Unit 2. Supervised Learning (10 Hrs.)

<u>Objective:</u>

✓ *Design and implement supervised learning algorithms to solve real world problems.*

2.1. Types of Supervised Learning

Supervised machine learning involves training a model on labeled data to learn patterns and relationships, which it then uses to make accurate predictions on new data.

Supervised learning is a category of machine learning that uses labeled datasets to train algorithms to predict outcomes and recognize patterns.

- Supervised learning algorithms are given labeled training to learn the relationship between the input and the outputs.
- The goal of supervised learning is to make accurate predictions when given new, unseen data.
- Supervised machine learning algorithms make it easier for organizations to create complex models that can make accurate predictions.
- Results of Supervised Machine Learning are widely used across various industries and fields, including healthcare, marketing, financial services, and more.

How does Supervised Learning Work?

- The data used in supervised learning are labeled (i.e. it contains examples of both inputs (called features) and correct outputs (labels or target variables)).
- The algorithms analyze a large dataset of training pairs to infer what a desired output value would be when asked to make a prediction on new data. This is achieved by adjusting the model's parameters to minimize the difference between its predictions and the actual labels.
- Over time, it adjusts itself to minimize errors and improve accuracy.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Example Case-2.1

Suppose you want to teach a model to identify pictures of animal. You provide a labeled dataset that contains many different examples of types of animals and the names of each species. You let the algorithm try to define what set of characteristics belongs to each animal based on the labeled outputs. You can then test the model by showing it an animal picture and asking it to guess what species it is. If the model provides an incorrect answer, you can continue training it and adjusting its parameters with more examples to improve its accuracy and minimize errors.

Once the model has been trained and tested, you can use it to make predictions on unknown data based on the previous knowledge it has learned.

Common supervised learning examples includes,

- Image Classification: For example, an algorithm might be used to recognize a person in an image and automatically tag them on a social media platform.
- Fraud Detection: For example, Models are trained on datasets that contain both fraudulent and non-fraudulent activity so they can be used to flag suspicious activity in real time.
- Recommendation systems: For example, online platforms and streaming services to power recommendations based on previous customer behavior or shopping history.
- Risk assessment: For example, banks and other financial services companies determine whether customers are likely to default loans, helping to minimize risk in their portfolios.

Supervised Learning can be applied to mainly **two types** of problems:

2.1.1. Regression

2.1.2. Classification

2.2. Regression

Regression is supervised learning techniques where the goal is to predict a continuous numerical value as an output based on one or more independent features in which machine learning algorithm is trained with both input features and output labels.

- Regression finds the relationships between variables by estimating how one variable affects the other so that predictions can be made.
- Regression algorithms are used to predict a real or continuous value, where the algorithm detects a relationship between two or more variables.
- There are two types of variables used in regression
  1. ***Dependent Variable or Target Variable (Y)***: variable we are trying to predict (for e.g. price of house)
  2. ***Independent Variable or Features or predictors (X):*** variables that influence the prediction (in above example 2.3, area, bedrooms, age and location are independent variables)

Mathematically: Regression is statistical methods that allow to predict a continuous outcome (y) based on the value of one or more predictor variables (x).

Example-2.2

Let's predict a salary of person based on work experience. For instance, a supervised learning algorithm would be fed inputs related to work experience (e.g., duration of time, the industry or field, location, etc.) and the corresponding assigned salary amount. After the model is trained, it could be used to predict the average salary based on work experience.

Example-2.3

Imagine you are trying to predict the price of a house. You might consider various factors like its size, location, age and number of bedrooms. In a house price dataset, the independent variables are columns used to predict, such as the "Area", "Bedrooms", "Age", and "Location". The Dependent variable will be the "Price" column – the feature to be predicted.

In Summary: Regression is a statistical method used in machine learning to model and analyze the relationships between a dependent variable (target) and one or more independent variables (predictors). The primary purpose of regression is to predict a continuous outcome based on input features.

Regression can be ***classified into different types*** based on the number of predictor variables and the nature of the relationship between variables.

2.2.1. Linear Regression

Linear regression is a statistical technique used to find the relationship between a dependent variable and one or two or more than two independent variables.

- In an ML context, linear regression finds the relationship between features and a label. And can be used for both prediction and understanding the nature of the relationship.
- Also called simple regression.
- Regression allows you to estimate how a dependent variable changes as the independent variable(s) change.
- Can be used in predicting house prices, stock prices, or sales based on multiple factors like location, interest rates, and marketing spend.

2.2.1.1. Simple and multiple regression

Simple Linear Regression

Simple linear regression is a statistical method used to model the relationship between two variables: a dependent variable and an independent variable.

- The dependent variable is the variable we want to predict, while the independent variable is the variable that we use to make the prediction.
- In simple linear regression, we assume that there is a linear relationship between the two variables i.e. the change in the independent variable is directly proportional to the change in the dependent variable.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

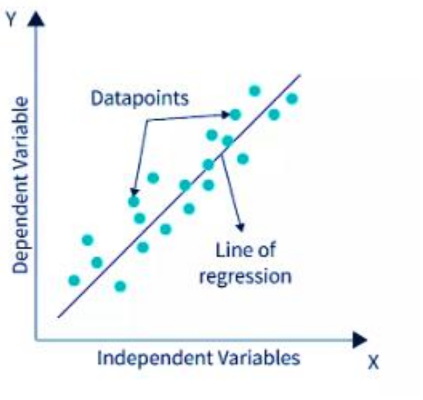


Figure: Linear Regression (best fit line)

In algebraic terms (or you can say mathematically),

Simple linear regression can be expressed as $y = mx + b$

where, y is the dependent variable / predicted value

x is the independent variable / input variable

m is the slope of line (how much y changes when x changes

b is the intercept (the value of y when x=0)

For example, in the linear regression formula of $y = 4x + 7$, there is only one possible outcome of "y" if "x" is defined as 2.

In Machine Learning,

We write the equation for a linear regression model as follows:

$y' = b + w_1x_1 + e$

Where,

y′ is the predicted label / output.

b is the **bias** of the model. In ML, bias is sometimes referred to as w_0.

w_1 is the **weight** of the feature.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

x_1 is a **feature** / input. And e is error of the estimation

During training, the model calculates the weight and bias that produce the best model.

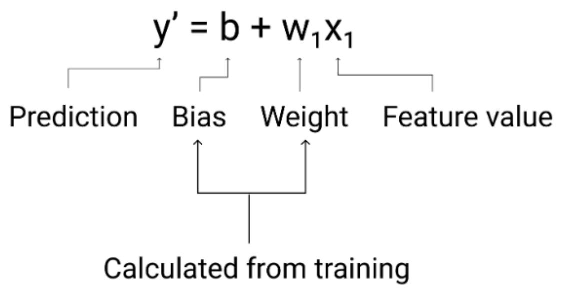


Figure: Mathematical Representation of linear model

<u>Multiple Linear Regression</u>

Multiple linear regression is a statistical method used to model the relationship between two variables: a dependent variable and two or more predictors or independent variable.

The idea behind multiple linear regression is similar to simple linear regression, except that we now have multiple independent variables that we use to make our prediction.

Example:

let’s say we want to predict a person’s salary based on their age, education, and years of experience. In this case, salary is the dependent variable, while age, education, and years of experience are the independent variables. We would collect data on these variables for a sample of individuals and use this data to create a regression model. The model would allow us to predict a person’s salary based on their age, education, and years of experience.

$y' = b + w_1x_1 + w_2x_2 + w_3x_3 \ldots\ldots\ldots\ldots + w_nx_n + e$

where y’, b w and x have similar meaning as in simple linear regression

e = model error (i.e. how much variation there is in your estimation of y’) in above example there are 3 independent variables so equation is up to w_3x_3. when the input variables are increased then the equation adjusts accordingly.

<u>What is the difference between Simple linear and multiple linear regression?</u>

The main difference between simple linear regression and multiple linear regression is the number of independent variables used in the model.

Another difference is the complexity of the model. Simple linear regression models are relatively simple and easy to interpret, as they involve only two variables. Multiple linear regression models, on the other hand, are more complex and require more computational power. They also require more careful interpretation, as the relationships between the independent variables and the dependent variable can be more difficult to understand.

2.2.1.2. Polynomial Regression

Polynomial regression is used when the relationship between the dependent and independent variables is nonlinear. It fits a polynomial equation (e.g., quadratic, cubic) to the data points, making it suitable for datasets with curves where the relationship between the variables is better represented by a curve rather than a straight line.

- Polynomial regression is an essential extension of linear regression used to model non-linear relationships in data.
- In many real-world scenarios, the relationship between variables isn't linear, making polynomial regression a suitable alternative for achieving better predictive accuracy.
- This technique allows machine learning models to capture curved patterns in data by fitting polynomial equations of higher degrees.
- Understanding polynomial regression equips data scientists with the tools needed to handle complex datasets effectively.
- There is no limit to the degree in a polynomial equation, and it can go up to the nth value, numerous kinds of polynomial regression exist.
- Generally, second degree of a polynomial equation is used.

As a key method in supervised learning, it can be used in Modeling complex relationships in data such as,

- predicting the progression of diseases or analyzing the impact of multiple factors on temperature changes.
- predict a non-linear trend like population growth over time etc.
- Predicting Tissue Growth Rates
- Estimating Mortality Rates based on factors like age, lifestyle, and environmental conditions
- Speed Control in Automated Systems such as those used in self-driving cars, speed adjustments are often influenced by factors like terrain, traffic, and weather conditions
- Finance, biology, physics where patterns are often non-linear.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Example:

Imagine you want to predict how many likes your new social media post will have at any given point after the publication. There is no linear correlation between the number of likes and the time that passes. Your new post will probably get many likes in the first 24 hours after publication, and then its popularity will decrease. Such kind of problems can be modeled by using polynomial regression model as linear regression.

Mathematically,

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modelled as an *nth-degree* polynomial.

When the relationship is non-linear, a polynomial regression model introduces higher-degree polynomial terms.

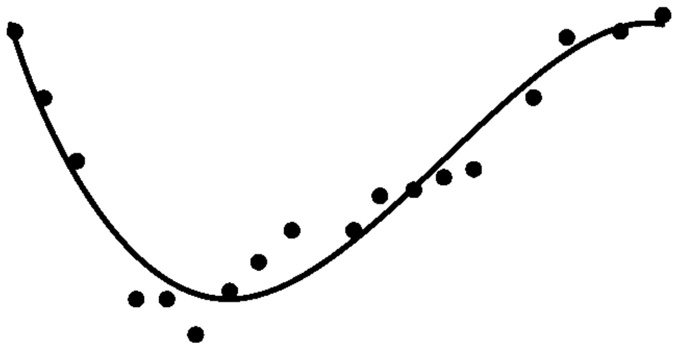


Figure: Polynomial Regression

The general form of a polynomial regression equation of degree n is:

$$y' = b + w_1x_1 + w_2x_2^2 + w_3x_3^3 \ldots\ldots\ldots\ldots + w_n x_n^n + e$$

where,

y is the dependent variable.

x is the independent variable.

$w_1, w_2, w_3 \ldots w_n$ are the weight / coefficients of the polynomial terms.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

n is the degree of the polynomial.

e- represents the error term.

Advantages of Polynomial Linear Regression

- Flexibility for Non-Linear Datasets
- Applicability Across Various Fields: Biology, Finance.
- Better Performance in Specific Scenarios

Dis-advantages of Polynomial Linear Regression

- Overfitting Risk with High-Degree Polynomials
- Computational Complexity
- Selecting the Optimal Polynomial Degree

2.2.2. Regularization Techniques

Imagine you're an architect, standing before a majestic blueprint of a future skyscraper. It's meant to be tall, imposing, and graceful. But there's one problem. It's at risk of becoming unstable. To prevent this, you need to ensure that its structural design doesn't get overloaded with unnecessary materials. Every beam, every bolt, must serve a purpose no more, no less.

Similarly, in machine learning, building a model that fits the data is much like building that skyscraper. Too much complexity and the model, like your building, will collapse under its own weight.

But, ***how do we manage the complexity of our machine learning models?***

- The answer is regularization.

