

MACHINE LEARNING LECTURE NOTES

**DEPARTMENT OF COMPUTER
SCIENCE AND ENGINEERING**

UNIT I

Topics:

Part I: Introduction: Representation and learning: Feature vectors, Feature spaces, Feature Extraction and Feature selection, Learning problem formulation.

What Is Machine Learning?

Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be *predictive* to make predictions in the future, or *descriptive* to gain knowledge from data, or both.

Arthur Samuel, an early American leader in the field of computer gaming and artificial intelligence, coined the term “Machine Learning” in 1959 while at IBM. He defined machine learning as “the field of study that gives computers the ability to learn without being explicitly programmed.” However, there is no universally accepted definition for machine learning. Different authors define the term differently.

Definition of learning

Definition

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

Examples

- i) Handwriting recognition learning problem
 - Task T: Recognising and classifying handwritten words within images
 - Performance P: Percent of words correctly classified
 - Training experience E: A dataset of handwritten words with given classifications
- ii) A robot driving learning problem
 - Task T: Driving on highways using vision sensors
 - Performance measure P: Average distance traveled before an error
 - training experience: A sequence of images and steering commands recorded while observing a human driver
- iii) A chess learning problem
 - Task T: Playing chess
 - Performance measure P: Percent of games won against opponents
 - Training experience E: Playing practice games against itself

Definition

A computer program which learns from experience is called a machine learning program or simply a learning program. Such a program is sometimes also referred to as a learner.

Components of Learning

Basic components of learning process

The learning process, whether by a human or a machine, can be divided into four components, namely, data storage, abstraction, generalization and evaluation. Figure 1.1 illustrates the various components and the steps involved in the learning process.

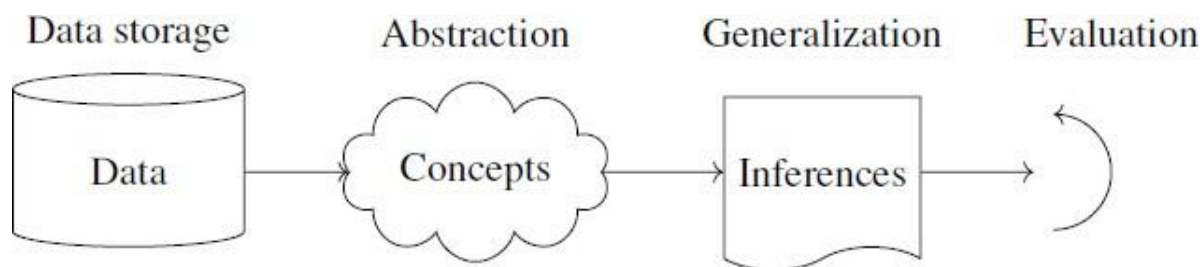


Figure 1.1: Components of learning process

1. Data storage

Facilities for storing and retrieving huge amounts of data are an important component of the learning process. Humans and computers alike utilize data storage as a foundation for advanced reasoning.

- In a human being, the data is stored in the brain and data is retrieved using electrochemical signals.
- Computers use hard disk drives, flash memory, random access memory and similar devices to store data and use cables and other technology to retrieve data.

2. Abstraction

The second component of the learning process is known as abstraction. Abstraction is the process of extracting knowledge about stored data. This involves creating general concepts about the data as a whole. The creation of knowledge involves application of known models and creation of new models.

The process of fitting a model to a dataset is known as training. When the model has been trained, the data is transformed into an abstract form that summarizes the original information.

3. Generalization

The third component of the learning process is known as generalisation. The term generalization describes the process of turning the knowledge about stored data into a form that can be utilized for future action. These actions are to be carried out on tasks that are similar, but not identical, to those what have been seen before. In generalization, the goal is to discover those properties of the data that will be most relevant to future tasks.

4. Evaluation

Evaluation is the last component of the learning process. It is the process of giving feedback to the user to measure the utility of the learned knowledge. This feedback is then utilised to effect improvements in the whole learning process

Applications of machine learning

Application of machine learning methods to large databases is called data mining. In data mining, a large volume of data is processed to construct a simple model with valuable use, for example, having high predictive accuracy.

The following is a list of some of the typical applications of machine learning.

1. In retail business, machine learning is used to study consumer behaviour.
2. In finance, banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market.
3. In manufacturing, learning models are used for optimization, control, and troubleshooting.

4. In medicine, learning programs are used for medical diagnosis.
5. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service.
6. In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is huge; it is constantly growing and searching for relevant information cannot be done manually.
7. In artificial intelligence, it is used to teach a system to learn and adapt to changes so that the system designer need not foresee and provide solutions for all possible situations.
8. It is used to find solutions to many problems in vision, speech recognition, and robotics.
9. Machine learning methods are applied in the design of computer-controlled vehicles to steer correctly when driving on a variety of roads.
10. Machine learning methods have been used to develop programmes for playing games such as chess, backgammon and Go.

Learning Models

Machine learning is concerned with using the right features to build the right models that achieve the right tasks. The basic idea of Learning models has divided into three categories. For a given problem, the collection of all possible outcomes represents the **sample space or instancespace**.

- Using a Logical expression. (**Logical models**)
- Using the Geometry of the instance space. (**Geometric models**)
- Using Probability to classify the instance space. (**Probabilistic models**)
- Grouping and Grading

Logical models

Logical models use a logical expression to divide the instance space into segments and hence construct grouping models. A **logical expression** is an expression that returns a Boolean value, i.e., a True or False outcome. Once the data is grouped using a logical expression, the data is divided into homogeneous groupings for the problem we are trying to solve. For example, for a classification problem, all the instances in the group belong to one class.

There are mainly two kinds of logical models: **Tree models** and **Rule models**.

Rule models consist of a collection of implications or IF-THEN rules. For tree-based models, the ‘if-part’ defines a segment and the ‘then-part’ defines the behaviour of the model for this segment. Rule models follow the same reasoning.

Logical models and Concept learning

To understand logical models further, we need to understand the idea of **Concept Learning**. Concept Learning involves learning logical expressions or concepts from examples. The idea of Concept Learning fits in well with the idea of Machine learning, i.e., inferring a general function from specific training examples. Concept learning forms the basis of both tree-based and rule-based models. More formally, Concept Learning involves acquiring the definition of a general category from a given set of positive and negative training examples of the category. A Formal Definition for Concept Learning is “*The inferring of a Boolean-valued function from training examples of its input and output.*” In concept learning, we only learn a description for the positive class and label everything that doesn’t satisfy that description as negative.

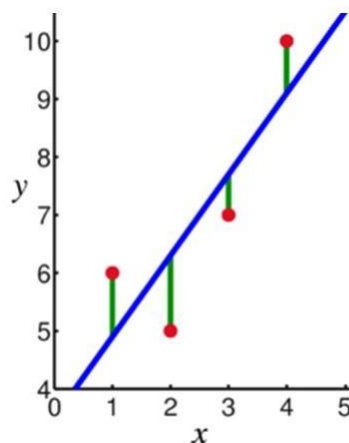
Geometric models

In the previous section, we have seen that with logical models, such as decision trees, a logical expression is used to partition the instance space. Two instances are similar when they end up in the same logical segment. In this section, we consider models that define similarity by considering the geometry of the instance space. In Geometric models, features could be described as points in two dimensions (x - and y -axis) or a three-dimensional space (x , y , and z). Even when features are not intrinsically geometric, they could be modelled in a geometric manner (for example, temperature as a function of time can be modelled in two axes). In geometric models, there are two ways we could impose similarity.

- We could use geometric concepts like **lines or planes to segment (classify)** the instance space. These are called **Linear models**.
- Alternatively, we can use the geometric notion of distance to represent similarity. In this case, if two points are close together, they have similar values for features and thus can be classed as similar. We call such models as **Distance-based models**.

Linear models

Linear models are relatively simple. In this case, the function is represented as a linear combination of its inputs. Thus, if x_1 and x_2 are two scalars or vectors of the same dimension and a and b are arbitrary scalars, then $ax_1 + bx_2$ represents a linear combination of x_1 and x_2 . In the simplest case where $f(x)$ represents a straight line, we have an equation of the form $f(x) = mx + c$ where c represents the intercept and m represents the slope.



Linear models are **parametric**, which means that they have a fixed form with a small number of numeric parameters that need to be learned from data. For example, in $f(x) = mx + c$, m and c are the parameters that we are trying to learn from the data. This technique is different from tree or rule models, where the structure of the model (e.g., which features to use in the tree, and where) is not fixed in advance.

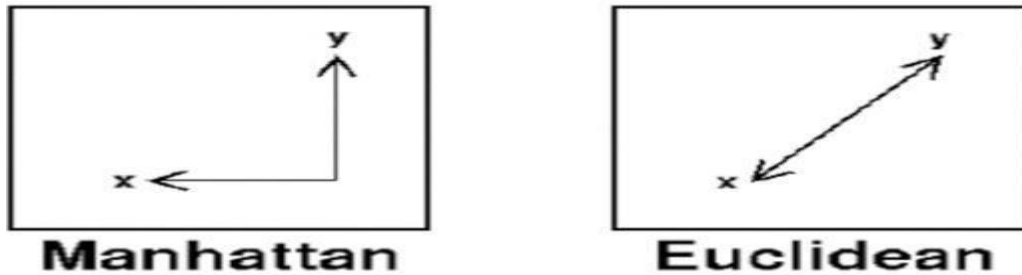
Linear models are **stable**, i.e., small variations in the training data have only a limited impact on the learned model. In contrast, **tree models tend to vary more with the training data**, as the choice of a different split at the root of the tree typically means that the rest of the tree is different as well. As a result of having relatively few parameters, Linear models have **low variance and high bias**. This implies that **Linear models are less likely to overfit the training data** than some other models. However, they are more likely to underfit. For example, if we want to learn the boundaries between countries based on labelled data, then linear models are not likely to give a good approximation.

Distance-based models

Distance-based models are the second class of Geometric models. Like Linear models, distance-based models are based on the geometry⁴ of data. As the name implies, distance-based models work on the concept of distance. In the context of Machine learning, the concept of distance is not based on merely the physical distance between two points. Instead, we could think of the distance

between two points considering the **mode of transport** between two points. Travelling between two cities by plane

covers less distance physically than by train because a plane is unrestricted. Similarly, in chess, the concept of distance depends on the piece used – for example, a Bishop can move diagonally. Thus, depending on the entity and the mode of travel, the concept of distance can be experienced differently. The distance metrics commonly used are **Euclidean**, **Minkowski**, **Manhattan**, and **Mahalanobis**.



Distance is applied through the concept of **neighbours and exemplars**. Neighbours are points in proximity with respect to the distance measure expressed through exemplars. Exemplars are either **centroids** that find a centre of mass according to a chosen distance metric or **medoids** that find the most centrally located data point. The most commonly used centroid is the arithmetic mean, which minimises squared Euclidean distance to all other points.

Notes:

- The **centroid** represents the geometric centre of a plane figure, i.e., the arithmetic mean position of all the points in the figure from the centroid point. This definition extends to any object in n -dimensional space: its centroid is the mean position of all the points.
- **Medoids** are similar in concept to means or centroids. Medoids are most commonly used on data when a mean or centroid cannot be defined. They are used in contexts where the centroid is not representative of the dataset, such as in image data.

Examples of distance-based models include the **nearest-neighbour** models, which use the training data as exemplars – for example, in classification. The **K-means clustering** algorithm also uses exemplars to create clusters of similar data points.

Probabilistic models

The third family of machine learning algorithms is the probabilistic models. We have seen before that the k-nearest neighbour algorithm uses the idea of distance (e.g., Euclidian distance) to classify entities, and logical models use a logical expression to partition the instance space. In this section, we see how the **probabilistic models use the idea of probability to classify new entities**.

Probabilistic models see features and target variables as random variables. The process of modelling represents and **manipulates the level of uncertainty** with respect to these variables. There are two types of probabilistic models: **Predictive and Generative**. Predictive probability models use the idea of a **conditional probability** distribution $P(Y|X)$ from which Y can be predicted from X . Generative models estimate the **joint distribution** $P(Y, X)$. Once we know the joint distribution for the generative models, we can derive any conditional or marginal distribution involving the same variables. Thus, the generative model is capable of creating new data points and their labels, knowing the joint probability distribution. The joint distribution looks for a relationship between two variables. Once this relationship is inferred, it is possible to infer new data points.

Naïve Bayes is an example of a probabilistic classifier.

We can do this using the **Bayes rule** defined as

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

The Naïve Bayes algorithm is based on the idea of **Conditional Probability**. **Conditional probability is based on finding the** probability that something will happen, *given that something else* has already happened. The task of the algorithm then is to look at the evidence and to determine the likelihood of a specific class and assign a label accordingly to each entity.

Some broad categories of models:

Geometric models

E.g. K-nearest neighbors, linear regression, support vector machine, logistic regression, ...

Probabilistic models

Naïve Bayes, Gaussian process regression, conditional random field, ...

Logical models

Decision tree, random forest, ...

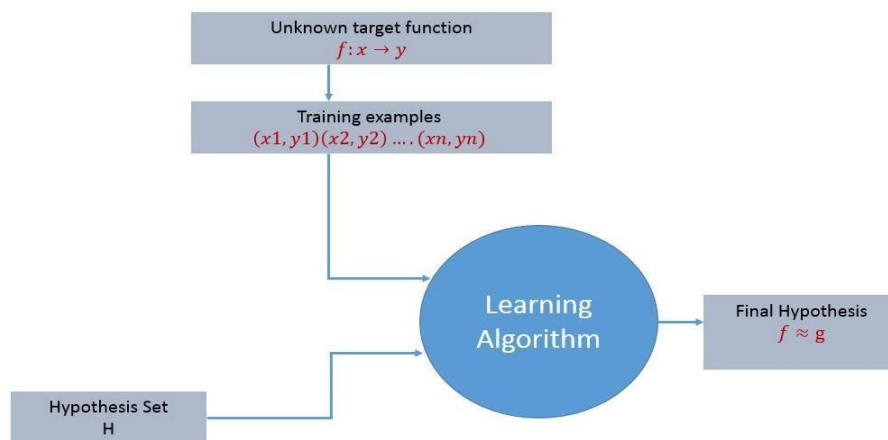
Grouping and Grading

Grading vs grouping is an orthogonal categorization to geometric-probabilistic-logical-compositional.

- Grouping models break the instance space up into groups or segments and in each segment apply a very simple method (such as majority class).
 - E.g. decision tree, KNN.
- Grading models form one global model over the instance space.
 - E.g. Linear classifiers – Neural networks

Designing a Learning System

For any learning system, we must be knowing the three elements — **T (Task)**, **P (Performance Measure)**, and **E (Training Experience)**. At a high level, the process of learning system looks as below.



The learning process starts with task T, performance measure P and training experience E and objective are to find an unknown target function. The target function is an exact knowledge to be learned from the training experience and its unknown. For example, in a case of credit approval, the learning system will have customer application records as experience and task would be to classify whether the given customer application is eligible for a loan. So in this case, the training examples can be represented as $(x1, y1)(x2, y2) \dots (xn, yn)$ where X represents customer application details and y represents the status of credit approval.

With these details, what is that exact knowledge to be learned from the training experience?

So the target function to be learned in the credit approval learning system is a mapping function $f: X \rightarrow y$. This function represents the exact knowledge defining the relationship between input variable X and output variable y.

Design of a learning system

Just now we looked into the learning process and also understood the goal of the learning. When we want to design a learning system that follows the learning process, we need to consider a few design choices. The design choices will be to decide the following key components:

1. Type of training experience
2. Choosing the Target Function
3. Choosing a representation for the Target Function
4. Choosing an approximation algorithm for the Target Function

5. The final Design

We will look into the game - checkers learning problem and apply the above design choices. For a checkers learning problem, the three elements will be,

1. *Task T: To play checkers*
2. *Performance measure P: Total percent of the game won in the tournament.*
3. *Training experience E: A set of games played against itself*

Type of training experience

During the design of the checker's learning system, the type of training experience available for a learning system will have a significant effect on the success or failure of the learning.

1. **Direct or Indirect training experience** — In the case of direct training experience, an individual board states and correct move for each board state are given. In case of indirect training experience, the move sequences for a game and the final result (win, loss or draw) are given for a number of games. How to assign credit or blame to individual moves is the credit assignment problem.
2. **Teacher or Not** — Supervised — The training experience will be labeled, which means, all the board states will be labeled with the correct move. So the learning takes place in the presence of a supervisor or a teacher. Unsupervised — The training experience will be unlabeled, which means, all the board states will not have the moves. So the learner generates random games and plays against itself with no supervisor or teacher involvement.
Semi-supervised — Learner generates game states and asks the teacher for help in finding the correct move if the board state is confusing.
3. **Is the training experience good** — Do the training examples represent the distribution of examples over which the final system performance will be measured? Performance is best when training examples and test examples are from the same/a similar distribution.

The checker player learns by playing against oneself. Its experience is indirect. It may not encounter moves that are common in human expert play. Once the proper training experience is available, the next design step will be choosing the Target Function.

Choosing the Target Function

When you are playing the checkers game, at any moment of time, you make a decision on choosing the best move from different possibilities. You think and apply the learning that you have gained from the experience. Here the learning is, for a specific board, you move a checker such that your board state tends towards the winning situation. Now the same learning has to be defined in terms of the target function.

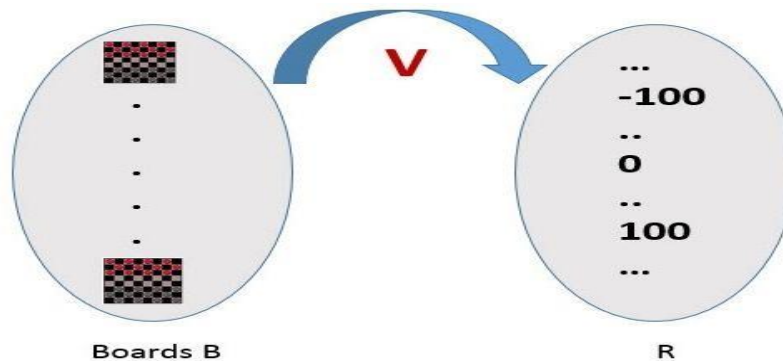
Here there are 2 considerations — direct and indirect experience.

- **During the direct experience**, the checker's learning system, it needs only to learn how to choose the best move among some large search space. We need to find a target function that will help us choose the best move among alternatives. Let us call this function ChooseMove

and use the notation **ChooseMove : B → M** to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

- **When there is an indirect experience**, it becomes difficult to learn such function. How about assigning a real score to the board state.

So the function be **V : B → R** indicating that this accepts as input any board from the set of legal board states B and produces an output a real score. This function assigns the higher scores to better board states.



If the system can successfully learn such a target function V, then it can easily use it to select the best move from any board position.

Let us therefore define the target value $V(b)$ for an arbitrary board state b in B, as follows:

1. if b is a final board state that is won, then $V(b) = 100$
2. if b is a final board state that is lost, then $V(b) = -100$
3. if b is a final board state that is drawn, then $V(b) = 0$
4. if b is not a final state in the game, then $V(b) = V(b')$, where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

The (4) is a recursive definition and to determine the value of $V(b)$ for a particular board state, it performs the search ahead for the optimal line of play, all the way to the end of the game. So this definition is not efficiently computable by our checkers playing program, we say that it is a nonoperational definition.

The goal of learning, in this case, is to discover an operational description of V ; that is, a description that can be used by the checkers-playing program to evaluate states and select moves within realistic time bounds.

It may be very difficult in general to learn such an operational form of V perfectly. We expect learning algorithms to acquire only some approximation to the target function \hat{V} .

Choosing a representation for the Target Function

Now that we have specified the ideal target function V, we must choose a representation that the learning program will use to describe the function \hat{V} that it will learn. As with earlier design choices, we again have many options. We could, for example, allow the program to represent using a large table with a distinct entry specifying the value for each distinct board state. Or we could allow it to represent using a collection of rules that match against features of the board state, or a quadratic polynomial function of predefined board features, or an artificial

neural network. In general, this choice of representation involves a crucial tradeoff. On one hand, we wish to pick a very expressive representation to allow representing as close an approximation as possible to the ideal target function V .

