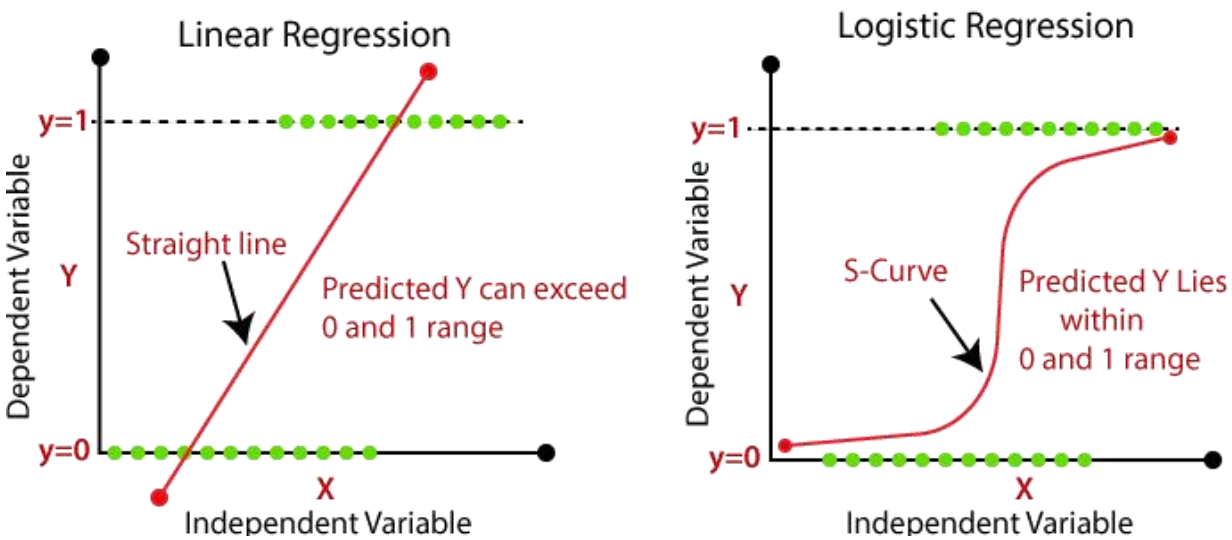
- List the difference between linear and logistic regression?



	Linear Regression	Logistic Regression
Response Variable	Continuous (e.g. price, age, height, distance)	Categorical (yes/no, male/female, win/not win)
Equation Used	$Y = \beta_0 + \beta_1X_1 + \beta_2X_2 + ...$	$p(Y) = e^{(\beta0 + \beta1X1 + \beta2X2 + ...)} / (1+ e^{(\beta0 + \beta1X1 + \beta2X2 + ...)})$
Method Used to Fit Equation	Ordinary Least Squares	Maximum Likelihood Estimation
Output to Predict	Continuous value ($150, 40 years, 10 feet, etc.)	Probability (0.741, 0.122, 0.345, etc.)

- Are there any assumptions of linear regression? If yes, what are they?

_Yes, linear regression makes several assumptions. Violation of these assumptions can impact the reliability and validity of the regression model. Here are the key assumptions of linear regression:

1. **Linearity:** The relationship between the independent variables (predictors) and the dependent variable (response) is linear. In other words, the change in the response variable is proportional to changes in the predictor variables.

Mathematically, it can be represented as:
\[Y = \beta_0 + \beta_1X_1 + \beta_2X_2 + \ldots + \beta_pX_p + \epsilon\]
where \(Y\) is the dependent variable, \(X_1, X_2, \ldots, X_p\) are the independent variables, \(\beta_0, \beta_1, \ldots, \beta_p\) are the model coefficients, and \(\epsilon\) represents the error term.

2. **Independence of Errors:** The errors (residuals) are assumed to be independent of each other. This means that the error for one observation should not depend on the errors of other observations. Violation of this assumption can lead to issues like autocorrelation.

3. **Homoscedasticity (Constant Variance):** The variance of the errors should be constant across all levels of the predictors. In other words, the spread of residuals should be roughly the same for all values of the independent variables. Heteroscedasticity, where the variance of errors varies, can lead to biased standard errors and affect hypothesis testing.

4. **Normality of Errors:** The errors are assumed to be normally distributed. This means that the distribution of the residuals should resemble a normal distribution. Deviations from normality can affect the accuracy of confidence intervals and hypothesis tests, especially with small sample sizes.

- Give mathematical insights to the logistic regression function?

$p(Y) = e^{(\beta 0 + \beta 1X1 + \beta 2X2 + \ldots)} / (1 + e^{(\beta 0 + \beta 1X1 + \beta 2X2 + \ldots)})$

- What are the outputs of the logistic model and the logistic function? What is the range of values of a logistic function?

In logistic regression, the output of the model is a probability score that represents the likelihood of a binary outcome (e.g., 0 or 1, Yes or No) based on the input features. The logistic function, also known as the sigmoid function, is used to transform the linear combination of input features into this probability score. Here's how it works:

1. **Logistic Model Output (Probability Score):**
   - The logistic model calculates a linear combination of the input features and their associated coefficients, plus an intercept term. This linear combination is often referred to as the "log-odds" or "logit."
   - Mathematically, the logistic model output (\(z\)) can be expressed as:
     \[z = \beta_0 + \beta_1x_1 + \beta_2x_2 + \ldots + \beta_nx_n\]
   - The probability of the positive class (class 1) is then calculated as the sigmoid (logistic) function of this log-odds value:
     \[P(Y=1) = \frac{1}{1 + e^{-z}}\]
   - Here, \(P(Y=1)\) represents the probability of the event occurring (e.g., the probability of a customer making a purchase), and \(e\) is the base of the natural logarithm (approximately equal to 2.71828).

2. **Logistic Function (Sigmoid Function):**
   - The logistic function, denoted as \(S(z)\) or \(σ(z)\), is used to transform the log-odds (\(z\)) into a probability score. It has an S-shaped curve.
   - The mathematical formula for the logistic (sigmoid) function is:
     \[S(z) = \frac{1}{1 + e^{-z}}\]

- This function takes any real-valued number (\(z\)) and maps it to a value between 0 and 1, which represents the probability of the positive class.
- As \(z\) approaches negative infinity, \(S(z)\) approaches 0, indicating a very low probability of the event occurring.
- As \(z\) approaches positive infinity, \(S(z)\) approaches 1, indicating a very high probability of the event occurring.

3. **Range of Values of the Logistic Function:**
- The logistic function always produces values between 0 and 1, making it suitable for modeling probabilities.
- In other words, the range of the logistic function is \([0, 1]\).
- This range is essential for binary classification because it allows us to interpret the output as the probability of an observation belonging to the positive class. For example, if \(P(Y=1) > 0.5\), the model predicts class 1; otherwise, it predicts class 0.

In summary, the logistic model in logistic regression produces a probability score for the positive class, and this probability score is computed using the logistic (sigmoid) function, which maps log-odds to values between 0 and 1. The sigmoid function's range ensures that the output is a valid probability.

- When Logistic Regression can be used?

Logistic regression can be used in a variety of machine learning and statistical modeling tasks. It is primarily employed in binary classification problems, where the goal is to predict one of two possible outcomes (e.g., yes/no, 1/0, true/false). Here are some common scenarios where logistic regression is a suitable choice:

1. **Medical Diagnosis:** Logistic regression can be used to predict whether a patient has a particular medical condition based on medical test results, patient demographics, or other relevant factors.

2. **Credit Risk Assessment:** Banks and financial institutions use logistic regression to assess the creditworthiness of loan applicants, determining whether an applicant is likely to default on a loan.

3. **Spam Email Detection:** Logistic regression can be applied to classify emails as spam or not spam based on the content, sender, and other features of the email.

4. **Customer Churn Prediction:** Businesses use logistic regression to predict whether a customer is likely to churn (cancel their subscription or stop using a service) based on historical customer behavior and demographics.

5. **Employee Attrition Prediction:** Companies use logistic regression to forecast whether an employee is likely to leave the organization based on factors like job satisfaction, salary, and tenure.

6. **Market Basket Analysis:** In retail, logistic regression can help predict whether a customer is likely to purchase a specific product based on their past shopping behavior and other attributes.

7. **Social Sciences:** Logistic regression is widely used in social sciences to model binary outcomes, such as predicting whether a student will pass or fail an exam based on study habits and socioeconomic factors.

8. **Natural Language Processing (NLP):** In sentiment analysis, logistic regression can be used to classify text sentiment as positive or negative based on the words and phrases used in the text.

9. **Click-Through Rate (CTR) Prediction:** In online advertising, logistic regression can predict whether a user will click on an ad based on user behavior, ad content, and other features.

10. **Fraud Detection:** Logistic regression can be used to identify fraudulent transactions based on transaction history and patterns of behavior.

It's important to note that logistic regression can also be extended to handle multiclass classification problems, where there are more than two possible classes. In such cases, techniques like "one-vs-all" or "softmax regression" can be employed to model and predict multiple classes.

Logistic regression is a versatile and widely used algorithm in the field of machine learning and statistics due to its simplicity, interpretability, and effectiveness in modeling binary and multiclass classification tasks.

- Why gradient descent is called an optimization algorithm

Gradient descent is called an optimization algorithm because its primary purpose is to find the minimum (or maximum) of a function. In machine learning and various other fields, the term "optimization" refers to the process of finding the best solution or the set of parameters that minimizes (or maximizes) a specific objective function.

Here's why gradient descent is considered an optimization algorithm:

1. **Minimizing a Cost Function:** In the context of machine learning, gradient descent is often used to minimize a cost or loss function. The goal is to find the set of model parameters (weights and biases) that minimize the error or cost associated with predictions on a training dataset. By adjusting these parameters iteratively, gradient descent moves toward the optimal parameter values that minimize the cost.

2. **Gradient-Based Updates:** Gradient descent operates by computing the gradient (a vector of partial derivatives) of the cost function with respect to the model parameters. The gradient points in the direction of the steepest ascent, and the negative gradient points in the direction of the steepest descent. By taking steps in the opposite direction of the gradient, the algorithm aims to descend to the minimum of the cost function.

3. **Iterative Optimization Process:** Gradient descent is an iterative algorithm. It starts with an initial guess for the parameter values and updates them in small steps (proportional to the gradient) until it converges to a minimum (or maximum) of the cost function. This process of iteratively refining the parameter values is a hallmark of optimization algorithms.

4. **Local and Global Optimization:** Gradient descent can be applied to both local and global optimization problems. In machine learning, it's often used to find the local minimum of the cost function that corresponds to the best possible model given the training data. However, with proper initialization and learning rates, it can also be effective in finding global optima in some cases.

5. **General Applicability:** Gradient descent is a versatile optimization technique used not only in machine learning but also in fields like mathematics, physics, engineering, and economics. It can be applied to optimize a wide range of functions and solve various types of optimization problems.

In summary, gradient descent is called an optimization algorithm because it's designed to systematically and iteratively minimize (or maximize) a target function by updating parameter values based on the gradient information. It is a fundamental tool in optimization and plays a crucial role in training machine learning models by finding the optimal parameters that minimize prediction errors.

- What happens If the learning rate is too high or too low

The learning rate is a crucial hyperparameter in gradient-based optimization algorithms like gradient descent. Its value can significantly impact the convergence and performance of the optimization process. Here's what happens when the learning rate is too high or too low:

1. **Learning Rate Too High:**
   - **Overshooting the Minimum:** If the learning rate is set too high, the optimization algorithm may take excessively large steps during each iteration. As a result, it can overshoot the minimum (or maximum) of the cost function. This can cause the optimization process to diverge, and the algorithm may fail to converge to a solution.
   - **Divergence:** In extreme cases, a very high learning rate can cause the algorithm to diverge rapidly, with the parameter values oscillating or increasing indefinitely. This results in unstable and unusable models.

2. **Learning Rate Too Low:**
   - **Slow Convergence:** If the learning rate is set too low, the optimization process progresses very slowly. It takes many iterations to make small adjustments to the parameter values. While this can lead to convergence, it may take an impractical amount of time to reach the minimum (or maximum) of the cost function.
   - **Getting Stuck in Local Minima:** In some cases, setting the learning rate too low can cause the optimization algorithm to get stuck in local minima (or maxima) and prevent it from finding a better global solution.

To choose an appropriate learning rate:

- **Experimentation:** It often requires experimentation and hyperparameter tuning. You can try a range of learning rates and monitor the behavior of the optimization algorithm, observing how quickly it converges and whether it converges to a satisfactory solution.

- **Learning Rate Schedules:** Learning rate schedules, such as learning rate decay, can help mitigate the problems associated with fixed learning rates. These schedules decrease the learning rate over time, starting with a larger rate to allow for faster convergence and then gradually reducing it to make smaller refinements.

- **Adaptive Learning Rates:** Some optimization algorithms, like Adam and RMSprop, use adaptive learning rates that adjust the learning rate based on the past gradients. These methods can automatically adapt to the characteristics of the cost function and the optimization process.

In practice, the choice of learning rate can be problem-dependent. There's no one-size-fits-all learning rate, and finding the right value often requires a balance between faster convergence and avoiding divergence or getting stuck in local optima. Experimentation and careful tuning are essential steps in selecting an appropriate learning rate for a given optimization task.

- For a practical application, which one would you choose : Gradient Descent method or Ordinary Least Squares method?

The choice between using the Gradient Descent method or the Ordinary Least Squares (OLS) method for a practical application depends on various factors, including the nature of the problem, the dataset size, computational resources, and your specific goals. Here are some considerations to help you decide:

Use Gradient Descent When:

1. **Large Datasets:** Gradient Descent is more scalable and computationally efficient when dealing with large datasets. It can handle datasets that don't fit in memory, making it suitable for big data scenarios.

2. **Non-Linear Models:** Gradient Descent can be used to optimize a wide range of machine learning models, including those with non-linearities (e.g., neural networks) or models with custom-defined cost functions.

3. **Online Learning:** If your application requires continuous learning from streaming data or incremental updates, online variants of Gradient Descent (e.g., Stochastic Gradient Descent) are suitable choices.

4. **Regularization:** Gradient Descent allows you to incorporate various forms of regularization (e.g., L1, L2 regularization) to prevent overfitting.

5. **Custom Cost Functions:** If your problem requires a custom-defined cost function, Gradient Descent provides the flexibility to optimize it.

Use Ordinary Least Squares (OLS) When:

1. **Linear Regression:** OLS is specifically designed for linear regression tasks, where you want to fit a linear model to your data. If your problem can be adequately modeled using a linear relationship between features and the target variable, OLS is a straightforward and interpretable choice.

2. **Small to Medium-sized Datasets:** OLS can efficiently handle small to medium-sized datasets that can fit in memory. It's relatively fast for these scenarios and doesn't require tuning hyperparameters like learning rates.

3. **Interpretability:** OLS provides interpretable coefficients for each feature, making it easier to understand the relationships between predictors and the target variable.

4. **Closed-Form Solution:** OLS has a closed-form solution, which means you can calculate the optimal coefficients analytically without the need for an iterative optimization process. This can be advantageous when computational efficiency is not a primary concern.

In summary, the choice between Gradient Descent and OLS depends on the specific requirements of your application. If you have a linear regression problem with a reasonably sized dataset and interpretability is important, OLS is a natural choice. However, if you're dealing with large datasets, non-linear models, custom cost functions, or complex optimization problems, Gradient Descent offers more flexibility and scalability. Ultimately, it's important to consider the trade-offs and characteristics of each method based on your unique problem and constraints.

- What are the evaluation metrics for the Regression Model and which is the most effective one?