In multiple and polynomial regression, if we rely too much on numerous features, the machine learning model works perfectly on training data but fail miserably on new / unseen data (overfitting). In such cases, model is not only learning the core patterns but also the noise in the data.

Regularization removes or shrinks unnecessary coefficients in the model, helping you build something strong and efficient.

Regularization is a method of adding a penalty term to the cost function of a linear regression model, which reduces the magnitude of the coefficients or weights. The penalty term can be either L1 or L2 norm, which correspond to different types of regularization: Lasso and Ridge, respectively. By adding regularization, you can prevent the model from fitting too closely to the training data and reduce the variance of the model. However, regularization also introduces some bias to the model, as it shrinks the coefficients towards zero or a constant.

Technically,

Regularization is a machine learning technique designed to prevent overfitting by adding a penalty term to the model’s loss function. This penalty reduces the model’s tendency to over-rely on specific features by shrinking the magnitude of the coefficients, leading to better generalization on unseen data. By controlling model complexity, regularization ensures the model captures true patterns rather than fitting noise in the training data, improving performance on new datasets.

In **linear regression**, we typically minimize the sum of squared errors (also known as the **Mean Squared Error**, or MSE) between the predicted and actual outputs. Regularization modifies this loss function / cost function by adding a penalty term based on the size of the coefficients (weights) associated with the features in the model.

$$\text{MSE}=\frac{1}{n}\sum_{i=1}^{n}(y_i - {y'}_i)^2$$

This equation represents the Mean Square Error for Linear Regression which measures how well the model predicts the target variable. Where,

y_i Actual value for the ith sample.

Y'_i Predicted value for the ith sample.

n Total number of training samples.

This is the traditional error term used in linear regression.

In practice, regularization involves adjusting the coefficients so that the model doesn’t place too much importance on any one feature. There are two main types of regularization in linear regression:

2.2.2.1. L2 Regularization (Ridge Regression):

Shrinks the coefficients by adding the sum of their squares to the loss function.

In Ridge Regression, we add the squared values of the coefficients to the loss function. This penalizes larger coefficients, forcing the model to maintain smaller weights. In essence, Ridge discourages the model from becoming overly dependent on any one feature.

The loss function / Cost function for Ridge Regression looks like this:

$$L_{\text{ridge}}=\frac{1}{n}\sum_{i=1}^{n}(y_i - y'_i)^2 + \lambda \sum_{j=1}^{p}{\beta_j}^2$$

Where, $\frac{1}{n}\sum_{i=1}^{n}(y_i - y'_i)^2$ is the MSE and $\lambda \sum_{j=1}^{p}{\beta_j}^2$ is the L2 Regularization term (penalty). This term Adds a penalty based on the size of the model coefficients.

Where,

Bj-The j-th model parameter (weight).

p: Total number of features (predictors).

λ: Regularization strength (hyperparameter). If

λ=0: No regularization → Equivalent to standard linear regression.

Λ large: More regularization → Coefficients are penalized more heavily → Model becomes simpler and possibly underfits.

The right value of λ\lambdaλ is typically chosen using **cross-validation**.

MSE ensures the model fits the data well and L2 Penalty prevents overfitting by shrinking the coefficient Bj towards zero (but not exactely zero).

Here, **λ (lambda)** is the regularization parameter that controls how much we penalize the model for large coefficients. A higher value of λ means more regularization, i.e., more emphasis on keeping the coefficients small.

2.2.2.2. L1 Regularization (Lasso Regression):

Shrinks the coefficients by adding the sum of their absolute values to the loss function.

Lasso Regression is similar to Ridge, but instead of adding the squared values of the coefficients, it adds their absolute values. This has the effect of shrinking some coefficients all the way to zero, effectively removing irrelevant features from the model.

The loss function for Lasso looks like this:

$$L_{\text{lasso}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - y'_i)^2 + \lambda \sum_{j=1}^{p}\left|\beta_j\right|$$

Lasso performs **feature selection** by eliminating features that don't contribute much to the model. This is particularly useful when you have a large number of features and want to identify the most important ones.

<u>How to choose the optimal level of regularization?</u>

The optimal level of regularization depends on the data and the problem you are trying to solve. One way to choose the optimal level of regularization is to use cross-validation, which is a technique of splitting the data into multiple subsets and using some of them for training and some of them for validation. By varying the regularization parameter, you can evaluate how well the model performs on the validation data, and choose the value that minimizes the validation error. Another way to choose the optimal level of regularization is to use grid search or random search, which are methods of exploring a range of possible values for the regularization parameter and finding the best one based on a scoring metric.

2.2.2.3. Bias-variance tradeoff

The bias-variance trade-off is a fundamental concept in machine learning, which describes the trade-off between the error due to the model's complexity and the error due to the model's sensitivity to the data.

- A high-bias model is one that is too simple and cannot capture the underlying patterns of the data, resulting in a large error on both the training and the test data.
- A high-variance model is one that is too complex and fits the training data too well, resulting in a small error on the training data but a large error on the test data.

- Ideally, you want to find a model that has low bias and low variance, which means that it can generalize well to new data.

How do we pick the best model? To address this question, we must first comprehend the trade-off between bias and variance.

- The mistake is due to the model's simple assumptions in fitting the data is referred to as bias. A high bias indicates that the model is unable to capture data patterns, resulting in under-fitting.
- The mistake caused by the complicated model trying to match the data is referred to as variance. When a model has a high variance, it passes over the majority of the data points, causing the data to overfit.

As the model complexity grows, the bias reduces while the variance increases, and vice versa. A machine learning model should, in theory, have minimal variance and bias. However, having both is nearly impossible. As a result, a trade-off must be made in order to build a strong model that performs well on both train and unseen data.

How does regularization affect the bias-variance trade-off?

Regularization is a way of controlling the complexity of a linear regression model, by penalizing the coefficients that are not important or relevant for the prediction. By doing so, regularization can reduce the variance of the model, as it prevents overfitting and makes the model more robust to noise and outliers. However, regularization also increases the bias of the model, as it forces the coefficients to be smaller or zero, which means that the model may not capture some of the true relationships between the variables. Therefore, regularization involves a trade-off between bias and variance, and you need to find the optimal level of regularization that minimizes the total error of the model.

2.2.3. Support Vector Regression

- Support Vector Regression (SVR) is a type of Support Vector Machine (SVM) algorithms, commonly used for regression analysis.

- SVMs are powerful supervised learning algorithms that are primarily used for classification problems.
- SVM aim to find the optimal *hyperplane* that best separates the two classes in the input data.
- A hyperplane is a flat subspace of dimension p-1 in a p-dimensional space, where p is the number of input features.
- The optimal hyperplane is the one that maximizes the *margin*, which is the distance between the hyperplane and the closest data points from each class, known as *support vectors* (for detailed explanation see on classification section below)

Similar to SVMs, SVR uses the concept of a hyperplane and margin but there are differences in their definitions.

- In SVR, the margin is defined as the error tolerance of the model, which is also called as the ε-*insensitive tube*.
- This tube allows some deviation of the data points from the hyperplane without being counted as errors.
- The hyperplane is the *best fit* possible to the data that fall within the ϵ-insensitive tube. The difference of SVM and SVR is summarized in the figure below.

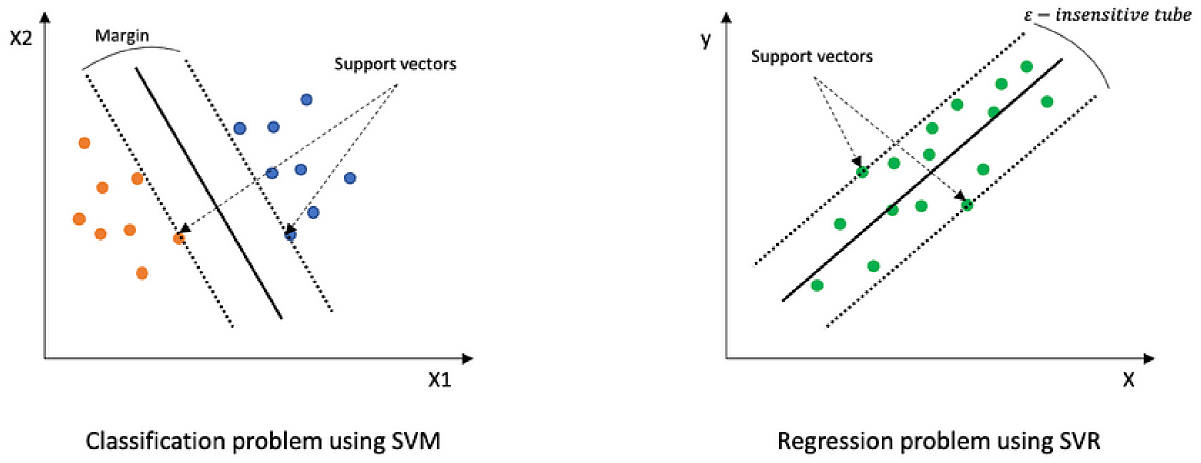

X2
Margin
Support vectors
X1
Classification problem using SVM
y
ε – insensitive tube
Support vectors
X
Regression problem using SVR

Figure: difference of SVM and SVR [IS: Medium]

In SVR, the goal is finding the best fit that accurately predicts the target variable while reducing complexity to avoid **overfitting**.

- To achieve this, an ε-insensitive tube around the hyperplane is defined, which permits some level of deviation between the actual and predicted target values, creating a balance between the complexity of the model and its generalization power.
- SVR can be mathematically formulated as a convex optimization problem.
- The objective of problem is to find a function *f(x)* that is as flat as possible while having a maximum deviation of ε from the actual targets for all the training data.
- Flatness of the function implies that it is less sensitive to small changes in the input data, which reduces the risk of overfitting.

For linear functions, flatness means having a small value of *w* in the best fit function *f(x) = wx +b.*

Mathematical Formulation in depth

SVR aims to find a function f(x)=w·ϕ(x)+b that minimizes prediction error while remaining as "flat" as possible. The optimization problem is defined as:

Minimize $\frac{1}{2}\|\mathrm{w}\|^2 + \mathrm{C}\sum_{\mathrm{i}=1}^{\mathrm{n}}(\xi_\mathrm{i} + \xi_\mathrm{i}^*)$

Subjected to $\begin{cases} y_i - (w.\phi(x_i) + b) \leq \in +\xi_i \, , \\ (w.\phi(x_i) + b) - y_i \leq \in +\xi_i \, , \\ \leq \in +\xi_i \; \geq 0 \; \forall_i \end{cases}$

where, w: Weight vector.

ϕ(x): Kernel function mapping input x**x** to a higher-dimensional space.

C: Regularization parameter balancing flatness and training error.

ε: Maximum allowable deviation (defines the ε-tube).

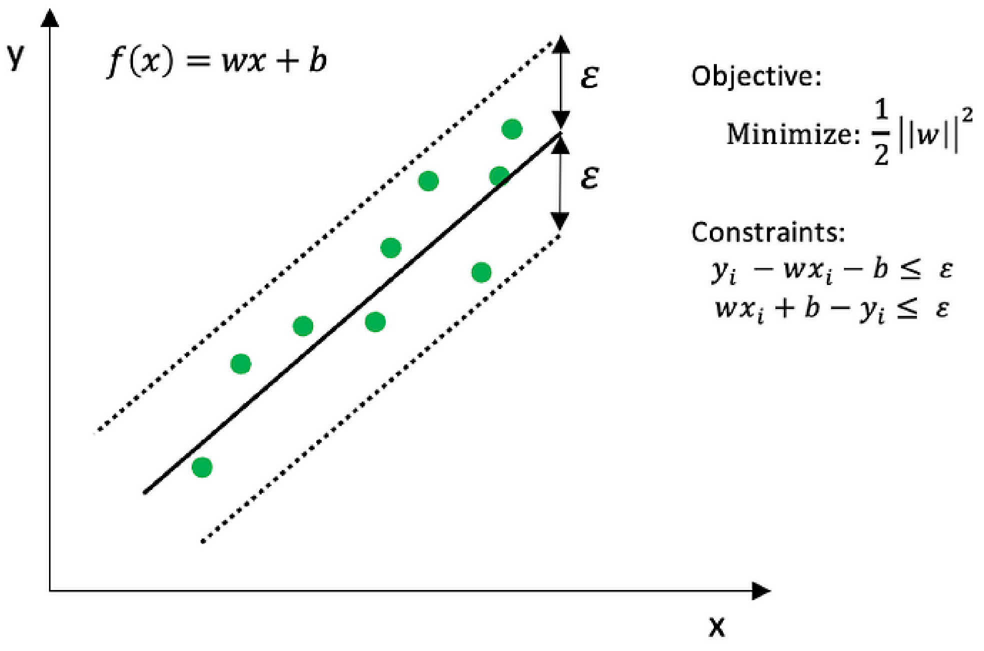
ξ_i, ξ_i^*: Slack variables for points outside the ε-tube.

