

EVALUATION METRICS

TEAM MEMBERS

**ABHIJITH
ANN MARYA
ANSIL
ILLIYAS
MUHAMMED ANSHEER
SHERIN
YADU**

1. Evaluation Metrics

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

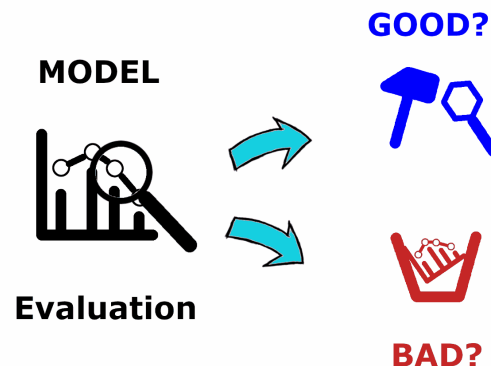
1.1 Loss function vs metrics

Metrics are different from loss functions.

A loss function, is more like an “error” function that calculates how far apart the output/predicted value of a learning function deviates/differs from the ground truth/actual value. The main focus is usually “**optimization (can either be a minimization or maximization problem)**”.

- “How far/close are we from the target destination? Are we moving in the right direction?”
- They're used to train a machine learning model (using some kind of optimization like Gradient Descent), and they're usually differentiable in the model's parameters.

An Evaluation Metric, also known as a “**Criterion**” is a method of evaluating/comparing the performance of a learning function. It's like a quantifier with which you can judge if a learning function for a given learning problem (or hypotheses class) is good or bad, based on standards.



NB: “A Higher value does not always indicate a good performance, and lower value does not always indicate a bad performance”

- Metrics are used to monitor and measure the performance of a model (during training and testing), and don't need to be differentiable.

However, if, for some tasks, the performance metric is differentiable, it can also be used as a loss function (perhaps with some regularizations added to it), such as MSE.

Key Differences

- A loss function is implemented during training to optimize a learning function. It is not a judge of overall performance.
- A Criterion/Evaluation Metric is used after training to measure overall performance.

Interesting Fact:

A confusion Matrix is neither a loss function nor an evaluation metrics. It's simply a diagnostic tool that maps the type of errors a given learning function is making or inclined towards (How many False Positives Versus False Negatives)

1.2 Libraries for model evaluation

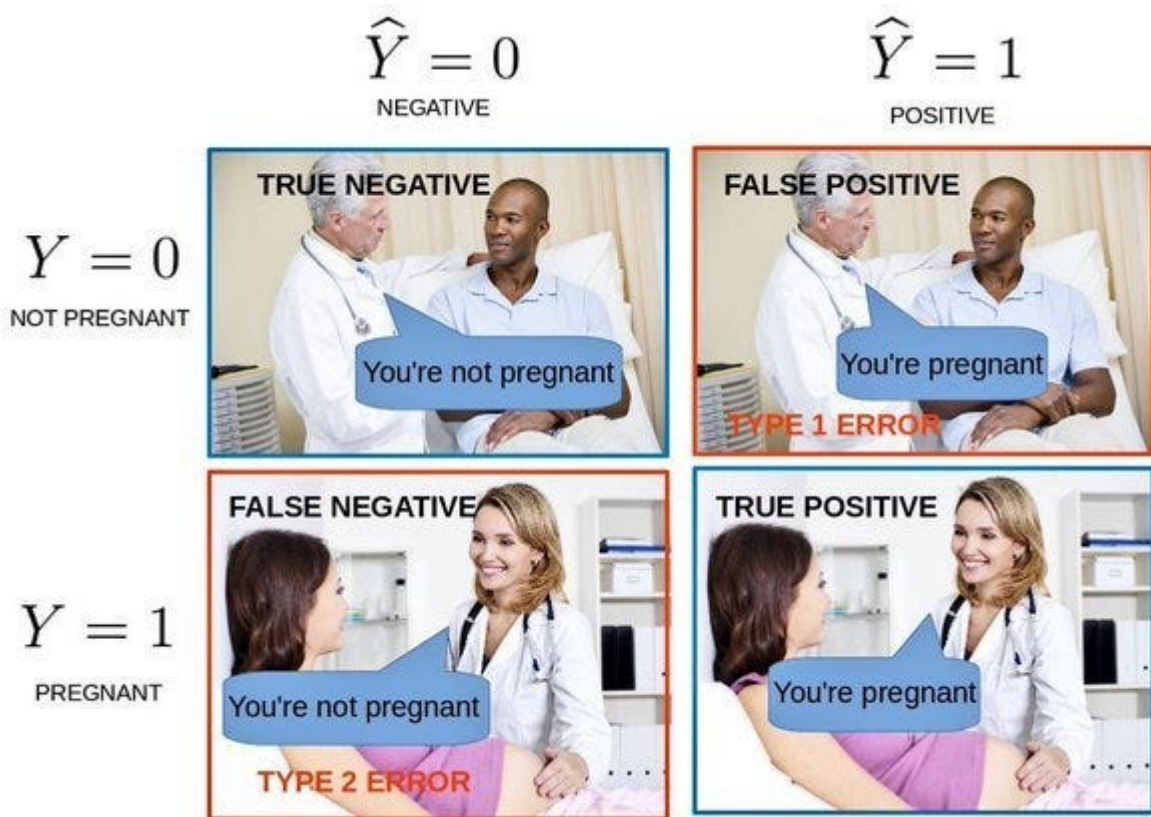
- [tf.keras.metrics](https://www.tensorflow.org/api_guides/python/keras_metrics)
- [Metrics and scoring: quantifying the quality of predictions](#)

2 Evaluation metrics for classification

Classification is about predicting the class labels given input data. There are many ways for measuring classification performance. Accuracy, confusion matrix, log-loss, and AUC-ROC are some of the most popular metrics.

For that we need the concept of [confusion matrix](#)

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative



2.1 Accuracy

2.1.1 What is Accuracy:

Accuracy is a fundamental evaluation metric used in machine learning to measure the performance of a classification model. It represents the ratio of correctly predicted instances to the total instances in the dataset.

2.1.2 How Accuracy is Calculated:

The accuracy of a model is calculated using the following formula:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100$$

In the context of a confusion matrix:

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{Total Population}} \times 100$$

2.1.3 Range of Values:

Accuracy values range from 0 to 1, or from 0% to 100%. A value of 1 (or 100%) indicates that all predictions are correct, while lower values indicate a higher number of misclassifications.

2.1.4 Libraries and Modules Used:

- [sklearn.metrics.accuracy_score](#)
- [tf.keras.metrics.Accuracy](#)

```
1 from sklearn.metrics import accuracy_score
2 accuracy_score(y_true, y_pred)
```

2.1.5 When to Use Accuracy:

Accuracy is suitable when:

- The classes are roughly balanced (i.e., similar number of instances in each class) and not skewed or No class imbalance
- Misclassifications of both positive and negative classes have similar consequences.

2.1.6 Applications of Accuracy:

Accuracy is used in various classification tasks, such as:

- Identifying spam emails (binary classification).
- Medical diagnoses (binary or multiclass classification).
- Sentiment analysis (binary or multiclass classification).

2.1.7 Advantages of Accuracy:

- **Simplicity:** It is easy to understand and compute.
- **Intuitive Interpretation:** It provides a straightforward measure of the model's overall performance.
- **Suitability for Balanced Classes:** When classes have similar sizes, accuracy can be a reliable indicator of performance.

2.1.8 Disadvantages of Accuracy:

- **Class Imbalance:** Accuracy can be misleading when classes have significant imbalances, as the model might predict the majority class excessively.
- **Misleading in Skewed Datasets:** In cases where one class is rare, a high accuracy could still lead to a poor model.
- **Unequal Misclassification Costs:** It treats false positives and false negatives equally, even if their consequences differ.

2.1.9 Relevant Reference Links:

- [Accuracy, Medium](#)

2.2 Precision:

2.2.1 What is it?

Precision evaluates how precise a model is in predicting positive labels. It provides insight into the proportion of true positive predictions (correctly predicted positives) out of all instances predicted as positive.

2.2.2 How it is Calculated:

Precision is calculated using the following formula:

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}$$

2.2.3 Range of Values:

Precision values range from 0 to 1, or from 0% to 100%. A precision value of 1 (or 100%) signifies that all predicted positives are correct, while lower values indicate a higher number of false positive predictions.