When evaluating regression models, there are several common evaluation metrics to assess how well the model's predictions align with the actual target values. The choice of the most effective metric depends on the specific characteristics of your regression problem, such as whether it's important to penalize certain types of errors more than others. Here are some widely used regression evaluation metrics:

1. **Mean Absolute Error (MAE):**
   - MAE is the average of the absolute differences between the predicted values and the actual values.
   - It measures the average magnitude of errors, where larger errors are penalized proportionally.
   - MAE is less sensitive to outliers compared to other metrics like RMSE.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - y_j|$$

2. **Mean Squared Error (MSE):**
   - MSE is the average of the squared differences between the predicted values and the actual values.
   - It penalizes larger errors more heavily than smaller errors.
   - MSE is sensitive to outliers due to the squaring of errors.

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)}^{2}$$

The square of the difference between actual and predicted

3. **Root Mean Squared Error (RMSE):**
   - RMSE is the square root of the MSE.

- It provides a measure of the average magnitude of errors in the same units as the target variable.
- RMSE is also sensitive to outliers.

4. **Mean Absolute Percentage Error (MAPE):**
- MAPE calculates the average percentage difference between predicted and actual values.
- It's particularly useful when you want to understand errors in terms of percentage.
- MAPE is sensitive to cases where actual values are close to zero, as it can result in very large percentages.

$$MAPE = \frac{1}{n} \times \sum \left| \frac{actual\ value\ - forecast\ value}{actual\ value} \right|$$

5. **Coefficient of Determination (R^2 or R-squared):**
- R^2 measures the proportion of the variance in the target variable that is predictable from the independent variables.
- It ranges from 0 to 1, where a higher R^2 indicates a better fit.
- However, R^2 may not provide insights into the magnitude or direction of errors.

$$R^2 = 1 - \frac{Residual\ sum\ of\ squares}{Total\ sum\ of\ squares}$$

$$\therefore R^2 = 1 - \frac{RSS}{TSS}$$

Where \(SSR\) is the sum of squared residuals (model's error), and \(SST\) is the total sum of squares (variance of the target variable).

6. **Adjusted R-squared:**
- Adjusted R^2 accounts for the number of predictors in the model and adjusts R^2 for model complexity.
- It penalizes the addition of irrelevant variables that do not improve the model's fit.
- Higher adjusted R^2 values indicate better models with more relevant predictors.

$$\text{Adjusted } R^2 = 1 - \frac{(n-1)}{[n-(k+1)]}(1-R^2)$$

Formula 9-6

where n = sample size
k = number of independent (x) variables

Where \(n\) is the number of observations and \(k\) is the number of predictors.

The choice of the most effective metric depends on your specific objectives and the characteristics of your data. There is no one-size-fits-all answer. MAE and RMSE are often good starting points, but you should consider the context of your problem and how you want to weigh different types of errors (e.g., large errors vs. small errors) when selecting the most appropriate metric for your regression evaluation.

- What is a residual in linear regression and how is it used in model evaluation?

In linear regression, a residual (also known as an error or residual error) is the difference between the observed or actual value of the dependent variable (the target) and the predicted value of the dependent variable based on the regression model. Mathematically, for each data point, the residual is calculated as:

\[\text{Residual}_i = \text{Actual Value}_i - \text{Predicted Value}_i \]

Here's how residuals are used in model evaluation in linear regression:

1. **Assessing Model Fit:**
   - Residuals are a measure of the model's performance in capturing the relationships between the independent variables and the dependent variable. A good model should have small residuals, indicating that it's making accurate predictions.
   - Large residuals suggest that the model is not accurately representing the data, indicating areas where the model may be underfitting or overfitting.

2. **Model Validation:**
   - Residuals are crucial for model validation. By comparing the distribution of residuals to certain assumptions (e.g., normality of errors), you can check whether the model's assumptions are met.
   - If the residuals exhibit patterns or trends (e.g., heteroscedasticity or non-linearity), it may indicate that the model is not appropriate for the data.

3. **Checking for Independence:**
   - Residuals should ideally be independent of each other, meaning that the error for one data point should not depend on the errors of other data points. Examining residual plots and conducting tests for autocorrelation can help detect violations of independence assumptions.

4. **Identifying Outliers:**
   - Large or extreme residuals can indicate outliers in the data. Outliers can significantly affect the model's fit and should be investigated.

5. **Measuring Model Performance:**
   - Metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are often computed using residuals to quantify the model's prediction accuracy.
   - Residuals can also be used to calculate the coefficient of determination (\(R^2\)) to understand how well the model explains the variance in the dependent variable.

6. **Detecting Heteroscedasticity:**
   - Heteroscedasticity is a situation where the variance of residuals varies across different levels of the independent variables. Residual plots can reveal patterns of heteroscedasticity, which may require addressing in the modeling process.

7. **Feature Engineering and Model Improvement:**
   - By analyzing residuals, you can gain insights into which features may be missing from the model or which transformations may improve the model's fit. For example, if residuals show a clear pattern, you might consider adding polynomial features.

In summary, residuals in linear regression are a critical component of model evaluation. They provide valuable information about the model's performance, help assess the validity of model assumptions, and guide improvements to the model. Analyzing residuals is an essential step in understanding how well a linear regression model fits the data and in diagnosing and addressing potential issues.

- What is the difference between linear regression and non-linear regression? Explain with an an example

Linear regression and **non-linear regression** are both techniques used for modeling relationships between variables, but they differ in the functional form of the model they use to fit data. Here's a comparison with an example for each:

Linear Regression:
- **Model Form**: Linear regression models assume a linear relationship between the independent variable(s) and the dependent variable. In simple linear regression, this relationship is described by a straight line equation. In multiple linear regression, it's a hyperplane in higher dimensions.
- **Example**: Suppose you want to predict a person's weight based on their height. A linear regression model might assume that weight (dependent variable) is directly proportional to height (independent variable), with a constant slope. The equation would be something like:
  ```
  Weight = (Slope) * Height + Intercept
  ```
  In this linear model, a unit increase in height would result in a constant change in weight.

Non-Linear Regression:non-linear regression is used for estimating continuous numerical values and allows for more complex, non-linear relationships between variables
- **Model Form**: Non-linear regression models allow for more complex, non-linear relationships between variables. The functional form of the model can take various shapes, including curves, exponentials, logarithms, and more.
- **Example**: Suppose you want to model the growth of a plant over time. The relationship between time (independent variable) and plant height (dependent variable) might not be linear. Instead, it could be described by a non-linear model, such as a logistic growth curve:
  ```
  Height = (Asymptote) / (1 + e^(-k*(Time - t0)))
  ```

```

In this non-linear model, the plant's height approaches an asymptotic limit over time, and the rate of growth is controlled by the parameters `Asymptote`, `k`, and `t0`.

**Key Differences**:

1. **Linearity**:
   - Linear regression assumes a linear relationship between variables, meaning the output changes linearly with changes in the input.
   - Non-linear regression allows for more complex, non-linear relationships, where the output can change in a non-linear fashion as the input changes.

2. **Model Complexity**:
   - Linear regression models are simpler and have fewer parameters since they involve straight lines or hyperplanes.
   - Non-linear regression models can be more complex, with a greater number of parameters, as they can take various non-linear forms.

3. **Fitting**:
   - Linear regression can often be fitted using simple least squares methods, where you minimize the sum of squared errors between the predicted and actual values.
   - Non-linear regression typically requires more advanced optimization techniques to estimate parameters that minimize the error between the model and the data.

In summary, the choice between linear and non-linear regression depends on the nature of the relationship between the variables in your data. Linear regression is appropriate when there is a linear relationship, while non-linear regression is used when the relationship is more complex and non-linear in nature.
logistic regression is linear in nature and is used for classification tasks

---

- How do we measure the strength of a linear relationship between two variables? Is it important for linear regression?

The strength of a linear relationship between two variables is typically measured using a statistical measure called the **correlation coefficient**. One of the most commonly used correlation coefficients is the Pearson correlation coefficient (Pearson's r), but there are others like the Spearman rank correlation and the Kendall tau rank correlation for non-parametric data. Pearson's correlation coefficient is used specifically to quantify the strength and direction of a linear relationship between two continuous variables.

Here's how Pearson's correlation coefficient works and its importance in the context of linear regression:

**Pearson Correlation Coefficient (Pearson's r):**
- Pearson's r measures the degree to which two continuous variables are linearly related. It ranges from -1 to 1.
```

- A positive r value (closer to 1) indicates a positive linear relationship, meaning that as one variable increases, the other tends to increase.
- A negative r value (closer to -1) indicates a negative linear relationship, meaning that as one variable increases, the other tends to decrease.
- An r value close to 0 indicates a weak or no linear relationship between the variables.

The formula for Pearson's correlation coefficient is as follows:

\[r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \]

Where:
- \(X_i\) and \(Y_i\) are the individual data points for the two variables.
- \(\bar{X}\) and \(\bar{Y}\) are the means (averages) of the two variables.
- \(n\) is the number of data points.

Importance in Linear Regression:
- The correlation coefficient is important in the context of linear regression for several reasons:
  1. **Model Selection:** It can help you decide whether a linear regression model is appropriate for your data. If the correlation between the dependent and independent variables is weak (close to 0), linear regression may not be the best choice.
  2. **Feature Selection:** In multiple linear regression, you can use the correlation coefficient to assess the strength of the relationship between each independent variable and the dependent variable. Variables with low correlations may be considered for removal.
  3. **Interpretation:** In linear regression, the magnitude and sign of the regression coefficients are directly related to the correlation between the variables. A positive correlation indicates a positive coefficient, and a negative correlation indicates a negative coefficient.
  4. **Multicollinearity:** High correlations among independent variables (multicollinearity) can cause instability in regression coefficient estimates. Detecting and addressing multicollinearity is important for model stability and interpretability.

In summary, measuring the strength of a linear relationship between variables using the correlation coefficient is crucial for assessing the appropriateness of linear regression, selecting relevant features, interpreting regression coefficients, and managing issues like multicollinearity. It provides valuable insights into the relationships within your data and guides the modeling process.

A model which is suffering from underfitting **(both training and test error are very high) will have high bias and low variance, resulting in a very high loss (hence the high error). An algorithm having access to a larger hypothesis space (as a result of different training data sets, features, and hyperparameters) would result in models having higher** variance**. In other words, these models will be high** variance **models.**

Model bias can be lowered using good machine learning algorithms, unbiased training data, and feature engineering.

Model variance can be lowered using good machine learning algorithms, a very large set of training data (more than required), and more relevant features or less noisy features if possible that are correlated to the target variable

---

- Is there an inductive bias of decision trees? If yes, what is it?

Inductive Bias of Decision Trees: Recursive partitioning is the primary inductive bias of decision trees. They create hierarchical rules through binary splits, prefer locally optimal choices, and favor features with high information gain while promoting simplicity and interpretability.

- How does a decision tree(DT) handle continuous attributes?

A decision tree (DT) can handle continuous attributes by making decisions based on threshold values. When dealing with continuous attributes, decision trees follow these general steps:

Selection of Splitting Attribute: The DT algorithm considers all possible values of the continuous attribute and selects a threshold value that optimizes a certain criterion, such as maximizing information gain (for ID3, C4.5) or minimizing mean squared error (for regression trees).

Thresholding: The continuous attribute's values are compared to the selected threshold. Data points with attribute values less than or equal to the threshold go down one branch of the tree, and data points with values greater than the threshold go down the other branch.

Recursion: The tree-building process continues recursively for each branch. Subsets of data are created based on the thresholding, and the algorithm selects the best attribute and threshold for each subset.

Leaf Nodes: The recursion stops when certain stopping criteria are met, such as reaching a maximum depth, all data points in a branch belong to the same class, or the number of data points in a branch is below a specified threshold. In such cases, a leaf node is created, and it represents a predicted class label (for classification) or a predicted numerical value (for regression).

A data segment in a DT is said to be **pure** if it contains data instances belonging to just one **response variable** (class or category). The goal while building a decision tree is to reach a state where **leaf nodes** attain a **pure** state.

- When do we pre-prune a decision tree

Pre-pruning a decision tree, also known as early stopping, involves setting certain criteria or conditions to halt the tree-building process before it becomes too complex or overfits the training data. Pre-pruning is typically done to prevent the decision tree from becoming too large and to improve its generalization to unseen data. Here are some common scenarios and conditions in which you might consider pre-pruning a decision tree:

1. **Maximum Depth:** You can set a maximum depth for the tree, limiting the number of levels it can grow. Once the tree reaches this depth, it stops splitting and creates leaf nodes. This is useful when you want to avoid deep, overly complex trees that may overfit the data.

2. **Minimum Samples per Leaf (Min Leaf Size):** You can specify a minimum number of training samples required to create a leaf node. If a branch has fewer samples than this threshold, it is not further split. This helps prevent the creation of very small leaf nodes that could lead to overfitting.

3. **Minimum Samples per Split (Min Split Size):** Similar to the minimum samples per leaf, you can set a minimum number of samples required to split a node. If a node has fewer samples than this threshold, it is not split further. This can prevent unnecessary splits on small subsets of data.

4. **Maximum Number of Leaf Nodes:** You can limit the total number of leaf nodes in the tree. Once this limit is reached, no further splitting occurs. This constraint helps control the complexity of the tree.

5. **Maximum Number of Features:** If you have a large number of features, you can restrict the number of features considered for splitting at each node. This is known as "max_features" and can help prevent overfitting when dealing with high-dimensional data.

6. **Cross-Validation:** You can use cross-validation to assess the performance of the tree on a validation set at each stage of the tree-building process. If performance on the validation set starts to degrade, you can stop growing the tree further.

7. **Impurity Threshold:** You can set a threshold for impurity measures (e.g., Gini impurity or entropy). If the impurity improvement from a split falls below this threshold, the split is not performed. This avoids splits that don't significantly improve the node's purity.

8. **Minimum Improvement in Impurity:** Similar to the impurity threshold, you can require that a split must result in a minimum improvement in impurity (e.g., a certain reduction in Gini impurity or entropy) for it to be considered.

Pre-pruning is an essential technique to prevent decision trees from becoming overly complex and overfitting the training data. The choice of which pre-pruning criteria to use depends on the specific characteristics of your dataset and problem. Experimentation and validation techniques (e.g., cross-validation) can help determine the most effective pre-pruning conditions for your decision tree model.

$$\min_{\mathbf{w},b,\boldsymbol{\xi}^+,\boldsymbol{\xi}^-} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-)$$

$$\begin{aligned} \text{s.t.} \quad & y_i - \mathbf{w}^T\mathbf{x}_i - b \leq \epsilon + \xi_i^+ \quad \forall i \\ & \mathbf{w}^T\mathbf{x}_i + b - y_i \leq \epsilon + \xi_i^- \quad \forall i \\ & \xi_i^+ \geq 0 \quad \forall i \\ & \xi_i^- \geq 0 \quad \forall i \end{aligned}$$

$$\min_{\mathbf{w},b} \quad \frac{1}{n}\sum_{i=1}^{n}\ell_\epsilon(y_i, \mathbf{w}^T\mathbf{x}_i + b) + \lambda\|w\|_2^2$$

In this equation:

w represents the weight vector.

b is the bias term.

ξ & ξ* are slack variables that allow some points to be on the wrong side of the margin or hyperplane but penalize them for doing so.

C is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the error. It is a regularization parameter, and larger values of C correspond to less tolerance for errors.

ϵ is a hyperparameter that defines the width of the epsilon-insensitive tube. Data points within this tube do not contribute to the error. The goal of SVR is to find the values of w and b that minimize the objective function while satisfying the constraints.

Q2. Is it a maximum margin classifier, justify

No, Support Vector Regression (SVR) is not a maximum margin classifier. It is a regression technique used for predicting continuous numerical values, and its objective is to minimize the error between predicted and actual values while maintaining a specified margin of tolerance (epsilon-insensitive tube).

Q3. What is the difference between SVR and SVM.