The solution depends on **support vectors** training points where errors exceed ε. The dual formulation uses Lagrange multipliers α_i, α_i^* , and the final prediction function is:

$f(x)=\sum_{i=0}^{n}(\alpha_i - \alpha_i^*)\, K(x_i, x) + b$

where, $K(x_i, x)= \phi(x_i) \cdot \phi(x)$ *is the kernel function*

Simply SVR can be represented by:

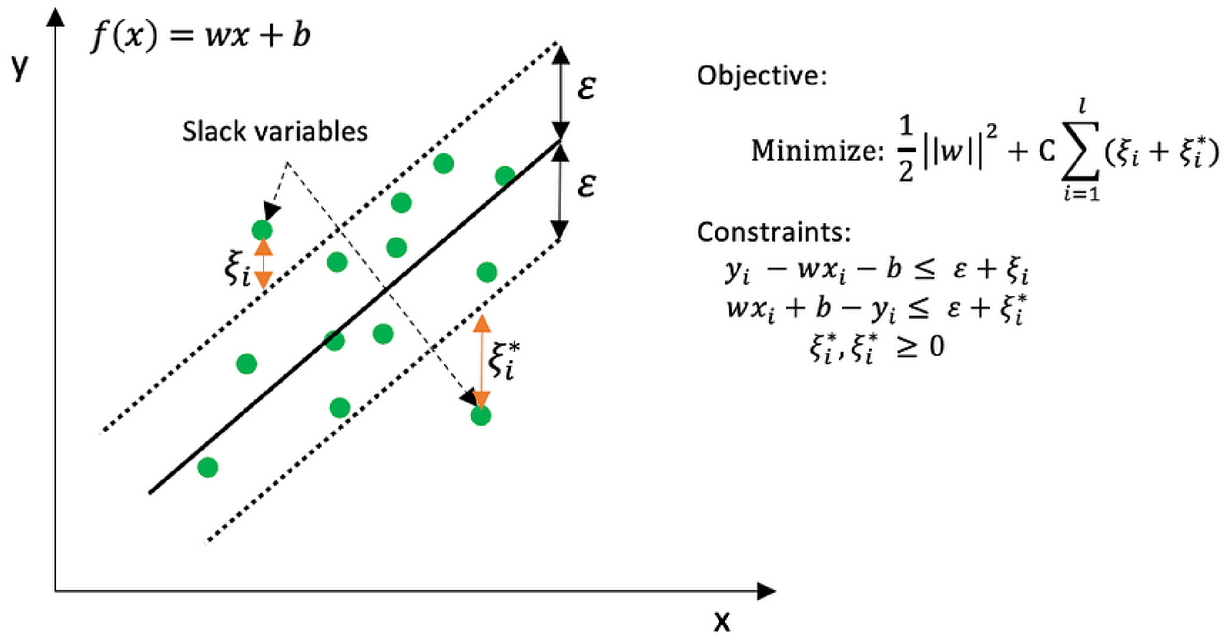

y
$f(x) = wx + b$
ε
ε
Objective:
Minimize: $\frac{1}{2}||w||^2$
Constraints:
$y_i - wx_i - b \leq \varepsilon$
$wx_i + b - y_i \leq \varepsilon$
x

Univariate linear SVR (with zero tolerance for error)

- Sometimes optimization problem is not feasible or we may want to allow for some errors. In that case, we introduce *slack variables,* which are the data points that fall outside of the ε-insensitive tube.
- The distance of slack variables from ε-insensitive tube boundary is represented as ξ.
- To balance the trade-off between the model complexity (i.e., flatness of *f(x)*) and total deviations beyond ε-insensitive tube, we utilize a regularization parameter $C > 0$.
- The strength of the regularization is inversely proportional to *C.*

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

SVR assigns zero prediction error to the points that lie inside the ε-insensitive tube, whereas it penalizes the slack variables proportionally to their ξ. This feature of SVR enables it to handle overfitting more effectively than ordinary regression models.


$f(x) = wx + b$
y
x
Slack variables
ε
ε
ξ_i
ξ_i^*
Objective:
Minimize: $\frac{1}{2}||w||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$
Constraints:
$y_i - wx_i - b \leq \varepsilon + \xi_i$
$wx_i + b - y_i \leq \varepsilon + \xi_i^*$
$\xi_i^*, \xi_i^* \geq 0$

Univariate linear SVR (allowing for errors)

SVR is ideal for scenarios where data exhibits nonlinear patterns, dimensionality is high, or robustness to outliers is critical. However, its computational demands and parameter-tuning requirements make it less suitable for large-scale or real-time applications. SVR can be used for both linear and non-linear regression problems by using various kernel functions. In cases where the data is non-linearly separable, kernel helps in finding function *f(x)* in higher dimensional space where a linear regression problem can be solved. Common kernel functions used in SVR include linear, polynomial, radial basis function (RBF), and sigmoid. SVR provides a powerful alternative to traditional regression methods in complex predictive modeling tasks.

2.3. Classification

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Classification is a supervised machine learning method where the model tries to predict the correct label (class) of a given input data. Those classes can be targets, labels or categories. In classification, the model is fully trained using the training data, and then it is evaluated on test data before being used to perform prediction on new unseen data.

For example, a spam detection machine learning algorithm would aim to classify emails as either "spam" or "not spam."

Types of learners in classification:

There are **two types** of learners in machine learning classification.

1. **Eager learners** are machine learning algorithms that first build a model from the training dataset before making any prediction on future datasets. They spend more time during the training process because of their eagerness to have a better generalization during the training from learning the weights, but they require less time to make predictions.

   Most machine learning algorithms are eager learners, and below are some examples:

   - Logistic Regression.
   - Support Vector Machine.
   - Decision Trees.
   - Artificial Neural Networks.
2. **Lazy learners or instance-based learners**, on the other hand, do not create any model immediately from the training data, and this is where the lazy aspect comes from. They just memorize the training data, and each time there is a need to make a prediction, they search for the nearest neighbor from the whole training data, which makes them very slow during prediction. Some examples of this kind are:
   - K-Nearest Neighbor.
   - Case-based reasoning.

Types of Classification:

Classification-based predictive modeling tasks are distinguished from each other based on the number of categories and the degree to which the categories are exclusive:

- **Binary classification** sorts data into two exclusive categories.
- **Multiclass classification** sorts data into more than two exclusive categories.
- **Multilabel classification** sorts data into nonexclusive categories.
- **Imbalanced** classification has an unequal distribution of data points across categories.

2.3.1 Logistic Regression

- Logistic regression is a supervised machine learning algorithm that accomplishes binary classification tasks by predicting the probability of an outcome, event, or observation.
- The model delivers a binary or dichotomous outcome limited to two possible outcomes: it can be either yes/no, 0/1, or true/false but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Logical regression analyzes the relationship between one or more independent variables and classifies data into discrete classes.
- It is extensively used in predictive modeling, where the model estimates the mathematical probability of whether an instance belongs to a specific category or not.
- Can be used in fraud detection (e.g. spam email detection), disease prediction (e.g. determine the probability of heart attacks), and many more like possibility of enrolling into a university etc.

For example, we have two classes Class A and Class B if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class A otherwise it belongs to Class B. It's referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

Logistic Regression uses a special function called the **sigmoid function / activation function / logistic function** to map predictions and their probabilities. The sigmoid function refers to an S-shaped curve that converts any real value to a range between 0 and 1.

Moreover, if the output of the sigmoid function (estimated probability) is greater than a predefined threshold on the graph, the model predicts that the instance belongs to that class. If the estimated probability is less than the predefined threshold, the model predicts that the instance does not belong to the class.

For example, if the output of the sigmoid function is above 0.5, the output is considered as 1. On the other hand, if the output is less than 0.5, the output is classified as 0. Also, if the graph goes further to the negative end, the predicted value of y will be 0 and vice versa. In other words, if the output of the sigmoid function is 0.65, it implies that there are 65% chances of the event occurring; a coin toss, for example.

The ***sigmoid function is referred to as an activation function for logistic regression*** and is defined as: $\sigma(z) = \frac{1}{1+ e^{-z}}$

Where, e is the base of natural logarithms and z = $b+w_1x_1+w_2x_2+\cdots+w_nx_n$ is the linear combination of input features and coefficients. i.e. output of the linear equation, also called **log odds**.

This equation $\sigma(z) = \frac{1}{1+ e^{-z}}$ becomes y=$\frac{e^{(\beta 0+\beta 1x)}}{1+ e^{(\beta 0+\beta 1x)}}$ in logistic regression. Where, x is input value y is predicted output, b0 is bias or intercept term and b1 is coefficient for input (x)

Log Odds: it refers to the ways of expressing probabilities. Log odds are different from probabilities. Odds refer to the ratio of success to failure, while probability refers to the ratio of success to everything that can occur.

For example, consider that you play twelve tennis games with your friend. you won 5 games and you lost 7 games. Here, the odds of you winning are 5 to 7 (or 5/7), while the probability of you winning is 5 to 12 (as the total games played = 12).

Probability = no of wins / total games = 5/12 = 0.417 i.e. you have a 41.7 % chance of winning.

Odds = no of wins / no of losses = 5/7 = 0.7142 i.e. odds of your winnings are 71.42 %

Now, convert odds to log odds,

We have odds of winning = 0.714

Then convert odds to log odds,

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

log odds (logit) = log (odds) = log (0.714) = -0.336 (negative log odds means that the chance of winning is less than 50 %.

Again, convert odds to probability:

Probability (p) = $\frac{odds}{1+odds} = \frac{0.714}{1+0.714}$ = 0.417 i.e. chances of win are 41.7 %

If we convert log odds to back to probability, probability = $\frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\,0.336}}$ = 0.417

Now, the question is why logistic regression uses log odds (logit) instead of probability directly?

Suppose you try to use linear regression to predict probability: $p=\beta_0+\beta_1 x$

This looks fine, but it has a serious problem i.e. Probability must be between 0 and 1, but a linear equation like $\beta_0+\beta_1 x$ can output any real number (e.g., –3, 2.5, etc.). So, the model might predict values less than 0 or greater than 1, which are invalid probabilities.

Solution is: use log odds.

Instead of modeling probability p directly, logistic regression models the mog of the odds: log $\left(\frac{p}{1-p}\right) = \beta_0+\beta_1 x$

why this works better? Because Log Odds are Unbounded,

- While probability (p) is between 0 and 1,
- Odds can range from 0 to ∞,
- And log(odds) (logit) can range from –∞ to +∞.

So now, the linear function on the right-hand side can safely take any value, and we can convert it back to a valid probability using the sigmoid function: p = $\frac{1}{1+\,e^{-(\,\beta 0+\beta 1x)}}$

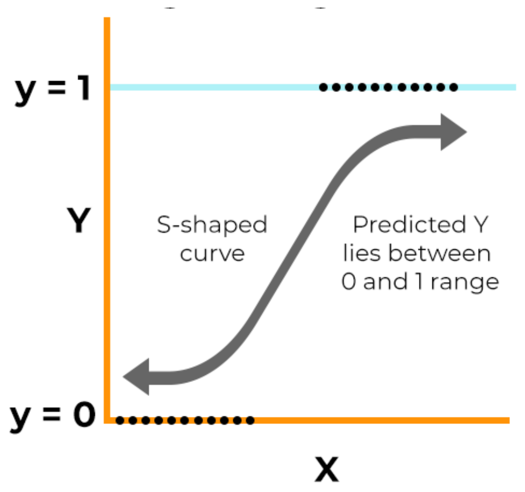


Figure: Logistic Regression curve (S-Shape curve / sigmoid curve)

There are four main classification tasks in Machine learning: binary, multi-class, multi-label, and imbalanced classifications.