2.2.4 Libraries and Modules Used:

- [sklearn.metrics.precision_score](#)
- [tf.keras.metrics.Precision](#)

```
1 from sklearn.metrics import precision_score
2 precision_score(y_true, y_pred)
```

2.2.5 When to Use Precision:

Precision is particularly useful when:

- The emphasis is on minimizing false positives, as in medical diagnoses where false positives could lead to unnecessary treatments.
- Class imbalance is present, and strict control over false positives is needed.

2.2.6 Applications of Precision:

- **Medical Diagnoses:** In medical settings, precision is essential to minimize false positive diagnoses, which can lead to unnecessary medical interventions or patient distress.
- **Credit Card Fraud Detection:** Reducing false positives is crucial to avoid inconveniencing legitimate cardholders while detecting fraudulent transactions.
- **Information Retrieval Systems:** In search engines, precision ensures that retrieved results are relevant to the user's query.

2.2.7 Advantages of Precision:

- **Class Imbalance Handling:** Precision excels in situations with imbalanced classes, allowing the model to focus on correctly predicting positives while controlling false positives.
- **Positive Predictive Value:** Precision reflects the positive predictive value, which is critical in domains where false positives have significant consequences.

2.2.8 Disadvantages of Precision:

- **Neglects False Negatives:** Precision doesn't consider false negatives, potentially leading to missed positive instances (low recall).
- **Cautious Model Behavior:** An overly cautious model might achieve high precision by making very few positive predictions, which can lead to missed opportunities.

2.2.9 Relevant Reference Links:

- [Precision and Recall in Machine Learning](#)
- [Understanding Accuracy, Precision, Recall, and F1 Score](#)

2.3 Recall (Sensitivity or True Positive Rate):

2.3.1 What is it?

Recall, also referred to as Sensitivity or True Positive Rate, is an essential evaluation metric in machine learning that emphasizes a model's ability to correctly identify positive instances. It quantifies the proportion of true positive predictions (correctly predicted positives) out of all actual positive instances.

Unlike precision that only comments on the correct positive predictions out of all positive predictions, recall provides an indication of missed positive predictions.

2.3.2 How it is Calculated:

Recall is calculated using the following formula:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

2.3.3 Range of Values:

Recall values span the range of 0 to 1, or equivalently, 0% to 100%. A recall value of 1 (or 100%) signifies that all actual positive instances are correctly predicted by the model, while lower values indicate that some positive instances were missed.

2.3.4 Libraries and Modules Used:

- [sklearn.metrics.recall_score](#)
- [tf.keras.metrics.Recall](#)

```
1 from sklearn.metrics import recall_score
2 recall_score(y_true, y_pred)
```

2.3.5 When to Use Recall:

Recall becomes particularly valuable when:

- The priority is on minimizing false negatives, such as in medical diagnoses where missing a positive case could have severe consequences.
- Dealing with imbalanced datasets, especially when the positive class is rare, and capturing all actual positives is of utmost importance.

2.3.6 Applications of Recall:

Recall finds application in multiple scenarios, including:

- **Medical Diagnoses:** In the medical field, recall ensures that the model identifies as many true positive cases as possible to avoid missing critical diagnoses.
- **Search and Retrieval Systems:** In information retrieval, recall guarantees that relevant documents are not overlooked when responding to user queries.

2.3.7 Advantages of Recall:

- **Sensitive to Missed Positives:** Recall is particularly sensitive to false negatives, making it a robust metric for identifying positive instances in imbalanced or critical situations.
- **Effective for Rare Classes:** Recall is well-suited for addressing class imbalance, especially when the positive class is rare and requires special attention.

2.3.8 Disadvantages of Recall:

- **Neglects False Positives:** Recall does not account for false positives, which could lead to an increase in incorrect positive predictions and reduced precision.
- **Trade-off with Precision:** Aiming for higher recall may lead to lower precision as the model becomes more cautious and predicts more positive instances.

2.3.9 Relevant Reference Links:

- [Understanding Accuracy, Precision, Recall, and F1 Score](#)
- [Precision and Recall in Machine Learning](#)

2.4 Specificity (True Negative Rate):

2.4.1 What is it?

Specificity, also referred to as the True Negative Rate, is an important evaluation metric in binary classification that measures a model's ability to correctly identify negative instances.

2.4.2 How it is Calculated:

Specificity is calculated using the following formula:

$$Specificity = \frac{True\ Negatives\ (TN)}{True\ Negatives\ (TN) + False\ Positives\ (FP)}$$

2.4.3 Range of Values:

Specificity values also range from 0 to 1, or from 0% to 100%. A Specificity value of 1 (or 100%) indicates perfect identification of negative instances, while lower values suggest a higher number of false positive predictions.

That is high specificity means that the model is correctly identifying most of the negative results, while a low specificity means that the model is mislabeling a lot of negative results as positive.

2.4.4 Libraries and Modules Used:

You can calculate Specificity using various programming languages and libraries. In Python, scikit-learn's confusion matrix and relevant functions can be used.

```
1 from sklearn.metrics import confusion_matrix
2 tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()
```

2.4.5 When to Use Specificity:

Specificity is particularly useful when:

- Emphasizing the importance of accurately identifying negative instances, especially in scenarios where false positives have significant consequences.
- Evaluating models in domains where the negative class is of particular interest.

2.4.6 Applications of Specificity:

- Medical testing, where accurately identifying healthy patients is crucial to avoid unnecessary treatments.
- Quality control processes, where identifying non-defective products is a priority.

2.4.7 Advantages of Specificity:

- **Focus on Negative Predictions:** Specificity provides insight into a model's ability to correctly predict negative instances, making it important in scenarios where avoiding false positives is crucial.

2.4.8 Disadvantages of Specificity:

- **Neglects Positive Predictions:** Specificity doesn't consider true positive predictions, potentially leading to an incomplete evaluation of the model's overall performance.

2.4.9 Relevant Reference Links:

- [Understanding Specificity, Sensitivity, and the ROC Curve](#)

2.5 F_β -Score:

2.5.1 What is it?

Fbeta-measure is a configurable single-score metric for evaluating a binary classification model based on the predictions made for the positive class.

Combines both precision and recall into a single value. It provides a balance between the two metrics and is particularly useful when there's a trade-off between precision and recall.

2.5.2 How it is Calculated:

$$F_{\beta} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

The F-measure is calculated as the harmonic mean of precision and recall, giving each the same weighting.

In general, there are three most common values for the beta parameter:

- **F0.5 score** (beta = 0.5): Such a beta makes a Precision value more important than a Recall one. In other words, it focuses on minimizing False Positives than minimizing False Negatives;
- **F1 score** (beta = 1): True harmonic mean of Precision and Recall. In the best-case scenario, if Precision and Recall are equal to 1, the F-1 score will also be equal to 1;

$$F_1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN}$$

- **F2 score** (beta = 2): Such a beta makes a Recall value more important than a Precision one. In other words, it focuses on minimizing False Negatives than minimizing False Positives.

2.5.3 Range of Values:

The F1-Score values range from 0 to 1, similar to precision and recall. A higher F1-Score indicates a better balance between precision and recall, with a perfect F1-Score of 1 representing a model that achieves both high precision and high recall.

2.5.4 Libraries and Modules Used:

- [`sklearn.metrics.fbeta_score`](#)
- [`sklearn.metrics.f1_score`](#)

```
1 from sklearn.metrics import fbeta_score
2 fbeta_score(y_true, y_pred, beta=0.5)
```

2.5.5 When to Use F1-Score:

The F1-Score is particularly useful when:

- There's an imbalance between precision and recall, and you seek a balanced evaluation metric.
- The cost of false positives and false negatives is significant and needs to be minimized equally.

2.5.6 Applications of F1-Score:

- Text classification tasks, where precision and recall need to be balanced.
- Disease detection, where both minimizing missed diagnoses and avoiding unnecessary treatments are important.

2.5.7 Advantages of F1-Score:

- **Balanced Metric:** The F1-Score offers a balanced perspective by combining both precision and recall, making it suitable when there's a trade-off between these metrics.
- **Addressing Imbalance:** The F1-Score is effective in handling class imbalance, ensuring that both false positives and false negatives are considered.

2.5.8 Disadvantages of F1-Score:

- **Influenced by Imbalance:** Like precision and recall, the F1-Score can also be affected by class imbalance, potentially leading to misleading results.