On the other hand, the more expressive the representation, the more training data the program will require in order to choose among the alternative hypotheses it can represent. To keep the discussion brief, let us choose a simple representation:

for any given board state, the function \hat{V} will be calculated as a linear combination of the following board features:

- ☐ $x_1(b)$ — number of black pieces on board b
- ☐ $x_2(b)$ — number of red pieces on b
- ☐ $x_3(b)$ — number of black kings on b
- ☐ $x_4(b)$ — number of red kings on b
- ☐ $x_5(b)$ — number of red pieces threatened by black (i.e., which can be taken on black's next turn)
- ☐ $x_6(b)$ — number of black pieces threatened by red

$$\hat{V} = w_0 + w_1 \cdot x_1(b) + w_2 \cdot x_2(b) + w_3 \cdot x_3(b) + w_4 \cdot x_4(b) + w_5 \cdot x_5(b) + w_6 \cdot x_6(b)$$

Where w_0 through w_6 are numerical coefficients or weights to be obtained by a learning algorithm.

Weights w_1 to w_6 will determine the relative importance of different board features.

Specification of the Machine Learning Problem at this time — Till now we worked on choosing the type of training experience, choosing the target function and its representation. The checkers learning task can be summarized as below.

- ☐ **Task T : Play Checkers**
- ☐ **Performance Measure : % of games won in world tournament**
- ☐ **Training Experience E : opportunity to play against itself**
- ☐ **Target Function : $V : \text{Board} \rightarrow \mathbb{R}$**
- ☐ **Target Function Representation : $\hat{V} = w_0 + w_1 \cdot x_1(b) + w_2 \cdot x_2(b) + w_3 \cdot x_3(b) + w_4 \cdot x_4(b) + w_5 \cdot x_5(b) + w_6 \cdot x_6(b)$**

The first three items above correspond to the specification of the learning task, whereas the final two items constitute design choices for the implementation of the learning program.

Choosing an approximation algorithm for the Target Function

Generating training data —

To train our learning program, we need a set of training data, each describing a specific board state b and the training value $V_{\text{train}}(b)$ for b . Each training example is an ordered pair $\langle b, V_{\text{train}}(b) \rangle$

For example, a training example may be $\langle (x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0), +100 \rangle$. This is an example where black has won the game since $x_2 = 0$ or red has no remaining pieces. However, such clean values of $V_{\text{train}}(b)$ can be obtained only for board value b that are clear win, loss or draw.

In above case, assigning a training value $V_{\text{train}}(b)$ for the specific boards b that are clean win, loss or draw is direct as they are direct training experience. But in the case of indirect training experience, assigning a training value $V_{\text{train}}(b)$ for the intermediate boards is difficult. In such case, the training values are updated using temporal difference learning. **Temporal difference (TD) learning is a concept central to reinforcement learning, in which learning happens through the iterative correction of your estimated returns towards a more accurate target return.**

Let $\text{Successor}(b)$ denotes the next board state following b for which it is again the program's turn to move. \hat{V} is the learner's current approximation to V . Using these information, assign the training value of $V_{\text{train}}(b)$ for any intermediate board state b as below :

$$V_{\text{train}}(b) \leftarrow \hat{V}(\text{Successor}(b))$$

Adjusting the weights

Now its time to define the learning algorithm for choosing the weights and best fit the set of training examples. One common approach is to define the best hypothesis as that which minimizes the squared error E between the training values and the values predicted by the hypothesis \hat{V} .

$$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in \text{training examples}} (V_{train}(b) - \hat{V}(b))^2$$

The learning algorithm should incrementally refine weights as more training examples become available and it needs to be robust to errors in training data. Least Mean Square (LMS) training rule is the one training algorithm that will adjust weights a small amount in the direction that reduces the error.

The LMS algorithm is defined as follows:

For each training example b

1. Compute $\text{error}(b) = V_{train}(b) - \hat{V}(b)$
2. for each board feature x_i , update weight w_i

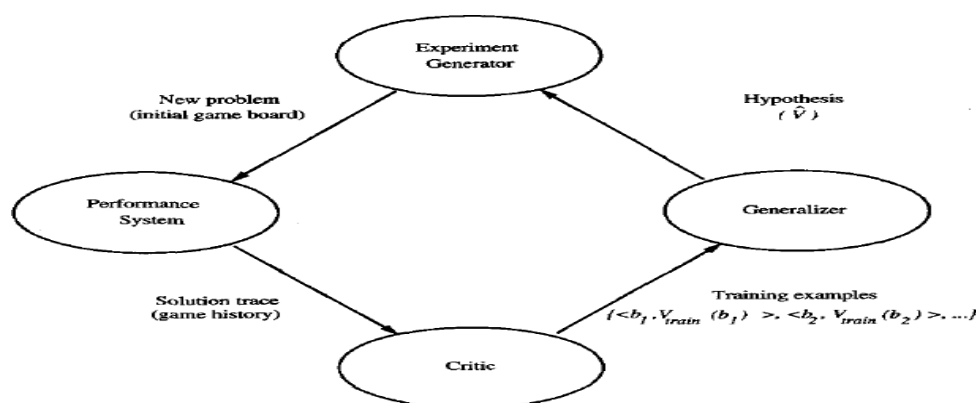
$$w_i \leftarrow w_i + \mu \cdot \text{error}(b) \cdot x_i$$

Here μ is a small constant (e.g., 0.1) that moderates the size of the weight update

Final Design for Checkers Learning system

The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems.

1. The performance System — Takes a new board as input and outputs a trace of the game it played against itself.
2. The Critic — Takes the trace of a game as an input and outputs a set of training examples of the target function.
3. The Generalizer — Takes training examples as input and outputs a hypothesis that estimates the target function. Good generalization to new cases is crucial.
4. The Experiment Generator — Takes the current hypothesis (currently learned function) as input and outputs a new problem (an initial board state) for the performance system to explore.



Final design of the checkers learning program.

Feature, Feature vector, feature space, feature extraction and feature selection

Feature: is a list of numbers eg: age, name, height, weight etc., that means every column is a feature in relational table.

Feature Vector is representation of particular row in relational table. Each row is a feature vector, row 'n' is a feature vector for the 'n'th sample.

Feature Set: Help to predict the output variable.

Example: To predict the age of particular person we need to know the year of birth. Here Feature Set = Year of Birth.

Normally good feature set can be identified using expert domain knowledge or mathematical approach.

ID	First Name	Last Name	Email	Year of Birth
1	Peter	Lee	plee@university.edu	1992
2	Jonathan	Edwards	jedwards@university.edu	1994
3	Marilyn	Johnson	mjohnson@university.edu	1993
6	Joe	Kim	jkim@university.edu	1992
12	Haley	Martinez	hmartinez@university.edu	1993
14	John	Mfume	jmfume@university.edu	1991
15	David	Letty	dletty@university.edu	1995

Table: Students

Feature
Vector

In machine learning, a **feature vector** is a numerical representation of an object or entity in a dataset, which is used as input to a machine learning algorithm. A feature vector is a one-dimensional array of numbers, where each number corresponds to a specific feature or attribute of the object being represented. The feature vector is derived from the raw data of the object, through a process of feature extraction or feature engineering.

The feature space is the set of all possible feature vectors that can be created from the raw data. Each feature vector represents a point in the feature space. The dimensionality of the feature space is determined by the number of features in the feature vector. For example, if the feature vector has three features, the feature space would be three-dimensional.

Let's consider an example to illustrate this concept. Suppose we have a dataset of houses for sale, and we want to predict their prices based on a set of features. The features we are interested in are the square footage, the number of bedrooms, the number of bathrooms, and the location of the house.

We can represent each house as a feature vector, where the first element is the square footage, the second element is the number of bedrooms, the third element is the number of bathrooms, and the fourth element is a one-hot encoding of the location. For example, if there are three possible locations (A, B, C), the fourth element of the feature vector would be a vector of length three, where the element corresponding to the location of the house is 1 and the other elements are 0.

Suppose we have a dataset of 100 houses, with the following feature vectors:

Square Footage	Bedrooms	Bathrooms	Location
		12	

Square Footage	Bedrooms	Bathrooms	Location
1500	2	1	A
2000	3	2	B
1200	2	1	C
...

The feature space for this dataset would be four-dimensional, with each house represented as a point in this four-dimensional space. The machine learning algorithm would use this feature space to learn a function that maps the feature vectors to the corresponding prices of the houses. The goal of the algorithm is to find a function that accurately predicts the price of a house based on its features.

Feature selection and feature extraction are two different techniques used in machine learning to reduce the dimensionality of a dataset by selecting or creating a subset of the most relevant features that are likely to contribute the most to the predictive performance of a model. Although they have a similar goal, they differ in how they achieve it.

Feature selection involves selecting a subset of the most important features from the original dataset, while discarding the remaining features. This can be done by using statistical methods to measure the correlation or relevance of each feature to the target variable, or by using domain knowledge to manually select the most informative features. The selected features can then be used as input to a machine learning algorithm to build a model.

On the other hand, **feature extraction** involves creating new features by transforming or combining the original features in some way, such that the new features capture the most important information from the original features. This can be done using techniques such as principal component analysis (PCA), linear discriminant analysis (LDA), or non-negative matrix factorization (NMF). The new features can then be used as input to a machine learning algorithm to build a model.

In both cases, the goal is to reduce the dimensionality of the dataset, while preserving the most relevant information for building an accurate machine learning model. However, feature selection tends to be more straightforward and interpretable, as it simply involves selecting a subset of the original features. Feature extraction can be more complex, as it involves creating new features that may not have a direct interpretation in terms of the original dataset.

Learning problem formulation:

In machine learning, the problem formulation refers to the process of defining the specific task that the machine learning algorithm is intended to solve. The formulation of the problem includes defining the input and output data, selecting the appropriate algorithm, and determining the evaluation metrics.

Here are the key steps involved in formulating a machine learning problem:

1. Define the problem: The first step is to clearly define the problem you want to solve. For example, you may want to predict the price of a house based on its location, size, and number of rooms.
2. Collect the data: The next step is to collect the data that will be used to train and test the machine learning model. This data should be representative of the problem you are trying to solve.
3. Prepare the data: The data collected may not be in the appropriate format for machine learning. Therefore, it may need to be preprocessed, which could include tasks such as cleaning, normalization, and feature engineering.
4. Choose the algorithm: After the data has been preprocessed, you need to select the appropriate algorithm to solve the problem. This may involve choosing from supervised, unsupervised, or reinforcement learning algorithms, depending on the type of data and the problem you are trying to solve.
5. Train the model: Once the algorithm has been chosen, the next step is to train the model using the prepared data.
6. Evaluate the model: After the model has been trained, it is important to evaluate its performance to determine how well it is solving the problem. This can be done by using evaluation metrics such as accuracy, precision, recall, and F1 score.
7. Deploy the model: Once the model has been trained and evaluated, it can be deployed to make predictions on new, unseen data.

In summary, the problem formulation involves defining the problem, collecting and preparing the data, selecting the appropriate algorithm, training and evaluating the model, and finally deploying it for making predictions on new data.

UNIT-I (Part-II)

Topics: Types of Machine Learning Algorithms: Parametric and Nonparametric Machine Learning Algorithms. Supervised, Unsupervised, Semi-Supervised and Reinforced Learning.

Types of Machine learning algorithms:

In machine learning, parametric and non-parametric are two different approaches for modeling the underlying relationship between the input and output variables.

Parametric learning involves making assumptions about the functional form of the relationship between the input and output variables, and then estimating the parameters of that function using the training data. In other words, the model structure is fixed and the parameters are learned from the data. Examples of parametric models include linear regression, logistic regression, and neural networks. Once the parameters are learned, the model can be used to make predictions on new data.

Non-parametric learning, on the other hand, makes no assumptions about the functional form of the relationship between the input and output variables. Instead, it uses the training data to build a flexible model that can fit any underlying relationship between the variables. Non-parametric models include decision trees, k-nearest neighbors, and support vector machines. Non-parametric models generally require more data than parametric models, but they can be more accurate when the underlying relationship is complex and difficult to model with a fixed functional form.

Here are some key differences between parametric and non-parametric learning:

- **Model flexibility:** Parametric models have a fixed model structure, which limits their ability to capture complex relationships between variables. Non-parametric models are more flexible and can fit a wider range of relationships between variables.
- **Parameter estimation:** In parametric models, the parameters are estimated using the training data. Non-parametric models do not have fixed parameters, but instead build a flexible model based on the training data.
- **Data requirements:** Parametric models generally require less data than non-parametric models, as they make assumptions about the functional form of the relationship between variables. Non-parametric

models require more data, as they need to build a flexible model that can fit any underlying relationship between variables.

- **Interpretability:** Parametric models are generally more interpretable than non-parametric models, as they have a fixed model structure that can be easily understood. Non-parametric models can be more difficult to interpret, as they do not have a fixed model structure.

In summary, parametric learning makes assumptions about the functional form of the relationship between the input and output variables, while non-parametric learning makes no such assumptions and builds a flexible model based on the training data.

Regenerate response

Types of Learning

In general, machine learning algorithms can be classified into three types.

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Supervised learning

A training set of examples with the correct responses (targets) is provided and, based on this training set, the algorithm generalises to respond correctly to all possible inputs. This is also called learning from exemplars. Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

In supervised learning, each example in the training set is a pair consisting of an input object (typically a vector) and an output value. A supervised learning algorithm analyzes the training data and produces a function, which can be used for mapping new examples. In the optimal case, the function will correctly determine the class labels for unseen instances. Both classification and regression problems are supervised learning problems. A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems.

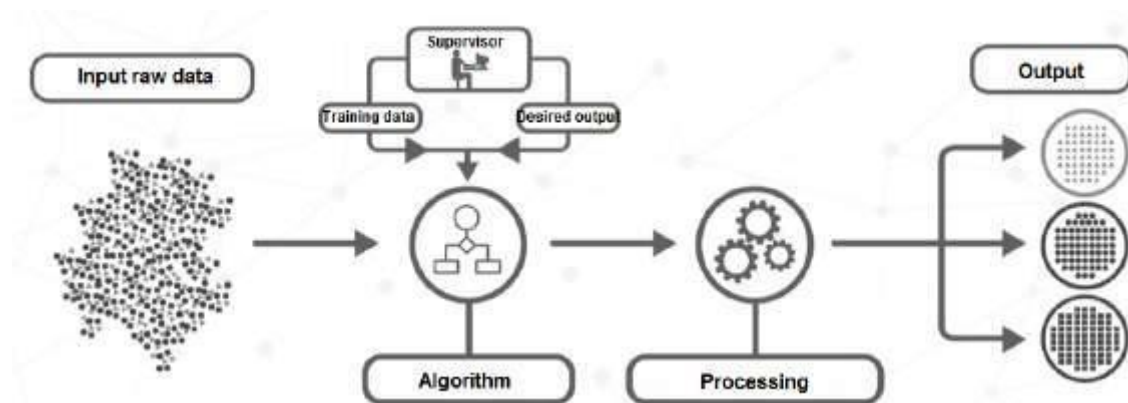


Figure 1.4: Supervised learning

Remarks

A “supervised learning” is so called because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers (that is, the correct outputs), the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Example

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients and each patient is labeled as “healthy” or “sick”.

gender	age	label
M	48	sick
M	67	sick
F	53	healthy
M	49	healthy
F	34	sick
M	21	healthy

Unsupervised learning

Correct responses are not provided, but instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorised together. The statistical approach to unsupervised learning is known as density estimation.

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. In unsupervised learning algorithms, a classification or categorization is not included in the observations. There are no output values and so there is no estimation of functions. Since the examples given to the learner are unlabeled, the accuracy of the structure that is output by the algorithm cannot be evaluated. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns Or grouping in data.

Example

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients.

gender	age
M	48
M	67
F	53
M	49
F	34
M	21

Based on this data, can we infer anything regarding the patients entering the clinic?

Reinforcement learning

This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right. Reinforcement learning is sometime called learning with a critic because of this monitor that scores the answer, but does not suggest improvements.

Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards. A learner (the program) is not told what actions to take as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situations and, through that, all subsequent rewards.

Example

Consider teaching a dog a new trick: we cannot tell it what to do, but we can reward/punish it if it does the right/wrong thing. It has to find out what it did that made it get the reward/punishment. We can use a similar method to train computers to do many tasks, such as playing backgammon or chess, scheduling jobs, and controlling robot limbs. Reinforcement learning is different from supervised learning. Supervised learning is learning from examples provided by a knowledgeable expert.

Supervised Learning	Unsupervised Learning
Supervised learning algorithms are trained using labeled data.	Unsupervised learning algorithms are trained using unlabeled data.
Supervised learning model takes direct feedback to check if it is predicting correct output or not.	Unsupervised learning model does not take any feedback.
Supervised learning model predicts the output.	Unsupervised learning model finds the hidden patterns in data.
In supervised learning, input data is provided to the model along with the output.	In unsupervised learning, only input data is provided to the model.
The goal of supervised learning is to train the model so that it can predict the output when it is given new data.	The goal of unsupervised learning is to find the hidden patterns and useful insights from the unknown dataset.
Supervised learning needs supervision to train the model.	Unsupervised learning does not need any supervision to train the model.
Supervised learning can be categorized in Classification and Regression problems.	Unsupervised Learning can be classified in Clustering and Associations problems.
Supervised learning can be used for those cases where we know the input as well as corresponding outputs.	Unsupervised learning can be used for those cases where we have only input data and no corresponding output data.
Supervised learning model produces an accurate result.	Unsupervised learning model may give less accurate result as compared to supervised learning.
Supervised learning is not close to true Artificial intelligence as in this, we first train the model for each data, and then only it can predict the correct output.	Unsupervised learning is more close to the true Artificial Intelligence as it learns similarly as a child learns daily routine things by his experiences.
It includes various algorithms such as Linear Regression, Logistic Regression, Support Vector Machine, Multi-class Classification, Decision tree, Bayesian Logic, etc.	It includes various algorithms such as Clustering, KNN, and Apriori algorithm.

Understanding data

Since an important component of the machine learning process is data storage, we briefly consider in this section the different types and forms of data that are encountered in the machine learning process.

Unit of observation

By a *unit of observation* we mean the smallest entity with measured properties of interest for a study.

Examples

- A person, an object or a thing
- A time point
- A geographic region

- A measurement

Sometimes, units of observation are combined to form units such as person-years.

Examples and features

Datasets that store the units of observation and their properties can be imagined as collections of data consisting of the following:

- **Examples**

An “example” is an instance of the unit of observation for which properties have been recorded. An “example” is also referred to as an “instance”, or “case” or “record.” (It may be noted that the word “example” has been used here in a technical sense.)

- **Features**

A “feature” is a recorded property or a characteristic of examples. It is also referred to as “attribute”, or “variable” or “feature.”

Examples for “examples” and “features”

1. Cancer detection

Consider the problem of developing an algorithm for detecting cancer. In this study we note the following.

- The units of observation are the patients.
- The examples are members of a sample of cancer patients.
- The following attributes of the patients may be chosen as the features:
 - gender
 - age
 - blood pressure
 - the findings of the pathology report after a biopsy

2. Pet selection

Suppose we want to predict the type of pet a person will choose.

- The units are the persons.
- The examples are members of a sample of persons who own pets.
- The features might include age, home region, family income, etc. of persons who ownpets.

year	model	price	mileage	color	transmission
2011	SEL	21992	7413	Yellow	AUTO
2011	SEL	20995	10926	Gray	AUTO
2011	SEL	19995	7351	Silver	AUTO
2011	SEL	17809	11613	Gray	AUTO
2012	SE	17500	8367	White	MANUAL
2010	SEL	17495	25125	Silver	AUTO
2011	SEL	17000	27393	Blue	AUTO
2010	SEL	16995	21026	Silver	AUTO
2011	SES	16995	32655	Silver	AUTO

Figure 1.2: Example for “examples” and “features” collected in a matrix format (data relates to automobiles and their features)

3. Spam e-mail

Let it be required to build a learning algorithm to identify spam e-mail.

- (a) The unit of observation could be an e-mail messages.
- (b) The examples would be specific messages.
- (c) The features might consist of the words used in the messages.