1. binary classification
2. multi class classification
3. multi label classification
4. imbalance classification

2.3.1.1. Binary classification

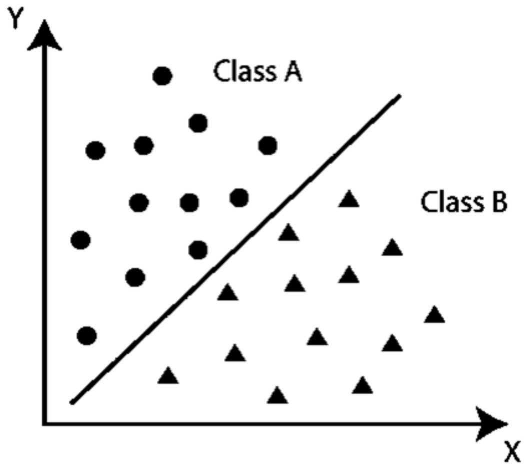


Figure: Binary Classification

Binary classification is a type of machine learning task where the goal is to categorize data into one of two distinct types. This classification problem is fundamental in supervised learning, where a model is trained on labeled data to predict categorical outcomes.

In binary classification, the output is binary, meaning there are only two possible outcomes. For example:

- A coin flip resulting in head or tail: Outcome= Head or Tail
- Categorizing an email as spam or not spam: Outcome= Spam or ham
- Deciding on whether or not to offer a loan to a bank customer: Outcome = yes or no.
- Evaluating the risk of cancer: Outcome = high or low.
- Predicting a team's win in a football match: Outcome = yes or no. etc.

<u>Popular Algorithms for Binary Classification</u>

- K-Nearest Neighbors (KNN)
- Logistic Regression *
- Support Vector Machine *
- Decision Trees
- Naive Bayes

<u>Mathematically,</u>

Given an input feature vector x= (x_1, x_2 ..., x_p), binary classification aims to learn a function f(x) that predicts the class label y∈ {0,1}.

Decision Rule

A general form of the decision function is:

$$f(x)=\begin{cases} 1\ if\ g(x) \geq T \\ 0\ if\ g(x) < T \end{cases}$$

where g(x)$g(\mathbf{x})$ is a scoring function (e.g., probability or score) and 'T' is a threshold, often set to 0.5.

Example: Linear Classifier

A common example is a linear classifier where g(x) = $\frac{1}{1+e^{-z}}$, with z = $\beta_0 + \sum_{j=0}^{p} \beta_j \, x_j$

Here, $\sigma(z)$ is the sigmoid function converting the linear combination z into a probability between 0 and 1 (logistic regression). The predicted class is 1 if $\sigma(z) \geq 0.5$, else 0.

2.3.1.2. Multi-class classification

- Multi-class classification can be treated as an extension of **binary classification** to more than two classes.
- **Multiclass classification** is a machine learning task where the goal is to assign instances to one of multiple predefined classes or categories
- The output of Multi class classification is a single class label assigned to each instance, indicating the most probable or correct class.

An example of a multi-class problem is a handwritten digit recognition, where the task is to identify which digit (0 through 9) by taking an image of a handwritten digit.

Other applications include speech recognition, sentiment analysis, and image classification into multiple categories.

- Multiclass classification involves distinguishing between multiple classes or categories. The fundamental idea is to teach a model to assign the most appropriate class label to each instance based on its features.

For Example:

Classifier to analyze an individual's mood into more than positive or negative. Instead, it will use multiple categories such as, happy, sad, depressed, excited, etc. There is no limit to the number of classes used in multiclass classification.

The most common algorithms which are used for Multi-Class Classification are:

- K-Nearest Neighbors (KNN)
- Naive Bayes
- Decision trees

- Gradient Boosting
- Random Forest
- SVM with One Vs One or One Vs Rest strategies

One Vs Rest – The main task here is to fit one model for each class which will be versus all the other classes

One Vs One – The main task here is to define a binary model for every pair of classes.

Mathematically,

The Softmax function is a key component in multiclass classification, transforming raw model outputs (logits) into probabilities that sum to 1. This enables interpretable predictions across multiple exclusive classes.

Softmax formula converts a vector of values z = [z_1, z_2, ……z_k] into probabilities using:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

In this equation, Exponentiation (e^{z_i})Amplifies differences between values and Normalization ($\sum_{j=1}^{k} e^{z_j}$) divides each exponentiated value by their sum to ensure probabilities total 1.

Let's understand more simply,

Let, an input feature vector x= (x_1, x_2,…… xp), the task is to predict the class label $y \in \{1, 2,...,K\}$, where $K \geq 3$ is the number of classes.

Model Output

A multiclass classifier typically outputs a vector of scores or probabilities for each class:

p= (p_1, p_2 ,..., p_K), where p_k=P (y=k |x, $\sum_{k=1}^{k} p_k = 1$

The predicted class is the one with the highest probability:

y'= argmax p_k

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Example:

A common approach to produce these probabilities is the **Softmax function**, which generalizes the sigmoid function used in binary logistic regression:

$$P_k = \frac{e^{z_k}}{\sum_{i=0}^{n} e^{z_j}} \text{ where } z_k = \beta_{k0} + \sum_{i=0}^{n} \beta_{kj} x_j$$

Here, z_k is the linear score for class k, and β_{kj} are the model parameters for class k.

Multi-label Classification (*not in syllabus but must know*)

People often get confused between multiclass and multi-label classification. But these two terms are very different and cannot be used interchangeably.

- Multi-label refers to a data point that may belong to more than one class.

For example, you wish to watch a movie with your friends but you have a different choice of genres that you all enjoy. Some of your friends like comedy and others are more into action and thrill. Therefore, you search for a movie that fulfills both the requirements and here, your movie is supposed to have multiple labels. Whereas, in multiclass or binary classification, your data point can belong to only a single class.

Some more examples of the multi-label dataset could be protein classification in the human body, or music categorization according to genres. It can also one of the concepts highly used in photo classification.

2.3.2. K-Nearest Neighbors (KNN)

KNN is a supervised machine learning algorithm that can be used to solve both classification and regression problems.

- It is one of the simplest yet powerful algorithms.
- KNN is also known as a lazy or instance-based 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.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- It works by finding the "k" nearest data points (neighbors) to a given input (i.e. training dataset) and uses their class to predict the class or value of new data point based on the majority class (for classification) or the average value (for regression).

Complexity of KNN:

Training time: O (1): essentially zero, since KNN just stores data.

Prediction time: O(N×D) times per query (compute distance to every one of the NNN training points in DDD dimensions), plus O (N log N) if you sort.

Space: O(N×D) to store the entire training set.

KNN commonly used for:

- Disease prediction – Predicting the likelihood of diseases based on symptoms.
- Handwriting recognition – Recognizing handwritten characters.
- Image classification – Categorizing and recognizing images.
- Data preprocessing – to find missing values in data
- Recommendation engine – recommend additional content to user

Step of Execution:

1. Prepare the data
2. Calculate distances

- The K-nearest neighbor algorithm forms a majority vote between the K most similar instances, and it uses a distance metric between the two data points to define them as identical.
- The most popular choice is Euclidean distance and it is limited to real-valued vectors. Using the below formula, it measures a straight line between the query point and the other point being measured.

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

also, Manhattan distance, Minkowski distance and Hamming distance can be used in different context in KNN (see more details about his at https://www.ibm.com/think/topics/knn).

3. Select K-Nearest Neighbors
    - K in KNN is the hyperparameter we can choose to get the best possible fit for the dataset.
    - Suppose we keep the smallest value for K, i.e., K=1. In that case, the model will show low bias but high variance because our model will be overfitted.
    - A more significant value for K, k=10, will surely smoothen our decision boundary, meaning low variance but high bias.
    - So, we always go for a trade-off between the bias and variance, known as a bias-variance trade-off.
    - The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k.
    - Overall, it is recommended to have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset.

4. Majority Voting (for classification)

Advantages

- KNN makes no assumptions about the distribution of classes i.e. it is a non-parametric classifier
- It is one of the methods that can be widely used in multiclass classification
- It does not get impacted by the outliers
- This classifier is easy to use and implement

Disadvantages

- K value is difficult to find as it must work well with test data also, not only with the training data
- It is a lazy algorithm as it does not make any models
- It is computationally extensive because it measures distance with each data point

Example:

Suppose you are predicting the movies genre, and you have the following data: (following data are real data extracted from IMBD)

IMDB Rating	Duration	Genre
8.8 (Fight Club)	148	Action
9.0 (The Dark Knight)	152	Action
7.5 (Deadpool & Wolverine)	128	Comedy
6.9 (Hi, Mom)	128	Comedy
7.6 (Deadpool 2)	119	Comedy

Let's predict the genre of "Ted" movie with IMDB rating 6.9- and 106-min duration.

<u>Now, Calculate the distance:</u>

Euclidean distance between the movie needs to be predicted and each move in the dataset.

Where ted has value (6.9, 106) = let (x1, y1)

Then Distance to (8.8, 148) = let (x2, y2) is calculated as

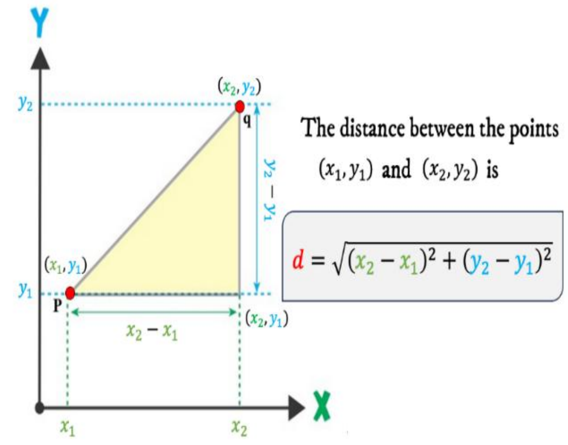
$\sqrt{(8.8-6.9)^2+(148-106)^2}$= 42.042

Similarly, Distance from (9.0, 152) = 46.047

Distance from (7.5,128) =22.0081

Distance from (6.9, 128) =22.00

Distance from (7.6,119) =13.018


Y
X
y_2
y_1
x_1
x_2
(x_2, y_2)
q
(x_1, y_1)
P
(x_2, y_1)
$x_2 - x_1$
$y_2 - y_1$
The distance between the points (x_1, y_1) and (x_2, y_2) is
$d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

<u>Again, Select K-Nearest Neighbor</u>

1. If K=1, check only one neighbor which is nearest i.e. 13.018, here, 13.018 is the score of Deadpool 2. Therefore, ted is comedy movie. (for which comedy is true label) (this is the best case, but always in real k=1

is not applicable in all case because there might be some noise in data so have to check in different k.

2. If k =3, check 3 nearest neighbor, then we have 13.081, 22.00, 22.008. here all score is of comedy movie and all votes are in comedy genre. therefore, ted is comedy movie (in case of multiple K we use voting).

2.3.3. Support Vector Machine (SVM)

Support Vector Machines (SVM) is a supervised machine learning algorithm commonly used for classification tasks.

- Also, SVM can be used for the regression task.
- SVM works by finding the optimal hyperplane or set of hyperplanes in a high-dimensional space that separates data points into the different classes.
- A good separation is achieved by the hyperplane that has the largest margin (i.e. meaning the maximum distance between data points of different classes)

OR

SVM tries to find the "best" margin (distance between the line and the support vectors) that separates the classes.

Application of SVM:

- Text and hypertext categorization
- Bioinformatics: genetic structure of the patients based on their biological problems.
- Handwriting recognition and text categorization
- Face detection and expression classification
- Data classification
- Steganography detection
- Cancer detection etc.

2.3.3.1. Hyperplane and Support Vectors

Hyperplane (*in red*): A hyperplane is a decision boundary that separates data points into different classes in a high-dimensional space.