2.5.9 Relevant Reference Links:

- [Precision, Recall, and F1 Score](#)

2.6 Area Under the ROC Curve (AUC-ROC):

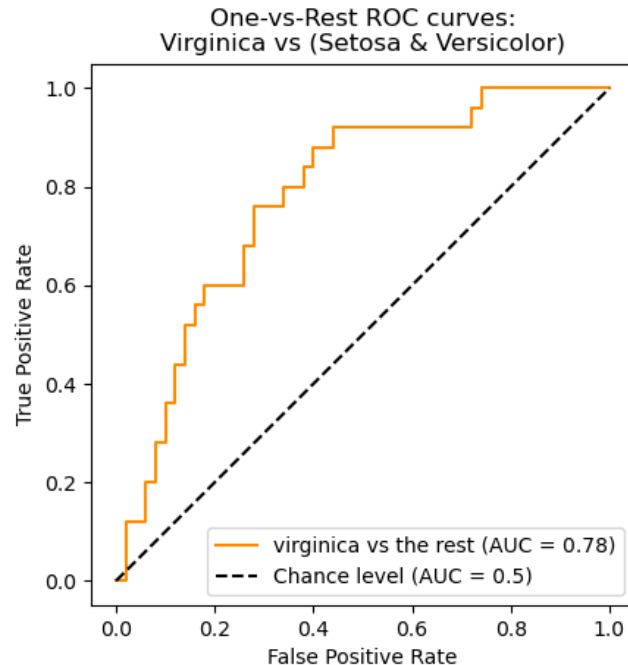
2.6.1 What is it?

The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) is a popular evaluation metric used for binary classification models. It measures the model's ability to distinguish between positive and negative classes across various probability thresholds.

It is a probability curve that plots the **TPR** against **FPR** at various threshold values and essentially **separates the 'signal' from the 'noise.'** In other words, it shows the performance of a classification model at all classification thresholds. The **Area Under the Curve (AUC)** is the measure of the ability of a binary classifier to distinguish between classes and is used as a summary of the ROC curve.

2.6.2 How it is Calculated:

The AUC-ROC is calculated by plotting the True Positive Rate (Recall) against the False Positive Rate (1 - Specificity) at different probability thresholds and computing the area under the resulting curve.



2.6.3 Range of Values:

The AUC-ROC values range from 0 to 1. A higher AUC-ROC indicates better discrimination between the positive and negative classes, with a value of 1 indicating perfect separation and 0.5 indicating random guessing.

2.6.4 Libraries and Modules Used:

- [sklearn.metrics.roc_auc_score](#)

```
1 from sklearn.metrics import roc_auc_score
2 roc_auc_score(y, y_pred)
```

2.6.5 When to Use AUC-ROC:

AUC-ROC is particularly useful when:

- You want to evaluate a model's performance across different probability thresholds.
- The class distribution is imbalanced, and the model needs to handle varying levels of false positives and false negatives.

2.6.6 Applications of AUC-ROC:

- Medical tests, where adjusting the threshold balances sensitivity and specificity.
- Fraud detection, where controlling false positives while catching as many fraud cases as possible is important.

2.6.7 Advantages of AUC-ROC:

- **Threshold Independence:** AUC-ROC considers the overall model performance across thresholds, providing a threshold-independent evaluation.
- **Imbalance Handling:** AUC-ROC is effective in handling class imbalance by assessing the model's ability to rank positive instances higher than negative instances.

2.6.8 Disadvantages of AUC-ROC:

- **Insensitive to Class Distribution:** AUC-ROC might not be suitable for highly imbalanced datasets where the negative class dominates, as it can still yield a high value even if the model's performance on the positive class is poor.

2.6.9 Relevant Reference Links:

- [Understanding ROC and AUC](#)

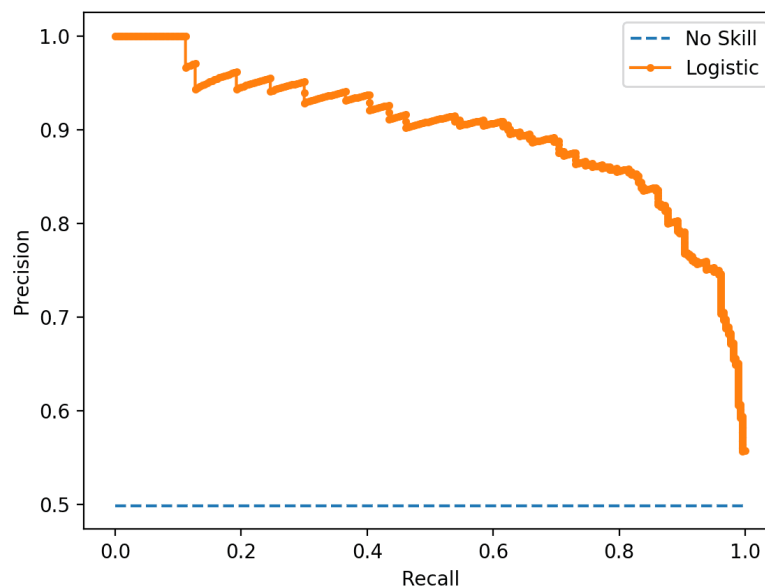
2.7 Area Under the Precision-Recall Curve (AUC-PR):

2.7.1 What is it?

The Area Under the Precision-Recall Curve (AUC-PR) is an evaluation metric used for binary classification models, particularly in scenarios where class imbalance exists. It measures the model's ability to balance precision and recall across different probability thresholds.

2.7.2 How it is Calculated:

The AUC-PR is calculated by plotting precision against recall at various probability thresholds and computing the area under the resulting curve.



2.7.3 Range of Values:

The AUC-PR values also range from 0 to 1. A higher AUC-PR indicates better precision-recall trade-off, with a value of 1 representing perfect precision-recall balance.

2.7.4 Libraries and Modules Used:

- [sklearn.metrics.precision_recall_curve](#)

```
1 from sklearn.metrics import precision_recall_curve
2 precision, recall, thresholds = precision_recall_curve(y_true, y_pred)
```

2.7.5 When to Use AUC-PR:

AUC-PR is particularly useful when:

- You are dealing with imbalanced datasets where the positive class is rare.
- You want to evaluate a model's performance in scenarios where precision and recall are both important.

2.7.6 Applications of AUC-PR:

- Anomaly detection, where accurately identifying rare anomalies is crucial.
- Information retrieval tasks, where achieving a balance between relevant and non-relevant documents is desired.

2.7.7 Advantages of AUC-PR:

- **Class Imbalance Handling:** AUC-PR is effective in handling class imbalance, focusing on the performance of the minority class.
- **Emphasis on Positive Class:** AUC-PR provides insights into a model's ability to accurately predict positive instances while minimizing false positives.

2.7.8 Disadvantages of AUC-PR:

- **Threshold Dependency:** AUC-PR can be sensitive to the chosen probability threshold, potentially leading to different results based on threshold selection.

2.7.9 Relevant Reference Links:

2.8 Log Loss (Cross-Entropy Loss):

2.8.1 What is it?

Log Loss, also referred to as Cross-Entropy Loss, is a widely used evaluation metric for binary and multiclass classification models. It quantifies the difference between predicted probabilities and actual class labels.

2.8.2 How it is Calculated:

For binary classification, the Log Loss is calculated using the following formula:

$$\text{Log Loss} = -\left(\frac{1}{N}\right) \sum [y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

Where,

N is the number of instances, y_i is the true label (0 or 1) of instance i , and p_i is the predicted probability of class 1 for instance i .

For a multiclass scenario, the formula is adapted accordingly to account for multiple classes.

2.8.3 Range of Values:

Log Loss values range from 0 to positive infinity. A Log Loss of 0 indicates perfect predictions, while higher values suggest poorer predictions.

2.8.4 Libraries and Modules Used:

- [sklearn.metrics.log_loss](#)

```
1 from sklearn.metrics import log_loss
2 log_loss(y_true, y_pred)
```

2.8.5 When to Use Log Loss:

Log Loss is particularly useful when:

- Training and evaluating models that output probability estimates.
- Seeking a probabilistic measure that captures the confidence of the model's predictions.

2.8.6 Applications of Log Loss:

- Spam email classification, where predicted probabilities can be used to rank emails by likelihood of being spam.
- Image classification with multiple classes, where model confidence is important.

2.8.7 Advantages of Log Loss:

- **Probabilistic Interpretation:** Log Loss provides a probabilistic interpretation of a model's predictions, making it suitable for tasks where understanding prediction confidence is crucial.

2.8.8 Disadvantages of Log Loss:

- **Sensitivity to Misclassification:** Log Loss is sensitive to misclassification, potentially leading to higher penalties for confident but incorrect predictions.

2.8.9 Relevant Reference Links:

- [Introduction to Log Loss \(Logarithmic Loss\) and Cross-Entropy](#)
- [Understanding the Mathematics behind Logarithmic Loss](#)

2.9 Cohen's Kappa:

2.9.1 What is it?

Cohen's Kappa is a statistical measure used to assess the level of agreement between two evaluators when dealing with categorical or nominal data. It accounts for the agreement that might occur by chance.