Certainly, here's a tabular comparison between Support Vector Machines (SVM) and Support Vector Regression (SVR):

Aspect	Support Vector Machines (SVM)	Support Vector Regression (SVR)
Purpose	Classification tasks	Regression tasks
Output	Class labels	Continuous numerical values
Loss Function	Hinge loss or variants for classification	Loss function for regression within an epsilon-insensitive tube
Application	Image classification, text categorization, spam detection, etc.	Stock price prediction, disease outcome forecasting, time-series forecasting, etc.
Output Interpretation	Class label (e.g., "Yes" or "No")	Numerical prediction (e.g., temperature, stock price)
Objective	Maximize margin between classes	Minimize error within a specified margin
Decision Boundary	Hyperplane separating classes	Regression function fitting data points
Supervised Learning Algorithm	Yes	Yes

This table summarizes the key differences between SVM and SVR in terms of their objectives, outputs, loss functions, applications, and more.

Q4. What is the significance of ε and

In Support Vector Regression (SVR), ε (epsilon) and ξ (xi) are important parameters that play a significant role in shaping the SVR model's behavior and determining the trade-off between fitting the data and controlling the margin of error. These parameters are crucial for fine-tuning the SVR model's performance for specific applications.

ε (Epsilon):

Epsilon, denoted as ε, defines the width of the epsilon-insensitive tube around the regression line. This tube represents a region where errors (deviations from the actual target values) are not penalized. Data points within this tube do not contribute to the loss function.
ε defines the tolerance for errors in SVR, allowing for some deviations from the regression line without penalty,
Points outside the tube are penalized linearly based on how far they deviate from the tube boundaries. The greater the deviation, the larger the penalty.
Adjusting the value of ε allows you to control the tolerance for errors in the model. Smaller values of ε result in a smaller insensitive tube and a stricter fit, while larger values of ε allow for more errors within the tube.

ξ (Xi):

ξ (xi) represents slack variables in the SVR optimization problem. These slack variables allow some data points to be on the wrong side of the margin or hyperplane, but they are penalized for doing so.
In the SVR objective function, ξ and ξ* (dual variables) are introduced for each data point in the training set. ξ represents the distance of a data point below the lower margin boundary, while ξ* represents the distance above the upper margin boundary.
ξ and ξ* help in handling data points that cannot be fitted within the epsilon-insensitive tube due to noise or outliers. By introducing these slack variables, SVR finds a balance between fitting the data closely and maintaining a margin of error.
The regularization parameter C in SVR controls the trade-off between minimizing ξ and maximizing the margin. Larger values of C penalize ξ more heavily, leading to a smaller margin but potentially better fitting to the training data.
ξ and C control the balance between fitting the data closely and maintaining a margin of error, helping the SVR model generalize well to new data while handling outliers and noise in the training set.
Q5. State two applications of SVR

Financial Forecasting: SVR is often used in finance for tasks such as stock price prediction, commodity price forecasting, and exchange rate prediction. Financial data is typically noisy and exhibits non-linear patterns, making it suitable for SVR. SVR can capture complex dependencies between various financial indicators and provide accurate forecasts, which is crucial for making informed investment decisions.

Medical Outcome Prediction: SVR can be applied in the field of healthcare for tasks like disease progression modeling, patient outcome prediction, and medical image analysis. SVR can handle high-dimensional data and non-linear relationships between medical features, allowing it to make predictions based on patient data, genetic information, and other relevant factors. For example, SVR can be used to predict the progression of diseases like Alzheimer's or to forecast patient survival rates based on clinical data.

- Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called **overfitting**. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a **test set**. Does it solve the problem? Then why do you cross validate?

- Define support vectors, hyperplane and margin.

Support vectors are data points from the training dataset that lie closest to the decision boundary, also known as the hyperplane. These are the critical data points that influence the position and orientation of the hyperplane.
Support vectors are the data points that have non-zero weights (also called Lagrange multipliers or coefficients) in the SVM's optimization problem. They are the most challenging and informative points for the SVM algorithm because they define the margin and the decision boundary.

In SVM, a hyperplane is a high-dimensional linear decision boundary that separates data points belonging to different classes. For a binary classification problem, the hyperplane is the surface that maximizes the margin between the two classes.
Mathematically, a hyperplane can be represented as a linear equation: w * x + b = 0, where w is the weight vector (normal to the hyperplane) and b is the bias term.

The margin in an SVM is the distance between the hyperplane and the nearest data point (support vector) from either of the two classes. It represents the separation or generalization capacity of the SVM.
The goal of the SVM training process is to maximize this margin. In other words, it aims to find the hyperplane that not only separates the data but also maximizes the minimum distance between the hyperplane and any data point.
A larger margin typically indicates better generalization to unseen data because it suggests a greater degree of separation between classes, reducing the risk of misclassification.

- The main goal of SVMs is to divide the datasets into a number of classes in order to find a maximum marginal hyperplane (MMH). Why?

The main goal of Support Vector Machines (SVMs) is indeed to divide datasets into classes by finding a maximum margin hyperplane (MMH). This is done for several important reasons:

1. **Optimal Generalization**:
   - The primary motivation behind SVMs is to create a classification model that generalizes well to unseen data. A wider margin (i.e., a larger distance between the classes and the hyperplane) provides better separation between classes and, therefore, a higher likelihood of correct classification for new, unseen data points.
   - The MMH maximizes the margin, making it less likely for the model to make errors on new data points.

2. **Robustness to Outliers**:
   - By maximizing the margin, SVMs are inherently robust to outliers or noisy data points. Outliers that fall far from the MMH have less influence on the position of the hyperplane. This robustness improves the model's ability to handle data with errors or anomalies.

3. **Reduced Overfitting**:
   - A wide margin often results in a simpler decision boundary. Simpler models are less prone to overfitting, which occurs when a model captures noise in the training data rather than the underlying patterns.

- The MMH encourages a model with lower complexity, reducing the risk of overfitting and improving its ability to generalize.

4. **Clear Decision Boundary**:
   - The MMH provides a clear and well-defined decision boundary between classes. This is desirable because it minimizes ambiguity in the model's predictions, making it easier to understand and interpret.

5. **Tolerance to Class Imbalance**:
   - SVMs can handle class imbalance (when one class has significantly fewer samples than the other) effectively by giving more weight to support vectors, which are often present in the minority class. This ensures that even with imbalanced data, the MMH can be found to maximize separation.

6. **Kernel Trick for Non-Linearity**:
   - SVMs can be extended to handle non-linearly separable data by using kernel functions. The MMH can still be sought in the higher-dimensional feature space, allowing SVMs to capture complex decision boundaries in various data types.

In summary, the primary goal of SVMs is to find a maximum margin hyperplane because it leads to a more robust, generalizable, and well-defined classification model. The MMH is a fundamental concept in SVMs that balances the trade-off between maximizing the margin and minimizing classification errors, resulting in an optimal separation of classes.

- What is hard classification in a Hard Support Vector Machine

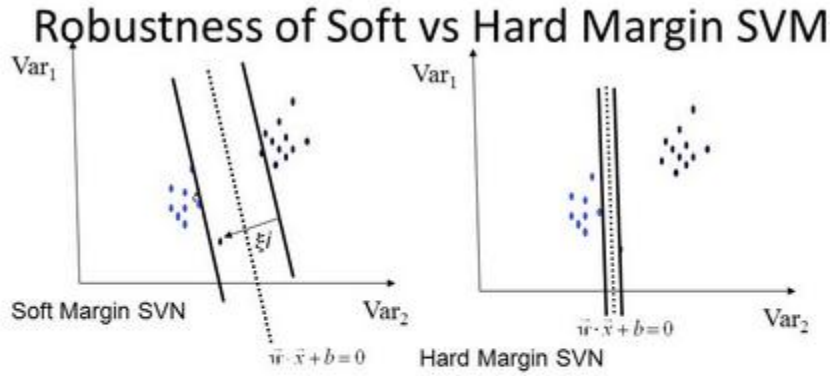
Decision Boundary:

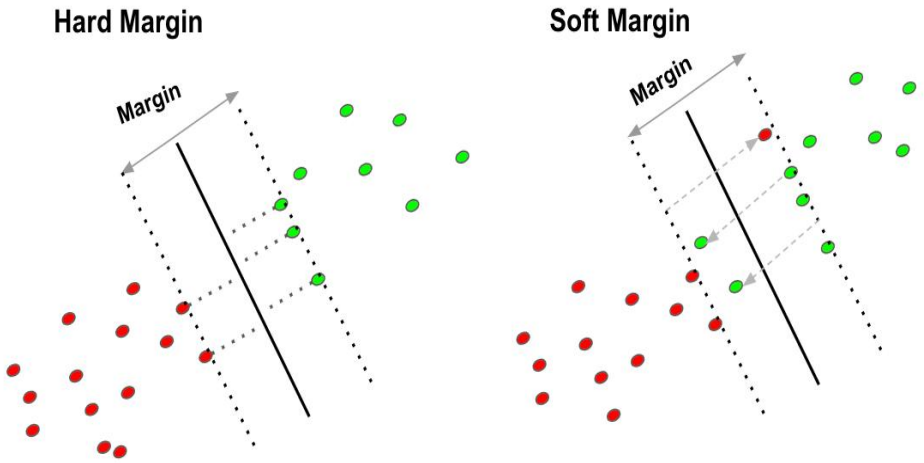
The Hard SVM aims to find a hyperplane (decision boundary) that maximizes the margin between the two classes, as well as ensures that all training data points are correctly classified.
Strict Separation:

In hard classification, the decision boundary is chosen in such a way that every data point is unambiguously assigned to one of the two classes.
Data points lying on or inside the margin are also assigned to one of the classes, and there is no allowance for errors or misclassification in the training data.
No Margin Violation:

Data points are classified in a way that ensures no margin violation occurs. In other words, there is zero tolerance for any data points being inside the margin or on the wrong side of the decision boundary.
While hard classification provides a clear and strict decision boundary, it can be sensitive to noisy data or outliers, and it may not always be suitable for real-world scenarios where data may not be perfectly separable. In cases where data is not linearly separable or contains noise, Hard SVMs may struggle to find a feasible solution without any margin violations.

- Differentiate with soft classifier





The correct choices for the sentences are:

1. Most non-linear datasets are usually in **low** dimensional space in the real-world scenario. Mapping these features further to higher dimensional (kernel space) can result in even **theoretically infinite**-dimensional feature space.

2. As a result, the processing time and power required on this kernel space are **very high**, making this approach inefficient. Another problem associated with higher dimensional mapping is that the model obtained on such a higher dimension is **very complex**.

3. Therefore, we employ a technique known as the kernel trick to make things **less complex**. This technique enables us to work in the input space without visiting the kernel space.

$$K(\vec{x}, \vec{l}^{i}) = e^{-\frac{||\vec{x}-\vec{l}^{i}||^2}{2\sigma^2}}$$

Which kernel is this. : **The Gaussian RBF Kernel in Non Linear SVM**

Explain each parameter in the equation :

1. $K(x, x')$: This represents the kernel function, which computes the similarity (or inner product) between two data points x and x' after they've been mapped to a higher-dimensional space. This similarity score is used in the SVM to make decisions about class boundaries.
2. x **and** x': These are the input data points you want to compute the similarity for. In the context of an SVM, these are feature vectors representing your data.
3. $||x - x'||^2$: This term computes the squared Euclidean distance between the two data points x and x'. It measures how far apart the data points are in the original feature space.
4. σ^2: The σ^2 (sigma squared) parameter is a hyperparameter of the Gaussian RBF kernel. It controls the width or spread of the Gaussian bell curve. Specifically, it determines the influence of each training example on the similarity score. A smaller σ^2 makes the kernel more sensitive to local variations in the data, while a larger σ^2 makes it more globally smooth.
   - A smaller σ^2 leads to a more complex, fine-grained decision boundary.
   - A larger σ^2 results in a smoother, broader decision boundary.

The choice of the σ^2 parameter is crucial and often requires hyperparameter tuning using techniques like cross-validation to find the best value for your specific dataset and problem. A good choice of σ^2 can significantly impact the performance of the SVM with a Gaussian RBF kernel.

- Why do you require ensembles in ML

Ensemble learning is a powerful technique in machine learning that involves combining the predictions of multiple individual models (base learners) to create a more accurate and robust final model. Ensembles are employed for several important reasons:

1. **Improved Predictive Performance**:
   - Ensembles often result in higher predictive accuracy compared to single models. They can reduce overfitting and bias by combining diverse models, each of which may capture different aspects of the data.

2. **Robustness to Noise and Variability**:
   - Ensembles are less susceptible to outliers and noisy data because the errors made by individual models may cancel each other out when combined. This makes ensembles more robust and less likely to make incorrect predictions due to random fluctuations in the data.

3. **Enhanced Generalization**:
   - Ensembles can generalize better to unseen data. By combining models that have been trained on different subsets of the data or with different algorithms, ensembles capture a broader range of patterns and relationships in the data.

4. **Model Stability**:
   - Ensembles can help stabilize the learning process. Individual models might be sensitive to small changes in the training data, but an ensemble's predictions are often more stable.

5. **Reduced Risk of Overfitting**:
   - When using ensembles, particularly techniques like bagging (Bootstrap Aggregating) and random forests, the risk of overfitting is reduced because the final prediction is the result of averaging or voting among multiple models.

6. **Handling Complex Relationships**:
   - Ensembles can better handle complex relationships and non-linearities in data. By combining different models, they can approximate complex decision boundaries more effectively.

7. **Model Selection and Tuning**:
   - Ensembles can provide insights into which features are important and which models perform well. This can guide further model selection and parameter tuning.

8. **Versatility**:
   - Ensembles can be applied to a wide range of machine learning algorithms and problem types. They are not limited to a specific learning algorithm and can be used with decision trees, neural networks, support vector machines, and more.

9. **State-of-the-Art Performance**:
   - In many machine learning competitions and real-world applications, ensemble methods have consistently achieved state-of-the-art performance, demonstrating their effectiveness in practice.

Overall, ensembles are a valuable tool in machine learning because they combine the strengths of multiple models while mitigating their individual weaknesses, leading to more accurate and robust predictions.

- Write the Differences between Bagging and boosting Comparison

Aspect	Bagging	Boosting
Goal	Reduce variance, improve stability	Reduce bias, improve accuracy, correct errors
Base Model Training	Independent, often parallel training	Sequential training, correcting errors
Data Sampling	Bootstrap sampling with replacement	Weighted sampling with focus on misclassifications
Weighted Voting (Combining Predictions)	Equal weight averaging or voting	Weighted averaging or voting with higher weights for better-performing models
Model Diversity	Models are diverse due to random subsets	Models are diverse due to sequential correction of errors

Robustness to Noise	Some noise reduction, but not as robust	More robust due to error correction
Overfitting	Reduces overfitting by averaging	May be prone to overfitting if not controlled
Parallelization	Highly parallelizable	Limited parallelization due to sequential nature
Examples	Random Forests, Bagged Decision Trees	AdaBoost, Gradient Boosting, XGBoost, LightGBM

- EXPLAIN HOW RANDOM FOREST HELPS IN REDUCING THE VARIANCE OF A MODEL.

Random Forest is an ensemble learning method that helps reduce the variance of a model through a combination of bagging and feature randomization techniques. Here's an explanation of how Random Forest achieves this variance reduction:

1. **Bagging (Bootstrap Aggregating)**:
   - Random Forest builds multiple decision trees, typically known as "trees" in the forest. These trees are grown independently using bootstrapped subsets of the original training data.
   - Bootstrapping involves randomly selecting samples from the training dataset with replacement. As a result, each tree in the Random Forest is trained on a slightly different subset of the data.
   - By training each tree on a different subset, Random Forest introduces diversity into the ensemble. This diversity reduces the variance because individual trees may make different errors due to their different training data.

2. **Feature Randomization**:
   - In addition to bootstrapping, Random Forest also introduces feature randomization. When creating each node in a decision tree, instead of considering all features for splitting, only a random subset of features (a subset of the available predictors) is considered.
   - The choice of a random subset of features at each node prevents any single feature from dominating the decision-making process across all trees. This feature selection randomness adds another layer of diversity among the trees.