Mathematically, Hyperplane can be express as w. x +b=0 in linear classification.

Where, where w is the weight vector, x is the input feature vector, and b is the bias term

- In 2D space, a hyperplane is simply a line that separates the data points into two classes.
- In 3D space, a hyperplane is a plane that separates the data points into two classes. Similarly, in N-dimensional space, a hyperplane has (N-1)-dimensions and it becomes more complex flat subspace.

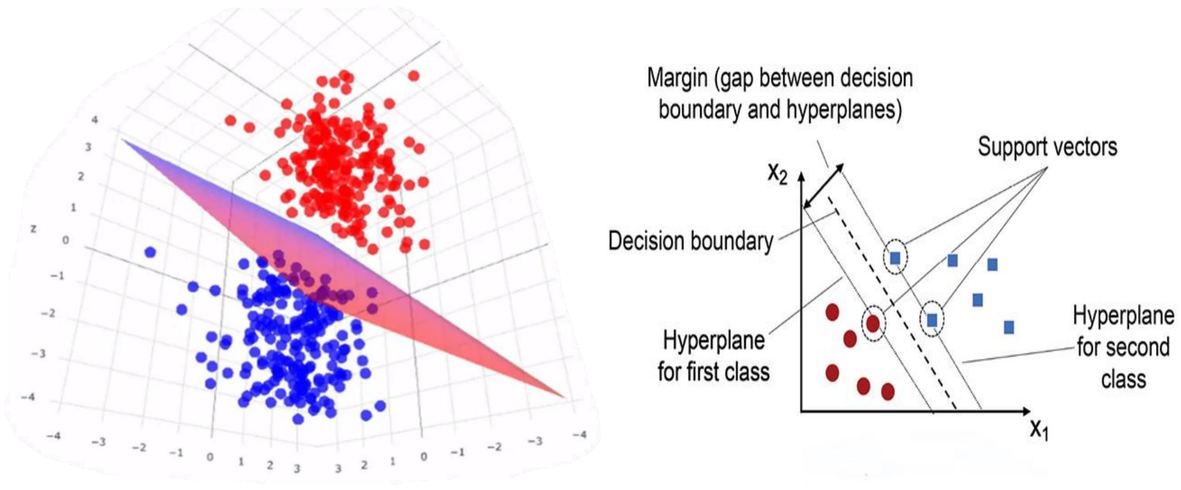

Margin (gap between decision boundary and hyperplanes)
Support vectors
X_2
Decision boundary
Hyperplane for first class
Hyperplane for second class
X_1

Figure: Hyperplane into 3-Dimensional Space and 2-Dimensional Space [IS: Medium]

- It can be used to make predictions on new data points by evaluating which side of the hyperplane they fall on.
- Data points on one side of the hyperplane are classified as belonging to one class, while data points on the other side of the hyperplane are classified as belonging to another class.

Support Vectors: Support Vectors are the data points that lie closest to the decision boundary (hyperplane).

- These data points are important because they determine the position and orientation of the hyperplane, and thus have a significant impact on the classification accuracy of the SVM.
- In fact, SVMs are named after these support vectors because they “*support*” or define the decision boundary.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- The support vectors are used to calculate the margin, which is the distance between the hyperplane and the closest data points from each class. The goal of SVMs is to maximize this margin while minimizing classification errors.

Margin: Margin is the distance between the decision boundary (hyperplane) and the closest data points from each class.

- The goal of SVMs is to maximize this margin while minimizing classification errors.
- A larger margin indicates a greater degree of confidence in the classification (i.e better generalization), as it means that there is a larger gap between the decision boundary and the closest data points from each class.
- The margin is a measure of how well-separated the classes are in feature space. SVMs are designed to find the hyperplane that maximizes this margin, which is why they are sometimes referred to as maximum-margin classifiers.

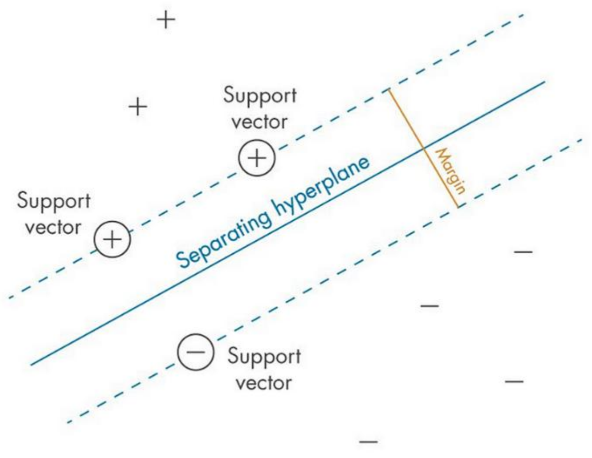
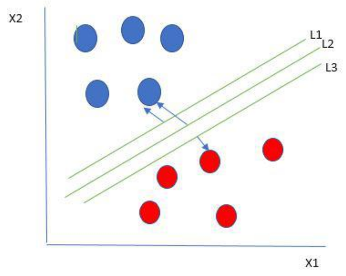


Figure: Support vector and Margin in SVM

Weight vector *w*: The direction of this vector indicates how the hyperplane is oriented, and its magnitude determines how steep the slope of the separation boundary is.

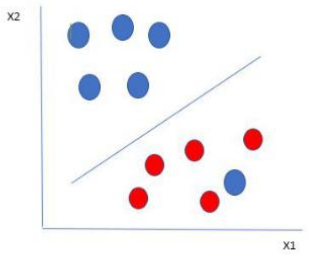
Working Principle of SVM:



The key idea behind the SVM algorithm is to find the hyperplane that best separates two classes by maximizing the margin between them.

The best hyperplane also known as the **"hard margin"** is the one that maximizes the distance between the hyperplane and the nearest data points from both classes which ensures a clear separation between the classes. So, from the figure a side, we choose L2 as hard margin.

BUT What if we have the data like in following figure?



Here, we have one blue ball in the boundary of the red ball. The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.

A soft margin allows for some misclassifications or violations of the margin to improve generalization. The SVM optimizes the following equation to balance margin maximization and penalty minimization:

$$\text{Objective Function}= \left(\frac{1}{margin}\right)+\lambda \sum \text{penalty}$$

The penalty used for violations is often hinge loss which has the following behavior:

- If a data point is correctly classified and within the margin there is no penalty (loss = 0).
- If a point is incorrectly classified or violates the margin the hinge loss increases proportionally to the distance of the violation.

Now, Let's discuss what if data are linearly not separable? Solution is - Use of Kernel Trick (see 2.3.3.2)

Mathematical Computation of SVM:

Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y. The equation for the linear hyperplane can be written as:

$w^{T} \cdot x + b = 0$

Where:

w is the normal vector to the hyperplane (the direction perpendicular to it).

b is the offset or bias term representing the distance of the hyperplane from the origin along the normal vector w.

Now, to calculate the distance from a Data Point to the Hyperplane,

The distance between a data point x_i and the decision boundary can be calculated as: $d_i = \frac{w^T x_i \ + b}{\|w\|}$

where ||w|| represents the Euclidean norm of the weight vector w. Euclidean norm of the normal vector W

<u>for Linear SVM Classifier:</u>

Distance from a Data Point to the Hyperplane: $\hat{y} = \begin{cases} 1: \ \mathrm{wT} \cdot \mathrm{x} + \mathrm{b} \geq 0 \\ 0: \ \mathrm{wT} \cdot \mathrm{x} + \mathrm{b} < 0 \end{cases}$

Where $\hat{y}$ is the predicted label of a data point.

For a linearly separable dataset the goal is to find the hyperplane that maximizes the margin between the two classes while ensuring that all data points are correctly classified. This leads to **optimization problem** for SVM.

$\underset{w,b}{minimize} \frac{1}{2}(\|w\|)^2$ subjected to the constraint $y_i(w^T xi+b) \geq 1$ fori=1,2,3,⋯,m

Where:

y_i is the class label (+1 or -1) for each training instance.

x_i is the feature vector for the i-th training instance.

m is the total number of training instances.

The condition $y_i(w^T xi+b) \geq 1$ ensures that each data point is correctly classified and lies outside the margin.

Soft Margin in Linear SVM Classifier

In the presence of outliers or non-separable data the SVM allows some misclassification by introducing slack variables ζ_i. The optimization problem is modified as:

$$\underset{w,b}{minimize} \frac{1}{2}(\|w\|)^2 + \mathrm{C} \sum_{i=0}^{m} \zeta_i$$

Subjected to $y_i(w^T xi+b) \geq 1- \zeta_i$ and $\zeta_i \geq 0$ for i= 1,2,…..,m

Where,

C is a regularization parameter that controls the trade-off between margin maximization and penalty for misclassifications.

$\zeta i \zeta i$ are slack variables that represent the degree of violation of the margin by each data point.

Dual Problem for SVM

The dual problem involves maximizing the Lagrange multipliers associated with the support vectors. This transformation allows solving the SVM optimization using kernel functions for non-linear classification.

The dual objective function is given by:

$$\underset{\propto}{maximize} \frac{1}{2} \sum_{i=1}^{m} . \sum_{j=1}^{m} \propto_i \propto_j \, t_i t_j \, K(\, x_{i,} x_j \,) - \sum_{i=1}^{m} \propto_i$$

Where:

α_i are the Lagrange multipliers associated with the ith training sample.

ti is the class label for the ith th training sample.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

$K(x_i,x_j)$ is the kernel function that computes the similarity between data points x_i and x_j. The kernel allows SVM to handle non-linear classification problems by mapping data into a higher-dimensional space.

The dual formulation optimizes the Lagrange multipliers $\alpha i \alpha i$ and the support vectors are those training samples where $\alpha_i>0$.

<u>SVM Decision Boundary</u>

Once the dual problem is solved, the decision boundary is given by:

$$W= \sum_{i=1}^{m} \propto_i t_i K(x_{i,}, x) + b$$

Where w is the weight vector, x is the test data point and b is the bias term. Finally the bias term b is determined by the support vectors, which satisfy:

$$t_i(w^T x_i+b) = 1 \rightarrow b = w^T x_i - t_i$$

Where x_i is any support vector.

This completes the mathematical framework of the Support Vector Machine algorithm which allows for both linear and non-linear classification using the dual problem and kernel trick.

[for more details and depth of mathematics involved in SVM: see https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/]

<u>2.3.3.3. SVM for Linear and Non-linear Classification</u>

Based on the shape of the decision boundary or training set, SVM can be classified as

1. Linear SVM / Simple SVM
2. Non-linear SVM / Kernel SVM

Linear SVM

A linear SVM refers to the SVM type used for classifying linearly separable data. This implies that when a dataset can be segregated into categories or classes with the help of a single straight line, it is termed a linear SVM, and the data is referred to as linearly distinct or separable. Moreover, the classifier that classifies such data is termed a linear SVM classifier.

- A simple SVM is typically used to address classification and regression analysis problems.
- It is computationally efficient and easier to interpret.
- The decision function for classification is: $f(x)= w^T \cdot x + b$, where *w* is the normal vector to the hyperplane (the direction perpendicular to it) and *b* is the offset or bias term representing the distance of the hyperplane from the origin along the normal vector w*w*.

If $f(x)>0$, the data point belongs to class +1; if $f(x)<0$, it belongs to class -1

Non-Linear SVM

Non-linear data that cannot be segregated into distinct categories with the help of a straight line is classified using a kernel or non-linear SVM. Here, the classifier is referred to as a non-linear classifier. The classification can be performed with a non-linear data type by adding features into higher dimensions rather than relying on 2D space. Here, the newly added features fit a hyperplane that helps easily separate classes or categories.

- Kernel SVM is typically used to handle optimization problems that have multiple variables.