Like many other evaluation metrics, Cohen's kappa is calculated based on the confusion matrix. However, in contrast to calculating overall accuracy, Cohen's kappa takes imbalance in class distribution into account and can, therefore, be more complex to interpret.

2.9.2 How it is Calculated:

Cohen's Kappa is calculated using the following formula:

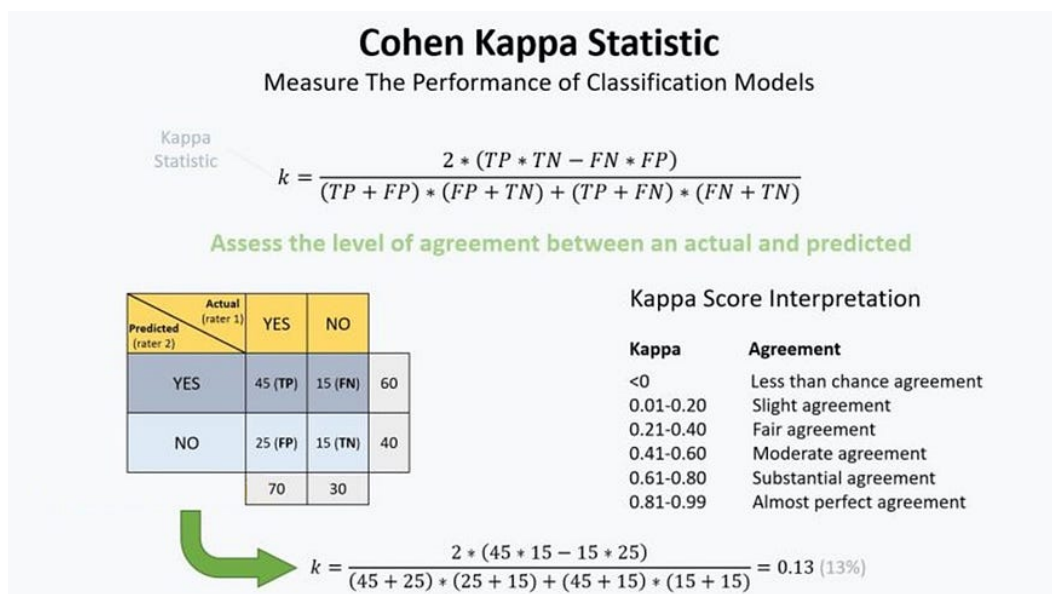
$$\kappa = \frac{po - pe}{(1 - pe)}$$

where:

- po is the observed agreement between the raters.
- pe is the expected agreement under chance.

2.9.3 Range of Values:

Cohen's Kappa values range from -1 to 1. A positive value indicates agreement better than chance, while a negative value suggests agreement worse than chance. A value of 0 indicates agreement that is no better than random chance.



2.9.4 Libraries and Modules Used:

- [sklearn.metrics.cohen_kappa_score](#)

```
1 from sklearn.metrics import cohen_kappa_score
2
3 #define array of ratings for both raters
4 rater1 = [0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0]
5 rater2 = [0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0]
6
7 #calculate Cohen's Kappa
8 cohen_kappa_score(rater1, rater2)
```

The Cohen Kappa score comes out to be **0.21053**

2.9.5 When to Use Cohen's Kappa:

Cohen's Kappa is particularly useful when:

- Assessing inter-rater reliability in scenarios where human judgment or evaluation is involved.
- Evaluating the performance of annotators or raters in labeling tasks.

2.9.6 Applications of Cohen's Kappa:

- Medical diagnosis, where multiple doctors diagnose the same condition.
- Natural language processing, for measuring agreement between human annotators in labeling text data.

2.9.7 Advantages of Cohen's Kappa:

- **Chance Correction:** Cohen's Kappa accounts for agreement that could happen by chance, providing a more accurate measure of inter-rater agreement.
- **Suitable for Nominal Data:** It is applicable to categorical or nominal data, making it versatile for various types of evaluations.

2.9.8 Disadvantages of Cohen's Kappa:

- **Dependent on Prevalence:** Cohen's Kappa can be influenced by the distribution of categories in the data, potentially leading to different values based on prevalence.
- **Sensitive to Agreement Frequency:** The interpretation of Cohen's Kappa can be challenging when the frequency of agreement is significantly higher or lower than expected.

2.9.9 Relevant Reference Links:

- [Understanding and Interpreting Cohen's Kappa](#)
- [Cohen's Kappa and Inter-Rater Reliability](#)

2.10 Matthews Correlation Coefficient (MCC):

2.10.1 What is it?

The Matthews Correlation Coefficient (MCC) is a balanced evaluation metric used to measure the quality of binary classification models, particularly in imbalanced datasets. It takes into account true positives, true negatives, false positives, and false negatives.

2.10.2 How it is Calculated:

MCC is calculated using the following formula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

where:

- TP is the count of true positives.
- TN is the count of true negatives.
- FP is the count of false positives.
- FN is the count of false negatives.

2.10.3 Range of Values:

MCC values range from -1 to 1. A value of 1 indicates perfect prediction, 0 indicates random prediction, and -1 indicates total disagreement between predictions and actual values.

2.10.4 Libraries and Modules Used:

- [sklearn.metrics.matthews_corrcoef](#)

```
1 from sklearn.metrics import matthews_corrcoef
2 matthews_corrcoef(y_true, y_pred)
```

2.10.5 When to Use MCC:

MCC is particularly useful when:

- Evaluating models on imbalanced datasets where one class is dominant.
- Seeking a balanced measure that considers true positives, true negatives, false positives, and false negatives.

2.10.6 Applications of MCC:

- Medical diagnosis, where class imbalance and different misclassification costs are prevalent.

- Fraud detection, where capturing rare fraud cases while minimizing false positives is essential.

2.10.7 Advantages of MCC:

- **Balanced Metric:** MCC provides a balanced evaluation by considering all four possible outcomes, making it suitable for imbalanced datasets.
- **Handles Class Imbalance:** MCC effectively handles class imbalance by accounting for true positives, true negatives, false positives, and false negatives.

2.10.8 Disadvantages of MCC:

- **Sensitivity to Imbalance:** While MCC handles imbalance, extreme class imbalance can still influence the MCC value.

2.10.9 Relevant Reference Links:

- [Introduction to Matthews Correlation Coefficient](#)
- [Math Behind MCC](#)

2.11 Gain-Lift Chart

2.11.1 What is it?

A Gain-Lift Chart, also known simply as a Lift Chart, is a graphical representation used in predictive modeling and marketing analytics to evaluate the performance of a predictive model, particularly in binary classification or response modeling tasks. The chart helps assess how well a model ranks or segments a target variable by comparing the model's predictions against a random or baseline model.

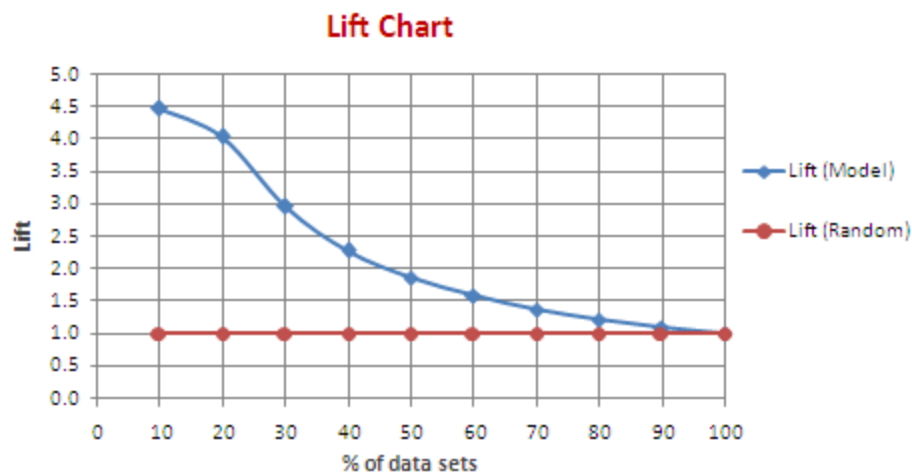
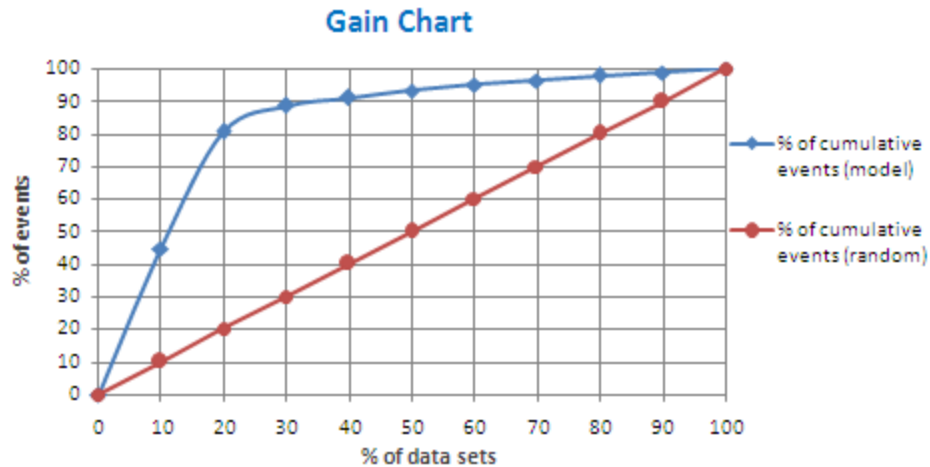
Purpose: The Gain-Lift Chart allows you to understand how effectively a model is targeting a specific outcome or response variable. It helps to visualize the performance improvement gained by using the predictive model compared to a random guess.