Examples and features are generally collected in a “matrix format”. Fig. 1.2 shows such a data set.

Different forms of data

1. Numeric data

If a feature represents a characteristic measured in numbers, it is called a numeric feature.

2. Categorical or nominal

A categorical feature is an attribute that can take on one of a limited, and usually fixed, number of possible values on the basis of some qualitative property. A categorical feature is also called a nominal feature.

3. Ordinal data

This denotes a nominal variable with categories falling in an ordered list. Examples include clothing sizes such as small, medium, and large, or a measurement of customer satisfaction on a scale from “not at all happy” to “very happy.”

Examples

In the data given in Fig.1.2, the features “year”, “price” and “mileage” are numeric and the features “model”, “color” and “transmission” are categorical.

Steps for PCA Algorithm

1. Standardize the data: PCA requires standardized data, so the first step is to standardize the data to ensure that all variables have a mean of 0 and a standard deviation of 1.
2. Calculate the covariance matrix: The next step is to calculate the covariance matrix of the standardized data. This matrix shows how each variable is related to every other variable in the dataset.
3. Calculate the eigenvectors and eigenvalues: The eigenvectors and eigenvalues of the covariance matrix are then calculated. The eigenvectors represent the directions in which the data varies the most, while the eigenvalues represent the amount of variation along each eigenvector.
4. Choose the principal components: The principal components are the eigenvectors with the highest eigenvalues. These components represent the directions in which the data varies the most and are used to transform the original data into a lower-dimensional space.
5. Transform the data: The final step is to transform the original data into the lower-dimensional space defined by the principal components.

- PCA is used to visualize multidimensional data.
- It is used to reduce the number of dimensions in healthcare data.
- PCA can help resize an image.
- It can be used in finance to analyze stock data and forecast returns.
- PCA helps to find patterns in the high-dimensional datasets.

Advantages of PCA

In terms of data analysis, PCA has a number of benefits, including:

1. **Dimensionality reduction:** By determining the most crucial features or components, PCA reduces the dimensionality of the data, which is one of its primary benefits. This can be helpful when the initial data contains a lot of variables and is therefore challenging to visualize or analyze.
2. **Feature Extraction:** PCA can also be used to derive new features or elements from the original data that might be more insightful or understandable than the original features. This is particularly helpful when the initial features are correlated or noisy.
3. **[Data visualization](#):** By projecting the data onto the first few principal components, PCA can be used to visualize high-dimensional data in two or three dimensions. This can aid in locating data patterns or clusters that may not have been visible in the initial high-dimensional space.
4. **Noise Reduction:** By locating the underlying signal or pattern in the data, PCA can also be used to lessen the impacts of noise or measurement errors in the data.
5. **Multicollinearity:** When two or more variables are strongly correlated, there is multicollinearity in the data, which PCA can handle. PCA can lessen the impacts of multicollinearity on the analysis by identifying the most crucial features or components.

Disadvantages of PCA

1. **Interpretability:** Although principal component analysis (PCA) is effective at reducing the dimensionality of data and spotting patterns, the resulting principal components are not always simple to understand or describe in terms of the original features.
2. **Information loss:** PCA involves choosing a subset of the most crucial features or components in order to reduce the dimensionality of the data. While this can be helpful for streamlining the data and lowering noise, if crucial features are not included in the components chosen, information loss may also result.
3. **Outliers:** Because PCA is susceptible to anomalies in the data, the resulting principal components may be significantly impacted. The covariance matrix can be distorted by outliers, which can make it harder to identify the most crucial characteristics.
4. **Scaling:** PCA makes the assumption that the data is scaled and centralized, which can be a drawback in some circumstances. The resulting principal components might not correctly depict the underlying patterns in the data if the data is not scaled properly.
5. **Computing complexity:** For big datasets, it may be costly to compute the eigenvectors and eigenvalues of the covariance matrix. This may restrict PCA's ability to scale and render it useless for some uses.

Uses of PCA

PCA is a widely used technique in data analysis and has a variety of applications, including:

1. **Data compression:** PCA can be used to reduce the dimensionality of high-dimensional datasets, making them easier to store and analyze.
2. **Feature extraction:** PCA can be used to identify the most important features in a dataset, which can be used to build predictive models.
3. **Visualization:** PCA can be used to visualize high-dimensional data in two or three dimensions, making it easier to understand and interpret.
4. **Data pre-processing:** PCA can be used as a pre-processing step for other machine learning algorithms, such as clustering and classification.

How Does Principal Component Analysis Work?

1. Normalize the Data

Standardize the data before performing PCA. This will ensure that each feature has a mean = 0 and variance = 1.

2. Build the Covariance Matrix

Construct a square matrix to express the correlation between two or more features in a multidimensional dataset.

3. Find the Eigenvectors and Eigenvalues

Calculate the eigenvectors/unit vectors and eigenvalues. Eigenvalues are scalars by which we multiply the eigenvector of the covariance matrix.

4. Sort the Eigenvectors in Highest to Lowest Order and Select the Number of Principal Components.

What Is Principal Component Analysis?

Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to

explore and visualize and make analyzing data points much easier and faster for machine learning algorithms without extraneous variables to process.

So, to sum up, the idea of PCA is simple — **reduce the number of variables of a data set, while preserving as much information as possible.**

Step-by-Step Explanation of PCA

STEP 1: STANDARDIZATION

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

More specifically, the reason why it is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (for example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

$$z = \frac{value - mean}{standard\ deviation}$$

Once the standardization is done, all the variables will be transformed to the same scale.

STEP 2: COVARIANCE MATRIX COMPUTATION

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them. Because sometimes, variables are highly correlated in such a way that they contain redundant information. So, in order to identify these correlations, we compute the covariance matrix.

The covariance matrix is a $p \times p$ symmetric matrix (where p is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with 3 variables x , y , and z , the covariance matrix is a 3×3 data matrix of this form:

$$\begin{bmatrix} Cov(x, x) & Cov(x, y) & Cov(x, z) \\ Cov(y, x) & Cov(y, y) & Cov(y, z) \\ Cov(z, x) & Cov(z, y) & Cov(z, z) \end{bmatrix}$$

Covariance Matrix for 3-Dimensional Data.

Since the covariance of a variable with itself is its variance ($Cov(a,a)=Var(a)$), in the main diagonal (Top left to bottom right) we actually have the variances of each initial variable. And since the covariance is commutative ($Cov(a,b)=Cov(b,a)$), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

What do the covariances that we have as entries of the matrix tell us about the correlations between the variables?

It's actually the sign of the covariance that matters:

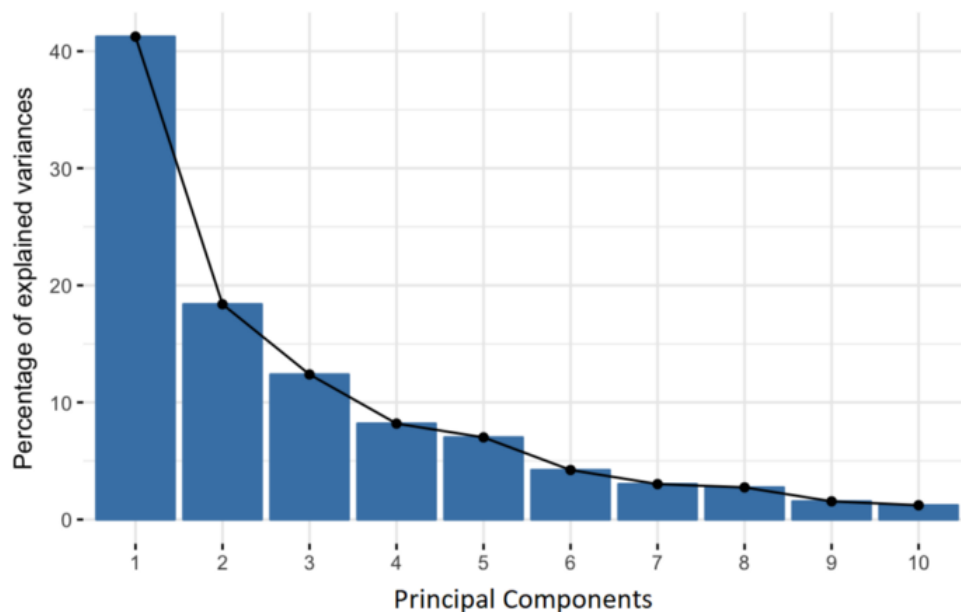
- If positive then: the two variables increase or decrease together (correlated)
- If negative then: one increases when the other decreases (Inversely correlated)

Now that we know that the covariance matrix is not more than a table that summarizes the correlations between all the possible pairs of variables, let's move to the next step.

STEP 3: COMPUTE THE EIGENVECTORS AND EIGENVALUES OF THE COVARIANCE MATRIX TO IDENTIFY THE PRINCIPAL COMPONENTS

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the **principal components** of the data. Before getting to the explanation of these concepts, let's first understand what do we mean by principal components.

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on, until having something like shown in the scree plot below.



Percentage of Variance (Information) for each by PC.

Organizing information in principal components this way, will allow you to reduce dimensionality without losing much information, and this by discarding the components with low information and considering the remaining components as your new variables.

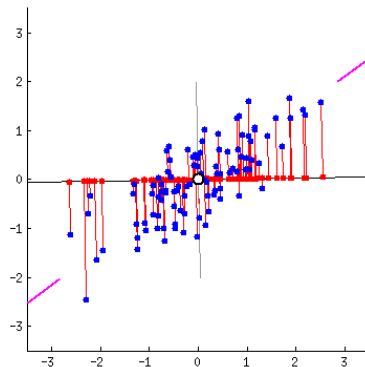
An important thing to realize here is that the principal components are less interpretable and don't have any real meaning since they are constructed as linear combinations of the initial variables.

Geometrically speaking, principal components represent the directions of the data that explain a **maximal amount of variance**, that is to say, the lines that capture most information of the data. The relationship between variance and information here, is that, the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more information it has. To put all this simply, just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible.

How PCA Constructs the Principal Components

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the **largest possible variance** in the data set. For example, let's assume that the scatter plot of our data set is as shown below, can we guess the first principal component? Yes, it's approximately the line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out. Or

mathematically speaking, it's the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).



The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

Now that we understand what we mean by principal components, let's go back to eigenvectors and eigenvalues. What you first need to know about them is that they always come in pairs, so that every eigenvector has an eigenvalue. And their number is equal to the number of dimensions of the data. For example, for a 3-dimensional data set, there are 3 variables, therefore there are 3 eigenvectors with 3 corresponding eigenvalues.

Without further ado, it is eigenvectors and eigenvalues who are behind all the magic explained above, because the eigenvectors of the Covariance matrix are actually *the directions of the axes where there is the most variance* (most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the *amount of variance carried in each Principal Component*.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Principal Component Analysis Example:

Let's suppose that our data set is 2-dimensional with 2 variables x, y and that the eigenvectors and eigenvalues of the covariance matrix are as follows:

$$v_1 = \begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix} \quad \lambda_1 = 1.284028$$

$$v_2 = \begin{bmatrix} -0.7351785 \\ 0.6778736 \end{bmatrix} \quad \lambda_2 = 0.04908323$$

If we rank the eigenvalues in descending order, we get $\lambda_1 > \lambda_2$, which means that the eigenvector that corresponds to the first principal component (PC1) is v_1 and the one that corresponds to the second principal component (PC2) is v_2 .

After having the principal components, to compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues. If we apply this on the example above, we find that PC1 and PC2 carry respectively 96 percent and 4 percent of the variance of the data.

STEP 4: FEATURE VECTOR

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call *Feature vector*.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only p eigenvectors (components) out of n , the final data set will have only p dimensions.

Principal Component Analysis Example:

Continuing with the example from the previous step, we can either form a feature vector with both of the eigenvectors v_1 and v_2 :

$$\begin{bmatrix} 0.6778736 & -0.7351785 \\ 0.7351785 & 0.6778736 \end{bmatrix}$$

Or discard the eigenvector v_2 , which is the one of lesser significance, and form a feature vector with v_1 only:

$$\begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix}$$

Discarding the eigenvector v_2 will reduce dimensionality by 1, and will consequently cause a loss of information in the final data set. But given that v_2 was carrying only 4 percent of the information, the loss will be therefore not important and we will still have 96 percent of the information that is carried by v_1 .

So, as we saw in the example, it's up to you to choose whether to keep all the components or discard the ones of lesser significance, depending on what you are looking for. Because if you just want to describe your data in terms of new variables (principal components) that are uncorrelated without seeking to reduce dimensionality, leaving out lesser significant components is not needed.

STEP 5: RECAST THE DATA ALONG THE PRINCIPAL COMPONENTS AXES

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

UNIT-I (Part-III)

Topics: Overfitting, Training, Testing, and Validation Sets, The Confusion Matrix, Accuracy Metrics Evaluation Measures: SSE, RMSE, R2, confusion matrix, precision, recall, F-Score, Receiver Operator Characteristic (ROC) Curve. Unbalanced Datasets. Some basic statistics: Averages, Variance and Covariance, The Gaussian, the bias-variance tradeoff.

Preliminaries:

When we talk about the Machine Learning model, we actually talk about how well it performs and its accuracy which is known as prediction errors. Let us consider that we are designing a machine learning model. A model is said to be a good machine learning model if it generalizes any new input data from the problem domain in a proper way. This helps us to make predictions about future data, that the data model has never seen. Now, suppose we want to check how well our machine learning model learns and generalizes to the new data. For that, we have overfitting and underfitting, which are majorly responsible for the poor performances of the machine learning algorithms.

Before diving further let's understand two important terms:

- **Bias:** Assumptions made by a model to make a function easier to learn. It is actually the error rate of the training data. When the error rate has a high value, we call it High Bias and when the error rate has a low value, we call it low Bias.
- **Variance:** The difference between the error rate of training data and testing data is called variance. If the difference is high then it's called high variance and when the difference of errors is low then it's called low variance. Usually, we want to make a low variance for generalized our model.

Underfitting: A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data, i.e., it only performs well on training data but performs poorly on testing data. *(It's just like trying to fit undersized pants!)* Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have fewer data to build an accurate model and also when we try to build a linear model with fewer non-linear data. In such cases, the rules of the machine learning model are too easy and flexible to be applied to such minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data and also reducing the features by feature selection.

In a nutshell, Underfitting refers to a model that can neither performs well on the training data nor generalize to new data.

Reasons for Underfitting:

1. High bias and low variance
2. The size of the training dataset used is not enough.
3. The model is too simple.
4. Training data is not cleaned and also contains noise in it.

Techniques to reduce underfitting:

1. Increase model complexity
2. Increase the number of features, performing feature engineering
3. Remove noise from the data.
4. Increase the number of epochs or increase the duration of training to get better results.

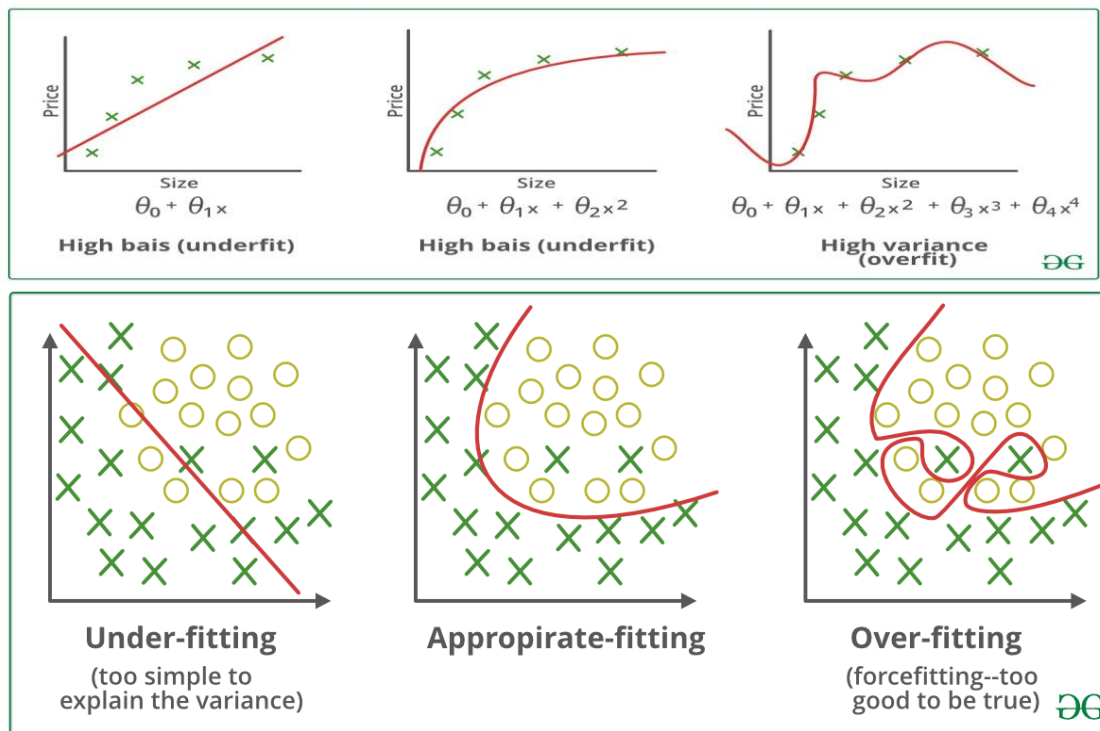
Overfitting: A statistical model is said to be overfitted when the model does not make accurate predictions on testing data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. And when testing with test data results in High variance. Then the model does not categorize the data correctly, because of too many details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

In a nutshell, Overfitting is a problem where the evaluation of machine learning algorithms on training data is different from unseen data.

Reasons for Overfitting are as follows:

1. High variance and low bias
2. The model is too complex
3. The size of the training data

Examples:



Techniques to reduce overfitting:

1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
4. Ridge Regularization and Lasso Regularization
5. Use dropout for neural networks to tackle overfitting.

Good Fit in a Statistical Model: Ideally, the case when the model makes the predictions with 0 error, is said to have a *good fit* on the data. This situation is achievable at a spot between overfitting and underfitting. In order to understand it, we will have to look at the performance of our model with the passage of time, while it is learning from the training dataset.

With the passage of time, our model will keep on learning, and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to the presence of noise and less useful details. Hence the performance of our model will decrease. In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point, the model is said to have good skills in training datasets as well as our unseen testing dataset.

Training, Testing and Validation datasets

Training Dataset

Training Dataset: The sample of data used to fit the model.

The actual dataset that we use to train the model (weights and biases in the case of a Neural Network). The model *sees* and *learns* from this data.

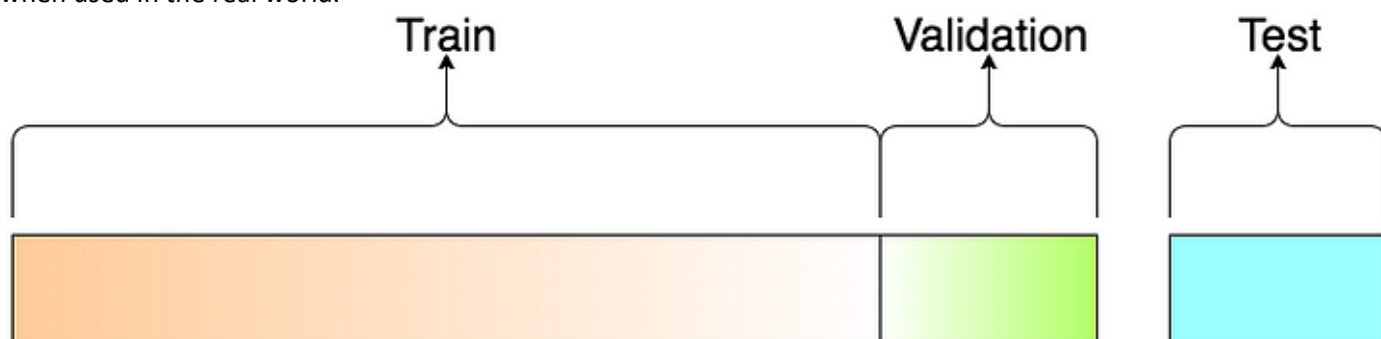
Validation Dataset

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

The validation set is used to evaluate a given model, but this is for frequent evaluation. We, as machine learning engineers, use this data to fine-tune the model hyperparameters. Hence the model occasionally *sees* this data, but never does it “*Learn*” from this. We use the validation set results, and update higher level hyperparameters. So the validation set affects a model, but only indirectly. The validation set is also known as the Dev set or the Development set. This makes sense since this dataset helps during the “development” stage of the model.