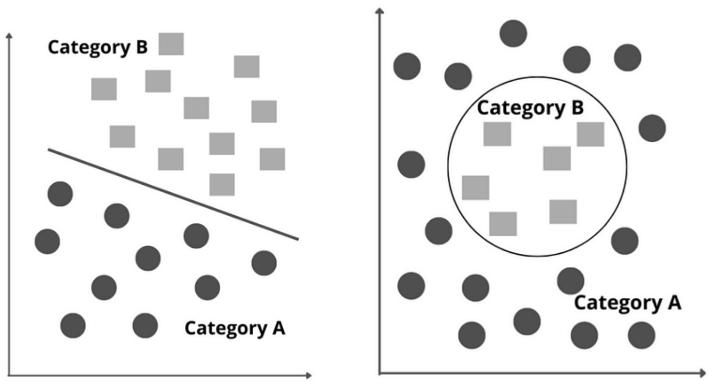


Figure: Linear and Non-linear SVM

2.3.3.2. Kernels and its Types: Linear, Polynomial, Radial Basis Function (RBF)

Kernels are functions that implicitly map input data into higher-dimensional spaces without explicitly computing the coordinates in that space. This is known as the "kernel trick," allowing SVM to handle complex, non-linear data efficiently.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Commonly used kernels are:

1. Linear Kernel: This is the simplest kernel and does not transform the data; it's equivalent to a linear SVM. Suitable for linearly separable data.

Kernel function for Linear Kernel is: K $(x, y) = (x^T)\, y$

2. Polynomial Kernel: This kernel allows for polynomial decision boundaries. It is useful when data follow polynomial relationships. Allows the algorithm to fit data in a higher-dimensional space.

Kernel function for Polynomial Kernel is: $K(x, y) = (x^T y + 1)^d$ Where, 1 or c is a constant and d is the degree of the polynomial.

3. Radial Basis Function (RBF) / Gaussian Kernel: This kernel maps data into an infinite-dimensional space and can handle very complex boundaries. It's the most widely used kernel and works well when there is no prior knowledge of the data's structure (suits highly non-linear data).

Kernel function for RBF or GK is: $K(x, y) = \exp(-\gamma \|x - y\|^2)$

4. Sigmoid Kernel: This function (based on the hyperbolic tangent) resembles the activation function of a neural network neuron. It can be used when one thinks in terms of neural-network-style non-linear separation. Kernel function is Similar to a neural network's activation function.

Kernel function for Sigmoid Kernel is: $K(x, y) = \tanh(\alpha\, x^T y + c)$

Different parameters like γ, d, c used in kernel are tuned during training.

Where,

C: Regularization parameter controlling the trade-off between maximizing the margin and minimizing the classification error.

- High C: Low bias and high variance (overfitting).
- Low C: High bias and low variance (underfitting).

Gamma (γ): Defines how far the influence of a single training example reaches.

- A low value of gamma results in a smoother decision boundary
- High value causes the boundary to be more complex and fit the data tightly.

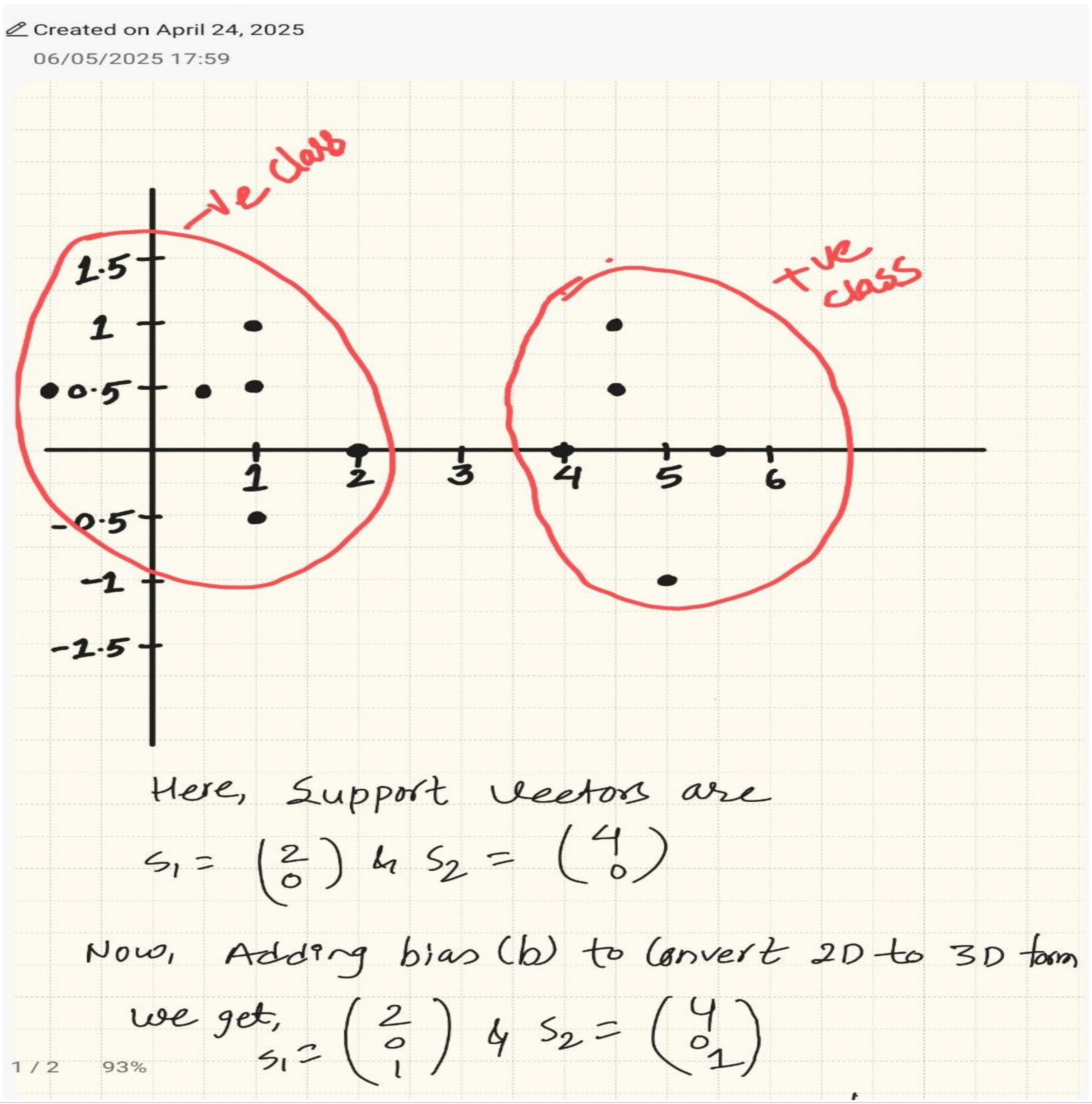
Real Life example using SVM Classifier:

Consider the following set of data points give, find the optimal hyperplane for these data points for Linear SVM.

(1,0.5), (1,1), (1, -0.5), (-0.5,0.5), (0.5,0.5), (2,0), (4,0), (4.5, 1), (4.5, 0.5), (5, -1), (5.5, 0)

Solution:

Step-1: Draw the points on plane



Created on April 24, 2025
06/05/2025 17:59

Here, Support vectors are

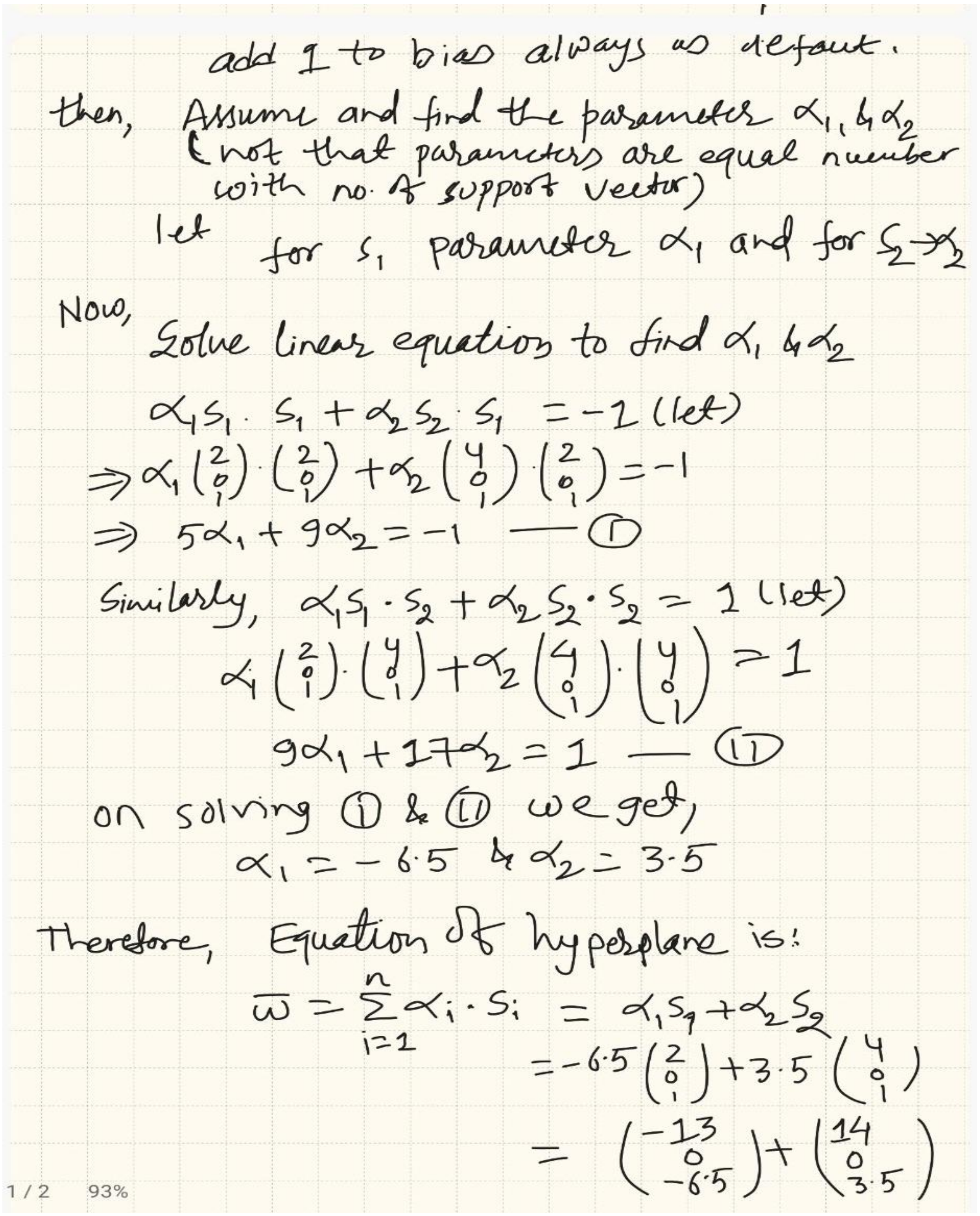
$s_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$ & $s_2 = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$

Now, Adding bias (b) to convert 2D to 3D form we get,

$s_1 = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}$ & $s_2 = \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}$

1 / 2 93%

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com



add 1 to bias always as default.

then, Assume and find the parameter α_1 & α_2 (not that parameters are equal number with no. of support vector)

let for s_1 parameter α_1 and for $s_2 \rightarrow \alpha_2$

Now, Solve linear equation to find α_1 & α_2

$\alpha_1 s_1 \cdot s_1 + \alpha_2 s_2 \cdot s_1 = -1$ (let)

$\Rightarrow \alpha_1 \begin{pmatrix}2\\0\\1\end{pmatrix} \cdot \begin{pmatrix}2\\0\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\0\\1\end{pmatrix} \cdot \begin{pmatrix}2\\0\\1\end{pmatrix} = -1$

$\Rightarrow 5\alpha_1 + 9\alpha_2 = -1$ — (I)

Similarly, $\alpha_1 s_1 \cdot s_2 + \alpha_2 s_2 \cdot s_2 = 1$ (let)

$\alpha_1 \begin{pmatrix}2\\0\\1\end{pmatrix} \cdot \begin{pmatrix}4\\0\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\0\\1\end{pmatrix} \cdot \begin{pmatrix}4\\0\\1\end{pmatrix} = 1$

$9\alpha_1 + 17\alpha_2 = 1$ — (II)

on solving (I) & (II) we get,

$\alpha_1 = -6.5$ & $\alpha_2 = 3.5$

Therefore, Equation of hyperplane is:

$$\bar{w} = \sum_{i=1}^{n} \alpha_i \cdot s_i = \alpha_1 s_1 + \alpha_2 s_2$$

$$= -6.5 \begin{pmatrix}2\\0\\1\end{pmatrix} + 3.5 \begin{pmatrix}4\\0\\1\end{pmatrix}$$

$$= \begin{pmatrix}-13\\0\\-6.5\end{pmatrix} + \begin{pmatrix}14\\0\\3.5\end{pmatrix}$$

1 / 2 93%

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

$\therefore \bar{\omega} = \begin{pmatrix} 1 \\ 0 \\ -3 \end{pmatrix}$

i.e $b = -3$ (we had Assumed $b = 1$)

$\Rightarrow b + 3 = 0$ is eq^n of bias.