2.11.2 Construction:

A Gain-Lift Chart is constructed by dividing the dataset into different segments or groups based on the predicted probabilities or scores provided by the model. These segments typically contain an equal number of observations, although other criteria can be used.

For each segment, the chart displays two lines:

1. **Cumulative Percentage of Positive Outcomes:** This line shows the cumulative percentage of positive outcomes (e.g., conversions, responses) in the segment as you move through the segments from left to right.
2. **Lift Line (Ratio of Actual to Expected):** This line represents the lift, which is the ratio of the actual percentage of positive outcomes in a segment to the expected percentage of positive outcomes under random guessing. A lift value greater than 1 indicates that the model is performing better than random guessing for that segment.



2.11.3 Use Cases:

The Gain-Lift Chart allows you to understand how much better the model is at identifying positive outcomes compared to random guessing, for each segment. It helps you identify which segments benefit the most from using the model's predictions.

2.11.4 Application:

Gain-Lift Charts are often used in marketing campaigns, customer targeting, and credit risk assessment to evaluate the performance of models in predicting outcomes like purchases, responses, or defaults.

2.11.5 Advantages:

- **Visual Comparison:** Gain-Lift Charts provide a visual way to compare the model's performance to a random baseline.
- **Segment-Specific Insights:** They help identify which segments receive the most benefit from using the model's predictions.

2.11.6 Limitations:

- **Dependence on Segmentation:** The chart's interpretation may be sensitive to how the dataset is segmented.
- **Baseline Assumption:** The lift values are relative to the assumed baseline of random guessing.

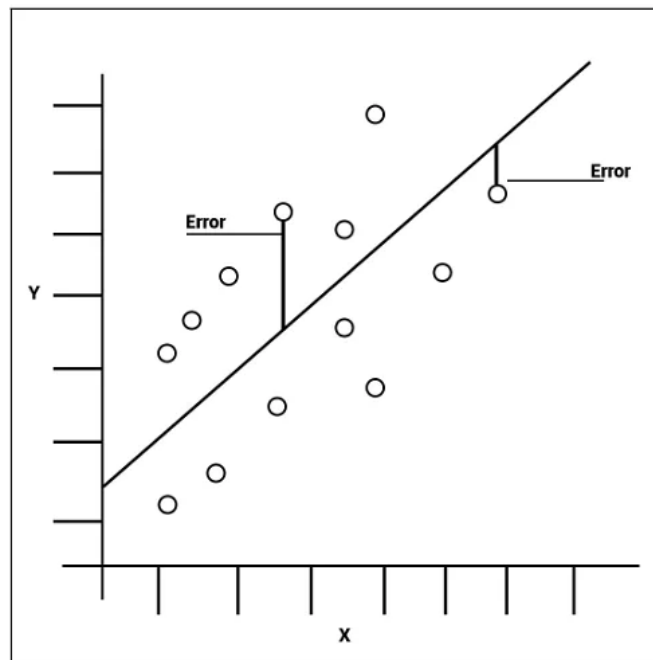
3 Evaluation metrics for Regression

There are several metrics that are commonly used for evaluating and reporting the performance of a regression model; they are:

3.1 Mean Absolute Error (MAE):

3.1.1 What is it?

Mean Absolute Error (MAE) is an evaluation metric used to measure the average magnitude of errors between predicted values and actual values in a regression problem. It quantifies the average absolute difference between the predicted values and the true values.



3.1.2 How it is Calculated:

MAE is calculated by taking the average of the absolute differences between each predicted value (\hat{y}_i) and its corresponding true value (y_i):

$$MAE = \frac{1}{n} \sum |y - \hat{y}|$$

Diagram illustrating the calculation of MAE:

- $\frac{1}{n}$: Divide by the total number of data points
- \sum : Sum of
- y : Actual output value
- \hat{y} : Predicted output value
- $|y - \hat{y}|$: The absolute value of the residual

3.1.3 Range of Values:

MAE values range from 0 to positive infinity. Lower MAE values indicate better model performance, with 0 indicating a perfect match between predictions and true values.

3.1.4 Libraries and Modules Used:

- [sklearn.metrics.median_absolute_error](#)

```
1 from sklearn.metrics import median_absolute_error
2 median_absolute_error(y_true, y_pred)
```

3.1.5 When to Use MAE:

MAE is used when you want to measure the average absolute error between predicted and actual values, giving equal weight to all errors.

3.1.6 Applications of MAE:

- Financial forecasting, such as predicting stock prices.
- Climate modeling to assess the accuracy of temperature predictions.
- Regression problems where equally weighting errors is appropriate.

3.1.7 Advantages of MAE:

- **Robust to Outliers:** MAE is less sensitive to outliers compared to other metrics like Mean Squared Error (MSE).
- **Easy Interpretation:** MAE represents an average absolute error, which is easy to understand and communicate.
- **Mathematical simplicity:** The absolute value function used in MAE makes it mathematically straightforward and computationally efficient to calculate, making it more accessible for use in various applications.
- **Balanced errors:** Unlike MSE, which penalizes larger errors more heavily, MAE treats all errors equally. This can be advantageous when you want to avoid overly emphasizing large errors in your analysis.
- **Resilience to data distribution:** MAE performs well across various data distributions, including both symmetric and skewed datasets, making it a versatile metric for evaluating models in different scenarios.

3.1.8 Disadvantages of MAE:

- **Lack of Sensitivity to Error Magnitude:** MAE treats all errors equally, which might not be suitable if larger errors should be penalized more.

3.1.9 Relevant Reference Links:

- [Mean Absolute Error \(MAE\) in Machine Learning](#)

3.2 Mean Absolute Scaled Error (MASE):

3.2.1 What is it?

Mean Absolute Scaled Error (MASE) is an evaluation metric used to measure the accuracy of a forecast in time series analysis. It quantifies the relative mean absolute

error of a forecast compared to the mean absolute error of a naive forecast (e.g., using the previous observation).

3.2.2 How it is Calculated:

MASE is calculated by taking the ratio of the mean absolute error of the forecast (MAE_f) to the mean absolute error of the naive forecast (MAE_{naive}):

$$MASE = \frac{MAE_f}{MAE_{naive}}$$

3.2.3 Range of Values:

MASE values can range from 0 to positive infinity. A value of 1 indicates that the forecast is as accurate as the naive forecast. Values less than 1 suggest the forecast is better than the naive forecast, while values greater than 1 indicate worse performance.

3.2.4 Libraries and Modules Used:

MASE can be calculated using various programming languages and libraries. Python packages like NumPy and pandas can be used for calculations.

3.2.5 When to Use MASE:

MASE is used in time series forecasting tasks to assess the accuracy of a forecast relative to a simple naive forecast. It provides a useful benchmark for evaluating forecast performance.

3.2.6 Applications of MASE:

- Demand forecasting for inventory management.
- Financial forecasting for sales or revenue.
- Any time series forecasting where benchmarking against a naive forecast is relevant.

3.2.7 Advantages of MASE:

- **Relative Performance:** MASE provides a measure of forecast accuracy relative to a baseline naive forecast.
- **Scale-Invariant:** MASE is scale-invariant, making it suitable for comparing forecasts across different datasets.

3.2.8 Disadvantages of MASE:

- **Dependency on Naive Forecast:** MASE's accuracy depends on the accuracy of the chosen naive forecast.