Test Dataset

Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained(using the train and validation sets). The test set is generally what is used to evaluate competing models (For example on many Kaggle competitions, the validation set is released initially along with the training set and the actual test set is only released when the competition is about to close, and it is the result of the the model on the Test set that decides the winner). Many a times the validation set is used as the test set, but it is not good practice. The test set is generally well curated. It contains carefully sampled data that spans the various classes that the model would face, when used in the real world.



Performance Metrics:

Confusion matrix, Accuracy metrics

Confusion Matrix in Machine Learning

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.
- It looks like the below table:

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

The above table has the following cases:

- **True Negative:** Model has given prediction No, and the real or actual value was also No.
- **True Positive:** The model has predicted yes, and the actual value was also true.
- **False Negative:** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive:** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error**.

Need for Confusion Matrix in Machine learning

- It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.
- It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.
- With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

Example: We can understand the confusion matrix using an example.

Suppose we are trying to create a model that can predict the result for the disease that is either a person has that disease or not. So, the confusion matrix for this is given as:

n = 100	Actual: No	Actual: Yes	
Predicted: No	TN: 65	FP: 3	68
Predicted: Yes	FN: 8	TP: 24	32
	73	27	

From the above example, we can conclude that:

- The table is given for the two-class classifier, which has two predictions "Yes" and "NO." Here, Yes defines that patient has the disease, and No defines that patient does not has that disease.
- The classifier has made a total of **100 predictions**. Out of 100 predictions, **89 are true predictions**, and **11 are incorrect predictions**.
- The model has given prediction "yes" for 32 times, and "No" for 68 times. Whereas the actual "Yes" was 27, and actual "No" was 73 times.

Calculations using Confusion Matrix:

We can perform various calculations for the model, such as the model's accuracy, using this matrix. These calculations are given below:

- **Classification Accuracy:** It is one of the important parameters to determine the accuracy of the classification problems. It defines how often the model predicts the correct output. It can be calculated as the ratio of the number of correct predictions made by the classifier to all number of predictions made by the classifiers. The formula is given below:

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

- **Misclassification rate:** It is also termed as Error rate, and it defines how often the model gives the wrong predictions. The value of error rate can be calculated as the number of incorrect predictions to all number of the predictions made by the classifier. The formula is given below:

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

- **Precision:** It can be defined as the number of correct outputs provided by the model or out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula:

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

- **Recall:** It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

$$\text{Recall} = \frac{TP}{TP+FN}$$

- **F-measure:** If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

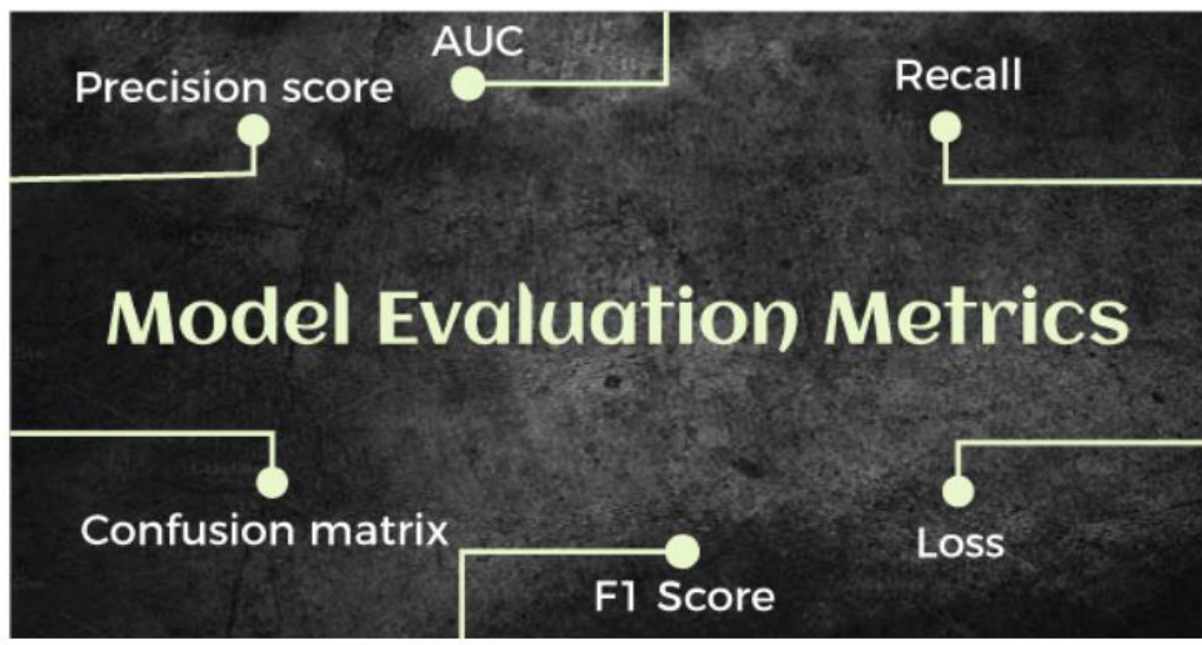
$$\text{F-measure} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

Other important terms used in Confusion Matrix:

- **Null Error rate:** It defines how often our model would be incorrect if it always predicted the majority class. As per the accuracy paradox, it is said that "*the best classifier has a higher error rate than the null error rate.*"
- **ROC Curve:** The ROC is a graph displaying a classifier's performance for all possible thresholds. The graph is plotted between the true positive rate (on the Y-axis) and the false Positive rate (on the x-axis).

Performance Metrics in Machine Learning

Evaluating the performance of a Machine learning model is one of the important steps while building an effective ML model. *To evaluate the performance or quality of the model, different metrics are used, and these metrics are known as performance metrics or evaluation metrics.* These performance metrics help us understand how well our model has performed for the given data. In this way, we can improve the model's performance by tuning the hyper-parameters. Each ML model aims to generalize well on unseen/new data, and performance metrics help determine how well the model generalizes on the new dataset.



In machine learning, each task or problem is divided into **classification** and **Regression**. Not all metrics can be used for all types of problems; hence, it is important to know and understand which metrics should be used. Different evaluation metrics are used for both Regression and Classification tasks. In this topic, we will discuss metrics used for classification and regression tasks.

1. Performance Metrics for Classification

In a classification problem, the category or classes of data is identified based on training data. The model learns from the given dataset and then classifies the new data into classes or groups based on the training. It predicts class labels as the output, such as *Yes or No*, *0 or 1*, *Spam or Not Spam*, etc. To evaluate the performance of a classification model, different metrics are used, and some of them are as follows:

- **Accuracy**
- **Confusion Matrix**

- **Precision**
- **Recall**
- **F-Score**
- **AUC(Area Under the Curve)-ROC**

I. Accuracy

The accuracy metric is one of the simplest Classification metrics to implement, and it can be determined as the number of correct predictions to the total number of predictions.

It can be formulated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total number of predictions}}$$

To implement an accuracy metric, we can compare ground truth and predicted values in a loop, or we can also use the scikit-learn module for this.

Firstly, we need to import the *accuracy_score* function of the scikit-learn library as follows:


1. `from sklearn.metrics import accuracy_score`
- 2.
3. Here, metrics is a class of sklearn.
- 4.
5. Then we need to pass the ground truth and predicted values in the function to calculate the accuracy.
- 6.
7. `print(f'Accuracy Score is {accuracy_score(y_test,y_hat)}')`

Although it is simple to use and implement, it is suitable only for cases where an equal number of samples belong to each class.

When to Use Accuracy?

It is good to use the Accuracy metric when the target variable classes in data are approximately balanced. For example, if 60% of classes in a fruit image dataset are of Apple, 40% are Mango. In this case, if the model is asked to predict whether the image is of Apple or Mango, it will give a prediction with 97% of accuracy.

When not to use Accuracy?

It is recommended not to use the Accuracy measure when the target variable majorly belongs to one class. For example, Suppose there is a model for a disease prediction in which, out of 100 people, only five people have a disease, and 95 people don't have one. In this case, if our model predicts every person with no disease (which means a bad prediction), the Accuracy measure will be 95%, which is not correct. 

II. Confusion Matrix

A confusion matrix is a tabular representation of prediction outcomes of any binary classifier, which is used to describe the performance of the classification model on a set of test data when true values are known.

The confusion matrix is simple to implement, but the terminologies used in this matrix might be confusing for beginners.

A typical confusion matrix for a binary classifier looks like the below image(However, it can be extended to use for classifiers with more than two classes).

n=165	Predicted: NO	Predicted: YES
	50	10
Actual: NO		
Actual: YES	5	100

We can determine the following from the above matrix:

- In the matrix, columns are for the prediction values, and rows specify the Actual values. Here Actual and prediction give two possible classes, Yes or No. So, if we are predicting the presence of a disease in a patient, the Prediction column with Yes means, Patient has the disease, and for NO, the Patient doesn't have the disease.
- In this example, the total number of predictions are 165, out of which 110 time predicted yes, whereas 55 times predicted No.
- However, in reality, 60 cases in which patients don't have the disease, whereas 105 cases in which patients have the disease.

In general, the table is divided into four terminologies, which are as follows:

1. **True Positive(TP):** In this case, the prediction outcome is true, and it is true in reality, also.
2. **True Negative(TN):** in this case, the prediction outcome is false, and it is false in reality, also.
3. **False Positive(FP):** In this case, prediction outcomes are true, but they are false in actuality.
4. **False Negative(FN):** In this case, predictions are false, and they are true in actuality.

III. Precision

The precision metric is used to overcome the limitation of Accuracy. The precision determines the proportion of positive prediction that was actually correct. It can be calculated as the True Positive or predictions that are actually true to the total positive predictions (True Positive and False Positive).

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

IV. Recall or Sensitivity

It is also similar to the Precision metric; however, it aims to calculate the proportion of actual positive that was identified incorrectly. It can be calculated as True Positive or predictions that are actually true to the total number of positives, either correctly predicted as positive or incorrectly predicted as negative (true Positive and false negative).

The formula for calculating Recall is given below:

$$\text{Recall} = \frac{TP}{TP + FN}$$

When to use Precision and Recall?

From the above definitions of Precision and Recall, we can say that recall determines the performance of a classifier with respect to a false negative, whereas precision gives information about the performance of a classifier with respect to a false positive.

So, if we want to minimize the false negative, then, Recall should be as near to 100%, and if we want to minimize the false positive, then precision should be close to 100% as possible.

In simple words, *if we maximize precision, it will minimize the FP errors, and if we maximize recall, it will minimize the FN error.*

V. F-Scores

F-score or F1 Score is a metric to evaluate a binary classification model on the basis of predictions that are made for the positive class. It is calculated with the help of Precision and Recall. It is a type of single score that represents both Precision and Recall. So, *the F1 Score can be calculated as the harmonic mean of both precision and Recall, assigning equal weight to each of them.*

The formula for calculating the F1 score is given below:

$$F1 - score = 2 * \frac{precision * recall}{precision + recall}$$

When to use F-Score?

As F-score make use of both precision and recall, so it should be used if both of them are important for evaluation, but one (precision or recall) is slightly more important to consider than the other. For example, when False negatives are comparatively more important than false positives, or vice versa.

VI. AUC-ROC

Sometimes we need to visualize the performance of the classification model on charts; then, we can use the AUC-ROC curve. It is one of the popular and important metrics for evaluating the performance of the classification model.

Firstly, let's understand ROC (Receiver Operating Characteristic curve) curve. **ROC represents a graph to show the performance of a classification model at different threshold levels.** The curve is plotted between two parameters, which are:

- **True Positive Rate**
- **False Positive Rate**

TPR or true Positive rate is a synonym for Recall, hence can be calculated as:

$$TPR = \frac{TP}{TP + FN}$$

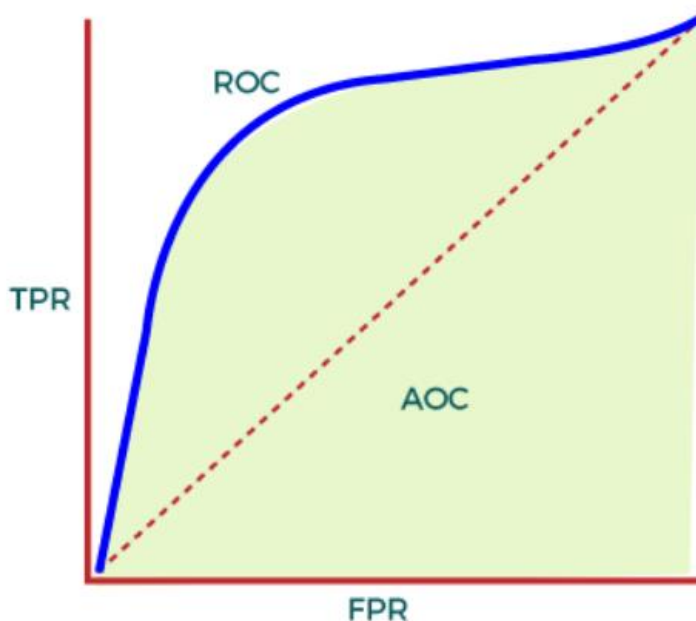
FPR or False Positive Rate can be calculated as:

$$FPR = \frac{FP}{FP + TN}$$

To calculate value at any point in a ROC curve, we can evaluate a logistic regression model multiple times with different classification thresholds, but this would not be much efficient. So, for this, one efficient method is used, which is known as AUC.

AUC: Area Under the ROC curve

AUC is known for **Area Under the ROC curve**. As its name suggests, AUC calculates the two-dimensional area under the entire ROC curve, as shown below image:



AUC calculates the performance across all the thresholds and provides an aggregate measure. The value of AUC ranges from 0 to 1. It means a model with 100% wrong prediction will have an AUC of 0.0, whereas models with 100% correct predictions will have an AUC of 1.0.

When to Use AUC

AUC should be used to measure how well the predictions are ranked rather than their absolute values. Moreover, it measures the quality of predictions of the model without considering the classification threshold.

When not to use AUC

As AUC is scale-invariant, which is not always desirable, and we need calibrating probability outputs, then AUC is not preferable.

Further, AUC is not a useful metric when there are wide disparities in the cost of false negatives vs. false positives, and it is difficult to minimize one type of classification error.

2. Performance Metrics for Regression

Regression is a supervised learning technique that aims to find the relationships between the dependent and independent variables. A predictive regression model predicts a numeric or discrete value. The metrics used for regression are different from the classification metrics. It means we cannot use the Accuracy metric (explained above) to evaluate a regression model; instead, the performance of a Regression model is reported as errors in the prediction. Following are the popular metrics that are used to evaluate the performance of Regression models.

- **Mean Absolute Error**
- **Mean Squared Error**
- **R2 Score**
- **Adjusted R2**

1. Mean Absolute Error (MAE)

Mean Absolute Error or MAE is one of the simplest metrics, which measures the absolute difference between actual and predicted values, where absolute means taking a number as Positive.

To understand MAE, let's take an example of Linear Regression, where the model draws a best fit line between dependent and independent variables. To measure the MAE or error in prediction, we need to calculate the difference between actual values and predicted values. But in order to find the absolute error for the complete dataset, we need to find the mean absolute of the complete dataset.

The below formula is used to calculate MAE:

$$MAE = 1/N \sum |Y - Y'|$$

Here,

Y is the Actual outcome, Y' is the predicted outcome, and N is the total number of data points.

MAE is much more robust for the outliers. One of the limitations of MAE is that it is not differentiable, so for this, we need to apply different optimizers such as Gradient Descent. However, to overcome this limitation, another metric can be used, which is Mean Squared Error or MSE.

II. Mean Squared Error

Mean Squared error or MSE is one of the most suitable metrics for Regression evaluation. It measures the average of the Squared difference between predicted values and the actual value given by the model.

Since in MSE, errors are squared, therefore it only assumes non-negative values, and it is usually positive and non-zero.

Moreover, due to squared differences, it penalizes small errors also, and hence it leads to over-estimation of how bad the model is.

MSE is a much-preferred metric compared to other regression metrics as it is differentiable and hence optimized better.

The formula for calculating MSE is given below:

$$MSE = 1/N \sum (Y - Y')^2$$

Here,

Y is the Actual outcome, Y' is the predicted outcome, and N is the total number of data points.

III. R Squared Score

R squared error is also known as Coefficient of Determination, which is another popular metric used for Regression model evaluation. The R-squared metric enables us to compare our model with a constant baseline to determine the performance of the model. To select the constant baseline, we need to take the mean of the data and draw the line at the mean.

The R squared score will always be less than or equal to 1 without concerning if the values are too large or small.

$$R^2 = 1 - \frac{MSE(Model)}{MSE(Baseline)}$$

IV. Adjusted R Squared

Adjusted R squared, as the name suggests, is the improved version of R squared error. R square has a limitation of improvement of a score on increasing the terms, even though the model is not improving, and it may mislead the data scientists.

To overcome the issue of R square, adjusted R squared is used, which will always show a lower value than R². It is because it adjusts the values of increasing predictors and only shows improvement if there is a real improvement.

We can calculate the adjusted R squared as follows:

$$R_a^2 = 1 - \left[\left(\frac{n-1}{n-k-1} \right) \times (1 - R^2) \right]$$

Here,

n is the number of observations

k denotes the number of independent variables

and R_a^2 denotes the adjusted R^2

Unbalanced Dataset

Imbalanced Data

A classification data set with skewed class proportions is called [imbalanced](#). Classes that make up a large proportion of the data set are called [majority classes](#). Those that make up a smaller proportion are [minority classes](#).

What counts as imbalanced? The answer could range from mild to extreme, as the table below shows.

Degree of imbalance

Mild

Moderate

Extreme

Proportion of Minority Class

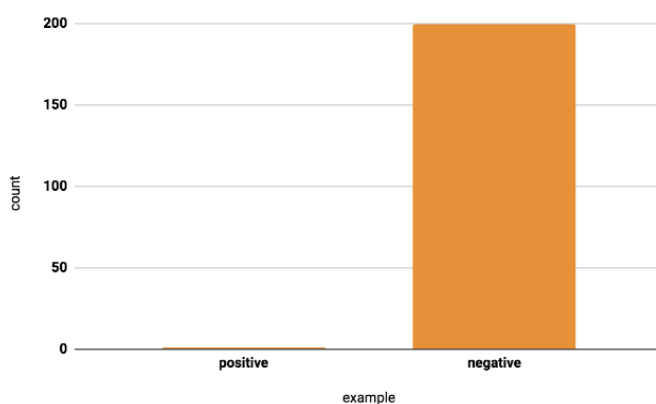
20-40% of the data set

1-20% of the data set

<1% of the data set

Why look out for imbalanced data? You may need to apply a particular sampling technique if you have a classification task with an imbalanced data set.

Consider the following example of a model that detects fraud. Instances of fraud happen once per 200 transactions in this data set, so in the true distribution, about 0.5% of the data is positive.



Why would this be problematic? With so few positives relative to negatives, the training model will spend most of its time on negative examples and not learn enough from positive ones. For example, if your batch size is 128, many batches will have no positive examples, so the gradients will be less informative.

