(Q1) In logistic regression, what is the logistic function (sigmoid function) and how is it used to compute probabilities?

It is crucial for in transforming the linear combination of input features into probabilities between 0 and 1. (because it deals with binary classification tasks, e.g., 0 /1, spam / not spam). Computing Probabilities with Logistic Regression:

$f(y)=1/1+e^{-y}$

$y=b_0+b_1x$

After substitution formula will be: **$f(y)=1/1+e^{-(b0+b1x)}$**

(Q2) When constructing a decision tree, what criterion is commonly used to split nodes, and how is it calculated?

The two criterions used for best splitting nodes in decision tree are: Information Gain and Entropy

Information Gain: Measures the decrease in uncertainty/impurity in features If, IG is high purity is high

Formula: **$IG=Entropy(s)-sum(s_1/s) * entropy(s_1)$**

Entropy: Measure the uncertainty/impurity in features

If, Entropy is high purity is low

Formula: **$Entropy=-(sum(p_i log_2(p_i)))$**

(Q3) Explain the concept of entropy and information gain in the context of decision tree construction?

Aim: Increasing homogeneous groups based on their features.

Entropy: It measures the impurity or uncertainty within a node

If, Entropy is high purity is low

Formula: **$Entropy=-(sum(p_i log_2(p_i)))$**

Information Gain: It measures the reduction in uncertainty by splitting a node on a particular feature.

If, IG is high purity is high

Formula: **$IG=Entropy(s)-sum(s_1/s) * entropy(s_1)$**

(Q4) How does the random forest algorithm utilize bagging and feature randomization to improve classification accuracy?

Random forests improve classification accuracy by two techniques: bagging and feature randomization.

Bagging (Bootstrap Aggregation):

- Bagging also called as Bootstrap Aggregation.
- It involves creating multiple decision trees.
- A bootstrap sample is a random sample with replacement (some data points may appear multiple times while others are omitted)
- The diversity in training data leads to individual trees with different biases and strengths. (it reduces the variance of the model and becomes less prone to overfitting. Hence, significantly improve accuracy on unseen data/new data.)

Feature Randomization:

- Random subset of features is considered for splitting.
- Different trees may use different features to arrive at the same decision (further adding diversity to the ensemble).
- Improve accuracy by preventing any **single feature from dominating the decision-making process** (it reduces the model's reliance on noisy or irrelevant features and leading to better performance on unseen data).

(Q5) What distance metric is typically used in k-nearest neighbors (KNN) classification, and how does it impact the algorithm's performance?

In KNN classification, the choice of distance metric significantly impacts the algorithm's performance. There are several commonly used distance metrics. They are:

Euclidean Distance:

Measures the straight-line distance between two data points in feature space.
Impact: Works well for data with uniform feature scales. However, it can be sensitive to features with vastly different scales or units, where features with larger scales dominate the distance calculation.

Formula: $\mathbf{d= \sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}}$ **Manhattan Distance:**

Impact: Less sensitive to outliers and features with different scales compared to Euclidean distance. However, it can be less effective for data with highly correlated features.

Formula: $\mathbf{d=|X_1 - X_2| + |Y_1 - Y_2|}$ **Minkowski Distance:**

Generalization of Euclidean and Manhattan distances, where the power p determines the importance of larger differences.
Impact: Allows for more flexibility in distance calculation. For example, p = 1 becomes Manhattan distance, p = 2 becomes Euclidean distance.

Mahalanobis Distance:

Takes into account the correlations between features, making it suitable for non- spherical data distributions.
Impact: More accurate for data with complex relationships between features. However, it can be computationally expensive and requires more information about the data distribution.

(Q6) Describe the Naïve-Bayes assumption of feature independence and its implications for classification?

The Naive Bayes classifier is a popular machine learning algorithm for classification tasks.
It works by applying Bayes' theorem and simplifying the computation by assuming **feature independence**
Implications of the Feature Independence Assumptions:

- **Simplified Computation:** efficient calculation of probabilities, making Naive Bayes a computationally inexpensive algorithm.
- **Robustness to Missing Values:** Naive Bayes can handle missing values in individual features without significant performance degradation, as it treats them as independent.
- **Scalability to High Dimensions:** Work effectively with large numbers of features, making it suitable for high-dimensional dataset.