3.2.9 Relevant Reference Links:

- [Mean Absolute Scaled Error \(MASE\) in Time Series Forecasting](#)

3.3 Mean Percentage Error (MPE):

3.3.1 What is it?

Mean Percentage Error (MPE) is an evaluation metric used in forecasting and time series analysis to measure the average percentage difference between predicted values and actual values. MPE quantifies the average magnitude and direction (underestimation or overestimation) of errors as a percentage of the actual values.

3.3.2 How it is Calculated:

MPE is calculated by taking the average of the percentage differences between each predicted value (\hat{y}_i) and its corresponding actual value (x_i):

$$\text{MPE} = \frac{1}{n} \sum_{i=1}^n \frac{\hat{y}_i - x_i}{x_i} \times 100$$

3.3.3 Range of Values:

MPE values can be positive or negative, indicating overestimation or underestimation, respectively. A lower absolute MPE indicates better model performance, but its interpretation depends on the context of the problem.

3.3.4 Libraries and Modules Used:

MPE can be calculated using various programming languages and libraries. Python packages like NumPy and pandas can be used for calculations.

3.3.5 When to Use MPE:

MPE is used when you want to measure the average percentage difference between predicted and actual values in forecasting and time series analysis.

3.3.6 Applications of MPE:

- Sales forecasting for businesses.
- Demand forecasting for inventory management.
- Budget forecasting for financial planning.

3.3.7 Advantages of MPE:

- **Direction and Magnitude:** MPE provides insight into the direction (overestimation or underestimation) and average magnitude of errors.
- **Interpretability:** MPE is expressed in percentage terms, making it easy to interpret and communicate.

3.3.8 Disadvantages of MPE:

- **Asymmetry:** MPE treats overestimation and underestimation equally, which might not be suitable if one direction of error is more critical than the other.

3.3.9 Relevant Reference Links:

- [Mean Percentage Error \(MPE\) in Forecasting](#)

3.4 Mean Absolute Percentage Error (MAPE):

3.4.1 What is it?

Mean Absolute Percentage Error (MAPE) is a widely used evaluation metric in forecasting and time series analysis. It measures the average absolute percentage difference between forecasted values and actual values, providing insight into the accuracy of the forecasts in terms of percentage error.

3.4.2 How it is Calculated:

MAPE is calculated by taking the average of the absolute percentage differences between each forecasted value (y_i) and its corresponding actual value (x_i):

$$MAPE = 1/n \sum |(y_i - x_i)/x_i| \times 100$$

3.4.3 Range of Values:

MAPE values are expressed in percentage and can range from 0 to positive infinity. A lower MAPE value indicates better forecast accuracy.

3.4.4 Libraries and Modules Used:

- [sklearn.metrics.mean_absolute_percentage_error](#)
- [tf.keras.losses.MeanAbsolutePercentageError](#)

```
1 from sklearn.metrics import mean_absolute_percentage_error
2 mean_absolute_percentage_error(y_true, y_pred)
```

3.4.5 When to Use MAPE:

MAPE is used in forecasting and time series analysis to evaluate the accuracy of forecasts in terms of percentage errors. It is commonly used when comparing the performance of different forecasting methods.

3.4.6 Applications of MAPE:

- Sales forecasting for retail businesses.
- Demand forecasting for supply chain management.
- Financial forecasting for budget planning.

3.4.7 Advantages of MAPE:

- **Relative Measure:** MAPE provides a relative measure of forecast accuracy in percentage terms, making it easy to interpret and compare.

3.4.8 Disadvantages of MAPE:

- **Division by Zero:** MAPE can encounter division by zero when actual values are close to zero. In such cases, other measures like Mean Absolute Scaled Error (MASE) are preferred.

3.4.9 Relevant Reference Links:

- [Mean Absolute Percentage Error \(MAPE\) in Time Series Forecasting](#)

3.5 Median Absolute Error (MedAE):

3.5.1 What is it?

Median Absolute Error (MedAE) is an evaluation metric used in regression analysis to measure the median of the absolute differences between predicted values and actual values. MedAE is a robust alternative to Mean Absolute Error (MAE) that is less sensitive to the influence of outliers.

3.5.2 How it is Calculated:

MedAE is calculated by taking the median of the absolute differences between each predicted value (y_i) and its corresponding actual value (x_i):

$$MedAE = median(|y_1 - x_1|, |y_2 - x_2|, \dots, |y_n - x_n|)$$

3.5.3 Range of Values:

MedAE values can range from 0 to positive infinity. A lower MedAE indicates better model performance, with 0 indicating a perfect match between predictions and true values.

3.5.4 Libraries and Modules Used:

MedAE can be calculated using various programming languages and libraries. Python packages like NumPy and scikit-learn can be used for calculations.

3.5.5 When to Use MedAE:

MedAE is used when you want to measure the median magnitude of errors between predicted and actual values, with reduced sensitivity to outliers compared to MAE.

3.5.6 Applications of MedAE:

- Regression tasks in which outliers can significantly affect the accuracy assessment.
- Model evaluation when robustness to extreme errors is important.

3.5.7 Advantages of MedAE:

- **Robustness to Outliers:** MedAE is less sensitive to outliers compared to MAE, making it a robust metric for evaluating model performance.

3.5.8 Disadvantages of MedAE:

- **Lack of Scale Interpretation:** MedAE doesn't provide error values in the same units as the original data, which can make interpretation less intuitive.

3.5.9 Relevant Reference Links:

- [Median Absolute Error \(MedAE\) in Regression Analysis](#)

3.6 Mean Squared Error (MSE):

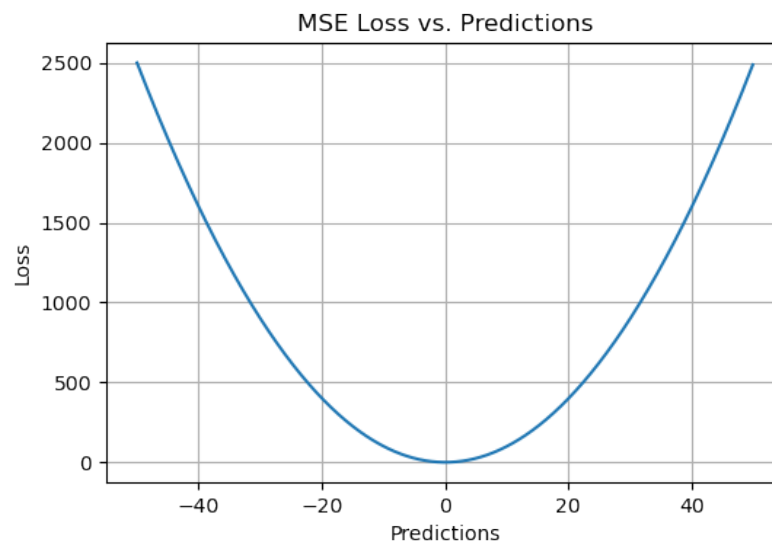
3.6.1 What is it?

Mean Squared Error (MSE) is a common evaluation metric used in regression analysis to measure the average of the squares of the errors or differences between predicted values and actual values. It quantifies the average squared deviation between the predicted and true values.

3.6.2 How it is Calculated:

MSE is calculated by taking the average of the squared differences between each predicted value (\hat{y}_i) and its corresponding true value (x_i):

$$MSE = \left(\frac{1}{n}\right) \sum (y_i - x_i)^2$$



3.6.3 Range of Values:

MSE values range from 0 to positive infinity. Lower MSE values indicate better model performance, with 0 indicating a perfect match between predictions and true values.

3.6.4 Libraries and Modules Used:

In Python, you can use the scikit-learn library or calculate it manually.

- [sklearn.metrics.mean_squared_error](#)

```
1 from sklearn.metrics import mean_squared_error
2 mse = mean_squared_error(y_true, y_pred)
```

3.6.5 When to Use MSE:

MSE is commonly used when you want to measure the average squared error between predicted and actual values. It is sensitive to large errors due to the squaring of errors.

3.6.6 Applications of MSE:

- Regression tasks such as predicting house prices or stock prices.
- Model evaluation where you want to emphasize larger errors.

3.6.7 Advantages of MSE:

- **Sensitivity to Large Errors:** MSE penalizes larger errors more heavily due to squaring, making it suitable when you want to focus on reducing large deviations.
- **Mathematical Properties:** The use of squared errors makes calculations and optimization techniques easier in certain cases.

3.6.8 Disadvantages of MSE:

- **Sensitivity to Outliers:** MSE is sensitive to outliers since it squares the errors, which can give excessive weight to outliers.

