alignment, Kernel Locally Linear Embedding, Kernel Local Tangent Space Alignment, Multidimensional Scaling, Isomap, Diffusion Maps, Laplacian Eigenmaps
- Online learning algorithms such as SGD-QN, Vowpal Wabbit
- Clustering algorithms: k-means and GMM
- Kernel Ridge Regression, Support Vector Regression
- Hidden Markov Models
- K-Nearest Neighbors
- Linear discriminant analysis
- Kernel Perceptrons.

Many different kernels are implemented, ranging from kernels for numerical data (such as gaussian or linear kernels) to kernels on special data (such as strings over certain alphabets). The currently implemented kernels for numeric data include:

- linear
- gaussian
- polynomial
- sigmoid kernels

The supported kernels for special data include:

- Spectrum
- Weighted Degree
- Weighted Degree with Shifts

The latter group of kernels allows processing of arbitrary sequences over fixed alphabets such as DNA sequences as well as whole e-mail texts.

Special features

As *Shogun* was developed with bioinformatics applications in mind it is capable of processing huge datasets consisting of up to 10 million samples. *Shogun* supports the use of pre-calculated kernels. It is also possible to use a combined kernel i.e. a kernel consisting of a linear combination of arbitrary kernels over different domains. The coefficients or weights of the linear combination can be learned as well. For this purpose *Shogun* offers a *multiple kernel learning* functionality.

Example of a FeedForward Neural Network built using Shogun :

Feedforward Network for Classification

Feedforward network or multi-layer perceptron defines a mapping $y=f(\mathbf{x};\theta)$

from an input \mathbf{x} to a category y and learns the value of parameters θ

by iterative training that results in the best function approximation. The network is a directed acyclic graph composed of an input layer, an output layer and a few hidden layers.

For example,

$$f(\mathbf{x})=f(3)(f(2)(f(1)(\mathbf{x})))$$

where \mathbf{x}

is the input layer, $f(1)$ and $f(2)$ are hidden layers and $f(3)$

is the output layer.

See chapter 6 in [GBC16] for a detailed introduction.

Example

Imagine we have files with training and test data. We create CDenseFeatures (here 64 bit floats aka RealFeatures) and CBinaryLabels as

```
features_train <- RealFeatures(f_feats_train)
```

```
features_test <- RealFeatures(f_feats_test)
```

```
labels_train <- BinaryLabels(f_labels_train)
```

```
labels_test <- BinaryLabels(f_labels_test)
```

We create instances of CNeuralInputLayer, CNeuralLinearLayer and NeuralSoftmaxLayer which are building blocks of CNeuralNetwork

```
num_feats <- features_train$get_num_features()
```

```
layers <- NeuralLayers()
```

```
layers <- layers$input(num_feats)
```

```
layers <- layers$rectified_linear(10)
```

```
layers <- layers$softmax(2)
```

```
all_layers <- layers$done()
```

We create a CNeuralNetwork instance by using the above layers and randomly initialize the network parameters by sampling from a gaussian distribution.

```
network <- NeuralNetwork(all_layers)
```

```
network$quick_connect()
```

```
network$initialize_neural_network()
```

Before training, we need to set appropriate parameters like regularization coefficient, dropout probabilities, learning rate, etc. as shown below. More parameters can be found in the documentation of CNeuralNetwork.

```
network$set_l2_coefficient(0.01)
```

```
network$set_dropout_hidden(0.5)
```

```
network$set_max_num_epochs(50)
```

```
network$set_gd_mini_batch_size(num_feats)
```

```
network$set_gd_learning_rate(0.1)
```

```
network$set_gd_momentum(0.9)
```

We train the model and apply it to some test data.

```
network$set_labels(labels_train)
```

```
network$train(features_train)
```

```
labels_predict <- network$apply_binary(features_test)
```

We can extract the parameters of the trained network.

```
parameters <- network$get_parameters()
```

Finally, we compute accuracy.

```
am <- AccuracyMeasure()
```

```
accuracy <- am$evaluate(labels_predict, labels_test)
```

Questions:

1. What are features of ACCORD?
2. How to use SHOGUN tool for machine learning.