3. **Combining Predictions**:
   - After growing all the individual trees in the forest, Random Forest combines their predictions. For regression tasks, this is typically done by averaging the predictions of all trees, and for classification tasks, it involves majority voting.
   - The ensemble prediction is more stable and less prone to overfitting because it incorporates the collective wisdom of all the trees, rather than relying on the prediction of a single tree.

4. **Out-of-Bag (OOB) Error Estimation**:
   - Random Forest can estimate its own generalization error using out-of-bag (OOB) samples. Since each tree is trained on a different subset of data, the data points not included in a particular tree's bootstrap sample can be used for testing that tree.
   - The OOB error estimate provides an unbiased assessment of the model's performance on unseen data and helps in monitoring whether the model is overfitting.
   - There will be a portion of the original training data that was not included in the bootstrap sample used to train that particular model. This portion of data is referred to as the "out-of-bag" data. the left out data of each sub dataset is used as validation set
Since the out-of-bag data points were not seen during the training of a specific model, they can be used as a kind of "held-out" or validation set specific to that model.

By combining the concepts of bagging, feature randomization, and ensemble averaging, Random Forest reduces the variance of the model. This reduction in variance leads to a more robust and generalizable model that is less sensitive to fluctuations or noise in the training data, ultimately improving its ability to make accurate predictions on new, unseen data.

- Can Random forest perform Regression tasks? What is the main difference between classification and regression using random forest.

Yes, Random Forest can indeed perform regression tasks in addition to classification tasks. Random Forest is a versatile ensemble learning algorithm that can be applied to both types of machine learning problems. The main difference between using Random Forest for classification and regression lies in the type of output or prediction that the algorithm provides:

Classification with Random Forest:

In classification tasks, the goal is to assign a discrete class label or category to each input data point. For example, classifying emails as spam or not spam.
When you use Random Forest for classification, the output consists of class labels or discrete categories.
Random Forest makes predictions by taking a majority vote among the individual decision trees in the ensemble. The class that receives the most votes is the predicted class.
Regression with Random Forest:

In regression tasks, the goal is to predict a continuous numerical value as the output. For example, predicting the price of a house based on its features.
When you use Random Forest for regression, the output consists of numerical values.
Random Forest makes predictions by averaging the predictions of individual decision trees in the ensemble. The final prediction is typically the mean (average) of the predictions from all trees.

- What is bagging? Justify its name.

Bagging, short for Bootstrap Aggregating, is an ensemble machine learning technique that justifies its name through a combination of two key concepts: bootstrapping and aggregation. Here's an explanation of how bagging works and why it's called "bagging":

Bootstrap Sampling (Bag):

Bagging begins with the concept of bootstrapping, which involves creating multiple subsets (bags) of the original dataset through random sampling with replacement.
When bootstrapping is applied to a dataset, each subset (bag) is created by randomly selecting data points from the original dataset with the possibility of selecting the same data point more than once. This random sampling process generates subsets that are similar to, yet different from, the original dataset.
Aggregation:

After generating multiple bootstrap samples (bags), the next step in bagging is to train a separate base model on each of these bags.
These base models can be any machine learning algorithm, but typically, decision trees are used in bagging. Each base model learns from a different subset of the data.
Combining Predictions:

Once all the base models are trained, bagging combines their individual predictions to make a final prediction.
In classification tasks, this often involves majority voting: the class that receives the most votes among the base models' predictions is the final predicted class.
In regression tasks, predictions from all models are typically averaged to obtain the final prediction.

Q3. What are the different ways to combine classifiers.

The choice of which method to use depends on the nature of the problem, the characteristics of the data, and the behavior of the individual base classifiers. Ensemble methods are powerful tools for improving predictive performance and robustness in various machine learning tasks

1. **Voting Methods**:
   - **Majority Voting**: In binary classification, each classifier's prediction is treated as a vote, and the class with the majority of votes is chosen as the final prediction. In multi-class classification, the class with the highest number of votes can be selected.
   - **Weighted Voting**: Similar to majority voting, but each classifier's vote is weighted based on its reliability or performance on the validation data.

2. **Averaging Methods**:
   - **Simple Average**: For regression tasks, the predictions of multiple regressors are averaged to obtain the final prediction.
   - **Weighted Average**: Similar to simple averaging, but each regressor's prediction is weighted based on its performance or reliability.

3. **Bagging (Bootstrap Aggregating)**:
   - Bagging combines multiple base classifiers by training them independently on bootstrapped subsets of the training data. For classification tasks, the final prediction can be obtained by majority voting.

4. **Boosting**:

- Boosting combines multiple base classifiers sequentially, with each classifier trained to correct the errors made by the previous ones. The final prediction is typically a weighted combination of the base classifier predictions.

5. **Random Forests**:
- Random Forests combine multiple decision trees. Each tree is trained on a subset of the data and a random subset of features. The final prediction is obtained through majority voting (for classification) or averaging (for regression).

8. **Adaptive Methods**:
- Methods like AdaBoost adaptively assign weights to data points during training to emphasize the samples that were misclassified by previous classifiers.

Stacking (Stacked Generalization), Stacked Ensembles, Bayesian Model Averaging , Clustering Ensembles are a few other methods.

Q3. What are the different ways to combine classifiers?

The choice of which method to use depends on the nature of the problem, the characteristics of the data, and the behavior of the individual base classifiers. Ensemble methods are powerful tools for improving predictive performance and robustness in various machine learning tasks

1. **Voting Methods**:
- **Majority Voting**: In binary classification, each classifier's prediction is treated as a vote, and the class with the majority of votes is chosen as the final prediction. In multi-class classification, the class with the highest number of votes can be selected.
- **Weighted Voting**: Similar to majority voting, but each classifier's vote is weighted based on its reliability or performance on the validation data.

2. **Averaging Methods**:
- **Simple Average**: For regression tasks, the predictions of multiple regressors are averaged to obtain the final prediction.
- **Weighted Average**: Similar to simple averaging, but each regressor's prediction is weighted based on its performance or reliability.

3. **Bagging (Bootstrap Aggregating)**:
- Bagging combines multiple base classifiers by training them independently on bootstrapped subsets of the training data. For classification tasks, the final prediction can be obtained by majority voting.

4. **Boosting**:
   - Boosting combines multiple base classifiers sequentially, with each classifier trained to correct the errors made by the previous ones. The final prediction is typically a weighted combination of the base classifier predictions.

5. **Random Forests**:
   - Random Forests combine multiple decision trees. Each tree is trained on a subset of the data and a random subset of features. The final prediction is obtained through majority voting (for classification) or averaging (for regression).

8. **Adaptive Methods**:
   - Methods like AdaBoost adaptively assign weights to data points during training to emphasize the samples that were misclassified by previous classifiers.

Stacking (Stacked Generalization), Stacked Ensembles, Bayesian Model Averaging , Clustering Ensembles are a few other methods.

Q2. What is the difference between BOOSTING REGRESSOR AND BOOSTING CLASSIFIER

Boosting is another ensemble learning technique used to improve the performance of machine learning models. Like bagging, boosting can be applied to both regression and classification problems, resulting in Boosting Regressors and Boosting Classifiers. The key difference between them is in how they train and combine base models:

1. Boosting Regressor:
   - Boosting Regressor is used for regression tasks, where the objective is to predict a continuous numeric value (e.g., predicting house prices, stock prices).
   - It works by training a sequence of base regression models, with each subsequent model focusing on the mistakes made by the previous ones.
   - Base models are typically decision trees with limited depth (weak learners).
   - During training, each base model is assigned a weight based on its performance, and the models' predictions are combined through a weighted sum to make the final prediction.
   - The process continues until a predefined number of base models (iterations) are trained or until the performance stops improving.

2. Boosting Classifier:

- Boosting Classifier is used for classification tasks, where the goal is to classify input data into one of several predefined classes or categories (e.g., spam detection, image recognition).
- Similar to Boosting Regressor, it trains a sequence of base classifiers, with each one focusing on correcting the mistakes of its predecessors.
- Base classifiers are typically decision trees with limited depth (weak learners).
- The final prediction in Boosting Classifier is determined by a weighted combination of the base classifiers' predictions, with more weight given to the models that perform better on the training data.
- The boosting process continues for a set number of iterations or until no further improvements in performance can be achieved.

- Multivariate nonlinear regression is a statistical method used to model the relationships between **multiple** independent variables and **many** dependent variables when the relationship is **nonlinear**. It is an extension of multivariate linear regression, which assumes that the relationship between the variables is linear.

- In nonlinear regression, we use **linear/nonlinear** functions to model the relationship between the independent variables and the dependent variable. These functions can take many forms, such as exponential, logarithmic, or power functions.

In nonlinear regression, various forms of functions, both linear and nonlinear, are used to model the relationship between independent variables and the dependent variable. These functions are chosen based on the specific characteristics of the data and the underlying assumptions about how the variables are related. Here are some common types of functions used in nonlinear regression:

Linear Functions: Linear functions can still be employed within a nonlinear regression framework. For example, a simple linear regression model is a special case of nonlinear regression where the relationship between variables is assumed to be a straight line (y = mx + b).

Exponential Functions: Exponential functions have the form y = a * e^(bx), where 'a' and 'b' are parameters. These functions are often used to model exponential growth or decay processes.

Logarithmic Functions: Logarithmic functions can take various forms, such as y = a + b * ln(x) or y = a * ln(bx). They are used when the relationship between variables is expected to be logarithmic in nature.

Power Functions: Power functions are of the form y = a * x^b, where 'a' and 'b' are parameters. These functions are used when there is a power-law relationship between the variables.

Polynomial Functions: Polynomial functions are a type of nonlinear function that includes terms with different powers of the independent variable. For example, a quadratic function is y = ax^2 + bx + c.

- The goal of multivariate nonlinear regression is to find the values of the model **parameters/coefficient**s that best fit the data. This is done by minimizing the sum of the squared differences between the predicted values and the actual values.

Goal of Multivariate Nonlinear Regression:
The goal of multivariate nonlinear regression is indeed to find the values of the model parameters or coefficients that best fit the data. However, it is essential to distinguish between the terms "multivariate" and "nonlinear" in this context:

Multivariate Regression: Multivariate regression involves predicting multiple dependent variables (also known as responses or outcomes) simultaneously based on one or more independent variables (predictors or features). Each dependent variable has its own set of model parameters.

Nonlinear Regression: Nonlinear regression is used when the relationship between the independent variables and the dependent variables is nonlinear. This means that the model is not a linear combination of the parameters but may involve nonlinear functions.

Model Fitting in Multivariate Nonlinear Regression:

The model parameters in multivariate nonlinear regression are typically estimated by minimizing the sum of squared differences between the predicted values and the actual values for each dependent variable. This minimization is done through optimization techniques that seek to find the parameter values that best describe the observed data.

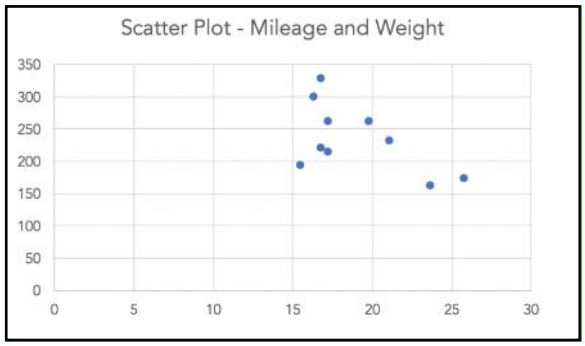
The objective function to be minimized can vary depending on the specific problem, but a common choice for nonlinear regression is to minimize the sum of squared residuals (the differences between predicted and actual values) across all dependent variables. The optimization process finds the parameter values that minimize this overall sum of squares.

- Linear regression is a subset of polynomial regression, as linear regression is just polynomial regression in the **second/ first** order

Linear regression is a subset of polynomial regression, but it is specifically polynomial regression of the first order, which means it involves fitting a straight line to the data

---

Motorcycle	Mileage	Dry Weight
RSV Mille	16.77	220.5
RSV Tuono	17.2	214.2
Mana 850	21.07	232.2
R1150RS	17.2	261.9
K1200GT	16.34	299.7
R1200GS	19.78	261.9
CBR600RR	15.48	193.05
ST1300	16.77	328.95
CBR300R	23.65	161.55
Ninja 300	25.8	173.7



1. Multivariate nonlinear regression is a statistical method used to model the relationships between **multiple** independent variables and **many** dependent variables when the relationship is **nonlinear**. It is an extension of multivariate linear regression, which assumes that the relationship between the variables is linear.

2. In nonlinear regression, we use **nonlinear** functions to model the relationship between the independent variables and the dependent variable. These functions can take many forms, such as exponential, logarithmic, or power functions.

3. The goal of multivariate nonlinear regression is to find the values of the model parameters that best fit the data. This is done by minimizing the sum of the squared differences between the predicted values and the actual values.

4. Linear regression is a subset of polynomial regression, as linear regression is just polynomial regression in the first order

5. What kind of a model will fit the data and why?

cubic polynomial regression

Visual Inspection: Plotting the data in a scatter plot helps visually assess the relationship between "mileage" and "dry weight."

the relationship seems nonlinear (e.g., curved),thus trying various degree values and 3 fits the most.

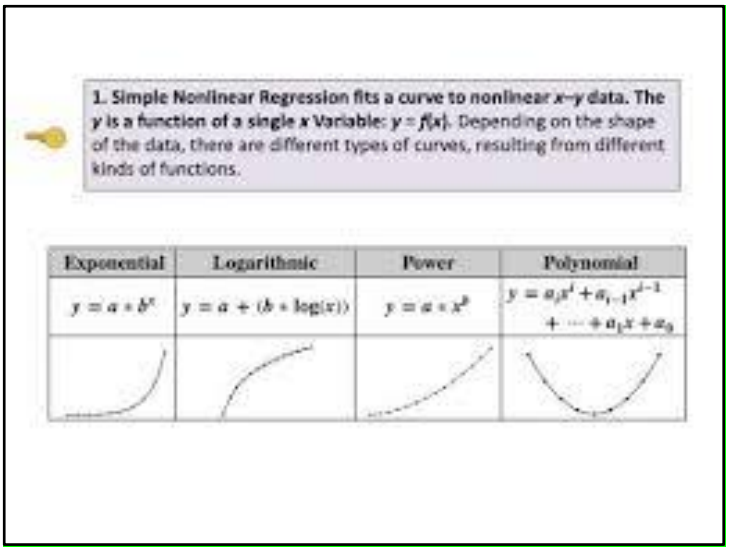
Model Selection: The choice of the model should be guided by the nature of the relationship between the variables.

6. Is it a multivariate/bivariate, linear/ nonlinear model?

multivariate and nonlinear because

Multivariate: The model is multivariate because it involves more than one variable. In this case, you are using both "Mileage" and "Dry Weight" as variables. You are predicting "Dry Weight" based on the "Mileage" variable, so it's a multivariate regression.

Nonlinear: The model is nonlinear because you are using polynomial regression, specifically polynomial features of degree 3. Polynomial regression introduces nonlinear relationships between the predictor variable (Mileage) and the response variable (Dry Weight) by fitting a polynomial function to the data. In contrast, linear regression assumes a linear relationship between variables, which is not the case in this code



- State the significance of k wrt n observations in a dataset

In the k-means algorithm, "k" represents the number of clusters you want to divide your dataset into, while "n" typically refers to the number of data points in your dataset. The significance of "k" with respect to "n" is as follows:

1. "k" is the user-defined parameter: You need to specify the value of "k" before running the k-means algorithm. The choice of "k" determines how many clusters the algorithm will try to create in your dataset.

2. Balancing clusters and data points: The value of "k" should be chosen carefully, taking into consideration the nature of your data and the problem you are trying to solve. If you choose a small "k," you may end up with too few clusters, and if you choose a large "k," you may create too many clusters. Finding the right balance is crucial to obtaining meaningful cluster assignments.