Note that,
in $\bar{\omega}$ top two elements gives the orientation of hypesplane

if value is $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow$ Vertical orientation

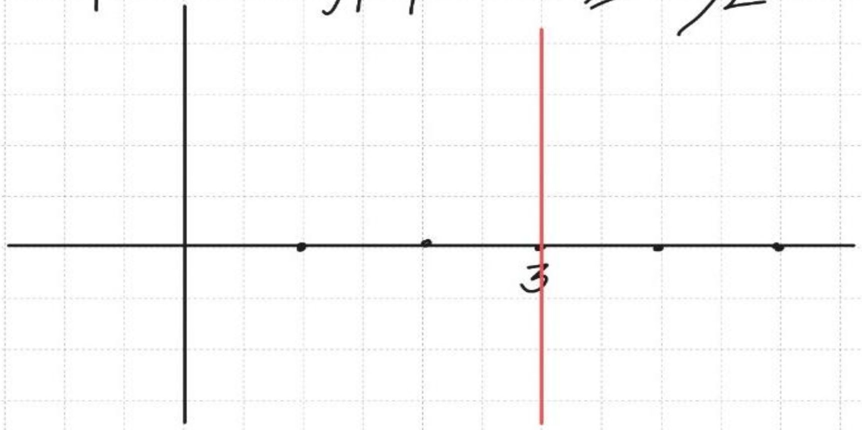
if value is $\begin{pmatrix} 0 \\ 1 \end{pmatrix} \rightarrow$ Horizontal "

if value is $\begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightarrow 45^\circ$ orientation

finally we have $b = -3$

i.e $b + 3 = 0$

This indicates that in +ve class at point 3 hyperplane exist.



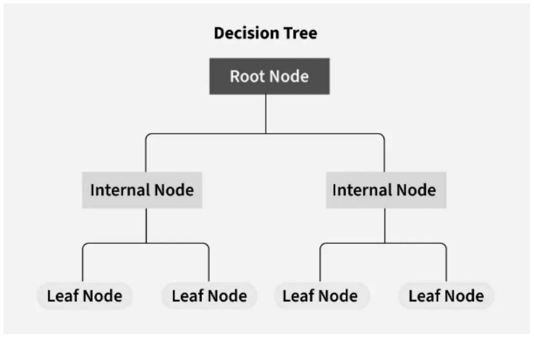
By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Q-2. Suppose you are given the +vely labeled data points (3.1), (3, -1), (6,1) and (6, -1). Similarly -vely labeled data points are (1,0), (0,1), (0, -1) and (-1,0). Find and plot the hyperplane with given data points. ***[Class Work]***

Q-3. Let the +vely labeled data points are (2,2), (2, -2), (-2, -2) and (-2,2). Similarly -vely labeled data points are (1,1), (1, -1), (-1, -1) and (-1,1). If the transformation function is given by $\emptyset \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{cases} 4 - x_2\,|x_1 - x_2| \\ 4 - x_1\,|x_1 - x_2| \end{cases}$ $if\ \sqrt{x_1 + x_2} > 2$ and by $\emptyset \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ otherwise. ***[Class Work]***

2.3.4. Decision Trees

A **decision tree** is a supervised learning algorithm that can be used for both classification and regression tasks. It works by recursively partitioning the data into subsets based on feature values, making decisions at each node to maximize a specific criterion (e.g., information gain or Gini index).



Components of DT

- **Root Node:** The top or starting node in the tree that represents the entire dataset or best feature to split the data.
- **Branches:** Line that connects the nodes representing the possible values of the features by showing the flow from one decision to another (i.e. attribute value).
- **Internal Nodes:** points where the decisions are made based on the input features (i.e. attribute tests)
- **Leaf / terminal Nodes:** Terminal nodes at the end of branches that represent the predicted / final outcome (class label or numerical value).

Types /Algorithms of / in DT:

1. **ID3:** This algorithm uses entropy and information gain as metrics to evaluate candidate splits.
2. **C4.5:** This algorithm is considered a later iteration of ID3, it can use information gain or gain ratios to evaluate split points within the decision trees.

3. **CART:** Abbreviation for "classification and regression trees" This algorithm typically utilizes Gini impurity to identify the ideal attribute to split on.

Application of DT

1. **Medical diagnosis:** for e.g. diabetes prediction based on lab test results. Input features like BMI, glucose levels, blood pressure etc.
2. **Education:** for e.g. university result prediction. Input features, attendance, study hour, previous class grade etc. Identifying factors affecting student dropout rates by considering financial, issues, number of backlogs, etc.
3. **Finance:** for e.g. Loan approval prediction. Input features like, income, credit score, employment status, loan history etc.

Assumptions of DT:

1. Binary Splits: i.e. each node divides the data into two subsets based on a single feature or condition
2. Recursive Partitioning: i.e. each node is divided into child nodes, and this process continues until a stopping criterion is met.
3. Feature used for splitting nodes are independent.
4. Each node is created with homogeneous sub nodes. i.e. samples within a node are as similar as possible regarding the target variable
5. Tree is constructed using top-down, greedy approach where each split is chosen to maximize information gain or minimize impurity at the current node.

Advantages and Dis-advantages of DT:

Advantages	**Dis-advantages**
Easy to understand	Risk of overfitting
Handles both numerical and categorical data	Unstable with small changes
No need for data scaling	Biased with imbalance data
Automated feature selection	Large tree can become hard to explain
Handles nonlinear relationship	Limited to axis-parallel splits

2.3.4.1. Construction and pruning of decision trees

To construct DT:

4. **Select the best attribute:** at each node, pick the pick the feature whose split yields the greatest "information gain" (i.e. most effectively reduces uncertainty). Using a metric like Gini impurity, entropy, or information gain, the best attribute to split the data is selected.
5. **Splitting the Dataset**: The dataset is split into subsets based on the selected attribute.
6. **Repeating the Process**: repeat recursively for each subset / child nodes, creating a new internal node or leaf node until one of two stopping criterion is met
    - All instances/ samples at the node belong to the same class (Pure node)
    - A pre-defined stopping rule is met (e.g., minimum samples per node, maximum depth).

To decide which feature to split on, use two main measures:

1. **Entropy (H):**
    - Entropy is a measure of uncertainty or disorder.
    - To check mathematically if any split is pure split or not, we use entropy or Gini impurity.
    - It's difficult to explain information gain without first discussing entropy. Entropy is a concept that stems from information theory, which measures the impurity of the sample values.
    - Entropy values range from 0 and 1.
    - If all samples in data set, S, belong to one class, then entropy will equal to 0 (i.e. low entropy indicates more ordered or homogeneous set)
    - If half of the samples are classified as one class and the other half are in another class, entropy will be at its highest / maximum at 1 (i.e. high entropy signifies greater disorder or diversity)

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

In order to select the best feature to split on and find the optimal decision tree, the attribute with the smallest amount of entropy should be used.

Entropy is calculated by using the equation:

Entropy for a dataset S, = H(S) = - $\sum_{i=1}^{c} p_i \, log_2 \, (p_i)$

Where, S represents the data set that entropy is calculated

c represents the classes in set, S

p_i represents the proportion of data points that belong to class c to the number of total data points in set, S

Thus, for different numbers of classes:

For **2 classes** (binary classification): maximum entropy is 1.

For **3 classes**: maximum entropy is log2(3) ≈1.585

For **4 classes**: maximum entropy is log2(4) =2 and so on

<u>Gini Impurity:</u>

- It is the probability of misclassifying a randomly chosen element in a set.
- The range of the Gini index is **[0, 0.5]**, where 0 indicates perfect purity and 0.5 indicates maximum impurity.
- Gini index is typically used in CART (Classification and Regression Trees) algorithms

Gini impurity is calculated by: for dataset S = G(S) = 1- $\sum_{i=1}^{c} {p_i}^2$

- It is computationally efficient than entropy because Gini index is a linear measure whereas entropy is a logarithmic measure.

2. **Information Gain (IG):**
    - IG helps to determine which features need to be selected.
    - The reduction in entropy is often referred to as Information Gain and is calculated as the difference between the entropy before and after the split. on a given attribute.

- The attribute with the highest information gain will produce the best split as it's doing the best job at classifying the training data according to its target classification.

Information Gain is calculated by the equation:

$$IG(S, A) = H(S) - \sum \frac{|S_v|}{|S|} H(S_v)$$

Where, A represents a specific attribute or class label

Entropy(S) is the entropy of dataset, S

S_v is the Subset of S where feature A has value v.

$|S_v|$ is the number of samples in S_v.

$|S|$ is the total number of samples in S.

$H(S_v)$ is the entropy of the subset S_v.

$\frac{|S_v|}{|S|}$ represents the proportion of the values in S_v to the number of values in dataset, S.

Note: Leaf node is the terminal node which represents the final output / predictions of DT. Once a data point reaches a leaf node, a decision or prediction is made based on the majority class (for classification) or the average value (for regression) of the data points that reach that leaf.

Pruning of DT:

Decision tree pruning involves the strategic removal of branches from a decision tree.

- This process aims to enhance the model's performance by eliminating parts that do not contribute significantly to its predictive power.
- By simplifying the tree, pruning helps in reducing the complexity of the model, making it more efficient and easier to interpret.
- A fully grown decision tree often fits the training data very well but may not generalize to unseen data. This phenomenon is called overfitting.

To reduce overfitting, we prune the tree by using following methods:

1. **Pre-Pruning:** by cutting off branches early which stops growth early (depth limits)
    - Offers simplicity and efficiency by stopping tree growth early.
    - However, it might miss capturing complex patterns in the data.
2. **Post-Pruning:** by trimming a fully grown tree.
    - Provides a thorough evaluation of the tree's structure, allowing for more informed pruning decisions.
    - It can be computationally intensive due to the need to build the full tree first.

Post pruning by cost-complexity ($\propto -\boldsymbol{Pruning}$)

- In cost-complexity pruning, we define a cost function for any subtree T:

$R_\alpha(T)=R(T)+\alpha\,|\text{leaves}(T)|$

where: R(T) = total number of misclassified training samples in subtree T.

$|\text{leaves}(T)|$ = number of leaves in T.

$\alpha\geq 0$ = complexity penalty (a user-chosen hyperparameter).

When $\alpha=0$, there is no penalty on complexity, so the fully grown tree is favored purely by misclassification count. As α increases, having more leaves (i.e., a more complex tree) becomes "costly," so simpler pruned subtrees may yield a lower overall cost $R_\alpha(T)$.