3.6.9 Relevant Reference Links:

- [Mean Squared Error \(MSE\) in Machine Learning](#)

3.7 Root Mean Squared Error (RMSE):

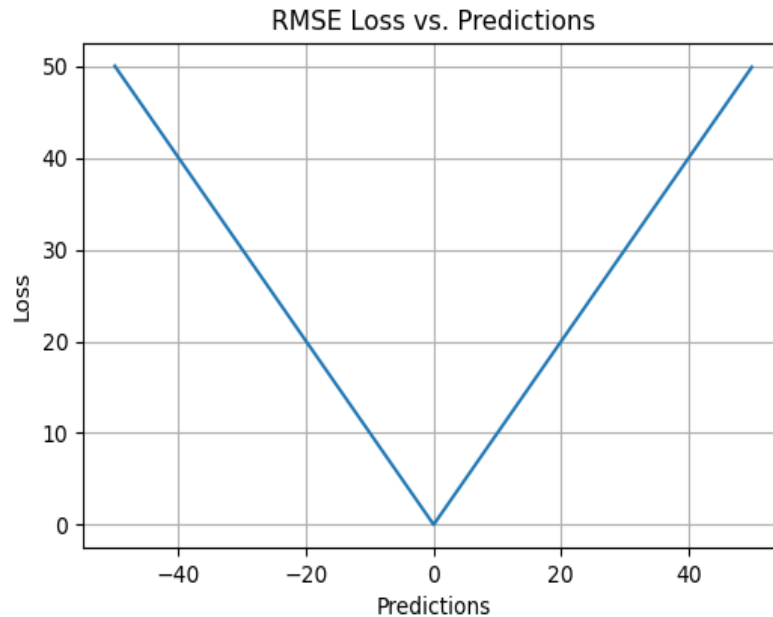
3.7.1 What is it?

Root Mean Squared Error (RMSE) is a commonly used evaluation metric in regression analysis to measure the square root of the average of the squared differences between predicted values and actual values. It is a variant of the Mean Squared Error (MSE) that provides the average absolute deviation between predictions and true values.

3.7.2 How it is Calculated:

RMSE is calculated by taking the square root of the average of the squared differences between each predicted value (\hat{y}_i) and its corresponding true value (x_i):

$$RMSE = \sqrt{\left(\frac{1}{n}\right) \sum (y_i - x_i)^2}$$



3.7.3 Range of Values:

RMSE values range from 0 to positive infinity, just like MSE. Lower RMSE values indicate better model performance, with 0 indicating a perfect match between predictions and true values.

3.7.4 Libraries and Modules Used:

```
1 from sklearn.metrics import mean_squared_error
2 MSE = mean_squared_error(y_actual, y_predicted)
3 RMSE = np.sqrt(MSE)
```

3.7.5 When to Use RMSE:

RMSE is used when you want to measure the average absolute error between predicted and actual values, but you also want the error values to be in the same units as the original values (not squared).

3.7.6 Applications of RMSE:

- Regression tasks like predicting sales or temperature.
- Model evaluation when both error magnitude and units are important.

3.7.7 Advantages of RMSE:

- **Same Units as Original Data:** RMSE provides error values in the same units as the original data, making it easier to interpret and communicate.

3.7.8 Disadvantages of RMSE:

- **Sensitivity to Outliers:** Like MSE, RMSE is sensitive to outliers due to squaring errors.

3.7.9 Relevant Reference Links:

- [Root Mean Squared Error \(RMSE\) in Machine Learning](#)

3.8 R-squared and Adjusted R-squared:

3.8.1 What are they?

R-squared (Coefficient of Determination) and Adjusted R-squared are evaluation metrics used in regression analysis to assess the goodness of fit of a regression model to the data. They indicate the proportion of the variance in the dependent variable that is explained by the independent variables in the model.

3.8.2 How they are Calculated:

1. **R-squared (Coefficient of Determination):**

R-squared is calculated as the ratio of the explained variance (SSR) to the total variance (SST):

$$R^2 = \frac{SSR}{SST} = 1 - \left(\frac{SSE}{SST} \right)$$

where:

- SSR is the sum of squared regression (explained) errors.
- SSE is the sum of squared errors (residuals).
- SST is the total sum of squares.

2. **Adjusted R-squared:**

Adjusted R-squared takes into account the number of predictors in the model and adjusts R-squared for the number of independent variables (p):

$$R^2 = 1 - \left(\frac{\frac{SSE}{n - p - 1}}{\frac{SST}{n - 1}} \right)$$

3.8.3 Range of Values:

R-squared and Adjusted R-squared values range from 0 to 1. A higher value indicates a better fit of the model to the data. However, a higher value doesn't necessarily mean a better model if overfitting is present.

3.8.4 Libraries and Modules Used:

- [sklearn.metrics.r2_score](#)


```
from sklearn.metrics import r2_score
A = [random.randint(10,100) for i in range(10)]
P = [random.randint(10,100) for i in range(10)]
r2 = r2_score(A, P)
print('r2 score for perfect model is', r2)
```

r2 score for perfect model is 0.6882240153391388

3.8.5 When to Use R-squared and Adjusted R-squared:

R-squared and Adjusted R-squared are used when you want to assess how well a regression model explains the variance in the dependent variable.

3.8.6 Applications of R-squared and Adjusted R-squared:

- **Model selection:** Comparing different models to choose the one that best fits the data.
- **Assessing explanatory power:** Evaluating how well the independent variables explain the variability in the dependent variable.

3.8.7 Advantages of R-squared and Adjusted R-squared:

- **Goodness of Fit:** They provide a quantifiable measure of how well the model fits the data.
- **Comparability:** They allow for easy comparison of different models.

3.8.8 Disadvantages of R-squared and Adjusted R-squared:

- **No Causation:** High R-squared values do not imply causation between variables.
- **Overfitting Concerns:** R-squared can increase with the addition of more variables, even if they don't contribute meaningfully to the model.

3.8.9 Relevant Reference Links:

- [R-squared and Adjusted R-squared in Regression Analysis](#)

3.8.10 MSE vs MAE vs R2

MSE / RSME	MAE	R2
Based on square of error	Based on absolute value of error	Based on correlation between actual and predicted value
Value lies between 0 to ∞	Value lies between 0 to ∞	Value lies between 0 and 1
Sensitive to outliers, punishes larger error more	Treat larger and small errors equally. Not sensitive to outliers	Not sensitive to outliers
Small value indicates better model	Small value indicates better model	Value near 1 indicates better model

3.9 Mean Squared Logarithmic Error (MSLE):

3.9.1 What is it?

Mean Squared Logarithmic Error (MSLE) is an evaluation metric commonly used in regression tasks, particularly when the target variable spans a wide range of values and the relative errors are important to consider. MSLE measures the mean squared difference between the natural logarithms of predicted values and the natural logarithms of actual values.

3.9.2 How it is Calculated:

MSLE is calculated by first taking the natural logarithm of both the predicted values (\hat{y}_i) and actual values (x_i), then calculating the mean squared difference:

$$\text{MSLE} = \frac{1}{n} \sum_{i=1}^n (\log(\hat{y}_i + 1) - \log(x_i + 1))^2$$

3.9.3 Range of Values:

MSLE values are always non-negative. Lower MSLE values indicate better model performance, with 0 indicating a perfect match between predictions and true values.

3.9.4 Libraries and Modules Used:

```
1 import numpy as np
2
3 def msle(y_true, y_pred):
4     squared_log_error = np.square(np.log1p(y_true) - np.log1p(y_pred))
5     return np.mean(squared_log_error)
6
```

3.9.5 When to Use MSLE:

MSLE is particularly useful when the target variable has a wide range of values and relative errors are more important to consider.

3.9.6 Applications of MSLE:

- Predictive modeling in fields like finance, where the magnitude of errors matters more than their direction.
- Any regression task where the target variable can take a wide range of values.

3.9.7 Advantages of MSLE:

- **Relative Error Focus:** MSLE emphasizes relative errors, making it suitable for scenarios where percentage errors are more meaningful than absolute differences.

3.9.8 Disadvantages of MSLE:

- **Insensitivity to Direction:** MSLE treats underpredictions and overpredictions similarly, which might not be desirable in all cases.