(Q7) In SVMs, what is the role of the kernel function, and what are some commonly used kernel functions?

In SVM, the kernel function plays a crucial role in enabling them to handle **non-linearly separable data**.
It acts as a bridge between the original input data space and a higher-dimensional feature space where the data becomes linearly separable Role of Kernel Function:

- Takes two input data points.
- Applies a mathematical operation on them.
- Outputs a value that measures their **similarity** in the **higher-dimensional feature space**.

Commonly used kernel functions:

- **Linear kernel:** The simplest kernel, suitable for linearly separable data.
- **Polynomial kernel:** Creates higher-order polynomial features from the original data, useful for capturing complex non-linear relationships.
- **Radial Basis Function (RBF kernel):** A popular choice, uses a Gaussian function to measure similarity, making it flexible for different data distributions.
- **Sigmoid kernel:** Similar to RBF, but with a different activation function, suitable for specific cases.

(Q8) Discuss the bias-variance trade off in the context of model complexity and overfitting?

The bias-variance trade off deals with the delicate balance between **model complexity** and **generalizability**.

It essentially states that we can't have a model that is both perfectly accurate on the training data (low variance) and perfectly generalizable to unseen data (low bias) at the same time.

Bias: Bias refers to the **systematic underfitting** of a model.

A high bias model makes simplifying assumptions and fails to capture the true complexities of the data. (This leads to **underestimating** the true relationship between the features and the target variable, resulting in consistently inaccurate predictions across all data, both training and unseen).

Variance: Variance refers to the **sensitivity of a model to the training data**.

A high variance model is very flexible and can fit the training data closely, potentially even memorizing noise and irrelevant details. (However, this ability to fit the training data tightly comes at the cost of **overfitting**, meaning the model performs well on the training data but fails to generalize to unseen data).

(Q9) How does TensorFlow facilitate the creation and training of neural networks?

TensorFlow offers several powerful features that make it a popular choice for creating and training neural networks:

1.**Ease of Use**:

- High-level API
- Automatic differentiation

2.**Flexibility and Customization**:

- Low-level API/High API on our need
- Wide range of supported models (CNN, RNN etc..,)

3.**Scalability and Efficiency**:

- Tensor Board
- TensorFlow Lite

4.**Community and Resources**:

- Large community
- Open-source

(Q10) Explain the concept of cross-validation and its importance in evaluating model performance?

Cross-validation is used to **evaluate the performance** of a model and **prevent overfitting**.

It involves splitting your data into multiple sets and using them in a specific way to assess the model's generalization ability.

Working:

1. **Data Split:** Divide your dataset into **folds** (typically 5 or 10).
2. **Train-Test Split:** For each fold:
   - Use **k-1 folds** as the **training set** to train your model.
   - Use the remaining **1 fold** as the **testing set** to evaluate the model's performance.
3. **Repeat:** Repeat steps 1 & 2 for all folds.
4. **Evaluation:** Calculate a **performance metric** (e.g., accuracy, precision, F1-score) for each fold.
5. **Average:** Take the **average** of the performance metrics across all folds to get an **overall estimate** of the model's performance.

Importance of cross-validation:

- Prevents overfitting
- Provides a more realistic estimate
- Enables comparing models

(Q11) What techniques can be employed to handle overfitting in machine learning models?

Overfitting is a model memorizes the training data too well when a model becomes too focused on the specific details of the training data (leading to poor performance on unseen data).

Techniques employ to handle overfitting:

- Increase Data Size
- Data Cleaning and Preprocessing
- Choose simpler models like decision trees or linear regression after that gradually increase the complexity of model

(Q12) What is the purpose of regularization in machine learning, and how does it work?

Regularization **aim** is to prevent overfitting and improve the generalizability of the model.

It helps avoid overfitting by introducing penalties or constraints that discourage the model from becoming overly complex or fitting the training data too closely.

Regularization works by adding a penalty or complexity term to the complex model.

$$\textbf{Regularized Cost} = \textbf{Original Cost} + \lambda \times \textbf{Regularization Term}$$

Here, ○ original cost function (which measures how well the model fits the training data) ○ regularization term (which penalizes complex models) ○ λ is the regularization parameter, which controls the strength of the regularization.