3. Impact on clustering quality: The value of "k" can significantly affect the quality of clustering. A higher "k" can lead to overfitting, where each data point gets its own cluster (poor generalization), while a lower "k" may result in underfitting, where distinct clusters are merged into one (loss of detail).

In summary, "k" in the k-means algorithm is a crucial parameter that determines the number of clusters in the output. It should be chosen based on the characteristics of your data and the goals of your analysis. The relationship between "k" and "n" reflects how many clusters you are trying to create in a dataset containing "n" data points.

- Compare k means with DBSCAN and MST

Aspect	K-Means	DBSCAN	MST
Type	Partitioning clustering	Density-based clustering	Minimum spanning tree (not a clustering algorithm)
Number of Clusters	Must be specified in advance	Automatically determined based on data density	Not applicable (MST finds tree structure, not clusters)
Cluster Shape	Assumes spherical, isotropic clusters	Can discover clusters of arbitrary shapes	Not applicable (MST focuses on graph structure)
Handling Noise	Assigns all points to clusters	Identifies noise points as outliers	Not applicable (MST doesn't distinguish noise points)
Parameter	Number of clusters (k)	Epsilon (Eps) and MinPoints	Not applicable (MST parameter is edge weight threshold)

Core Points	Not applicable (K-Means has centroids)	Identifies core points and border points	Not applicable (MST identifies edges in a graph)
Algorithm	Iterative centroid-based algorithm	Density-based algorithm with connected components	Graph-based algorithm that finds minimum spanning tree
Initialization	Random initial centroids	Not applicable (starts from a data point)	Not applicable (based on input graph)
Suitable Data	Data with spherical clusters	Data with arbitrary-shaped clusters, noise, and outliers	Data represented as a graph

---

- Is it necessary to provide the number of clusters prior to the implementation of DBSCAN

No, it is not necessary to provide the number of clusters prior to implementing DBSCAN (Density-Based Spatial Clustering of Applications with Noise). One of the advantages of DBSCAN is that it can automatically discover the number of clusters within the data based on the density of data points, making it a density-based clustering algorithm.

In DBSCAN, clusters are formed based on the density of data points rather than requiring a predefined number of clusters. The algorithm identifies clusters as areas of high data point density separated by areas of lower density. Therefore, you do not need to specify the number of clusters in advance, which is a common requirement for some other clustering algorithms like k-means.

However, you do need to specify two important parameters:

1. **Epsilon (Eps)**: This parameter defines the radius within which the algorithm searches for neighboring points. It determines the spatial extent of a data point's neighborhood.

2. **MinPoints**: This parameter specifies the minimum number of data points required to form a dense region or cluster. Data points with at least MinPoints neighbors within an Epsilon radius are considered core points.

The choice of appropriate values for Epsilon and MinPoints depends on the characteristics of your data and the desired clustering granularity. With well-tuned Epsilon and MinPoints values, DBSCAN can automatically discover and delineate clusters of varying shapes and sizes within your data without requiring you to specify the number of clusters in advance.

- Is the algorithm sensitive to outliers as is the case of K-Means clustering.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is less sensitive to outliers compared to the K-Means clustering algorithm. Here's why:

1. **K-Means Sensitivity to Outliers:**
   - In K-Means clustering, each data point is assigned to the nearest centroid. Outliers, which are data points significantly distant from the cluster centroids, can have a substantial impact on the clustering result.
   - Outliers can disproportionately influence the position and size of clusters because they can "pull" cluster centroids towards them.
   - K-Means aims to minimize the sum of squared distances between data points and their assigned centroids, so outliers can distort the means of clusters.

2. **DBSCAN's Robustness to Outliers:**
   - DBSCAN, on the other hand, is density-based rather than distance-based. It defines clusters as dense regions of data points separated by areas of lower point density.
   - Outliers, by definition, are typically located in areas of lower density. DBSCAN identifies such points as noise and doesn't assign them to any cluster.
   - DBSCAN's clustering result is less influenced by individual outliers because it primarily focuses on identifying dense regions and doesn't force every data point into a cluster.

In DBSCAN, there are three types of data points:

- **Core Points:** These are data points that have a sufficient number of neighbors within a specified radius (\(eps\)).

- **Border Points:** These are data points that are within the \(eps\) distance of a core point but do not have enough neighbors to be considered core points themselves.

- **Noise (Outlier) Points:** These are data points that are neither core points nor border points. They are typically isolated and considered noise or outliers.

Because DBSCAN explicitly identifies and labels noise points, it is more robust to the presence of outliers. Outliers are not forced into clusters but are treated as noise, allowing DBSCAN to naturally handle datasets with varying levels of noise.

However, it's important to note that DBSCAN's performance can be sensitive to the choice of its hyperparameters, particularly the \(eps\) (neighborhood radius) and the minimum number of points required to form a dense region (\(minPts\)). Proper parameter tuning is necessary to achieve good clustering results with DBSCAN.

- Explain the significance of Epsilon (Eps) and MinPoints.

In the context of DBSCAN (Density-Based Spatial Clustering of Applications with Noise), "Epsilon" (often denoted as "Eps") and "MinPoints" are two crucial parameters that significantly impact the clustering process. Here's an explanation of their significance:

1. **Epsilon (Eps)**:
   - **Definition**: Epsilon (Eps) is a positive real number that defines the radius around each data point within which the algorithm searches for neighboring points.
   - **Significance**: Epsilon determines the spatial extent of a data point's neighborhood. Data points within this distance are considered neighbors, and they play a central role in defining core points in DBSCAN.
   - **Impact**:
     - A larger Epsilon value results in a broader definition of what constitutes a neighbor, potentially leading to larger clusters.
     - A smaller Epsilon value restricts the neighborhood size, potentially leading to smaller and more tightly packed clusters.
   - **Tuning**: Choosing an appropriate Epsilon value depends on the specific characteristics of your dataset. It often requires domain knowledge or trial and error.

2. **MinPoints**:
   - **Definition**: MinPoints is a positive integer that specifies the minimum number of data points required to form a dense region or cluster.
   - **Significance**: MinPoints defines the density threshold that determines whether a data point is a core point or not. Core points must have at least MinPoints neighbors (including themselves) within an Epsilon radius.
   - **Impact**:
     - Increasing MinPoints requires denser clusters to be formed, which leads to more robust clustering results but may result in smaller clusters.
     - Decreasing MinPoints allows for the detection of sparser clusters but may increase the likelihood of noise points being included as part of a cluster.
   - **Tuning**: Like Epsilon, selecting an appropriate MinPoints value depends on the dataset and the desired cluster density. It may require experimentation to find the optimal value.

The significance of Epsilon and MinPoints lies in their ability to control the size, density, and granularity of clusters identified by DBSCAN. Properly setting these parameters is essential to achieve meaningful and accurate clustering results. A well-tuned combination of Epsilon and MinPoints helps DBSCAN

identify clusters of varying shapes and densities while effectively distinguishing noise points from meaningful clusters in spatial data.

- Explain the 3 types of data points used in the algorithm

1. **Core Points**:
   - Core points are data points that have at least a specified minimum number of other data points (a minimum density) within a certain distance (epsilon or ε) from them.
   - In DBSCAN, core points are the central points around which clusters are formed. They are characterized by their relatively high local density.

2. **Border Points**:
   - Border points are data points that are within the ε-distance of a core point but do not have enough neighboring data points to be considered core points themselves.
   - Border points are part of a cluster but are on the outer edges of the cluster and have lower local density compared to core points.

3. **Noise Points (Outliers)**:
   - Noise points, also known as outliers, are data points that do not belong to any cluster and do not satisfy the criteria for being a core or border point.
   - Noise points are typically isolated data points that do not fit well into any cluster and are considered noise in the dataset.

- Write the steps for MST algorithm

Kruskal's algorithm for finding the Minimum Spanning Tree:

1. **Initialize**: Start with an empty set to represent the MST.

2. **Sort Edges**: Sort all edges in non-decreasing order of their weights.

3. **Select Edge**: Take the edge with the smallest weight that does not create a cycle in the MST. Add it to the MST.

4. **Repeat**: Continue selecting the next smallest edge that doesn't create a cycle until there are (n-1) edges in the MST, where n is the number of vertices.

5. **Output**: The MST is complete and contains (n-1) edges with the minimum total weight.

- Explain in brief Agglomerative and Hierarchical clustering algorithms

Agglomerative Clustering:

- **Type**: Bottom-up, hierarchical clustering method.
- **Process**: Starts with each data point as its own cluster and merges clusters iteratively, grouping the closest clusters together.
- **Distance Metric**: Requires a distance or similarity metric to determine cluster proximity.
- **Merging Criteria**: Common merging criteria include single linkage (minimum pairwise distance), complete linkage (maximum pairwise distance), and average linkage (average pairwise distance).
- **Dendrogram**: Represents the hierarchy of clusters as a tree-like structure (dendrogram), which can be cut at various levels to obtain clusters of different sizes.
- **Complexity**: Time complexity can be high for large datasets, especially with single linkage, but it can be more interpretable than some other methods.

Hierarchical Clustering:

- **Type**: Top-down, hierarchical clustering method.
- **Process**: Starts with all data points in a single cluster and recursively divides clusters into smaller sub-clusters.
- **Distance Metric**: Requires a distance or similarity metric to determine cluster dissimilarity.
- **Dividing Criteria**: Common dividing criteria include minimizing the variance within clusters (divisive) or maximizing the similarity between clusters (agglomerative).
- **Hierarchy**: Represents the hierarchy of clusters as a tree-like structure, and each level of the hierarchy corresponds to a different partitioning of the data.
- **Interpretability**: Can provide a comprehensive view of the data's hierarchical structure but may be less straightforward to determine the number of clusters compared to some other methods.

In summary, Agglomerative clustering starts with individual data points and merges clusters iteratively, while Hierarchical clustering starts with all data points in a single cluster and recursively divides them. Both methods result in hierarchical cluster structures but use different approaches to construct them. The choice between them depends on the specific clustering goals and the nature of the data.

Certainly! In the context of Support Vector Machines (SVM), non-zero weights, often referred to as Lagrange multipliers or coefficients, play a crucial role in determining the support vectors and ultimately the hyperplane and margin. Here's a more detailed explanation:

1. **Lagrange Multipliers (λ)**:

- In SVM, the primary objective is to find a hyperplane that maximizes the margin between the two classes while correctly classifying the training data.

- This optimization problem involves finding the weight vector `w` and bias term `b` of the hyperplane. To do this, SVM uses a technique called Lagrange optimization, which involves introducing Lagrange multipliers (λ) to incorporate constraints into the optimization problem.

2. **Constraints**:

- The main constraint in SVM is that all data points should be correctly classified and that they should lie on the correct side of the decision boundary (hyperplane). These constraints can be expressed mathematically as: `yi(w * xi + b) >= 1` for all data points, where `yi` is the class label (+1 or -1), `xi` is a data point, and `w * xi + b` is the distance of the data point to the hyperplane.

3. **Non-Zero Lagrange Multipliers (Support Vectors)**:

- In the Lagrange optimization, each data point corresponds to a Lagrange multiplier (λ). Most of these multipliers will be zero for data points that are correctly classified and do not lie on the margin boundaries.

- The critical insight in SVM is that only a small subset of data points will have non-zero Lagrange multipliers. These are the support vectors, which are the data points closest to the decision boundary (hyperplane) and those that are incorrectly classified or very close to it.

- The non-zero Lagrange multipliers (λ) associated with the support vectors determine the orientation and position of the hyperplane. Specifically, they define the weights `w` of the hyperplane.

4. **Hyperplane and Margin**:

- Once the non-zero Lagrange multipliers (support vectors) are identified, they are used to calculate the weight vector `w` of the hyperplane. The hyperplane equation is of the form: `w * x + b = 0`.

- The margin is computed as the perpendicular distance from the hyperplane to the closest support vector. The goal of SVM training is to maximize this margin while ensuring that all support vectors are correctly classified.

In summary, Lagrange multipliers (λ) are introduced to incorporate constraints into the SVM optimization problem. Non-zero Lagrange multipliers correspond to support vectors, which are crucial for defining the hyperplane's orientation and position, as well as determining the margin

in SVM. By maximizing this margin, SVM finds the optimal hyperplane for separating the classes while minimizing the risk of misclassification on the training data.

Support vectors can indeed lie on the margin, but they can also be inside the margin or even on the wrong side of the margin. The key characteristic of support vectors is that they are the data points closest to the decision boundary (hyperplane), regardless of whether they are on the margin boundary, inside the margin, or on the wrong side of the margin. Here are the different possibilities:

1. **Support Vectors on the Margin**:

   - These are support vectors that lie exactly on the margin boundary, which means they are equidistant from the hyperplane on both sides.

   - Support vectors on the margin are typically the most influential in defining the hyperplane because they determine the orientation of the hyperplane.

2. **Support Vectors Inside the Margin**:

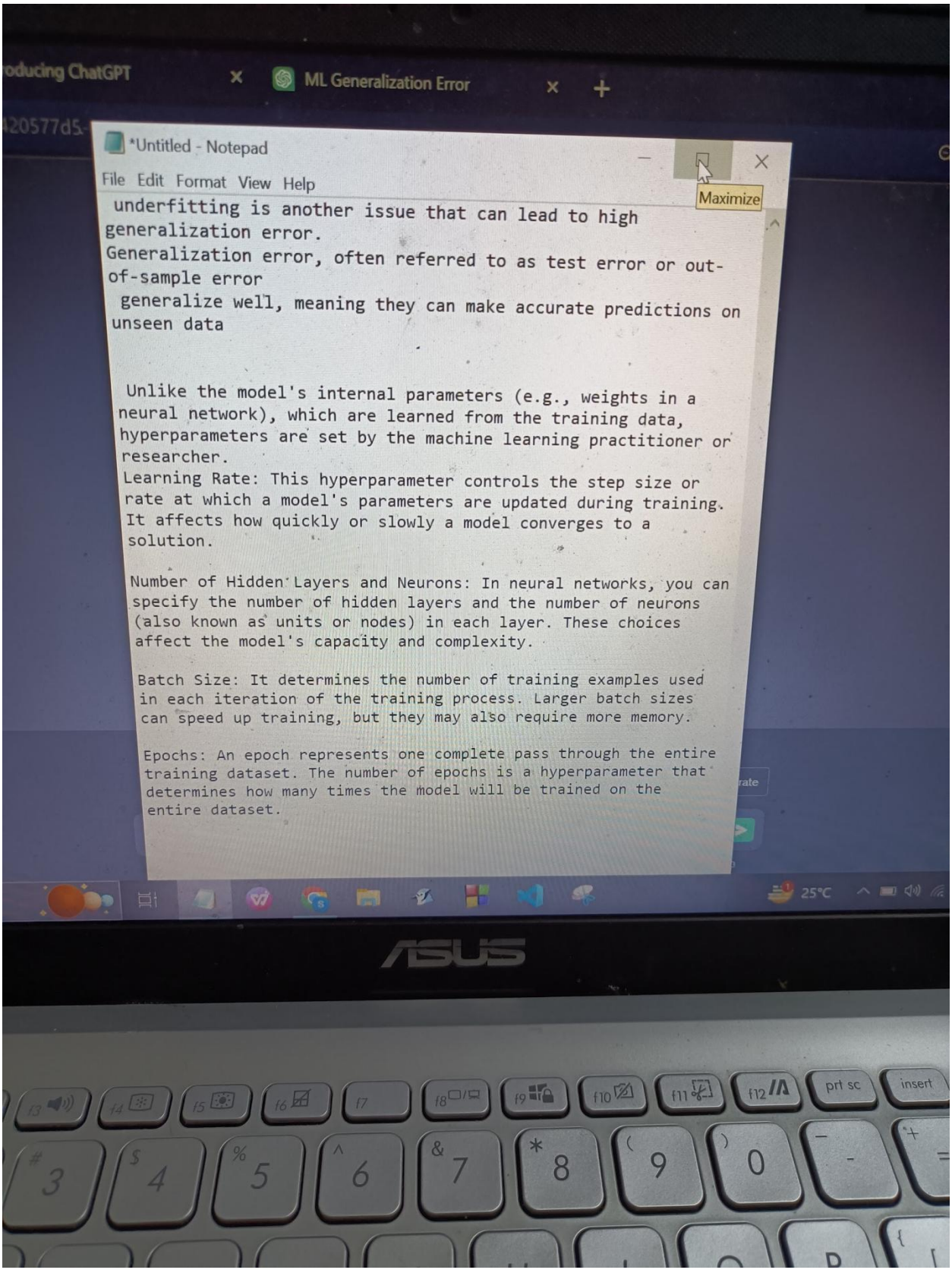
   - Some support vectors may be inside the margin but still closer to the hyperplane than other data points.