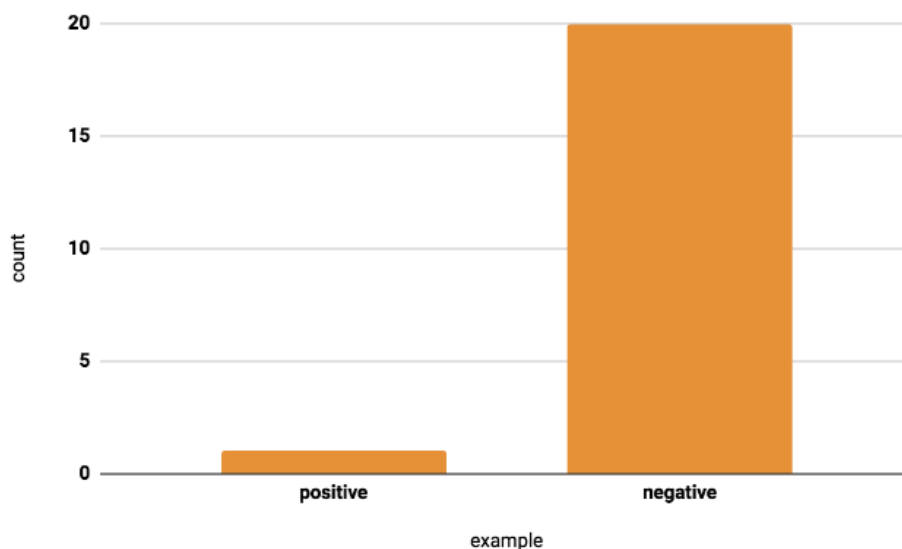
If you have an imbalanced data set, **first try training on the true distribution**. If the model works well and generalizes, you're done! If not, try the following downsampling and upweighting technique.

Downsampling and Upweighting

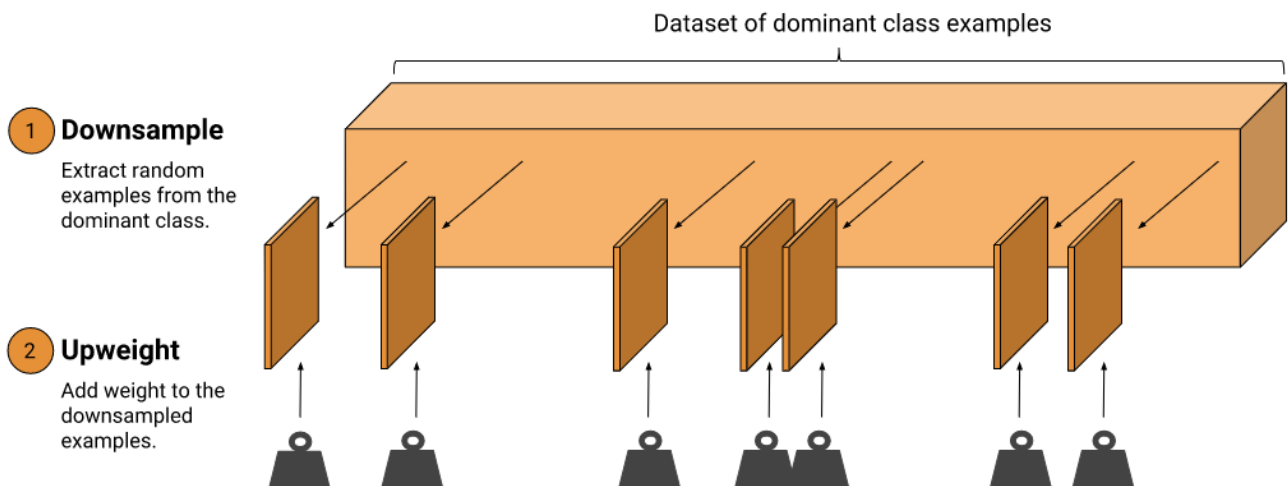
An effective way to handle imbalanced data is to downsample and upweight the majority class. Let's start by defining those two new terms:

- **Downsampling** (in this context) means training on a disproportionately low subset of the majority class examples.
- **Upweighting** means adding an example weight to the downsampled class equal to the factor by which you downsampled.

Step 1: Downsample the majority class. Consider again our example of the fraud data set, with 1 positive to 200 negatives. Downsampling by a factor of 20 improves the balance to 1 positive to 10 negatives (10%). Although the resulting training set is still *moderately imbalanced*, the proportion of positives to negatives is much better than the original *extremely imbalanced* proportion (0.5%).



Step 2: Upweight the downsampled class: The last step is to add example weights to the downsampled class. Since we downsampled by a factor of 20, the example weight should be 20.



You may be used to hearing the term *weight* when it refers to model parameters, like connections in a neural network. Here we're talking about *example weights*, which means counting an individual example more importantly during training. An example weight of 10 means the model treats the example as 10 times as important (when computing loss) as it would an example of weight 1.

The weight should be equal to the factor you used to downsample:

$$\{\text{example weight}\} = \{\text{original example weight}\} \times \{\text{downsampling factor}\}$$

Why Downsample and Upweight?

It may seem odd to add example weights after downsampling. We were trying to make our model improve on the minority class -- why would we upweight the majority? These are the resulting changes:

- **Faster convergence:** During training, we see the minority class more often, which will help the model converge faster.
- **Disk space:** By consolidating the majority class into fewer examples with larger weights, we spend less disk space storing them. This savings allows more disk space for the minority class, so we can collect a greater number and a wider range of examples from that class.
- **Calibration:** Upweighting ensures our model is still calibrated; the outputs can still be interpreted as probabilities.

Define Mean, Variance and Co-variance

Mean

The mean of a set of values or measurements is the sum of all the measurements divided by the sum of all the measurements in the set:

$$\text{Mean} = \frac{\sum_{i=1}^n x_i}{n}$$

If we compute the population's mean, we call it the parametric or population mean, denoted by μ (read "mu"). If we get the mean of the sample, we call it the sample mean, denoted by the \bar{x} .

Population vs. Sample

A population refers to the summation of all the elements of interest to the researcher.

- Examples: The number of people in a country, the number of hedge funds in the U.S., or even the total number of CFA candidates in a given year.

A sample is just a set of elements that represent the population as a whole. By analyzing sample data, we are able to make conclusions about the entire population.

- For example, if we sample the returns of 30 hedge funds spread across the U.S., we can use the results to make reasonable conclusions about the market as a whole (well over 10,000 hedge funds).

Variance

Variance is a measure of dispersion around the mean and is statistically defined as the average squared deviation from the mean. It is noted using the symbol σ^2 .

$$\sigma^2 = \frac{\sum_{i=1}^N (X_i - \mu)^2}{N}$$

Where μ is the population mean, and N is the population size.

The standard deviation, σ , is the square root of the variance and is commonly referred to as the volatility of the asset. Essentially, it is a measure of how far, on average, the observations are from the mean. A population's variance is given by:

The population standard deviation equals the square root of the population variance. The sample variance is given by:

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

Where \bar{X} is the sample mean and n is the sample size.

Note that the sample standard deviation equals the square root of the sample variance.

Covariance

Covariance is a measure of how closely two assets move together. In covariance, we focus on the relationship between the deviations of some two variables rather than the deviation from the mean of one variable.

If the means of random variables X and Y are known, then the covariance between the two random variables can be determined as follows:

$$\hat{\sigma}_{xy} = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_x) (y_i - \mu_y)$$

If we do not know the means, then the equation changes to:

$$\hat{\sigma}_{xy} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \hat{\mu}_x) (y_i - \hat{\mu}_y)$$

Correlation

Correlation is a concept that is closely related to covariance in the following way:

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

Correlation ranges between +1 and -1 and is much easier to interpret than covariance. Two variables are perfectly correlated if their correlation is equal to +1. Note that they are uncorrelated if their correlation equals 0, and move in perfectly opposite directions if their correlation equals -1.

The Gaussian and bias-variance trade-off

Gaussian Distribution:

In machine learning, Gaussian usually refers to Gaussian distributions, also known as normal distributions. A Gaussian distribution is a probability distribution that is widely used in machine learning and statistics to model continuous data.

A Gaussian distribution is characterized by two parameters: the mean (μ) and the standard deviation (σ). The mean represents the center of the distribution, while the standard deviation represents the spread of the distribution. The shape of the Gaussian distribution is bell-shaped, with the highest point at the mean, and the curve gradually tapering off towards both sides.

Gaussian distributions are used in many machine learning algorithms, such as Gaussian mixture models, Gaussian process regression, and Naive Bayes classifiers. They are particularly useful when dealing with continuous data, and can be used to model a wide range of phenomena, from the heights of people to the temperatures of cities.

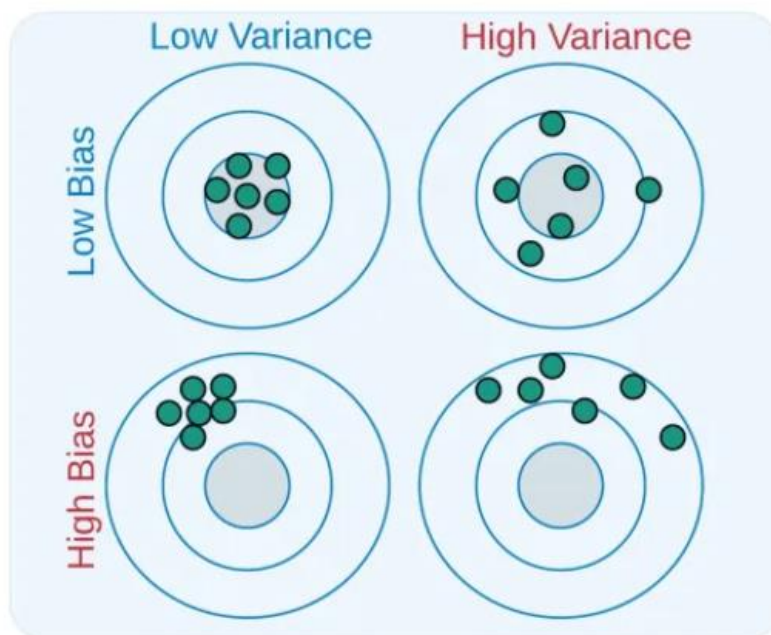
some examples and notes on Gaussian distributions in machine learning:

1. **Height of People:** The height of people in a population can be modeled using a Gaussian distribution. The mean of the distribution would be the average height of the population, and the standard deviation would represent how much the heights vary from the mean.
2. **Gaussian Mixture Models:** Gaussian mixture models are a type of unsupervised learning algorithm that assumes that the data is generated from a mixture of several Gaussian distributions. Each Gaussian distribution represents a different cluster or group in the data.
3. **Gaussian Process Regression:** Gaussian process regression is a non-parametric approach to regression that uses a Gaussian process to model the distribution of the data. The model can be used to make predictions for new inputs by computing the mean and variance of the Gaussian distribution at that point.
4. **Naive Bayes Classifiers:** Naive Bayes classifiers are a type of probabilistic classifier that assumes that the features are independent given the class label. In the case of continuous features, a Gaussian distribution is used to model the distribution of each feature for each class.

Overall, Gaussian distributions are a fundamental concept in machine learning and statistics, and understanding them is crucial for many machine learning algorithms and models.

Bias Variance Trade-Off

Understanding the bias-variance tradeoff is essential for developing accurate and reliable Machine Learning models. It can help to optimize the model performance and avoid common pitfalls such as underfitting and overfitting. One of the best ways to visualize the bias and variance concepts is through a dartboard like the one shown below.



The figure shows how variance and bias are related:

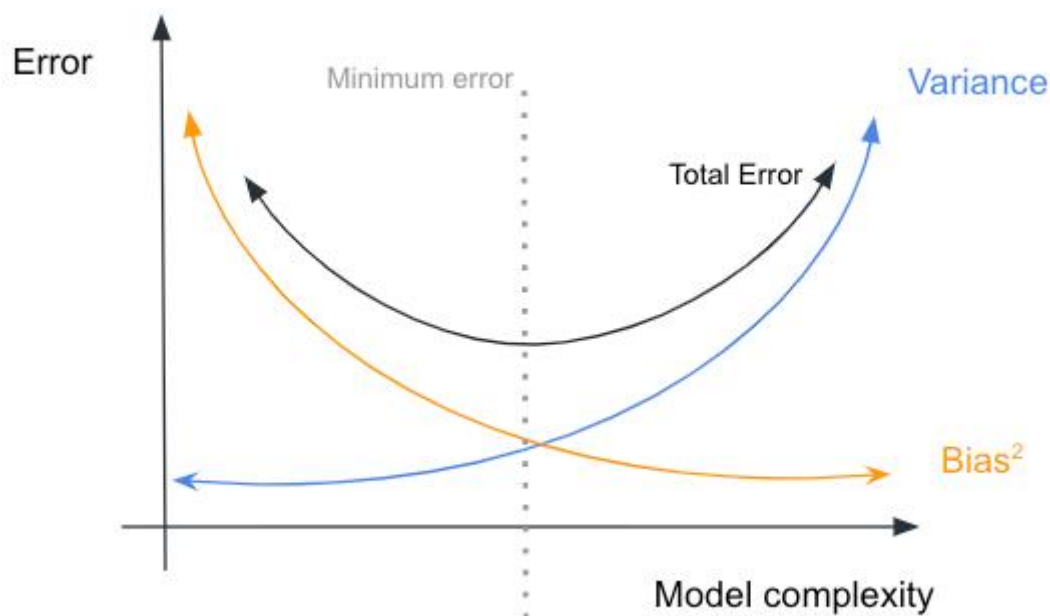
- A model with high bias and high variance is a model that makes a lot of mistakes and is very inconsistent.
- A model with high variance and low bias tends to be more accurate, but the results suffer a lot of variations.
- A model with high bias and low variance is a model that makes a lot of bad predictions but is very consistent in its results.
- Lastly, a model with low bias and variance makes good predictions and is consistent with its results.

Looking at the figure, it's intuitive that all the models should have a low bias and low variance, since this combination generates the best results. However, this is where the bias-variance tradeoff appears.

The tradeoff

The bias-variance tradeoff arises because increasing the model's complexity can reduce the bias but increase the variance. On the other hand, decreasing the complexity can reduce the variance but increase the bias. The goal is to find the optimal balance between bias and variance, which results in the best generalization performance on new, unseen data.

This is directly related to the complexity of the model used, as shown in the figure below.



Bias-variance tradeoff and error relationship (image by author)

The graph shows how the complexity of the model is related to the values of bias and variance. Models that have a low complexity can be too simple to understand the patterns of the data used in the training, a phenomenon called **underfitting**. Consequently, it won't be able to make good predictions on the test data, resulting in a high bias.

On the other hand, a model with too much degree of liberty can result in what is called ***overfitting***, which is when the model has an excellent performance in the training data, but has a significant decrease in performance when evaluating the test data. This happens when the model becomes extremely accustomed to the training data, thus losing its generalization capability and, when it needs to interpret a data sample never seen before, it cannot get a good result.

As the model's complexity increases, the bias decreases (the model fits the training data better) but the variance increases (the model becomes more sensitive to the training data). The optimal tradeoff occurs at the point where the error is minimized, which in this case is at a moderate level of complexity.

To help understanding, let's look at a practical example that illustrates the concept of bias-variance tradeoff.

UNIT II

Supervised Algorithms

Topics: Decision Trees: ID3, Classification and Regression Trees, Regression: Linear Regression, Multiple Linear Regression, Logistic Regression, Support Vector Machines: Linear and Non-Linear, Kernel Functions, K Nearest Neighbors. Introduction to clustering, K-means clustering, K-Mode Clustering.

Decision Tree

Introduction Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely **decision nodes and leaves**. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.



An example of a decision tree can be explained using above binary tree. Let's say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The

decision nodes here are questions like ‘What’s the age?’, ‘Does he exercise?’, and ‘Does he eat a lot of pizzas?’. And the leaves, which are outcomes like either ‘fit’, or ‘unfit’. In this case this was a binary classification problem (a yes no type problem). There are two main types of Decision Trees:

1. Classification trees (Yes/No types)

What we have seen above is an example of classification tree, where the outcome was a variable like ‘fit’ or ‘unfit’. Here the decision variable is **Categorical**.

2. Regression trees (Continuous data types)

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123. **Working** Now that we know what a Decision Tree is, we’ll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**. Before discussing the ID3 algorithm, we’ll go through few definitions. **Entropy** Entropy, also called as Shannon Entropy is denoted by $H(S)$ for a finite set S , is the measure of the amount of uncertainty or randomness in data.

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

Intuitively, it tells us about the predictability of a certain event. Example, consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there’s no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it’ll always be heads. In other words, this event has **no randomness** hence it’s entropy is zero. In particular, lower values imply less uncertainty while higher values imply high uncertainty. **Information Gain** Information gain is also called as Kullback-Leibler divergence denoted by $IG(S, A)$ for a set S is the effective change in entropy after deciding on a particular attribute A . It measures the relative change in entropy with respect to the independent variables

$$IG(S, A) = H(S) - H(S, A)$$

Alternatively,

$$IG(S, A) = H(S) - \sum_{i=0}^n P(x) H(x)$$

where $IG(S, A)$ is the information gain by applying feature A . $H(S)$ is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A , where $P(x)$ is the probability of event x . Let’s understand this with the help of an example Consider a piece of data collected over the course of 14 days where the features are Outlook, Temperature, Humidity, Wind and the outcome variable is whether Golf was played on the day. Now, our job is to build a predictive model which takes in above 4 parameters and predicts whether Golf will be played on the day. We’ll build a decision tree to do that using **ID3 algorithm**.

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes

D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3

ID3 Algorithm will perform following tasks recursively

1. Create root node for the tree
2. If all examples are positive, return leaf node „positive“
3. Else if all examples are negative, return leaf node „negative“
4. Calculate the entropy of current state $H(S)$
5. For each attribute, calculate the entropy with respect to the attribute „x“ denoted by $H(S, x)$
6. Select the attribute which has maximum value of $IG(S, x)$
7. Remove the attribute that offers highest IG from the set of attributes
8. Repeat until we run out of all attributes, or the decision tree has all leaf nodes.

Now we'll go ahead and grow the decision tree. The initial step is to calculate $H(S)$, the Entropy of the current state. In the above example, we can see in total there are 5 No's and 9 Yes's.

Yes	No	Total
9	5	14

$$Entropy(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

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

$$= 0.940$$

where „x“ are the possible values for an attribute. Here, attribute „Wind“ takes two possible values in the sampled data, hence $x = \{Weak, Strong\}$ we'll have to calculate:

1. $H(S_{weak})$
2. $H(S_{strong})$
3. $P(S_{weak})$
4. $P(S_{strong})$
5. $H(S) = 0.94$ which we had already calculated in the previous example

Amongst all the 14 examples we have **8 places where the wind is weak** and **6 where the wind is Strong**.

Wind = Weak	Wind = Strong	Total
8	6	14

$$\begin{aligned}
 P(S_{weak}) &= \frac{\text{Number of Weak}}{\text{Total}} \\
 &= \frac{8}{14} \\
 P(S_{strong}) &= \frac{\text{Number of Strong}}{\text{Total}} \\
 &= \frac{6}{14}
 \end{aligned}$$

Now out of the 8 Weak examples, 6 of them were „Yes“ for Play Golf and 2 of them were „No“ for „Play Golf“. So, we have,

$$\begin{aligned}
 Entropy(S_{weak}) &= -\left(\frac{6}{8}\right) \log_2 \left(\frac{6}{8}\right) - \left(\frac{2}{8}\right) \log_2 \left(\frac{2}{8}\right) \\
 &= 0.811
 \end{aligned}$$

Similarly, out of 6 Strong examples, we have 3 examples where the outcome was „Yes“ for Play Golf and 3 where we had „No“ for Play Golf.

$$\begin{aligned}
 Entropy(S_{strong}) &= -\left(\frac{3}{6}\right) \log_2 \left(\frac{3}{6}\right) - \left(\frac{3}{6}\right) \log_2 \left(\frac{3}{6}\right) \\
 &= 1.000
 \end{aligned}$$

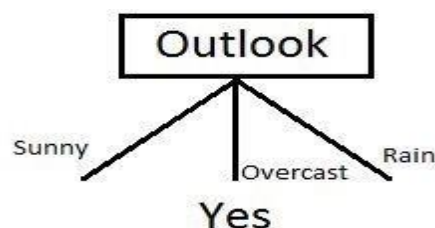
Remember, here half items belong to one class while other half belong to other. Hence we have perfect randomness. Now we have all the pieces required to calculate the Information Gain,

$$\begin{aligned}
 IG(S, Wind) &= H(S) - \sum_{i=0}^n P(x) * H(x) \\
 IG(S, Wind) &= H(S) - P(S_{weak}) * H(S_{weak}) - P(S_{strong}) * H(S_{strong}) \\
 &= 0.940 - \left(\frac{8}{14}\right) (0.811) - \left(\frac{6}{14}\right) (1.00) \\
 &= 0.048
 \end{aligned}$$

Which tells us the Information Gain by considering „Wind“ as the feature and give us information gain of **0.048**. Now we must similarly calculate the Information Gain for all the features.

$$\begin{aligned}
 IG(S, Outlook) &= 0.246 \\
 IG(S, Temperature) &= 0.029 \\
 IG(S, Humidity) &= 0.151 \\
 IG(S, Wind) &= 0.048 \text{ (Previous example)}
 \end{aligned}$$

We can clearly see that $IG(S, Outlook)$ has the highest information gain of 0.246, hence we chose **Outlook attribute as the root node**. At this point, the decision tree looks like.