Note: A higher λ value leads to stronger regularization and also the choice of the λ is crucial, and it is often determined through techniques like cross-validation.

Types:

L1 Regularization (Lasso):

- It adds the absolute values of the coefficients as a penalty term to the cost function.
- The regularization term is proportional to the sum of the absolute values of the model parameters.
- It tends to produce sparse models, meaning it encourages some of the coefficients to become exactly zero, effectively performing feature selection.

L2 Regularization (Ridge):

- It adds the squared values of the coefficients as a penalty term to the cost function.
- The regularization term is proportional to the sum of the squared values of the model parameters.

(Q13) Describe the role of hyper-parameters in machine learning models and how they are tuned for optimal performance?

Hyperparameters are like the knobs and dials of the model.

These are external settings that control the learning process and model complexity, ultimately impacting its performance. Unlike parameters, which are learned from the data during training, hyperparameters are set before training begins.

Examples:

- **Learning rate**: Controls the step size taken during gradient descent optimization in algorithms like linear regression and neural networks.
- **Number of hidden layers and neurons in neural networks**: Affects the model's capacity to learn complex non-linear relationships.
- **Regularization parameters**: (e.g., L1/L2 regularization strength) Control the model's complexity to prevent overfitting.

The Role of Hyperparameters:

- Controlling Model Complexity
- Tuning Performance

(Q14) What are precision and recall, and how do they differ from accuracy in classification evaluation?

Precision and recall are two key metrics used to evaluate the performance of classification models

Precision:

- Definition: Measures the proportion of positive predictions that are actually correct.
- Interpretation: Answers the question: "Out of all the instances the model classifies as positive, how many are truly positive?"
- Useful when: Dealing with imbalanced datasets (where one class is much smaller than the other) or when false positives have high costs. Recall:
- Definition: Measures the proportion of actual positive instances that are correctly identified by the model.
- Interpretation: Answers the question: "Out of all the truly positive instances, how many did the model correctly identify as positive?"
- Useful when: It's crucial to identify all true positives, even if it means some false positives occur.

For example, Imagine a model classifying emails as spam or not spam.

- High accuracy (80%) but low precision (60%): The model correctly classifies 80% of emails, but out of all emails it identifies as spam, only 60% are actually spam (many false positives).
- High recall (90%) but low precision (40%): The model identifies 90% of all spam emails correctly, but it also classifies many non-spam emails as spam (high false positive rate).

(Q15) Explain the ROC curve and how it is used to visualize the performance of binary classifiers?

The Receiver Operating Characteristic (ROC) curve is a valuable tool for evaluating the performance of binary classifiers.

It provides a visual representation of the trade-off between true positive rate (TPR) and false positive rate (FPR).

Components:

- True Positive Rate: The proportion of actual positive cases correctly identified as positive.
  TPR= True Positives/True Positives + False Negatives
- False Positive Rate: The proportion of actual negative cases incorrectly identified as positive.
  FPR= False Positives/False Positives + True Negatives
- Threshold Variation: The ROC curve is created by varying the decision threshold of the classifier, which determines how confident the model needs to be before classifying an instance as positive. By adjusting this threshold, you can generate different TPR and FPR values.
- Plotting the Curve: For each threshold setting, record the corresponding TPR and FPR values. The ROC curve is a plot of TPR against FPR. Typically, the x-axis represents the FPR, and the y-axis represents the TPR.
- AUC-ROC (Area Under the ROC Curve): The AUC-ROC is a single metric that summarizes the overall performance of the classifier. It represents the area under the ROC curve and ranges from 0 to 1. A model with an AUC-ROC of 0.5 indicates random performance, while a model with an AUC-ROC of 1.0 represents perfect performance.

Interpreting the ROC curve:

- Top-left corner (0,1): This point represents a perfect classifier with 100% sensitivity and 0% false positives.
- Bottom-right corner (1,0): This point represents a classifier that predicts all negatives correctly but fails to predict any positives.
- Diagonal line (random classifier): A random classifier would produce a diagonal line from the bottom-left to the top-right, and its AUC-ROC would be 0.5.