   - These support vectors are crucial because they contribute to maximizing the margin, even though they are not directly on the margin boundary.

3. **Support Vectors on the Wrong Side of the Margin**:

   - In some cases, support vectors may be misclassified and lie on the wrong side of the margin.

   - These support vectors are important because they highlight the difficulty of classifying certain data points. SVM aims to


oducing ChatGPT
ML Generalization Error
*Untitled - Notepad
File Edit Format View Help
Maximize
underfitting is another issue that can lead to high generalization error.
Generalization error, often referred to as test error or out-of-sample error
generalize well, meaning they can make accurate predictions on unseen data
Unlike the model's internal parameters (e.g., weights in a neural network), which are learned from the training data, hyperparameters are set by the machine learning practitioner or researcher.
Learning Rate: This hyperparameter controls the step size or rate at which a model's parameters are updated during training. It affects how quickly or slowly a model converges to a solution.
Number of Hidden Layers and Neurons: In neural networks, you can specify the number of hidden layers and the number of neurons (also known as units or nodes) in each layer. These choices affect the model's capacity and complexity.
Batch Size: It determines the number of training examples used in each iteration of the training process. Larger batch sizes can speed up training, but they may also require more memory.
Epochs: An epoch represents one complete pass through the entire training dataset. The number of epochs is a hyperparameter that determines how many times the model will be trained on the entire dataset.
25°C
ASUS

minimize the classification error while maximizing the margin, so these support vectors are critical in finding the best trade-off between these objectives.

So, support vectors can take different positions relative to the margin, but what they have in common is their role in influencing the hyperplane's position and orientation and, by extension, the margin. The presence of support vectors is what allows SVM to create a decision boundary that maximizes the margin while correctly classifying as many data points as possible.

GENERALIZATION ERROR:

HYPERPARAMETER:

HYPERPARAMETER TUNING:

Principal Component Analysis (PCA), Singular Value Decomposition (SVD), and Linear Discriminant Analysis (LDA) are all dimensionality reduction techniques that aim to reduce the number of features (or dimensions) in a dataset while preserving relevant information. Let's discuss what each of these techniques does and how they can be applied to a student dataset.

Before Applying Dimensionality Reduction:
Suppose you have a student dataset with various features like age, gender, test scores in multiple subjects, extracurricular activities, attendance, and more. This dataset may have a high dimensionality, meaning it has many features, which can lead to several issues:

1. **Curse of Dimensionality:** High-dimensional datasets can suffer from the curse of dimensionality, making it challenging to visualize, analyze, and process the data effectively.

2. **Overfitting:** High-dimensional data is more prone to overfitting when building machine learning models. Overfitting occurs when models capture noise rather than the underlying patterns in the data.

3. **Computation and Storage Costs:** Working with high-dimensional data can be computationally expensive, and it may require more storage resources.

After Applying Dimensionality Reduction:
Now, let's see what each of these dimensionality reduction techniques does and how they can help with the student dataset:

1. **Principal Component Analysis (PCA):**
   - **What it does:** PCA is a technique that reduces the dimensionality of data while retaining as much variance as possible. It identifies new orthogonal features (principal components) that are linear combinations of the original features.
   - **Application:** Applying PCA to the student dataset would result in a reduced set of principal components that capture the most important patterns in the data. This can help in data visualization, noise reduction, and potentially improving the performance of machine learning models.

2. **Singular Value Decomposition (SVD):**
   - **What it does:** SVD is a matrix factorization technique that can be used for dimensionality reduction. It decomposes the original data matrix into three matrices: U, Σ (a diagonal matrix), and V. Dimensionality reduction is achieved by truncating or selecting a subset of the diagonal values from Σ and the corresponding columns from U and V.
   - **Application:** By applying SVD, you would reduce the dimensionality of the student dataset while retaining the most significant information. This can lead to more efficient data analysis and modeling.

3. **Linear Discriminant Analysis (LDA):**
   - **What it does:** LDA is a supervised dimensionality reduction technique that aims to maximize the separability between different classes or groups in the data. It finds the linear combinations of features that best separate the classes.
   - **Application:** In the context of a student dataset, LDA could be used to reduce dimensionality while ensuring that features that discriminate between student categories (e.g., pass/fail) are preserved. It can be valuable in classification tasks.

In summary, PCA, SVD, and LDA are dimensionality reduction techniques that can help address the challenges associated with high-dimensional datasets. They can simplify data analysis, visualization, and modeling while preserving relevant information. The specific choice of technique depends on the goals of your analysis and the nature of your dataset, such as whether it's supervised or unsupervised.

PYQ

In machine learning, the concept of entropy is related to information theory and is used in the context of decision trees, particularly in algorithms like ID3 (Iterative Dichotomiser 3) and its successor, C4.5. Entropy in machine learning doesn't directly measure the disorder of a system as in thermodynamics, but it quantifies the impurity or disorder of a dataset with respect to the class labels in a classification problem. Let's clarify the concept of entropy in machine learning:

1. **Entropy in Decision Trees**:
   - Decision trees are a popular machine learning algorithm used for classification and regression tasks.
   - In classification problems, the goal is to partition the dataset into subsets in a way that the resulting subsets are as pure as possible, meaning that they contain as many data points from a single class as possible.
   - Entropy is used to measure the impurity or disorder of a dataset with respect to the class labels. It helps in deciding how to split the data effectively.
   - The formula for entropy in a binary classification problem is:
     ```
     Entropy(S) = -p(+) * log2(p(+)) - p(-) * log2(p(-))
     ```
     where `p(+)` is the proportion of positive class examples in the subset `S`, and `p(-)` is the proportion of negative class examples in the subset.

2. **Information Gain**:
   - Entropy is used to calculate the Information Gain, which helps in deciding which attribute to split on at each node of a decision tree.
   - Information Gain is a measure of how much the entropy decreases after a dataset is split on a specific attribute. It quantifies the reduction in disorder achieved by making the split.

3. **Decision Tree Splitting**:
   - At each node of a decision tree, the attribute that maximizes Information Gain is selected for splitting.
   - This attribute is chosen because it reduces the overall impurity or entropy of the dataset, leading to more pure child nodes.

In summary, in machine learning, entropy is used to assess the impurity or disorder of data with respect to class labels. It's a fundamental concept in the context of decision trees, where it plays a crucial role in determining how to split the data effectively to build a predictive model. The goal is to create a tree structure that minimizes entropy, leading to pure and accurate classifications.

Occam's Razor is the idea that the simplest explanation or model is often the best. In machine learning, it's applied to favor simpler models that generalize well. The Naive Bayes algorithm explicitly follows this principle, as do regularization techniques in various models.

Find predicted value of Y for one epoch and RMSE using Linear regression.

X	Y-Actual
2	1
3	3
4	6
5	9
6	11
7	13
8	15
9	17
10	20

Find the new revised *theta* for the given problem using Expectation -Maximization Algorithm for one epoch.

1	H	T	T	T	H	H	T	H	T	H
2	H	H	H	H	H	T	H	H	H	H
3	H	T	H	H	H	H	H	T	H	H
4	H	T	H	T	T	T	H	H	T	T
5	T	H	H	H	T	H	H	H	T	H

$\Theta_A = 0.6$ and $\Theta_B = 0.5$

Use Principal Component Analysis (PCA) to arrive at the transformed matrix for the given matrix A.

$$A^T = \begin{matrix} 2 & 1 & 0 & -1 \\ 4 & 3 & 1 & 0.5 \end{matrix}$$

Find optimal hyper plane for the following points:

{(1, 1), (2, 1), (1, -1), (2,-1), (4, 0), (5, 1), (6, 0)}

Issues in decision tree induction include:

1. Overfitting: Trees can become overly complex and fit noise in the data, leading to poor generalization.

2. Bias toward certain features: Decision trees may favor features with many values, potentially ignoring other relevant features.

3. Lack of interpretability: Deep trees can be hard to understand, diminishing their utility.

4. Instability: Small changes in data can lead to different trees, affecting robustness.

5. Handling missing data: Decision trees can struggle with missing values in features.

6. Imbalanced data: Trees may not handle imbalanced class distributions well.

7. Greedy nature: Decision trees make locally optimal decisions, which may not lead to the globally best tree.

The values of independent variable x and dependent value y are given below:

X	Y
0	2
1	3
2	5
3	4
4	6

Find the least square regression line y=ax+b. Estimate the value of y when x is 10.

The Radial Basis Function (RBF) kernel is a key component of Support Vector Machines (SVM) used to separate non-linear patterns. Here's its role:

1. **Non-Linear Transformation:** RBF is a kernel function that maps the original feature space into a higher-dimensional space where data may become linearly separable. This transformation helps SVM handle non-linear patterns.

2. **Complex Decision Boundaries:** In the higher-dimensional space, RBF kernel creates complex, non-linear decision boundaries that can effectively separate non-linear data distributions.

3. **Kernel Trick:** The RBF kernel uses a mathematical technique known as the "kernel trick" to avoid explicitly performing the transformation, as it can be computationally expensive. Instead, it computes the dot product in the higher-dimensional space efficiently.

4. **Tuning:** The RBF kernel has a parameter called gamma (γ) that influences the shape of the decision boundaries. Tuning this parameter allows for controlling the complexity of the decision boundaries, which can be critical in capturing the underlying non-linear patterns accurately.

In summary, the RBF kernel in SVM is instrumental in transforming the data into a higher-dimensional space, enabling the SVM to model and separate non-linear patterns by creating complex and flexible decision boundaries.

Use Principal Component analysis (PCA) to arrive at the transformed matrix for the given matrix A.

$A^T = \begin{matrix} 2 & 1 & 0 & -1 \\ 4 & 3 & 1 & 0.5 \end{matrix}$

Elements of reinforcement learning (RL) include:

1. **Agent:** The entity that interacts with an environment, making decisions and taking actions.

2. **Environment:** The external system with which the agent interacts. It includes everything the agent can observe and affect.

3. **State (S):** A representation of the environment at a given time, which the agent uses to make decisions.

4. **Action (A):** The set of possible choices or decisions the agent can make to influence the environment.

5. **Policy (π):** The strategy or set of rules that the agent follows to select actions based on states.

6. **Reward (R):** A numerical value that the agent receives from the environment after taking an action in a particular state, indicating the immediate desirability of that action.

7. **Value Function (V or Q):** A function that estimates the expected cumulative reward the agent can achieve from a given state (V) or a state-action pair (Q).

8. **Exploration vs. Exploitation:** The trade-off between exploring new actions to learn more about the environment and exploiting known actions that are likely to yield higher rewards.

9. **Learning Algorithm:** The method used by the agent to update its policy, value function, or model of the environment based on interactions and rewards.

10. **Markov Decision Process (MDP):** A mathematical framework that formally defines the RL problem, consisting of states, actions, transition probabilities, rewards, and a discount factor.

11. **Discount Factor (γ):** A value that represents the agent's preference for immediate rewards over delayed rewards in RL problems.

12. **Exploration Strategy:** Techniques used by the agent to decide which actions to explore, often involving randomness or heuristics.

13. **Episode:** A single run or sequence of interactions between the agent and the environment, typically starting from an initial state and ending with a terminal state.

14. **Policy Evaluation and Improvement:** Iterative processes to assess and update the agent's policy to improve decision-making.

These elements collectively form the foundation of reinforcement learning, a subfield of machine learning that focuses on training agents to make sequential decisions in dynamic environments to maximize cumulative rewards.

Issues in ml

1. Overfitting
2. Underfitting
4. Data Imbalance
5. Feature Engineering
6. Curse of Dimensionality
7. Computational Resources
8. Interpretability
9. Generalization
10. Model Selection
11. Hyperparameter Tuning

Key terminologies in reinforcement learning using temporal difference learning include:

1. **Agent:** The learner or decision-maker that interacts with the environment.

2. **Environment:** The external system or world with which the agent interacts.

3. **State (S):** A representation of the environment at a specific time.

4. **Action (A):** The choice or decision made by the agent in a particular state.

5. **Policy (π):** The strategy or set of rules that the agent uses to select actions.

6. **Reward (R):** A numerical value received by the agent after taking an action in a particular state, indicating immediate desirability.

7. **Value Function (V or Q):** A function that estimates the expected cumulative reward from a state (V) or a state-action pair (Q).

8. **Bellman Equation:** An equation that relates the value of a state to the values of its neighboring states.

9. **Temporal Difference (TD) Error:** The difference between the predicted value and the actual observed value, used for updating value functions.

10. **Exploration vs. Exploitation:** The trade-off between exploring new actions and exploiting known actions to maximize rewards.

11. **Episode:** A sequence of interactions between the agent and the environment, often starting from an initial state and ending in a terminal state.

12. **Q-Learning:** A specific temporal difference learning algorithm used for learning Q-values.

13. **SARSA:** Another specific temporal difference learning algorithm that estimates action-state-action values.

These terms are fundamental to understanding and working with reinforcement learning using temporal difference learning methods.

Key terminologies in reinforcement learning using temporal difference learning include:

1. **Agent:** The learner or decision-maker that interacts with the environment.

2. **Environment:** The external system or world with which the agent interacts.

3. **State (S):** A representation of the environment at a specific time.

4. **Action (A):** The choice or decision made by the agent in a particular state.

5. **Policy (π):** The strategy or set of rules that the agent uses to select actions.

6. **Reward (R):** A numerical value received by the agent after taking an action in a particular state, indicating immediate desirability.

7. **Value Function (V or Q):** A function that estimates the expected cumulative reward from a state (V) or a state-action pair (Q).

8. **Bellman Equation:** An equation that relates the value of a state to the values of its neighboring states.

9. **Temporal Difference (TD) Error:** The difference between the predicted value and the actual observed value, used for updating value functions.

10. **Exploration vs. Exploitation:** The trade-off between exploring new actions and exploiting known actions to maximize rewards.

11. **Episode:** A sequence of interactions between the agent and the environment, often starting from an initial state and ending in a terminal state.

12. **Q-Learning:** A specific temporal difference learning algorithm used for learning Q-values.

13. **SARSA:** Another specific temporal difference learning algorithm that estimates action-state-action values.

These terms are fundamental to understanding and working with reinforcement learning using temporal difference learning methods.

Well-Posed Learning Problem:
A well-posed learning problem is a problem in which all necessary elements are clearly and unambiguously defined. It includes a clear description of the task, the dataset, the performance measure, and the conditions under which the learner is expected to perform. A well-posed problem allows for a meaningful solution to be found using a machine learning algorithm.

Robot Driving Learning Problem:
The robot driving learning problem involves teaching a robot to navigate and control a vehicle autonomously.

A kernel in the context of machine learning, especially in Support Vector Machines (SVM), is a function that transforms the input data into a higher-dimensional space.
Kernels transform the input data into a higher-dimensional feature space where it might become linearly separable. In this new space, the SVM attempts to find a hyperplane that best separates the classes.

Instead of explicitly computing the transformation and representing the data in this high-dimensional space, the kernel function calculates the dot product (inner product) between the transformed feature vectors. This is known as the "kernel trick."