Here we observe that whenever the outlook is Overcast, Play Golf is always ‘Yes’, it’s no coincidence by any chance, the simple tree resulted because of **the highest information gain is given by the attribute Outlook**. Now how do we proceed from this point? We can simply apply **recursion**, you might want to look at the algorithm steps described earlier. Now that we’ve used Outlook, we’ve got three of them remaining Humidity, Temperature, and Wind. And, we had three possible values of Outlook: Sunny, Overcast, Rain. Where the Overcast node already ended up having leaf node ‘Yes’, so we’re left with two subtrees to compute: Sunny and Rain.

Next step would be computing $H(S_{sunny})$.

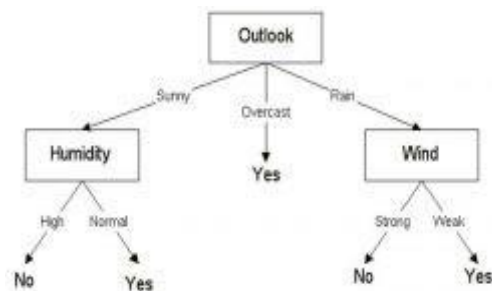
Table where the value of Outlook is Sunny looks like:

Temperature	Humidity	Wind	Play Golf
Hot	High	Weak	No
Hot	High	Strong	No
Mild	High	Weak	No
Cool	Normal	Weak	Yes
Mild	Normal	Strong	Yes

$$H(S_{sunny}) = \left(\frac{3}{5}\right) \log_2 \left(\frac{3}{5}\right) - \left(\frac{2}{5}\right) \log_2 \left(\frac{2}{5}\right) = 0.96$$

As we can see the **highest Information Gain is given by Humidity**. Proceeding in the same way with S_{rain}

will give us Wind as the one with highest information gain. The final Decision Tree looks something likethis. The final Decision Tree looks something like this.



Classification and Regression Trees

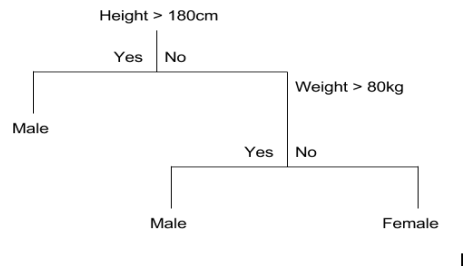
Classification Trees

A classification tree is an algorithm where the target variable is fixed or categorical. The algorithm is then used to identify the “class” within which a target variable would most likely fall.

An example of a classification-type problem would be determining who will or will not subscribe to a digital platform; or who will or will not graduate from high school.

These are examples of simple binary classifications where the categorical dependent variable can assume only one of two, mutually exclusive values. In other cases, you might have to predict among a number of different variables. For instance, you may have to predict which type of smartphone a consumer may decide to purchase.

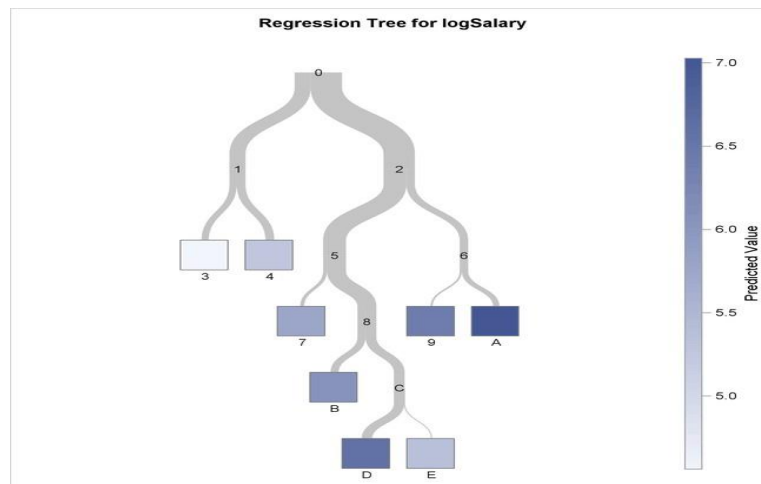
In such cases, there are multiple values for the categorical dependent variable. Here’s what a classic classification tree looks like



Regression Trees

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

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



When to use Classification and Regression Trees

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

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

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

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

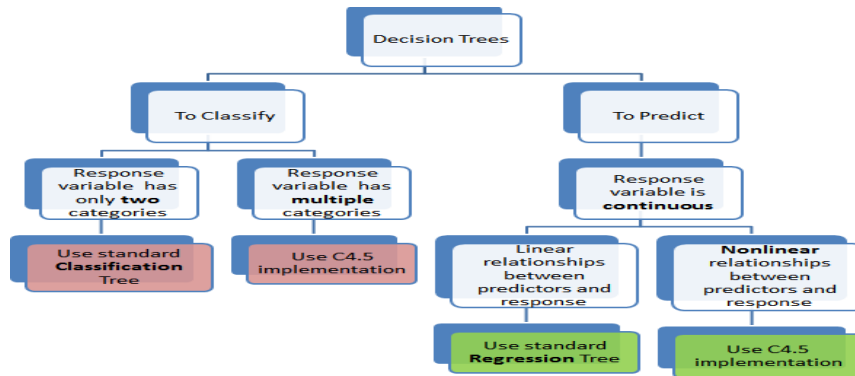
How Classification and Regression Trees Work

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

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

In a regression tree, a regression model is fit to the target variable using each of the independent variables. After this, the data is split at several points for each independent variable.

At each such point, the error between the predicted values and actual values is squared to get “A Sum of Squared Errors” (SSE). The SSE is compared across the variables and the variable or point which has the lowest SSE is chosen as the split point. This process is continued recursively.



Advantages of Classification and Regression Trees

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

(i) The Results are Simplistic

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

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

(ii) Classification and Regression Trees are Nonparametric & Nonlinear

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

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

Since there is no need for such implicit assumptions, classification and regression tree methods are well suited to data mining. This is because there is very little knowledge or assumptions that can be made beforehand about how the different variables are related.

As a result, classification and regression trees can actually reveal relationships between these variables that would not have been possible using other techniques.

(iii) Classification and Regression Trees Implicitly Perform Feature Selection

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

Limitations of Classification and Regression Trees

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

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

(i) Overfitting

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

(ii) High variance

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

(iii) Low bias

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

What is a CART in Machine Learning?

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

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

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

Summing up

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

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

Regression

Regression Analysis in Machine learning

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

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

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

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

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

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

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

Some examples of regression can be as:

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

Terminologies Related to the Regression Analysis:

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

Why do we use Regression Analysis?

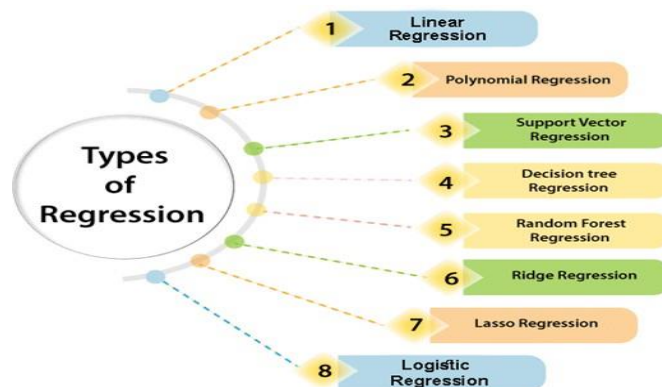
As mentioned above, Regression analysis helps in the prediction of a continuous variable. There are various scenarios in the real world where we need some future predictions such as weather condition, sales prediction, marketing trends, etc., for such case we need some technology which can make predictions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

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

Types of Regression

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

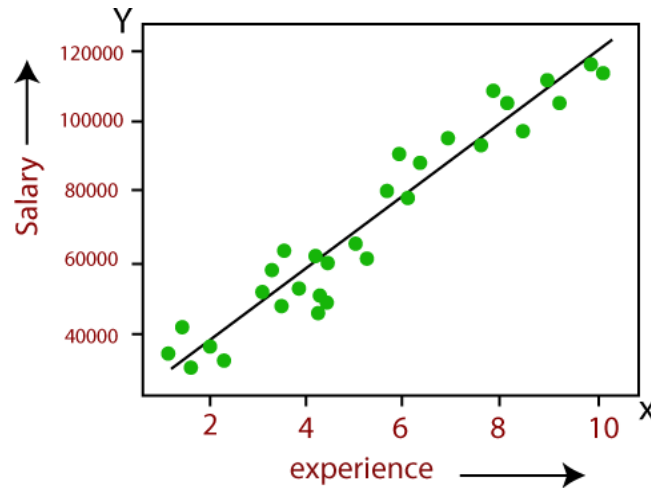
- **Linear Regression**
- **Logistic Regression**
- **Polynomial Regression**
- **Support Vector Regression**
- **Decision Tree Regression**
- **Random Forest Regression**
- **Ridge Regression**
- **Lasso Regression**



Linear Regression:

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

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



Below is the mathematical equation for Linear regression:

$$Y = aX + b$$

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

Some popular applications of linear regression are:

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

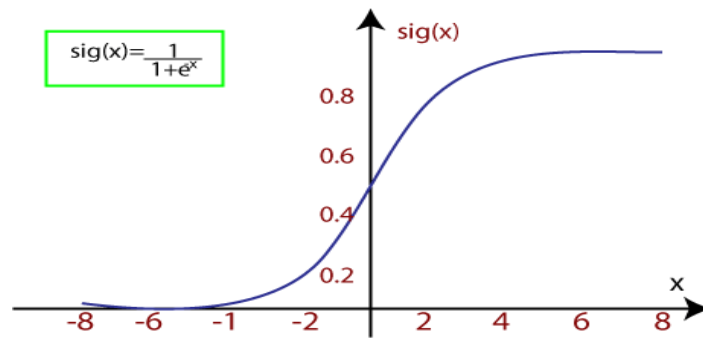
Logistic Regression:

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

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

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

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



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

There are three types of logistic regression:

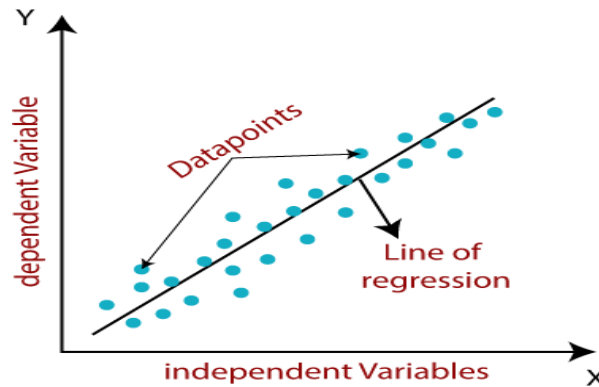
- **Binary(0/1, pass/fail)**
- **Multi(cats, dogs, lions)**
- **Ordinal(low, medium, high)**

Linear Regression in Machine Learning

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

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

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



Mathematically, we can represent a linear regression as:

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

Here,

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

a_0 = intercept of the line (Gives an additional degree of freedom)

a_1 = Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

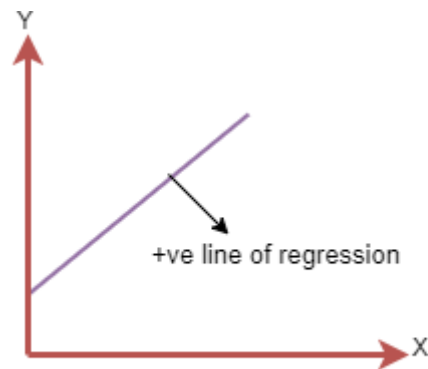
Linear regression can be further divided into two types of the algorithm:

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

Linear Regression Line:

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

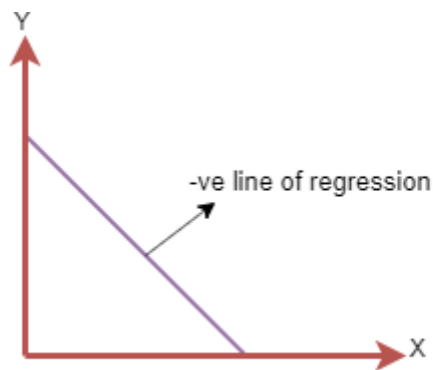
- **Positive Linear Relationship:**
If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



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

- **Negative Linear Relationship:**

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



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

Finding the best fit line:

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

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

Cost function-

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

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

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

$$MSE = \frac{1}{N} \sum_{i=1}^n (y_i - (a_1 x_i + a_0))^2$$

Where,

N=Total number of observation

Y_i = Actual value

(a₁x_i+a₀)= Predicted value.

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

Gradient Descent:

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

Model Performance:

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

1. R-squared method:

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

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

Assumptions of Linear Regression

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

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

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

- **Homoscedasticity Assumption:**
Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.
- **Normal distribution of error terms:**
Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.
- **No autocorrelations:**
The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

Simple Linear Regression in Machine Learning

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

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

Simple Linear regression algorithm has mainly two objectives:

- **Model the relationship between the two variables.** Such as the relationship between Income and expenditure, experience and Salary, etc.
- **Forecasting new observations.** Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

Simple Linear Regression Model:

The Simple Linear Regression model can be represented using the below

$$\text{equation: } y = a_0 + a_1x + \varepsilon$$

Where,

a_0 = It is the intercept of the Regression line (can be obtained putting $x=0$)

a_1 = It is the slope of the regression line, which tells whether the line is increasing or decreasing.

ε = The error term. (For a good model it will be negligible)

Multiple Linear Regressions

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

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

We can define it as:

“Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.”

Example:

Prediction of CO₂ emission based on engine size and number of cylinders in a car.

Some key points about MLR:

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

MLR equation:

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

$$Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n \quad (a)$$

Where,

Y= Output/Response variable

$b_0, b_1, b_2, b_3, b_n \dots$ = Coefficients of the model.

$x_1, x_2, x_3, x_4, \dots$ = Various Independent/feature variable

Assumptions for Multiple Linear Regression:

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

Difference between Linear Regression and Logistic Regression:

Linear Regression	Logistic Regression
Linear regression is used to predict the continuous dependent variable using a given set of independent variables.	Logistic Regression is used to predict the categorical dependent variable using a given set of independent variables.
Linear Regression is used for solving Regression problem.	Logistic regression is used for solving Classification problems.
In Linear regression, we predict the value of continuous variables.	In logistic Regression, we predict the values of categorical variables.
In linear regression, we find the best fit line , by which we can easily predict the output.	In Logistic Regression, we find the S-curve by which we can classify the samples.
Least square estimation method is used for estimation of accuracy.	Maximum likelihood estimation method is used for estimation of accuracy.
The output for Linear Regression must be a continuous value, such as price, age, etc.	The output of Logistic Regression must be a Categorical value such as 0 or 1, Yes or No, etc.
In Linear regression, it is required that relationship between dependent variable and independent variable must be linear .	In Logistic regression, it is not required to have the linear relationship between the dependent and independent variable.
In linear regression, there may be collinearity between the independent var	In logistic regression, there should not be collinearity between the independent variable.

What is Linear Discriminant Analysis (LDA)?

Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.

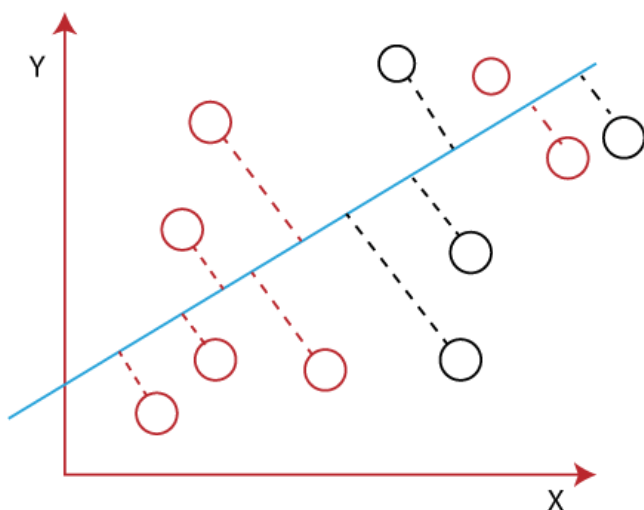
Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

How Linear Discriminant Analysis (LDA) works?

Linear Discriminant analysis is used as a dimensionality reduction technique in machine learning, using which we can easily transform a 2-D and 3-D graph into a 1-dimensional plane.

Let's consider an example where we have two classes in a 2-D plane having an X-Y axis, and we need to classify them efficiently. As we have already seen in the above example that LDA enables us to draw a straight line that can completely separate the two classes of the data points. Here, LDA uses an X-Y axis to create a new axis by separating them using a straight line and projecting data onto a new axis.

Hence, we can maximize the separation between these classes and reduce the 2-D plane into 1-D.



To create a new axis, Linear Discriminant Analysis uses the following criteria:

- **It maximizes the distance between means of two classes.**
- **It minimizes the variance within the individual class.**

Using the above two conditions, LDA generates a new axis in such a way that it can maximize the distance between the means of the two classes and minimizes the variation within each class.

In other words, we can say that the new axis will increase the separation between the data points of the two classes and plot them onto the new axis.

Why LDA?

- Logistic Regression is one of the most popular classification algorithms that perform well for binary classification but falls short in the case of multiple classification problems with well-separated classes. At the same time, LDA handles these quite efficiently.
 - LDA can also be used in data pre-processing to reduce the number of features, just as PCA, which reduces the computing cost significantly.
 - LDA is also used in face detection algorithms. In Fisherfaces, LDA is used to extract useful data from different faces. Coupled with eigenfaces, it produces effective results.
1. Linear Discriminant Analysis (LDA) is a supervised learning algorithm used for classification tasks in machine learning. It is a technique used to find a linear combination of features that best separates the classes in a dataset.
 2. LDA works by projecting the data onto a lower-dimensional space that maximizes the separation between the classes. It does this by finding a set of linear discriminants that maximize the ratio of between-class variance to within-class variance. In other words, it finds the directions in the feature space that best separate the different classes of data.
 3. LDA assumes that the data has a Gaussian distribution and that the covariance matrices of the different classes are equal. It also assumes that the data is linearly separable, meaning that a linear decision boundary can accurately classify the different classes.

LDA has several advantages, including:

It is a simple and computationally efficient algorithm.

It can work well even when the number of features is much larger than the number of training samples.

It can handle multicollinearity (correlation between features) in the data.

However, LDA also has some limitations, including:

It assumes that the data has a Gaussian distribution, which may not always be the case.

It assumes that the covariance matrices of the different classes are equal, which may not be true in some datasets.

It assumes that the data is linearly separable, which may not be the case for some datasets.

It may not perform well in high-dimensional feature spaces.

Linear Discriminant Analysis or **Normal Discriminant Analysis** or **Discriminant Function Analysis** is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space.

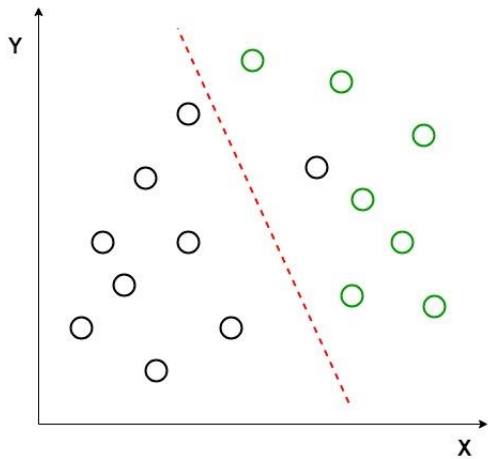
For example, we have two classes and we need to separate them efficiently. Classes can have multiple features.

Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.



Example:

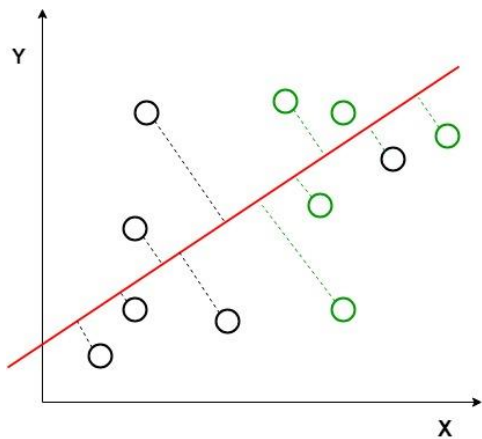
Suppose we have two sets of data points belonging to two different classes that we want to classify. As shown in the given 2D graph, when the data points are plotted on the 2D plane, there's no straight line that can separate the two classes of the data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.



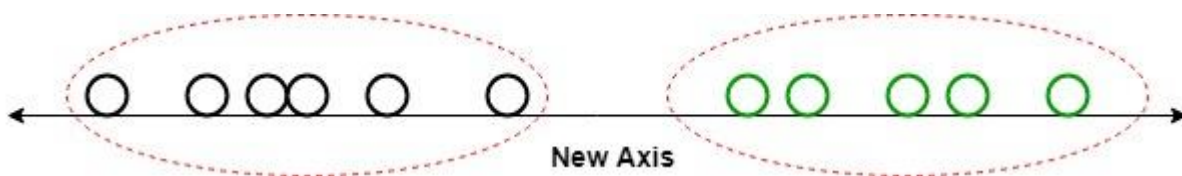
Here, Linear Discriminant Analysis uses both the axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reducing the 2D graph into a 1D graph.

Two criteria are used by LDA to create a new axis:

1. Maximize the distance between means of the two classes.
2. Minimize the variation within each class.



In the above graph, it can be seen that a new axis (in red) is generated and plotted in the 2D graph such that it maximizes the distance between the means of the two classes and minimizes the variation within each class. In simple terms, this newly generated axis increases the separation between the data points of the two classes. After generating this new axis using the above-mentioned criteria, all the data points of the classes are plotted on this new axis and are shown in the figure given below.



But Linear Discriminant Analysis fails when the mean of the distributions are shared, as it becomes impossible for LDA to find a new axis that makes both the classes linearly separable. In such cases, we use non-linear discriminant analysis.

Extensions to LDA:

1. **Quadratic Discriminant Analysis (QDA):** Each class uses its own estimate of variance (or covariance when there are multiple input variables).

2. **Flexible Discriminant Analysis (FDA):** Where non-linear combinations of inputs are used such as splines.
3. **Regularized Discriminant Analysis (RDA):** Introduces regularization into the estimate of the variance (actually covariance), moderating the influence of different variables on LDA.

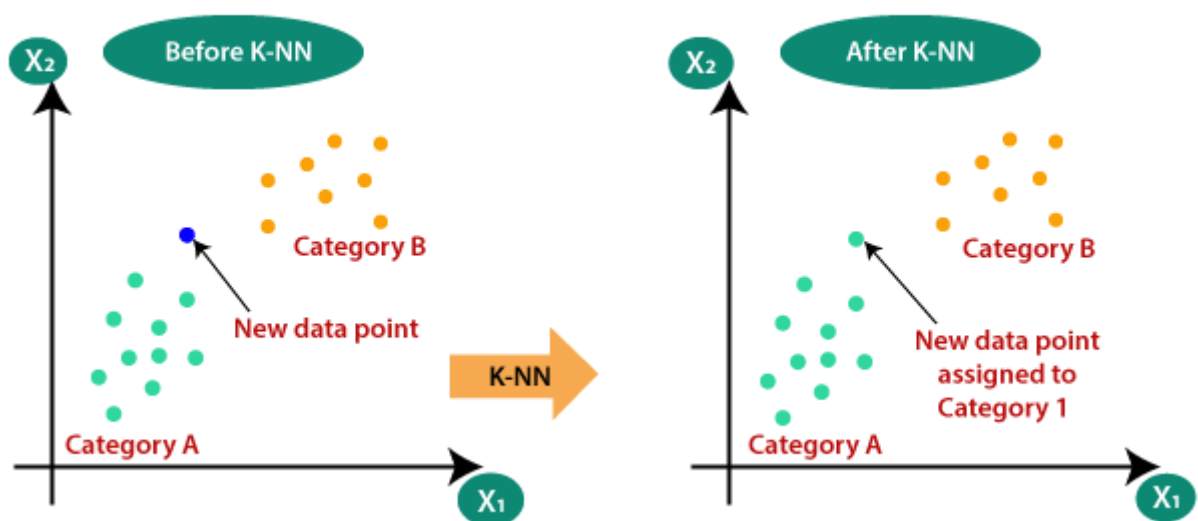
K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x_1 , so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

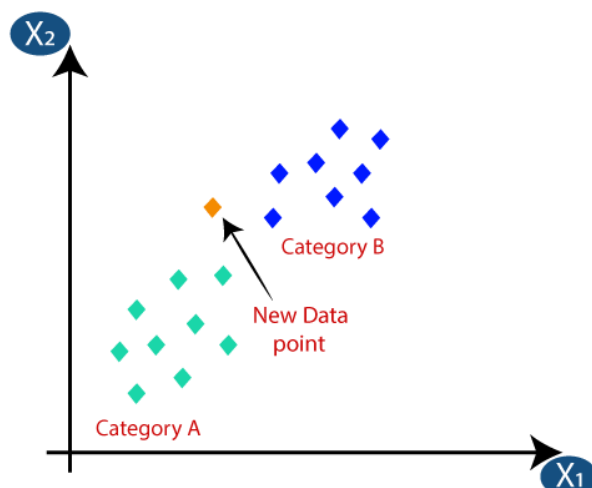


How does K-NN work?

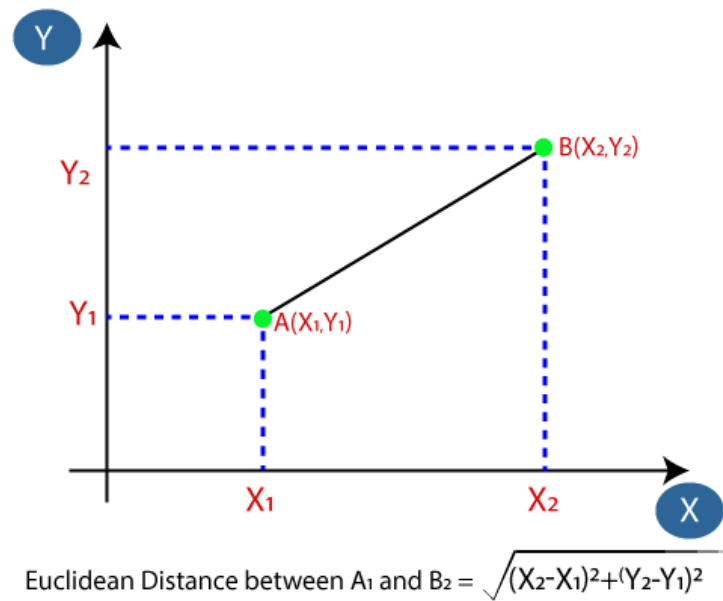
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

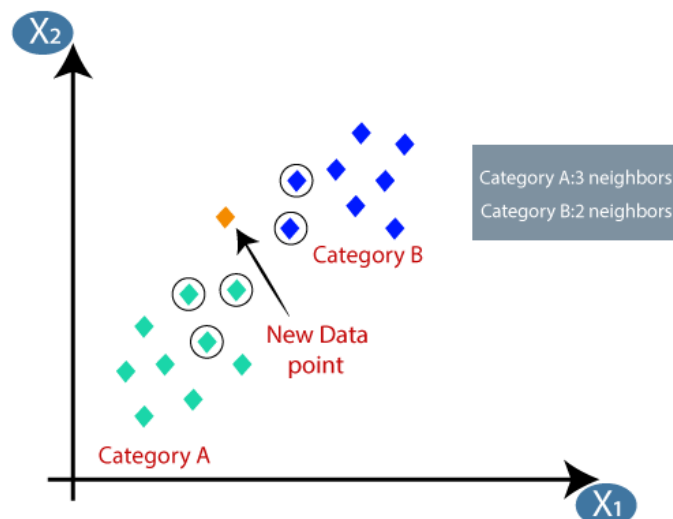
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the $k=5$.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm: There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.

- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data

- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.**
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles.**

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes:** It is called Bayes because it depends on the principle of [Bayes' Theorem](#).

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

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

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2

Total	10	5
-------	----	---

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

$$P(\text{Yes}|\text{Sunny}) = P(\text{Sunny}|\text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{Yes}) = 3/10 = 0.3$$

$$P(\text{Sunny}) = 0.35$$

$$P(\text{Yes}) = 0.71$$

$$\text{So } P(\text{Yes}|\text{Sunny}) = 0.3 * 0.71 / 0.35 = \mathbf{0.60}$$

$$P(\text{No}|\text{Sunny}) = P(\text{Sunny}|\text{No}) * P(\text{No}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{NO}) = 2/4 = 0.5$$

$$P(\text{No}) = 0.29$$

$$P(\text{Sunny}) = 0.35$$

$$\text{So } P(\text{No}|\text{Sunny}) = 0.5 * 0.29 / 0.35 = \mathbf{0.41}$$

So as we can see from the above calculation that $P(\text{Yes}|\text{Sunny}) > P(\text{No}|\text{Sunny})$

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

49

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

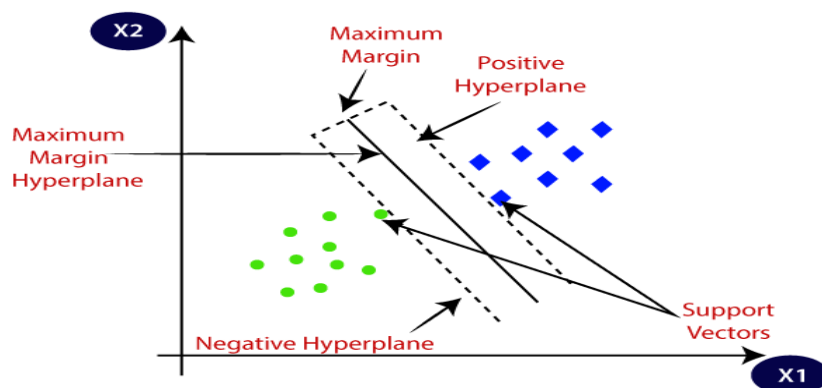
Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

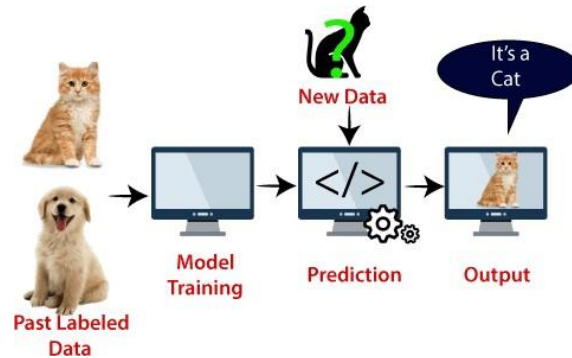
- **Gaussian:** The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial:** The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.
- **Bernoulli:** The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Support Vector Machines

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



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization**, etc. Types of SVM

SVM can be of two types:

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

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

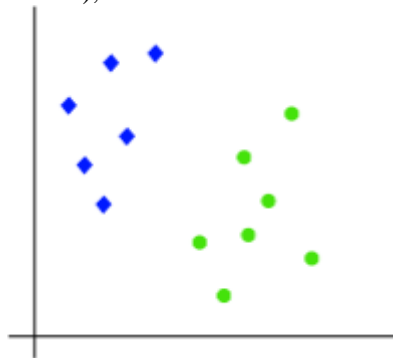
We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Support Vectors:

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

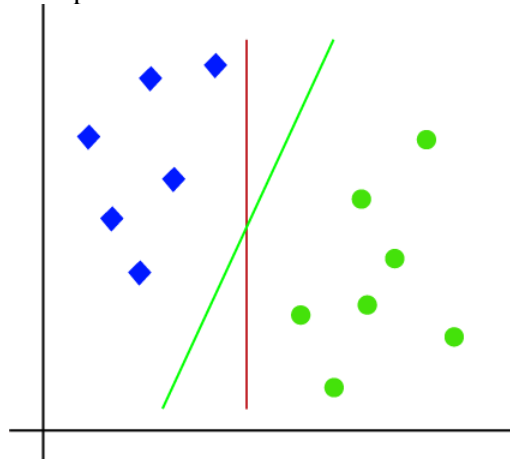
Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x_1 and x_2 . We want a



classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:

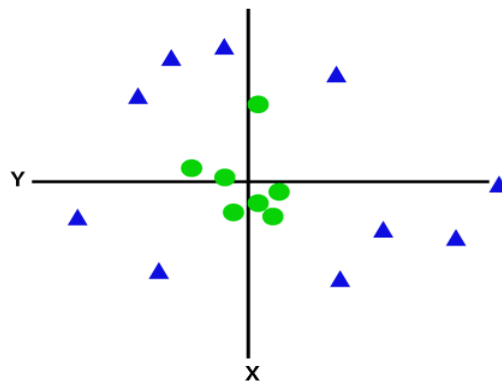
So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



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

Non-Linear SVM:

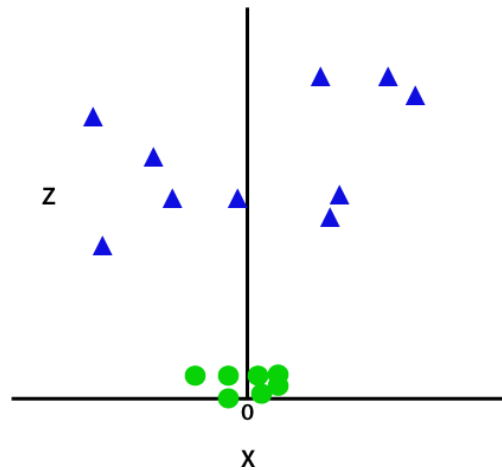
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



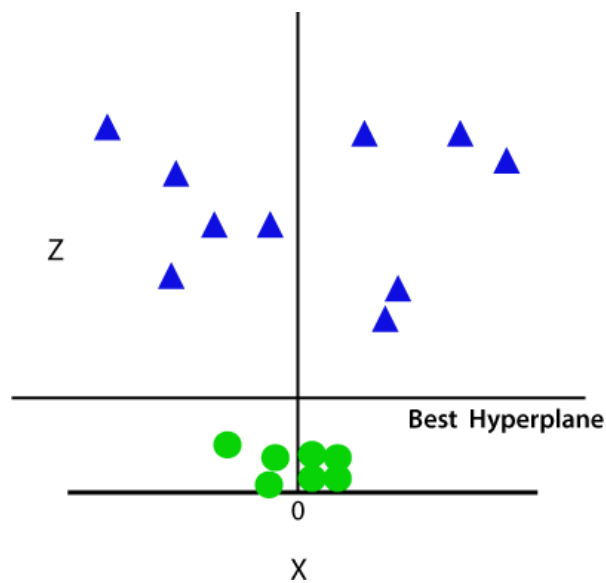
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z = x^2 + y^2$$

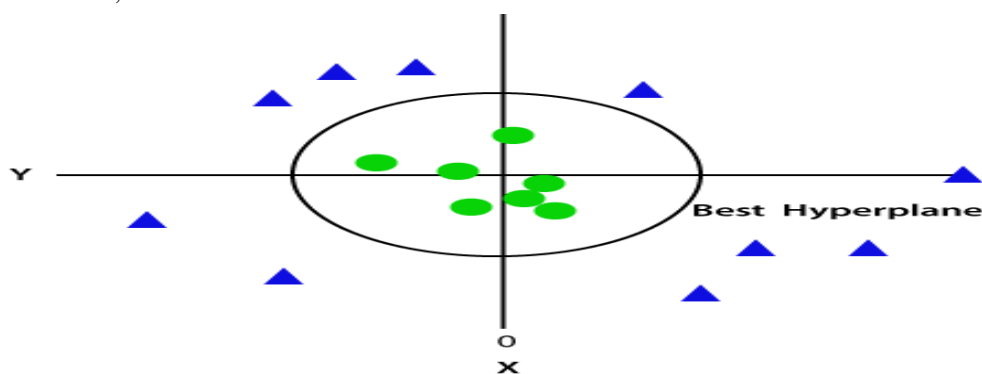
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2dspace with $z=1$, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

SVM Kernels

In practice, SVM algorithm is implemented with kernel that transforms an input data space into

the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non- separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate. The following are some of the types of kernels used by SVM.

Linear Kernel

It can be used as a dot product between any two observations. The formula of linear kernel is as below

$$K(x, x_i) = \sum (x * x_i)$$

From the above formula, we can see that the product between two vectors say x & x_i is the sum of the multiplication of each pair of input values.

UNIT III

Ensemble Learning

Ensemble learning usually produces more accurate solutions than a single model would. Ensemble Learning is a technique that create multiple models and then combine them them to produce improved results. Ensemble learning usually produces more accurate solutions than a single model would.

- Ensemble learning methods is applied to regression as well as classification.
 - Ensemble learning for regression creates multiple repressors i.e. multiple regressionmodels such as linear, polynomial, etc.
 - Ensemble learning for classification creates multiple classifiers i.e. multiple classificationmodels such as logistic, decision tress, KNN, SVM, etc.

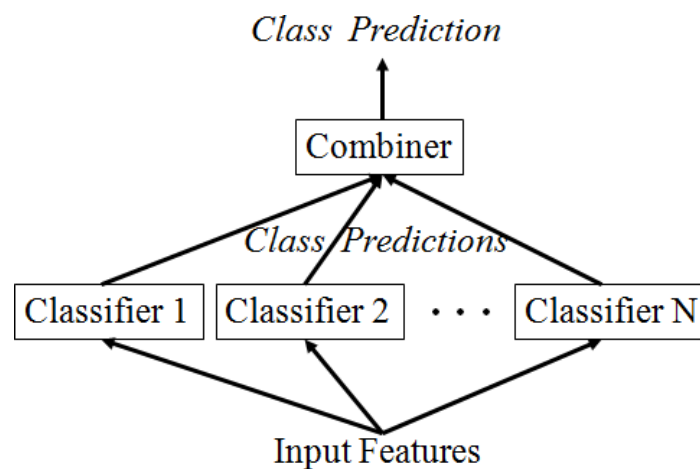


Figure 1: Ensemble learning view

Which components to combine?

- different learning algorithms
- same learning algorithm trained in different ways
- same learning algorithm trained the same way

There are two steps in ensemble learning:

Multiples machine learning models were generated using same or different machine learningalgorithm. These are called “base models”. The prediction perform on the basis of base models.

Techniques/Methods in ensemble learning

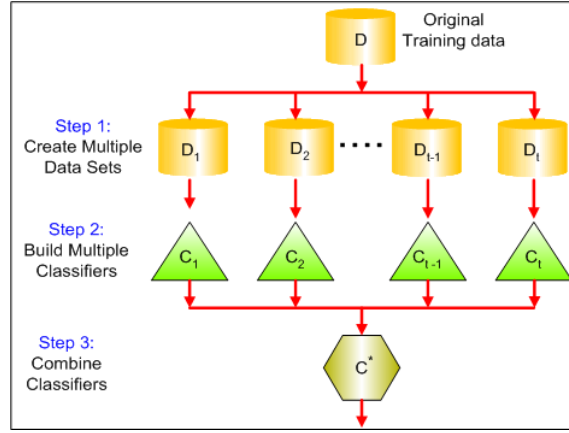
Voting, Error-Correcting Output Codes, Bagging: Random Forest Trees, Boosting: Adaboost, Stacking.

