**Difference between supervised and unsupervised ML?**

In Supervised ML, the model learns from the training set and then it is tested on the test set. Classification algorithms like Logistic regression, decision tree etc. come under supervised ML. Scenarios – customer churn prediction, weather forecast

In Unsupervised ML, the model learns itself with experience , it does not require any training set. Clustering algorithms like K-Means, Hierarchical clustering etc. comes under unsupervised ML. Scenarios – Facial emotion recognition, sentiment analysis, review segregation

**What is overfitting? How do you prevent it in a ML model?**

Overfitting occurs when the model learns the training set too good and performs bad in the test set. In this scenario, we say we have, low bias and high variance.

We can avoid overfitting by using large datasets and do hyperparameter tuning.

Example:- In a decision tree, overfitting occurs when the tree is too deep. So as to avoid this we can do pruning, where we control the no. of levels of the tree.

**What is cross validation? How is it useful in evaluating ML models?**

Cross validation is a way to check if a machine learning model is overfitting or underfitting on a given dataset. It is a statistical technique used to evaluate and validate the performance of ML models. It involves dividing the dataset into multiple subsets and then using them iteratively to train and validate the model.

The process involves randomly dividing the dataset into k folds, with k being a no. chosen by the user. Then the model is trained on k folds and validated on the remaining fold. This process is repeated k times, with each fold being used for validation exactly once. The result of each fold is averaged to get a more reliable estimate of the model’s performance.

There are several methods used for cross validation.

* Holdout validation
* Leave-one -out validation
* K-fold cross validation
* Stratified k-fold cross validation

Best Practices for Cross-Validation

To get the most out of cross-validation, it is important to follow some best practices, including:

* Use a large dataset, if possible, to get a more accurate estimate of the model’s performance.
* Use a reasonable value for k, such as 5 or 10, that balances the computational cost with the accuracy of the estimate.
* Stratify the data if it contains imbalanced classes to ensure that each fold contains a representative sample of each class.
* Shuffle the data before splitting it into folds to avoid any biases in the data ordering.
* Use the same preprocessing steps for each fold to ensure consistency in the results.

**What evaluation metrics would you use for classification problems and why? What about for regression problems?**

For classification problems, the goal is to predict categorical labels(ex – spam or not spam). I would use Accuracy score, precision, recall, F1-score, ROC-AUC curve, confusion matrix depending on the importance of false positives and false negatives.

For regression problems, the goal is to predict continuous numerical values. I would use Mean Absolute error, Mean Squared error, Root Mean Squared error, R2 - squared error depending on whether the goal is to minimize the error size or to understand the model's explanatory power.

**How would you handle imbalanced classes in a classification problem?**

Handling imbalanced classes in a classification problem is important because models can become biased toward the majority class, leading to poor performance on the minority class. Various ways to handle is:

* Oversampling the minority class - This involves increasing the number of instances of the minority class by replicating samples or generating synthetic data points. (SMOTE) - Generates synthetic instances by interpolating between existing minority class samples. This can help create a more balanced dataset and reduce overfitting compared to simple duplication.

Random oversampling - This simply duplicates instances from the minority class to balance the dataset but may increase the risk of overfitting.

* Undersampling the majority class - This involves reducing the number of instances of the majority class to balance the dataset. While it may help balance the dataset, it also discards potentially valuable data.

Random undersampling - Randomly removes samples from the majority class. This can lead to loss of information and potentially reduce model performance.

* Adjusting class weights
* Use ensemble methods like balanced random forest, easyEnsemble – they are designed to handle imbalanced datasets.
* Focus on metrics that handle imbalance like precision, recall, f1-score, roc-auc, confusion matrix
* Collect more data on minority class

**Explain what ROC-AUC is and how to interpret it.**

The ROC helps us in distinguishing between the two classes of a classification model.

It lets us visualize the model’s performance at all possible thresholds, not just a single one

As if we,

* If you set a high threshold, say 95%, the model will only mark obvious spam emails, ensuring your important emails don’t accidentally get flagged. However, this might mean some less-obvious spam emails might slip through into your main inbox.
* On the other hand, if you set a *lower threshold*, say 50%, the filter will catch a broader range of spam emails, but there’s also a higher chance it might incorrectly classify a legitimate email as spam.

In this scenario, the threshold determines the balance between ensuring you don’t miss important emails (False Positives) and ensuring spam doesn’t clutter your main inbox (True Positives). Adjusting this threshold affects the balance, representing the trade-off in decision-making.

How to create a ROC curve

1. Predict Probabilities: When you ask a classification model whether something is “spam” or “not spam”, it doesn’t simply guess. Internally, the model calculates a probability.
2. Test Different Thresholds: If the model directly classifies based on its internal score, we’d have to set a threshold. For instance, you might say “If the score is above 0.8, call it spam.” But is 0.8 the right threshold? What if 0.7 is better? Or 0.9?
3. Track TPR and FPR: TPR (True Positive Rate) TPR=TP/(TP+FN) is about of all actual spam emails, how many did we correctly call “spam.” FPR (False Positive Rate) FPR=FP/(FP+TN)  is about of all actual “not spam” emails, how many did we wrongly call “spam”?
4. Plot: For each threshold, plot TPR on the y-axis and FPR on the x-axis. Connect the dots to form a curve.

How to interpret it:

1. Top-Left Corner is Best: If the curve is closer to the top-left corner, that’s great! It means we’re correctly identifying spam without mislabeling many genuine emails. (Maximum True Positives, minimum false positives)
2. Above the Diagonal Line: If the curve is above the diagonal line (from bottom-left to top-right), our model is better than just random guessing because the diagonal line means where TPR = FPR.
3. Area Under Curve (AUC): The bigger the area under the curve (closer to 1), the better our model is.

What is AUC?

AUC, which stands for “Area Under the ROC Curve,” quantifies a classifier’s performance. The ROC Curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) over various thresholds. Instead of assessing a model through the ROC curve visually, AUC summarizes it into a single numerical value, with a higher AUC indicating superior model performance.

Why is AUC Important?

1. Comparison of Models: A single AUC value offers a quick comparison for multiple models. The one with a higher AUC generally excels in classification when evaluated on the same dataset.
2. Effective with Imbalanced Datasets: In datasets where one class significantly dominates, metrics like accuracy might not provide an accurate picture. AUC, which incorporates both TPR and FPR, serves as a more reliable metric in such instances.

Interpreting AUC Values

* AUC = 0.5: This implies that the classifier’s performance is equivalent to random guessing (akin to a coin toss). It corresponds to the diagonal line from (0,0) to (1,1) on the ROC plot.
* 0.5 < AUC < 1: Indicates that the classifier outperforms random guessing. The closer the value is to 1, the better the classifier is at differentiating between positive and negative instances.
* AUC = 1: A perfect classifier that flawlessly distinguishes between positive and negative classes. It’s a theoretical value and seldom achieved in real-world applications.

A diagram of a positive and negative rate

Description automatically generated

Reinforcement learning

Bias and variance tradeoff

Precision and recall tradeoff

Feature selection

Feature scaling

Logistic Regression

Decision tree

Random forest

XGBoost

K-Means

**Data visualization :-**

Matplotlib - Bar plot, Line chart, pie chart, scatter plot, hist plot

Seaborn - Count plot, Box plot, line chart, bar plot, Pair plot, heat map, hist plot, scatter plot

Plotly – heat map, 3d line plot, 3d scatter plot

Bokeh - scatter plot, hist plot, box plot, bar plot, line plot