3. No Need for High-Dimensional Data: With the kernel trick, you don't need to work with high-dimensional data explicitly, which is computationally expensive. The SVM operates in the original feature space but effectively utilizes the non-linear relationships captured by the kernel.

Q-Learning is a model-free reinforcement learning algorithm used for making decisions in an environment. It's a method for learning a Q-table, where "Q" stands for "Quality," which contains action values indicating the expected cumulative reward an agent can achieve by taking a specific action in a specific state.

Q-Learning Algorithm:

Initialize Q-Table: Create a Q-table with rows for states and columns for actions, initially set to arbitrary values.

Exploration vs. Exploitation: In each time step, the agent decides whether to explore new actions or exploit its current knowledge based on an exploration strategy (e.g., epsilon-greedy).

Action Selection: Select an action to take in the current state using the exploration strategy.

Perform Action: Execute the selected action, which transitions the agent to a new state.

Receive Reward: Observe the reward associated with the new state-action pair.

Update Q-Value: Update the Q-value for the previous state-action pair using the Q-learning update rule:

Q(state, action) = (1 - α) * Q(state, action) + α * [reward + γ * max(Q(next_state, all_actions))]

α (alpha) is the learning rate, controlling the impact of new information.
γ (gamma) is the discount factor, representing the agent's preference for immediate rewards over delayed ones.
Repeat: Continue the process by taking actions, receiving rewards, and updating the Q-table until convergence or a predefined number of iterations.

Difference from Reinforcement Learning (RL):

Reinforcement learning (RL) is a broader concept that includes various methods for an agent to learn how to make decisions in an environment by trial and error. Q-learning is a specific algorithm within RL. The primary differences are:

Q-Learning Specificity: Q-learning is a specific RL algorithm that focuses on learning Q-values and is suitable for discrete action spaces.

Tabular Approach: Q-learning uses a tabular approach (Q-table) to store and update action values, which is suitable for problems with a small state and action space.

Model-Free: Q-learning is model-free, meaning it doesn't require a model of the environment to make decisions, making it suitable for situations where the agent's actions have an impact but are not entirely predictable.

Certainly, here are brief explanations of the terms with respect to Reinforcement Learning:

1. **Delayed Rewards:** Delayed rewards in reinforcement learning refer to situations where the consequences of an action are not immediately apparent. The agent may need to take a sequence of actions over time before it receives a reward, making it challenging to associate actions with outcomes. The agent must learn to make decisions that lead to long-term cumulative rewards, considering the time delay between actions and rewards.

2. **Exploration:** Exploration is the process by which the agent tries new actions or explores unknown parts of the environment to gather information about which actions lead to better rewards. Balancing exploration and exploitation (selecting actions that are known to lead to rewards) is crucial in reinforcement learning to discover optimal strategies.

3. **Partially Observable States:** In partially observable states, the agent does not have complete information about the current state of the environment. It may have access to limited or noisy observations, making it challenging to make informed decisions. The agent must deal with uncertainty and use available information effectively to make the best choices.
While achieving a high accuracy of 96% in a cancer prediction model is a good start, it doesn't necessarily mean you should be entirely satisfied. Here are a few reasons why you might not want to be overly happy and what you can do to improve the model:

1. **Imbalanced Dataset:** If the dataset is imbalanced, meaning there are significantly more negative cases (non-cancer) than positive cases (cancer), the accuracy metric can be misleading. The model may predict negative for most cases and still achieve a high accuracy.

What to do: Consider using other evaluation metrics such as precision, recall, F1-score, or the area under the ROC curve (AUC) to get a more comprehensive view of the model's performance.

2. **False Negatives:** In a cancer prediction model, false negatives (predicting no cancer when cancer is present) can be more critical than false positives. Achieving high accuracy may not necessarily reduce false negatives.

What to do: Focus on increasing sensitivity (recall) to reduce false negatives. This can be achieved by adjusting the model's decision threshold, collecting more data, or using more sophisticated models.

3. **Overfitting:** A high accuracy on the training data doesn't guarantee similar performance on unseen data. Overfitting can be a concern.

What to do: Regularize the model, use cross-validation, and ensure that the model generalizes well to new data.

4. **Clinical Relevance:** The clinical or practical relevance of the model's predictions is crucial. A model with high accuracy but no real impact on patient outcomes may not be as valuable.

What to do: Collaborate with domain experts to ensure the model's predictions have real clinical significance. The model should lead to actionable insights or interventions.

5. **Data Quality:** Data quality issues, such as missing data, noise, or biases, can impact the model's performance and reduce its trustworthiness.

What to do: Thoroughly preprocess and clean the data, and consider collecting more data or using data augmentation techniques if necessary.

In summary, a high accuracy score is a good sign, but it's essential to consider other factors and evaluation metrics to gauge the model's effectiveness and relevance in a specific domain, especially in critical areas like cancer prediction. Always be open to improving the model and its impact on real-world outcomes.

Certainly, here are explanations of the terms you mentioned in the context of linear regression:

1. **Regression Line:**
   - A regression line is a straight line that represents the relationship between two variables in linear regression.
   - In simple linear regression, it is used to model the relationship between the dependent variable (usually denoted as "y") and an independent variable (usually denoted as "x").
   - The equation of the regression line is typically represented as: `y = b0 + b1 * x`, where "b0" is the intercept, "b1" is the slope, and "x" is the independent variable.

2. **Scatter Plot:**
   - A scatter plot is a graphical representation of data points in a two-dimensional space, often used to visualize the relationship between two variables.
   - Each data point is plotted on the graph, where one variable is represented on the x-axis, and the other variable is represented on the y-axis.
   - Scatter plots help visualize patterns, correlations, and outliers in the data.

3. **Error in Prediction (Residuals):**
   - In linear regression, error in prediction, often referred to as residuals, represents the vertical distance between each data point and the regression line.
   - It quantifies the difference between the actual observed values and the values predicted by the regression line.
   - Residuals are used to assess the model's performance and to validate the assumption that the errors are normally distributed and have constant variance.

4. **Best Fitting Line (Regression Line):**
   - The best-fitting line in linear regression is the line that minimizes the sum of squared residuals (errors) for all data points.
   - It represents the "best" linear approximation of the relationship between the two variables.
   - The best-fitting line is determined by finding the values of "b0" and "b1" in the regression line equation that minimize the sum of squared residuals, making it the line that best represents the data.

In summary, a regression line is a mathematical representation of the relationship between two variables, and it is determined by finding the best-fitting line

Multi-label classification involves assigning multiple labels or categories to an item. For instance, a news article can be categorized as "Technology" and "Business" simultaneously. This approach allows more flexible and nuanced classification, common in applications like text categorization, image tagging, and content recommendation. that minimizes the prediction errors (residuals) for the given data points. Scatter plots are used to visually inspect the data and relationships, and the best-fitting line helps make predictions based on that relationship.

A data point A is considered density-reachable from another data point B if there exists a chain of data points (including B and A) where each point is a core point, and the next point in the chain is within the specified radius of the previous point.
In other words, data point A is density-reachable from data point B if there is a path of core points leading from B to A within the defined distance. This concept is used in DBSCAN to connect core points and form clusters. Data points that are density-reachable from a core point become part of the same cluster.

Learners in Classification Problems

In the classification problems, there are two types of learners:

1. **Lazy Learners:** Lazy Learner firstly stores the training dataset and wait until it receives the test dataset. In Lazy learner case, classification is done on the basis of the most related data stored in the training dataset. It takes less time in training but more time for predictions.
   Example: K-NN algorithm, Case-based reasoning
2. **Eager Learners:** Eager Learners develop a classification model based on a training dataset before receiving a test dataset. Opposite to Lazy learners, Eager learners take less time in training and more time in prediction. **Example:** Decision Trees, Naïve Bayes, ANN.

Transfer learning
related tasks. cross utilize the knowledge learnt
activa non gear se gear se car seekhna

use in ml of tl:
use of pretrained models, instead of building from scratch since no enough data

idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones.

reuse the output for a first model as the starting pt for second model. Modify or add new layers on top of the pretrained model, which are specific to the target task. this is called fine tuning

eg. inception v3 model.

challenges:
domain mismatch needs domain adaptation (reduce the distribution shift between athe source and target domains by aligning them)
too much fine tuning can result in overfitting
complexity

---

algo
eval
opti

supervised/inductive ml

semi supervised:
teacher provides a few specific examples and explanations for a concept, but the majority of your learning is left for you to explore and understand on your own.

bias= prob/expectation of error

overfits. learns training data too well
so much so that it also learns noise. complicated model (#features too many) + less data. overcome: regularisation (l1 & l2) / k fold cross validation (k-CV) / ensemble.

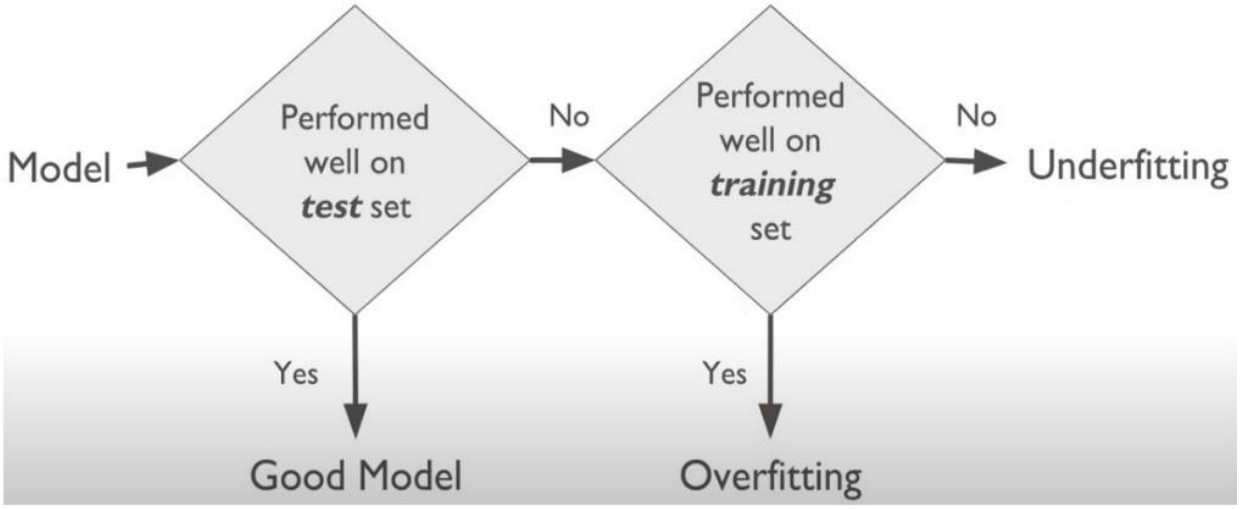
sensitivity to changes in data: variability

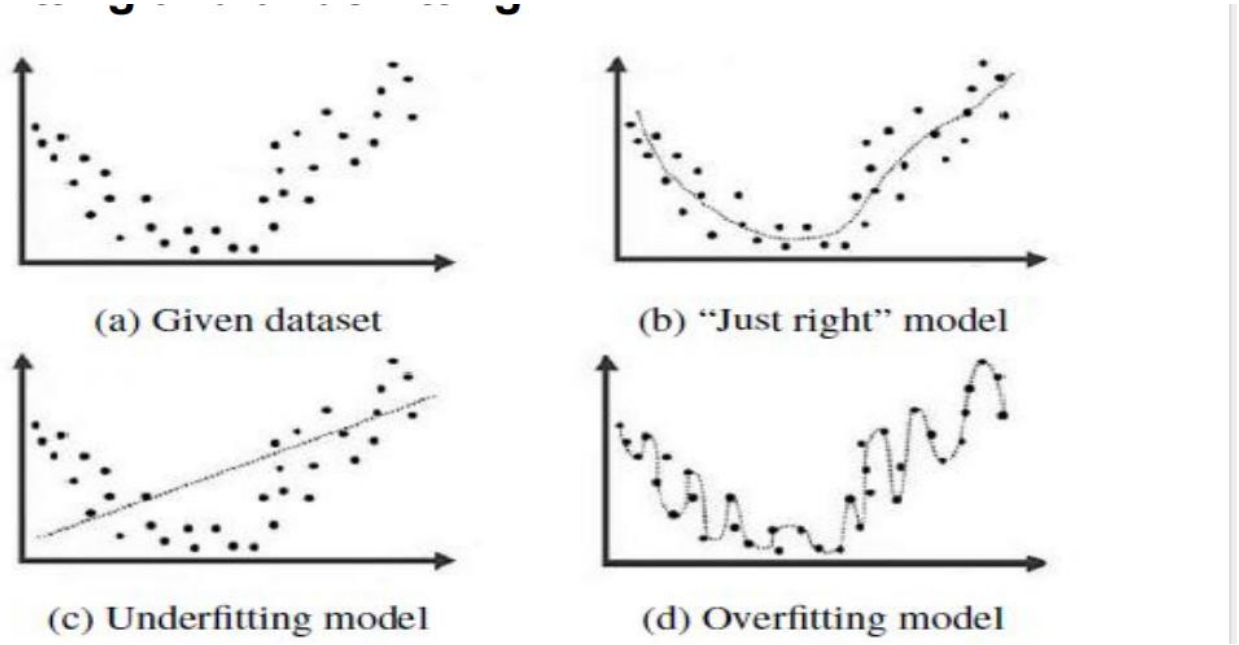
training error. error on training dataset
generalization error. error on test dataset.

he validation dataset is used during the model development phase. It helps you fine-tune hyperparameters and evaluate the model's performance during training. You can adjust the model based on the performance on the validation data.

Testing Dataset: It is used to assess the model's generalization performance. The generalization error is calculated by evaluating the model on this separate testing dataset.