Model Combination Schemes - Combining Multiple Learners

We discussed many different learning algorithms in the previous chapters. Though these are generally successful, no one single algorithm is always the most accurate. Now, we are going to discuss models composed of multiple learners that complement each other so that by combining them, we attain higher accuracy.

There are also different ways the multiple base-learners are combined to generate the final output:

Figure2: General Idea - Combining Multiple Learners



Multiexpert combination

Multiexpert combination methods have base-learners that work in parallel. These methods can in turn be divided into two:

- In the global approach, also called learner fusion, given an input, all base-learners generate an output and all these outputs are used.
Examples are voting and stacking.
- In the local approach, or learner selection, for example, in mixture of experts, there is a gating model, which looks at the input and chooses one (or very few) of the learners as responsible for generating the output.

Multistage combination

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

An example is *cascading*.

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

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

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

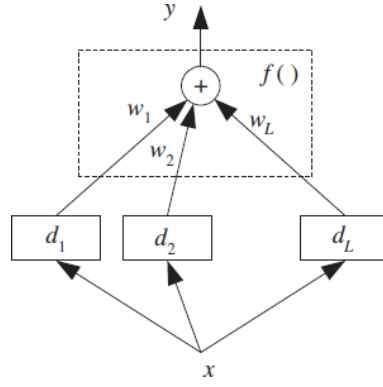


Figure 1: Base-learners are d_j and their outputs are combined using $f(\cdot)$. This is for a single output; in the case of classification, each base-learner has K outputs that are separately used to calculate y_i , and then we choose the maximum. Note that here all learners observe the same input; it may be the case that different learners observe different representations of the same input object or event.

When there are K outputs, for each learner there are $d_{ji}(x)$, $i = 1, \dots, K$, $j = 1, \dots, L$, and, combining them, we also generate K values, y_i , $i = 1, \dots, K$ and then for example inclassification, we choose the class with the maximum y_i value:

$$\text{Choose } C_i \text{ if } y_i = \max_{k=1}^K y_k$$

Voting

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

ers, Refer figure 1.

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

This is also known as *ensembles* and *linear opinion pools*. In the simplest case, all learners are given equal weight and we have *simple voting* that corresponds to taking an average. Still, taking a (weighted) sum is only one of the possibilities and there are also other combination rules, as shown in table 1. If the outputs are not posterior probabilities, these rules require that outputs be normalized to the same scale

Table 1 - Classifier combination rules

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L} \sum_{j=1}^L d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$

An example of the use of these rules is shown in table 2, which demonstrates the effects of different rules. Sum rule is the most intuitive and is the most widely used in practice. Median rule is more robust to outliers; minimum and maximum rules are pessimistic and optimistic, respectively. With the product rule, each learner has veto power; regardless of the other ones, if one learner has an output of 0, the overall output goes to 0. Note that after the combination rules, y_i do not necessarily sum up to 1.

Table 2: Example of combination rules on three learners and three classes

	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
Sum	0.2	0.5	0.3
Median	0.2	0.5	0.4
Minimum	0.0	0.4	0.2
Maximum	0.4	0.6	0.4
Product	0.0	0.12	0.032

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

When there are two classes, this is *majority voting* where the winning class gets more than half of the votes. If the voters can also supply the additional information of how much they vote for each class (e.g., by the posterior probability), then after normalization, these can be used as weights in a *weighted voting* scheme. Equivalently, if d_{ji} are the class posterior probabilities, $P(C_i | x, \mathcal{M}_j)$, then we can just sum them up ($w_j = 1/L$) and choose the class with maximum y_i .

In the case of regression, simple or weighted averaging or median can be used to fuse the outputs of base-regressors. Median is more robust to noise than the average.

Another possible way to find w_j is to assess the accuracies of the learners (regressor or classifier) on a separate validation set and use that information to compute the weights, so that we give more weights to more accurate learners.

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

$$P(C_i | x) = \sum_{\text{all models } \mathcal{M}_j} P(C_i | x, \mathcal{M}_j) P(\mathcal{M}_j)$$

Simple voting corresponds to a uniform prior. If we have a prior distribution preferring simpler models, this would give larger weights to them. We cannot integrate over all models; we only choose a subset for which we believe $P(\mathcal{M}_j)$ is high, or we can have another Bayesian step and calculate $P(C_i | x, \mathcal{M}_j)$, the probability of a model given the sample, and sample high probable models from this density.

Let us assume that d_j are iid with expected value $E[d_j]$ and variance $\text{Var}(d_j)$, then when we take a simple

average with $w_j = 1/L$, the expected value and variance of the output are

$$E[y] = E\left[\sum_j \frac{1}{L} d_j\right] = \frac{1}{L} L E[d_j] = E[d_j]$$

$$\text{Var}(y) = \text{Var}\left(\sum_j \frac{1}{L} d_j\right) = \frac{1}{L^2} \text{Var}\left(\sum_j d_j\right) = \frac{1}{L^2} L \text{Var}(d_j) = \frac{1}{L} \text{Var}(d_j)$$

We see that the expected value does not change, so the bias does not change. But variance, and therefore mean square error, decreases as the number of independent voters, L , increases. In the general case,

$$\text{Var}(y) = \frac{1}{L^2} \text{Var}\left(\sum_j d_j\right) = \frac{1}{L^2} \left[\sum_j \text{Var}(d_j) + 2 \sum_j \sum_{i < j} \text{Cov}(d_j, d_i) \right]$$

which implies that if learners are positively correlated, variance (and error) increase. We can thus view using different algorithms and input features as efforts to decrease, if not completely eliminate, the positive correlation.

Error-Correcting Output Codes

The **Error-Correcting Output Codes** method is a technique that allows a multi-class classification problem to be reframed as multiple binary classification problems, allowing the use of native binary classification models to be used directly.

Unlike [one-vs-rest and one-vs-one methods](#) that offer a similar solution by dividing a multi-class classification problem into a fixed number of binary classification problems, the error-correcting output codes technique allows each class to be encoded as an arbitrary number of binary classification problems. When an overdetermined representation is used, it allows the extra models to act as “error-correction” predictions that can result in better predictive performance.

In *error-correcting output codes* (ECOC), the main classification task is defined in terms of a number of subtasks that are implemented by the base-learners. The idea is that the original task of separating one class from all other classes may be a difficult problem. Instead, we want to define a set of simpler classification problems, each specializing in one aspect of the task, and combining these simpler classifiers, we get the final classifier.

Base-learners are binary classifiers having output $-1/+1$, and there is a *code matrix* \mathbf{W} of $K \times L$ whose K rows are the binary codes of classes in terms of the L base-learners d_j . For example, if the second row of \mathbf{W} is $[-1, +1, +1, -1]$, this means that for us to say an instance belongs to C_2 , the instance should be on the negative side of d_1 and d_4 , and on the positive side of d_2 and d_3 . Similarly, the columns of the code matrix defines the task of the base-learners. For example, if the third column is $[-1, +1, +1]^T$, we understand that the task of the third base-learner, d_3 , is to separate the instances of C_1 from the instances of C_2 and C_3 combined. This is how we form the training set of the base-learners. For example in this case, all instances labeled with C_2 and C_3 form X_3^+ and instances labeled with C_1 form X_3^- , and d_3 is trained so that $x' \in X_3^+$ give output $+1$ and $x' \in X_3^-$ give output -1 .

The code matrix thus allows us to define a polychotomy ($K > 2$ classification problem) in terms of dichotomies ($K = 2$ classification problem), and it is a method that is applicable using any learning

algorithm to implement the dichotomizer base-learners—for example, linear or multilayer perceptrons (with a single output), decision trees, or SVMs whose original definition is for two-class problems.

The typical one discriminant per class setting corresponds to the diagonal code matrix where $L = K$. For example, for $K = 4$,

we have

$$\mathbf{W} = \begin{bmatrix} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{bmatrix}$$

The problem here is that if there is an error with one of the baselearners, there may be a misclassification because the class code words are so similar. So the approach in error-correcting codes is to have $L > K$ and increase the Hamming distance between the code words. One possibility is *pairwise separation* of classes where there is a separate baselearner to separate C_i from C_j , for $i < j$. In this case, L

$= K(K - 1)/2$ and with $K = 4$, the code matrix is

$$\mathbf{W} = \begin{bmatrix} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}$$

where a 0 entry denotes “don’t care.” That is, d_1 is trained to separate C_1 from C_2 and does not use the training instances belonging to the other classes. Similarly, we say that an instance belongs to C_2 if $d_1 =$

-1 and $d_4 = d_5 = +1$, and we do not consider the values of d_2 , d_3 , and d_6 . The problem here is that L is $O(K^2)$, and for large K pairwise separation may not be feasible.

If we can have L high, we can just randomly generate the code matrix with $-1/+1$ and this will work fine, but if we want to keep L low, we need to optimize \mathbf{W} . The approach is to set L beforehand and then find \mathbf{W} such that the distances between rows, and at the same time the distances between columns, are as large as possible, in terms of Hamming distance. With K classes, there are $2^{(K-1)} - 1$ possible columns, namely, two-class problems. This is because K bits can be written in $2K$ different ways and complements (e.g., “0101” and “1010,” from our point of view, define the same discriminant) dividing the possible combinations by 2 and then subtracting 1 because a column of all 0s (or 1s) is useless. For example, when $K = 4$, we have

$$\mathbf{W} = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & +1 & +1 & +1 & +1 \\ -1 & +1 & +1 & -1 & -1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 \end{bmatrix}$$

When K is large, for a given value of L , we look for L columns out of the $2^{(K-1)} - 1$. We would like these columns of \mathbf{W} to be as different as possible so that the tasks to be learned by the base-learners are as different from each other as possible. At the same time, we would like the rows of \mathbf{W} to be as different as possible so that we can have maximum error correction in case one or more base-learners fail.

ECOC can be written as a voting scheme where the entries of \mathbf{W} , w_{ij} , are considered as vote weights:

$$y_i = \sum_{j=1}^L w_{ij} d_j$$

and then we choose the class with the highest y_i . Taking a weighted sum and then choosing the maximum instead of checking for an exact match allows d_j to no longer need to be binary but to take a value between -1 and $+1$, carrying soft certainties instead of hard decisions. Note that a value p_j between 0 and 1, for example, a posterior probability, can be converted to a value d_j between -1 and $+1$ simply as

$$d_j = 2p_j - 1$$

One problem with ECOC is that because the code matrix \mathbf{W} is set a priori, there is no guarantee that the subtasks as defined by the columns of \mathbf{W} will be simple.

Bagging

Bootstrap aggregating, often abbreviated as bagging, involves having each model in the ensemble vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set. As an example, the [random forest](#) algorithm combines random decision trees with bagging to achieve very high classification accuracy.

The simplest method of combining classifiers is known as bagging, which stands for bootstrap aggregating, the statistical description of the method. This is fine if you know what a bootstrap is, but fairly useless if you don't. A bootstrap sample is a sample taken from the original dataset with replacement, so that we may get some data several times and others not at all. The bootstrap sample is the same size as the original, and lots and lots of these samples are taken: B of them, where B is at least 50, and could even be in the thousands. The name bootstrap is more popular in computer science than anywhere else, since there is also a bootstrap loader, which is the first program to run when a computer is turned on. It comes from the nonsensical idea of 'picking yourself up by your bootstraps,' which means lifting yourself up by your shoelaces, and is meant to imply starting from nothing.

Bootstrap sampling seems like a very strange thing to do. We've taken a perfectly good dataset, mucked it up by sampling from it, which might be good if we had made a smaller dataset (since it would be faster), but we still ended up with a dataset the same size. Worse, we've done it lots of times. Surely this is just a way to burn up computer time without gaining anything. The benefit of it is that we will get lots of learners that perform slightly differently, which is exactly what we want for an ensemble method. Another benefit is that estimates of the accuracy of the classification function can be made without complicated analytic work, by throwing computer resources at the problem (technically, bagging is a variance reducing algorithm; the meaning of this will become clearer when we talk about bias and variance). Having taken a set of bootstrap samples, the bagging method simply requires that we fit a model to each dataset, and then combine them by taking the output to be the majority vote of all the classifiers. A NumPy implementation is shown next, and then we will look at a simple example.

```
# Compute bootstrap samples
```

```
samplePoints =  
np.random.randint(0,nPoints,(nPoints,nSamples))  
classifiers =  
[]
```

```
for i in range(nSamples):
```

```
    sample = []
```

```
    sampleTarget
```

```
= []for j in
```

```
range(nPoints):
```

```
    sample.append(data[samplePoints[j,i]])
```

```

sampleTarget.append(targets[samplePoints[j,i]])
# Train classifiers
classifiers.append(self.tree.make_tree(sample,sampleTarget,features))

```

The example consists of taking the party data that was used to demonstrate the decision tree, and restricting the trees to stumps, so that they can make a classification based on just one variable

When we want to construct the decision tree to decide what to do in the evening, we start by listing everything that we've done for the past few days to get a suitable dataset (here, the last ten days):

Deadline?	Is there a party?	Lazy?	Activity
Urgent	Yes	Yes	Party
Urgent	No	Yes	Study
Near	Yes	Yes	Party
None	Yes	No	Party
None	No	Yes	Pub
None	Yes	No	Party
Near	No	No	Study
Near	No	Yes	TV
Near	Yes	Yes	Party
Urgent	No	No	Study

The output of a decision tree that uses the whole dataset for this is not surprising: it takes the two largest classes, and separates them. However, using just stumps of trees and 20 samples, bagging can separate the data perfectly, as this output shows:

```

Tree Stump Prediction
['Party', 'Party', 'Party', 'Party', 'Pub', 'Party', 'Study', 'Study', 'Party', 'Study']
Correct Classes
['Party', 'Study', 'Party', 'Party', 'Pub', 'Party', 'Study', 'TV', 'Party', 'Study']
Bagged Results
['Party', 'Study', 'Party', 'Party', 'Pub', 'Party', 'Study', 'TV', 'Party', 'Study']

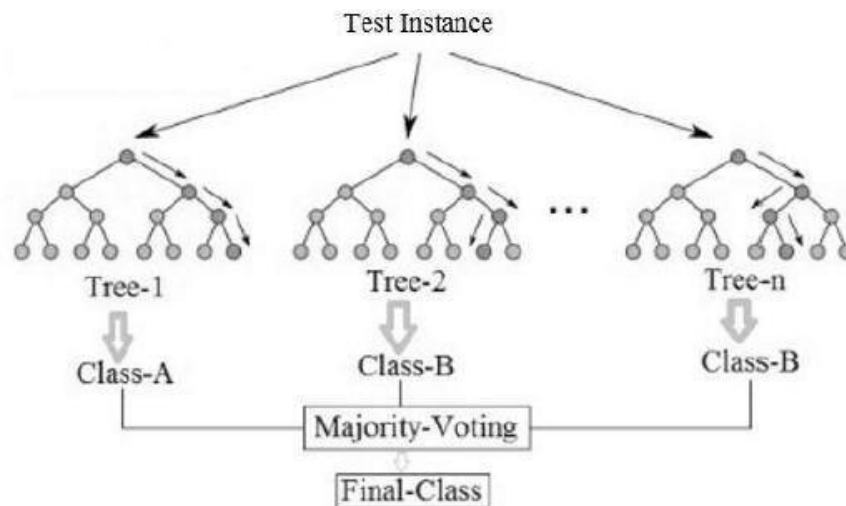
```

RANDOM FORESTS

A random forest is an ensemble learning method where multiple decision trees are constructed and then they are merged to get a more accurate prediction.

If there is one method in machine learning that has grown in popularity over the last few years, then it is the idea of random forests. The concept has been around for longer than that, with several different people inventing variations, but the name that is most strongly attached to it is that of Breiman, who also described the CART algorithm in unit 2.

Figure 3: Example of random forest with majority voting



The idea is largely that if one tree is good, then many trees (a forest) should be better, provided that there is enough variety between them. The most interesting thing about a random forest is the ways that it creates randomness from a standard dataset. The first of the methods that it uses is the one that we have just seen: bagging. If we wish to create a forest then we can make the trees different by training them on slightly different data, so we take bootstrap samples from the dataset for each tree. However, this isn't enough randomness yet. The other obvious place where it is possible to add randomness is to limit the choices that the decision tree can make. At each node, a random subset of the features is given to the tree, and it can only pick from that subset rather than from the whole set.

As well as increasing the randomness in the training of each tree, it also speeds up the training, since there are fewer features to search over at each stage. Of course, it does introduce a new parameter (how many features to consider), but the random forest does not seem to be very sensitive to this parameter; in practice, a subset size that is the square root of the number of features seems to be common. The effect of these two forms of randomness is to reduce the variance without effecting the bias. Another benefit of this is that there is no need to prune the trees. There is another parameter that we don't know how to choose yet, which is the number of trees to put into the forest. However, this is fairly easy to pick if we want optimal results: we can keep on building trees until the error stops decreasing.

Once the set of trees are trained, the output of the forest is the majority vote for classification, as with the other committee methods that we have seen, or the mean response for regression. And those are pretty much the main features needed for creating a random forest. The algorithm is given next before we see some results of using the random forest.

Algorithm

Here is an outline of the random forest algorithm.

1. The random forests algorithm generates many classification trees. Each tree is generated as follows:
 - a) If the number of examples in the training set is N , take a sample of N examples at random - but with replacement, from the original data. This sample will be the training set for generating the tree.
 - b) If there are M input variables, a number 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 generation of the various trees in the forest.

- c) Each tree is grown to the largest extent possible.
2. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree “votes” for that class. The forest chooses the classification

The implementation of this is very easy: we modify the decision to take an extra parameter, which is m , the number of features that should be used in the selection set at each stage. We will look at an example of using it shortly as a comparison to boosting.

Looking at the algorithm you might be able to see that it is a very unusual machine learning method because it is embarrassingly parallel: since the trees do not depend upon each other, you can both create and get decisions from different trees on different individual processors if you have them. This means that the random forest can run on as many processors as you have available with nearly linear speedup.

There is one more nice thing to mention about random forests, which is that with a little bit of programming effort they come with built-in test data: the bootstrap sample will miss out about 35% of the data on average, the so-called out-of-bootstrap examples. If we keep track of these datapoints then they can be used as novel samples for that particular tree, giving an estimated test error that we get without having to use any extra datapoints.

This avoids the need for cross-validation.

As a brief example of using the random forest, we start by demonstrating that the random forest gets the correct results on the Party example that has been used in both this and the previous chapters, based on 10 trees, each trained on 7 samples, and with just two levels allowed in each tree:

```
RF prediction
['Party', 'Study', 'Party', 'Party', 'Pub', 'Party', 'Study', 'TV', 'Party', 'Study']
```

As a rather more involved example, the car evaluation dataset in the UCI Repository contains 1,728 examples aiming to classify whether or not a car is a good purchase based on six attributes. The following results compare a single decision tree, bagging, and a random forest with 50 trees, each based on 100 samples, and with a maximum depth of five for each tree. It can be seen that the random forest is the most accurate of the three methods.

```

Tree
Number correctly predicted 777.0
Number of testpoints 864
Percentage Accuracy 89.9305555556

Number of cars rated as good or very good 39.0
Number correctly identified as good or very good 18.0
Percentage Accuracy 46.1538461538
-----
Bagger
Number correctly predicted 678.0
Number of testpoints 864
Percentage Accuracy 78.4722222222

Number of cars rated as good or very good 39.0
Number correctly identified as good or very good 0.0
Percentage Accuracy 0.0
-----
Forest
Number correctly predicted 793.0
Number of testpoints 864
Percentage Accuracy 91.7824074074

Number of cars rated as good or very good 39.0
Number correctly identified as good or very good 20.0
Percentage Accuracy 51.28205128

```

Strengths and weaknesses

Strengths

The following are some of the important strengths of random forests.

- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.
- Random forest run times are quite fast, and they are able to deal with unbalanced and missing data.
- They can handle binary features, categorical features, numerical features without any need for scaling.

Weaknesses

- A weakness of random forest algorithms is that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.

- The sizes of the models created by random forests may be very large. It may take hundreds of megabytes of memory and may be slow to evaluate.
- Random forest models are black boxes that are very hard to interpret.