Example of DT:

Let's predict the exam pass or fail using 10 students' data:

S.N.	Study Hours (SH)	Attendance (A)	Pass
1.	High	High	Yes
2.	High	High	Yes
3.	High	Low	No

4.	Low	High	No
5.	Low	Low	No
6.	High	High	Yes
7.	High	Low	No
8.	Low	High	No
9.	Low	Low	No
10.	High	High	Yes

Step-1: Root Node Selection

1.1. Calculate entropy of entire dataset

Here, Total Sample =10, yes= 4 (let's say +ve) and No =6 (let's say -ve)

H(S)= S {+4, -6} =$-\frac{4}{10}\log_2\frac{4}{10}-\frac{6}{10}\log_2\frac{6}{10}\approx0.971$

Step-2 Calculation information Gain for features

2.1.Split on Study Hours (SH)

High SH: 6 samples (Yes=4, No=2)

Low SH: 4 samples (Yes=0, No=4)

Entropy for High SH: H(high)=- $\frac{4}{6}\log_2\frac{4}{6}-\frac{2}{6}\log_2\frac{2}{6}$ = 0.918

Entropy for Low SH: H(Low)=0(all No)

Information Gain for SH = IG (SH)= H(S) - $\frac{6}{10}\ H(high)-\frac{4}{10}$ H (Low) = 0.4199

2.2.Split on Attendance (A)

High A: 6 samples (Yes=4, No=2)

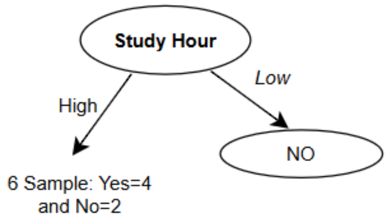
Low A: 4 samples (Yes=0, No=4)

Entropy for High A: H(High)= - $\frac{4}{6}\log_2\frac{4}{6}-\frac{2}{6}\log_2\frac{2}{6}$ = 0.9183

Entropy for Low A: H(Low)=0

Information Gain for A = IG (A)= H(S) - $\frac{6}{10}\ H(high)-\frac{4}{10}$ H (Low) = 0.420

Here, both features Study Hours and Attendance have the same IG (0.420). We can pick either as the root node. Let's choose Study Hours for this example.



Here, If SH = Low: all 4 samples are No → leaf node "No" and If SH = High: 6 samples (Yes=4, No=2) → need further splitting

Step-3: Split High SH branch by Attendance (A)

Samples in High SH branch:

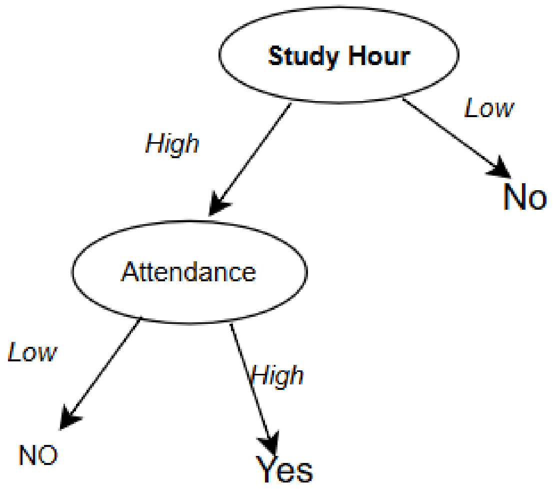
Attendance	Yes	No	Total
High	4	1	5
Low	0	1	1

Entropy for Attendance = High in High SH: H= $-\frac{4}{5}\log_2\frac{4}{5}-\frac{1}{5}\log_2\frac{1}{5}=0.722$

Entropy for Attendance = Low in High SH: H= 0 (all No)

Information Gain (IG) = H(S) - $\frac{5}{6}*0.722-\frac{1}{6}*\ 0\ =0.316$

Now, Final tree before Pruning is:



By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Why prune?

The branch Attendance = High under High SH has 5 samples but is not pure (4 Yes, 1 No). and Pruning can simplify the tree by merging these leaves if it doesn't reduce performance significantly.

Cost Complexity Pruning Calculation

Let α be the complexity parameter. For the unpruned subtree (High SH branch), Number of leaves, T=2

Total impurity (weighted entropy): $R(T)=\frac{5}{6}\times0.722+\frac{1}{6}\times0=0.602$

Total cost: $C(T)=R(T)+\alpha\times T=0.602+2\alpha$

Pruned subtree (merge into one leaf):

Entropy of merged node (High SH branch): H=0.918 and Number of leaves, T=1

Total cost: $C(T)=0.918+\alpha$

Pruning condition:

Prune if: $0.918+\alpha\leq0.602+2\alpha$

$0.918-0.602\leq2\alpha-\alpha$

$0.316\leq\alpha$

Therefore, if $\alpha\geq0.316$, prune the subtree (remove Attendance split under High SH). Otherwise, keep the split.

Example 2: Construct the DT from the following data to predict the football playing condition.

Day	Weather	Temperature	Humidity	Wind	Play Football?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Cloudy	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes

5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Cloudy	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Cloudy	Mild	High	Strong	Yes
13	Cloudy	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Solution: on Board

2.3.4.2. Ensemble methods: Bagging, Random Forests

Ensemble learning is a machine learning technique that combines multiple models (like decision trees, regressors, or neural networks) to produce more accurate and reliable predictions than any single model alone.

It works on the principle that a group of models (learners) performs better than an individual model which is similar to how a group opinion is often more accurate than one person's guess.

Ensemble models help to:

- Reduce variance (less overfitting). e.g. Bagging: Trains models on different random samples of data (with replacement) and averages predictions.
- Reduce bias (fixing underfitting). e.g. Boosting: Trains models sequentially, each one focusing on correcting errors made by the previous model.
- Improve overall prediction. e.g. Stacking: Combines predictions from different types of models using a new model (meta-learner) to make the final decision.

They help balance the bias-variance tradeoff, leading to better generalization.

Example:

Imagine you're trying to guess the weight of a watermelon. If you ask 10 people and take the average of their guesses, it's usually closer to the actual weight than relying on just one person's guess.
This is how ensemble learning improves accuracy. This is what ensemble method do in ML (i.e. combine prediction of several models to get better accuracy, less, variance and less overfitting)

Types of ensemble models:

1. **Parallel**: All models are trained independently. For example: Bagging, Random Forest.
2. **Sequential:** Models are trained one after another, improving step-by-step. Example: Boosting

Common Ensemble Techniques:

1. Bagging
2. Stacking
3. Boosting

Among these methods, **Bagging** (Bootstrap Aggregating) and **Random Forests** stand out for their ability to reduce overfitting while maintaining high performance.

Bagging:

Bagging involves training multiple instances of a base model (e.g., decision trees) on randomly sampled subsets of the training data. These subsets, called **bootstrap samples**, are created by sampling with replacement, allowing data points to appear multiple times in a single sample. Predictions from all models are aggregated through:

- **Majority voting** for classification tasks.
- **Averaging** for regression tasks

Therefore, Bagging = Bootstrap + Aggregation

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- **Bootstrap**: Randomly select subsets with replacement from the training data.
- **Aggregating**: Train a model on each subset and combine their predictions.

➢ In Bagging there exists sampling of data as well as features and build a model

➢ It is usually applied to decision tree methods.

<u>Steps in Bagging:</u>

1. Bootstrapping: Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.
2. Parallel Training: Each model is learned in parallel with each training set and independent of each other
3. Aggregation: The final predictions are determined by combining the predictions from all the models. In the case of regression, an average is taken of all the outputs predicted by the individual classifiers which is known as **soft voting**. For classification problems, the class with the highest majority of votes is accepted; this is known as **hard voting or majority voting.**

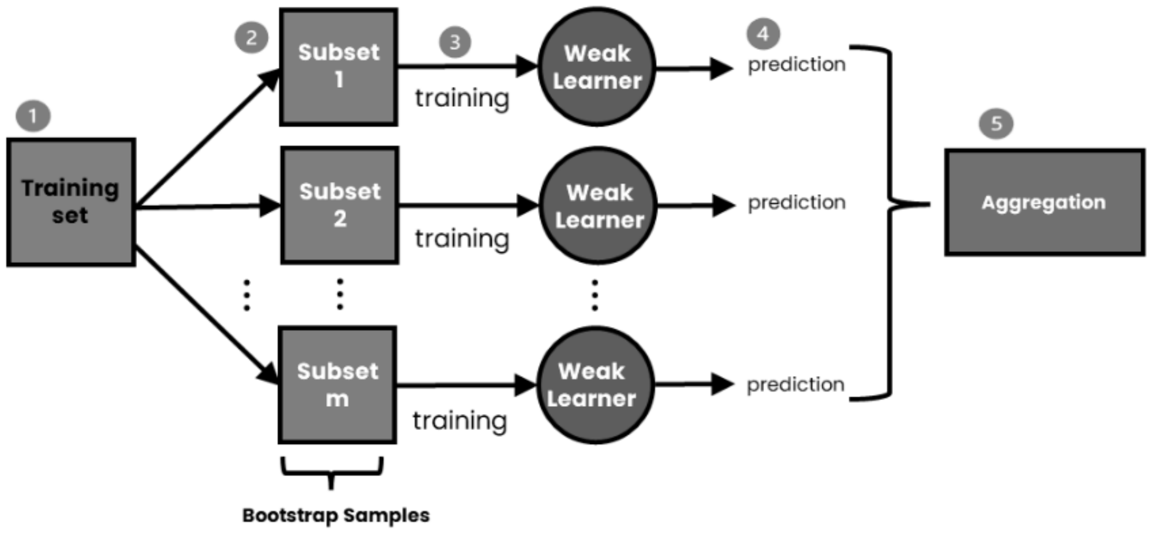


Figure: Process of Bagging (Bootstrap Aggregation)

Mathematics Behind Bagging in ML:

Let, D = {(x_1, y_1) ..., (x_n, y_n)} is the original dataset, hi(x): the prediction of the ith model and M: number of models then, the bagged prediction is:

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

For Classification: $\hat{y}$ = majority_vote ($h_1(x), h_2(x) ..., h_M(x)$)

For Regression: $\hat{y} = \frac{1}{M}\sum_{i=1}^{M} h_i(x)$

Random Forest:

Random Forest is an improved and advanced version of Bagging using Decision Trees, each trained on a random subset of data and features and RF is known for the high accuracy, low overfitting and robust performance.

- It's called a "forest" because it grows many trees, and "random" because it introduces randomness in the data and features used by each tree.
- Random forests are collection of decision trees combined to give one output, where each tree is slightly different from the others.
- Used in both classification and regression problem but works best in classification.

Example:

You wanted to choose a course after your +2, but you couldn't decide which course fit your skill set. So, you decided to consult various people like your cousins, teachers, parents, degree students, and working professionals. You asked them varied questions such as why you should choose a particular course, job opportunities related to it, course fees, etc. Finally, after consulting various people about the course, you decided to take the Computer Engineering suggested by most of them. *(though it turned out to be your worst decision, haha no offense, just for fun*).

Steps in Random Forest Algorithm:

1. Select a subset of data points and a subset of features to construct each decision tree. Simply take n random records and m features from a dataset containing k records.
2. Construct individual decision trees for each sample
3. Each decision tree will generate an output.
4. Consider the final output based on Majority Voting for classification and averaging for regression, respectively.

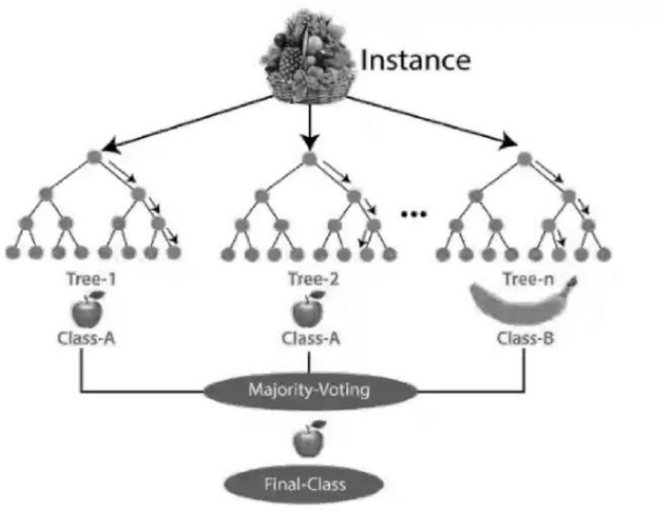


Figure: Example process of Random Forest

Hyperparameter for Random forests:

- **n_estimators:** Number of trees the algorithm builds before averaging the predictions.
- **max_features:** Maximum number of features random forest considers splitting a node.
- **mini_sample_leaf:** Determines the minimum number of leaves required to split an internal node.
- **criterion:** How to split the node in each tree? (Entropy/Gini impurity/Log Loss)
- **max_leaf_nodes**: Maximum leaf nodes in each tree.

End of Chapter