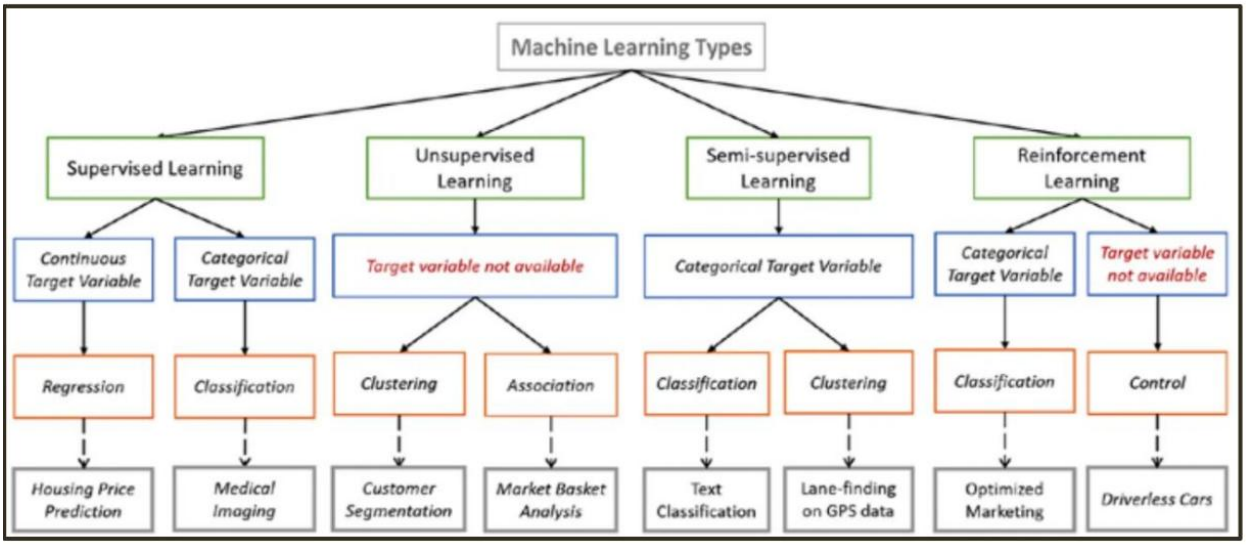
well posed learning problem: PET. PERF. EXP. TASKS

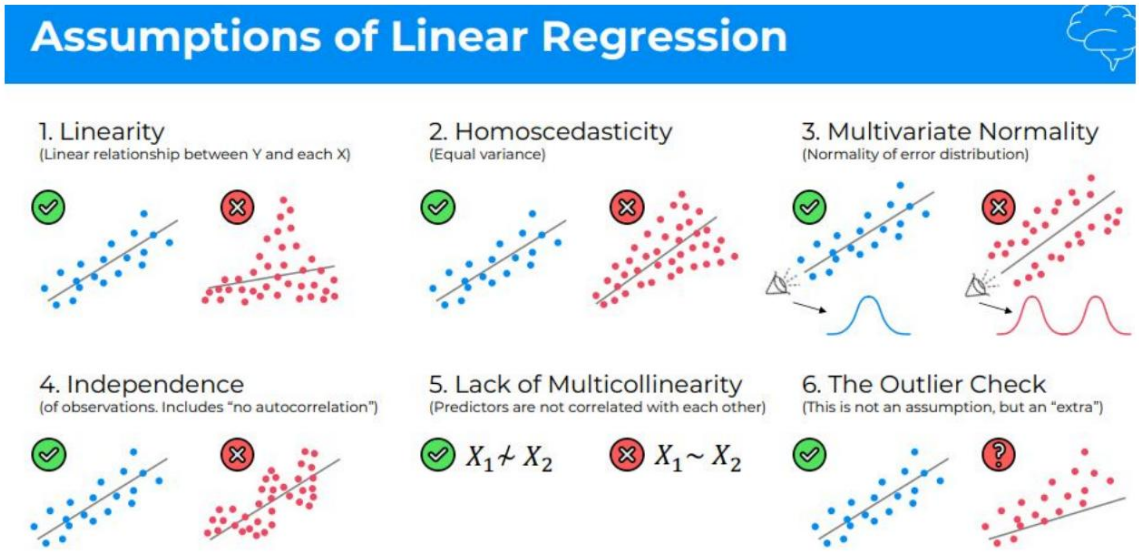


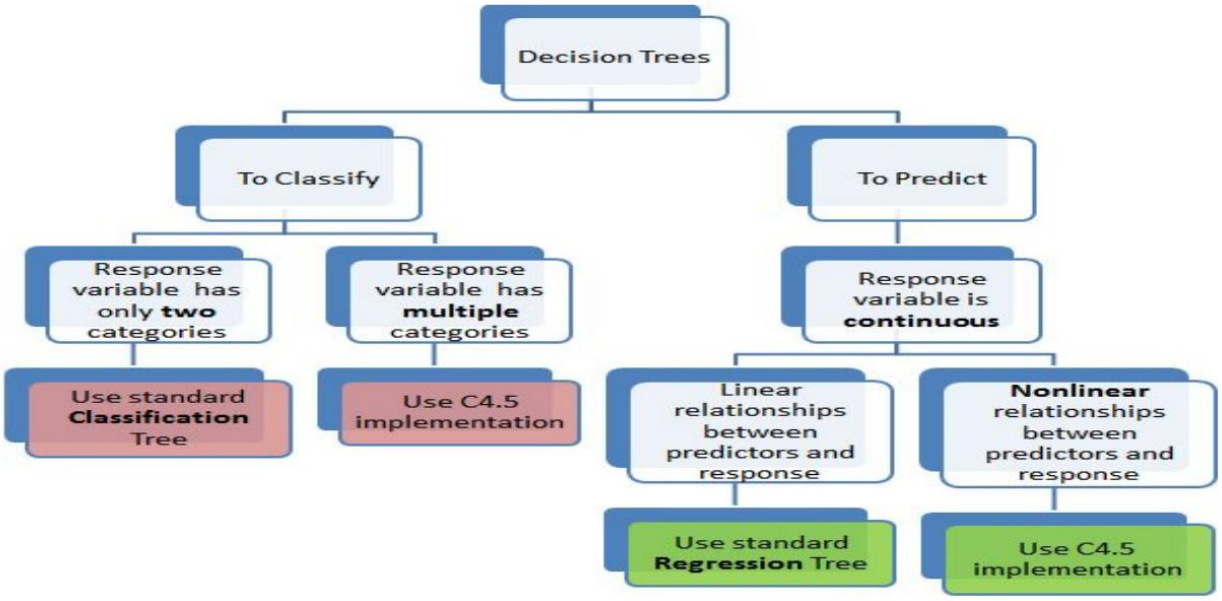


Precision is a useful metric in cases where False Positive is a higher concern than False Negatives.

Recall is a useful metric in cases where False Negative matters more than False Positive.







algo
eval
opti

supervised/inductive ml

semi supervised:
teacher provides a few specific examples and explanations for a concept, but the majority of your learning is left for you to explore and understand on your own.

bias= prob/expectation of error

overfits. learns training data too well
so much so that it also learns noise. complicated model (#features too many) + less data. overcome: regularisation (l1 & l2) / k fold cross validation (k-CV) / ensemble.

sensitivity to changes in data: variability

training error. error on training dataset
generalization error. error on test dataset.

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

he validation dataset is used during the model development phase. It helps you fine-tune hyperparameters and evaluate the model's performance during training. You can adjust the model based on the performance on the validation data.

Testing Dataset: It is used to assess the model's generalization performance. The generalization error is calculated by evaluating the model on this separate testing dataset.

well posed learning problem: PET. PERF. EXP. TASKS

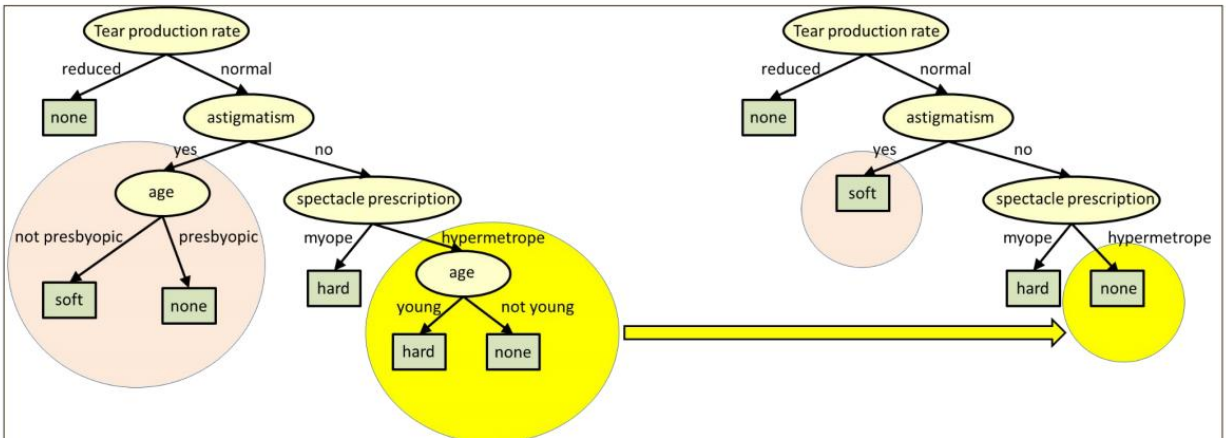
fp type1
fn type2

So, C4.5 algorithm solves most of problems in ID3. The algorithm uses gain ratios instead of gains. In this way, it creates more generalized trees and not to fall into overfitting. Moreover, the algorithm transforms continuous attributes to nominal ones based on gain maximization and in this way it can handle continuous data. Additionally, it can ignore instances including missing data and handle missing dataset. On the other hand, both ID3 and C4.5 requires high CPU and memory demand. Besides, most of authorities think decision tree algorithms in data mining field instead of machine learning.

Features	*ID3*	*C4.5*	*CART*
Type of data	Categorical	Continuous and Categorical	continuous and nominal attributes data
Speed	Low	Faster than ID3	Average
Boosting	Not supported	Not supported	Supported
Pruning	No	Pre-pruning	Post pruning
Missing Values	Can't deal with	Can't deal with	Can deal with
Formula	Use information entropy and information Gain	Use split info and gain ratio	Use Gini diversity index

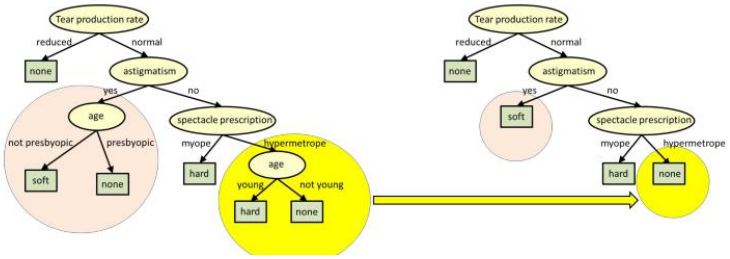
Pruning by Information Gain

- to prune out portions of the tree that result **in the least information gain**. This procedure does not require any additional data, and only bases the pruning on the information that is already computed when the tree is being built from training data.
- The process of IG-based pruning requires us to identify “twigs”, nodes whose children are all leaves. “Pruning” a twig removes all of the leaves which are the children of the twig, and makes the twig a leaf. The figure below illustrates this.



The algorithm for pruning is as follows:

1. Catalog all twigs in the tree
2. Count the total number of leaves in the tree.
3. While the number of leaves in the tree exceeds the desired number:
    1. Find the twig with the least Information Gain
    2. Remove all child nodes of the twig.
    3. Relabel twig as a leaf.
    4. Update the leaf count.



gmm uses em

A Gaussian Mixture Model (GMM) is a probabilistic model that describes data as a combination of multiple Gaussian distributions. It incorporates essential terminologies such as "mixture model," where data is a weighted sum of Gaussians. Each Gaussian has parameters like mean and variance. GMM is used for clustering or density estimation, assigning data points to clusters based on component probabilities. The Expectation-Maximization (EM) algorithm iteratively estimates Gaussian parameters from the data,

HMMs are particularly useful for modeling and analyzing sequences of data, state transition prob. HMMs assume the Markov property, which means that the probability of transitioning to a

future state depends only on the current state and not on past states. This is also known as the "memoryless" property.
At each state, the system emits an observation (data point) based on a probability distribution associated with that state. These observations may not directly reveal the underlying state, hence the term "hidden." HMMs have parameters that describe state transition probabilities and emission probabilities. These parameters are learned from data using algorithms like the Baum-Welch or Expectation-Maximization (EM).

The Slack Variable helps to define 3 types of data points:

- if $\xi = 0$ then the corresponding point ξ is on the margin or further away.
- if $0 < \xi < 1$ then the point ξ is within the margin and classified correctly (Correct side of the hyperplane).
- If $\xi \geq 1$ then the point is misclassified and present at the wrong side of the hyperplane.

The ξ is the misclassification penalty. Hence we want to minimize it during optimization.

Learners in Classification Problems

In the classification problems, there are two types of learners:

1. **Lazy Learners:** Lazy Learner firstly stores the training dataset and wait until it receives the test dataset. In Lazy learner case, classification is done on the basis of the most related data stored in the training dataset. It takes less time in training but more time for predictions.
   Example: K-NN algorithm, Case-based reasoning
2. **Eager Learners:** Eager Learners develop a classification model based on a training dataset before receiving a test dataset. Opposite to Lazy learners, Eager learners take less time in training and more time in prediction. **Example:** Decision Trees, Naïve Bayes, ANN.

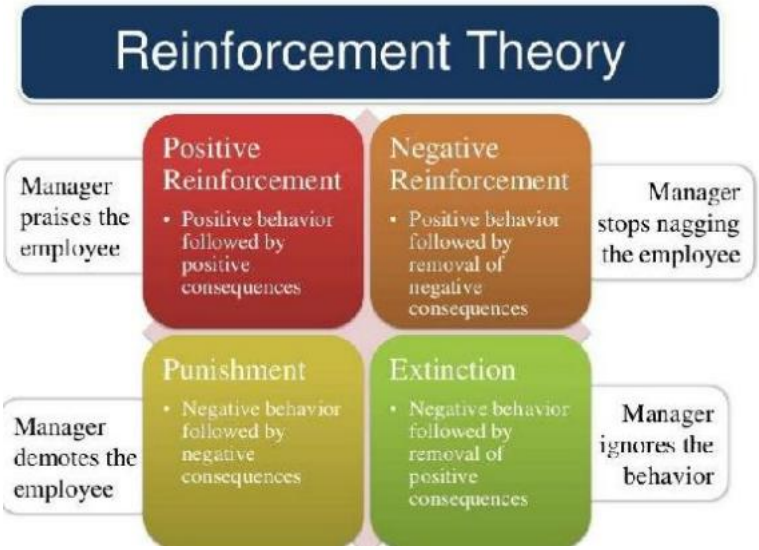
diversification introduced using el

All of these greedy rules work:

1. Add edges in increasing weight, skipping those whose addition would create a cycle. (**Kruskal's Algorithm**)
2. Run `TreeGrowing` starting with any root node, adding the frontier edge with the smallest weight. (**Prim's Algorithm**)
3. Start with all edges, remove them in decreasing order of weight, skipping those whose removal would disconnect the graph. (**"Reverse-Delete" Algorithm**)

---

kruskal allows non-adj



Feedback based ml: rl

Policy is agents behaviour.. what map it follows

Chess game, the players only focus on the current state and do not need to remember past actions or states.

-markov

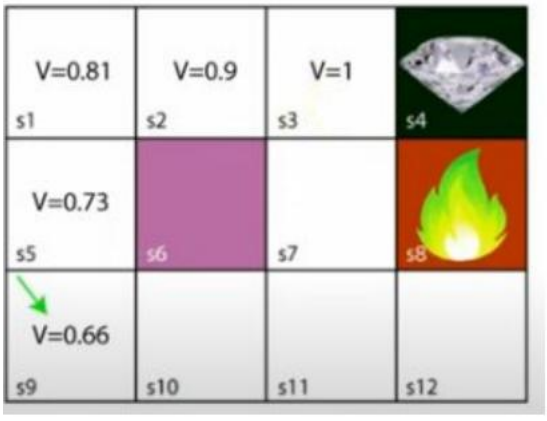
- Partial observability: agent indirectly observes environment:
  - A robot with camera vision isn't told its absolute location
  - A trading agent only observes current prices
  - A poker playing agent only observes public cards
- Now agent state $\neq$ environment state
- Formally this is a partially observable Markov decision process (POMDP)
- Agent must construct its own state representation S_t^a, e.g.

MDP (Bellman equation)

- The Bellman equation can be written as:

$$V(s) = \max[R(s,a) + \gamma V(s')]$$

- Where,
- $V(s) =$ *value calculated at a particular point.*
- $R(s,a) =$ *Reward at a particular state s by performing an action*
- $\gamma =$ *Discount factor*
- $V(s') =$ *The value at the previous state.*



rule based classifier: rule satisfication and coverage

rule pruning
rule extraction from dt. if then and or

ordered rule set= decision list (rule based ordering)
ranked wrt priority
default/ fallback rule for tuples not satisfying any rule,, fired when no other rule satisfied. evaluated at end

class based ordering . all the rules for
the most prevalent (or most frequent) class come first, the rules for the next prevalent class come next, and so on

size ordering (size of rule) scheme assigns the highest priority to the triggering rule that has the "toughest" requirements, where toughness is measured by the rule antecedent size. That is, the triggering rule with the most attribute tests is fired.

a record may trigger more than one rule. hence some ordering needed. not mutually exclusive

a record may not trigger any rule so default rule is needed. non exhaustive hua toh

for a rule to match, only lhs has to be same

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	**Single**	125K	**No**
2	No	Married	100K	**No**
3	No	**Single**	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	**Single**	85K	**Yes**
9	No	Married	75K	**No**
10	No	**Single**	90K	**Yes**

(Status = Single) → No

Coverage = 40%, Accuracy = 50%