3.9.9 Relevant Reference Links:

- [Mean Squared Logarithmic Error \(MSLE\) in Machine Learning](#)

3.10 Root Mean Squared Logarithmic Error (RMSLE):

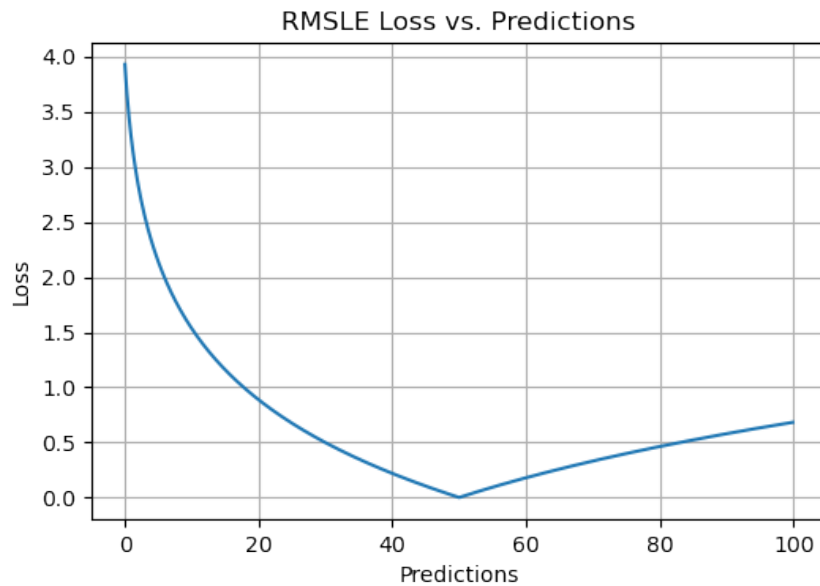
3.10.1 What is it?

Root Mean Squared Logarithmic Error (RMSLE) is an evaluation metric commonly used in regression tasks, particularly in situations where the target variable spans a wide range of values and is sensitive to relative errors. RMSLE measures the root mean squared difference between the natural logarithm of predicted values and the natural logarithm of actual values.

3.10.2 How it is Calculated:

RMSLE is calculated by first taking the natural logarithm of both the predicted values (\hat{y}_i) and actual values (x_i), then calculating the mean squared difference, taking the square root of the mean, and finally exponentiating to revert back to the original scale:

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (\log(x_i+1) - \log(\hat{y}_i+1))^2}$$



3.10.3 Range of Values:

RMSLE values are always non-negative. Lower RMSLE values indicate better model performance, with 0 indicating a perfect match between predictions and true values.

3.10.4 Libraries and Modules Used:

```
1 import numpy as np
2
3 def rmsle(y_true, y_pred):
4     # Calculate the squared logarithmic error
5     squared_log_error = np.square(np.log1p(y_true) - np.log1p(y_pred))
6
7     # Calculate the mean of squared logarithmic errors
8     mean_squared_log_error = np.mean(squared_log_error)
9
10    # Calculate the root mean squared logarithmic error
11    rmsle_value = np.sqrt(mean_squared_log_error)
12
13    return rmsle_value
14
```

3.10.5 When to Use RMSLE:

RMSLE is particularly useful when the target variable has a wide range of values and relative errors are important to consider.

3.10.6 Applications of RMSLE:

- Predictive modeling in fields like finance, where the magnitude of errors matters more than their direction.

- Any regression task where the target variable can take a wide range of values.

3.10.7 Advantages of RMSLE:

- **Relative Error Focus:** RMSLE emphasizes relative errors, making it suitable for scenarios where percentage errors are more meaningful than absolute differences.

3.10.8 Disadvantages of RMSLE:

- **Insensitivity to Direction:** RMSLE treats underpredictions and overpredictions similarly, which might not be desirable in all cases.

3.10.9 Relevant Reference Links:

- [Root Mean Squared Logarithmic Error \(RMSLE\) in Machine Learning](#)

3.11 Huber Loss:

3.11.1 What is it?

Huber Loss is a type of loss function used in regression tasks, particularly in robust regression, that combines the advantages of both Mean Squared Error (MSE) and Mean Absolute Error (MAE). Huber Loss is less sensitive to outliers compared to MSE while still maintaining some of the desirable properties of MAE.

3.11.2 How it is Calculated:

Huber Loss is defined as a piecewise function that transitions from quadratic (like MSE) for small errors to linear (like MAE) for larger errors. The formula for Huber Loss is:

$$L_i = \begin{cases} (z^{(i)})^2 & \text{for } |z^{(i)}| \leq \delta \\ 2\delta|z^{(i)}| - \delta^2 & \text{for } |z^{(i)}| > \delta \end{cases}$$

$$Z = y - f(x)$$

Here, y represents the true value, $f(x)$ represents the predicted value, and δ is a threshold that determines where the loss transitions from quadratic to linear behavior.

3.11.3 Advantages of Huber Loss:

- **Robust to Outliers:** Huber Loss is less sensitive to outliers compared to MSE, making it suitable for data with noisy or outlier-contaminated observations.
- **Smooth Transition:** The piecewise formulation of Huber Loss provides a smooth transition between quadratic and linear loss, balancing between the advantages of MSE and MAE.

3.11.4 Disadvantages of Huber Loss:

- **Hyperparameter Tuning:** The choice of the δ parameter requires careful tuning based on the characteristics of the data.

3.11.5 Libraries and Modules Used:

```
from scipy.special import huber
r = np.array([1., 2.5, 8., 10.])
deltas = np.array([[1.], [5.], [9.]])
print(r.shape, deltas.shape)
huber(deltas, r)

(4,) (3, 1)
array([[ 0.5 ,  2.   ,  7.5 ,  9.5 ],
       [ 0.5 ,  3.125, 27.5 , 37.5 ],
       [ 0.5 ,  3.125, 32.  , 49.5 ]])
```

3.11.6 When to Use Huber Loss:

Huber Loss is used when you want a loss function that is less sensitive to outliers than MSE but provides some of the advantages of both MSE and MAE.

3.11.7 Applications of Huber Loss:

- Regression tasks where the data may contain outliers or noisy observations.
- Robust parameter estimation when dealing with data that has varying levels of noise.

3.11.8 Relevant Reference Links:

- [Huber Loss in Machine Learning](#)

4 In a nut shell

Metric Type	Metric Name
Regression	Mean Squared Error (MSE)*
Regression	Root Mean Squared Error (RMSE)*
Regression	Mean Absolute scaled error (MASE)
Regression	Mean Absolute percentage error (MAPE)
Regression	Mean Absolute Error (MAE)*
Regression	R-squared (Adjusted R-squared)*
Regression	Explained Variance Score*
Regression	Huber Loss
Regression	Median Absolute Error (MedAE)*
Regression	RMSLE (Root Mean Squared Logarithmic Error)
Regression	Mean Bias Deviation (MBD)
Regression	Mean Percentage Error (MPE)
Regression	Symmetric Mean Absolute Percentage Error (sMAPE)
Regression	Theil's U Statistic
Regression	Kendall's Tau
Regression	Spearman's Rank Correlation
Regression	Mean Squared Logarithmic Error (MSLE)
Regression	Mean Percentage Deviation (MPD)
Regression	Coefficient of Variation (CV)
Regression	Determination Coefficient (determination)
Regression	Symmetric Mean Absolute Percentage Error (sMAPE)
Regression	Akaike Information Criterion (AIC)*
Classification	Accuracy*
Classification	Precision*
Classification	Recall (Sensitivity/True Positive Rate)*
Classification	F1-Score*
Classification	Specificity (True Negative Rate)
Classification	ROC-AUC (Receiver Operating Characteristic - Area Under the Curve)*
Classification	Log Loss (Cross-Entropy Loss)*
Classification	Cohen's Kappa*
Classification	Matthews Correlation Coefficient (MCC)*
Classification	Concordant – Discordant Ratio
Classification	Kolomogorov Smirnov Chart
Classification	Gain lift chart
Classification	Gini Coefficient/Gini Index*
Classification	Categorical Cross-Entropy Loss*
Classification	Hamming Loss*
Classification	Jaccard Index (Intersection over Union)*
Classification	Cramer's V*

Classification	Top-k Accuracy
Classification	Area Under Precision-Recall Curve (AUC-PRC)*
Classification	Receiver Operating Characteristic (ROC) Curve*
Classification	Cohen's H Score*
Classification	Kappa Architecture*
Classification	Informedness (Youden's J Index)*
Classification	Prevalence Threshold
Classification	Discrimination Threshold
Classification	Goodman-Kruskal Gamma
Classification	Normalized Mutual Information (NMI)
Classification	Bennett's S Score
Classification	Mutual Information (MI)
Classification	Zero-One Loss
Classification	Perplexity
Classification	Normalized Gini Coefficient (NGini)
Classification	Normalized Entropy
Classification	Consensus Score
Classification	Cohen's Linearly Weighted Kappa

5 Model Evaluation using Cross Validation

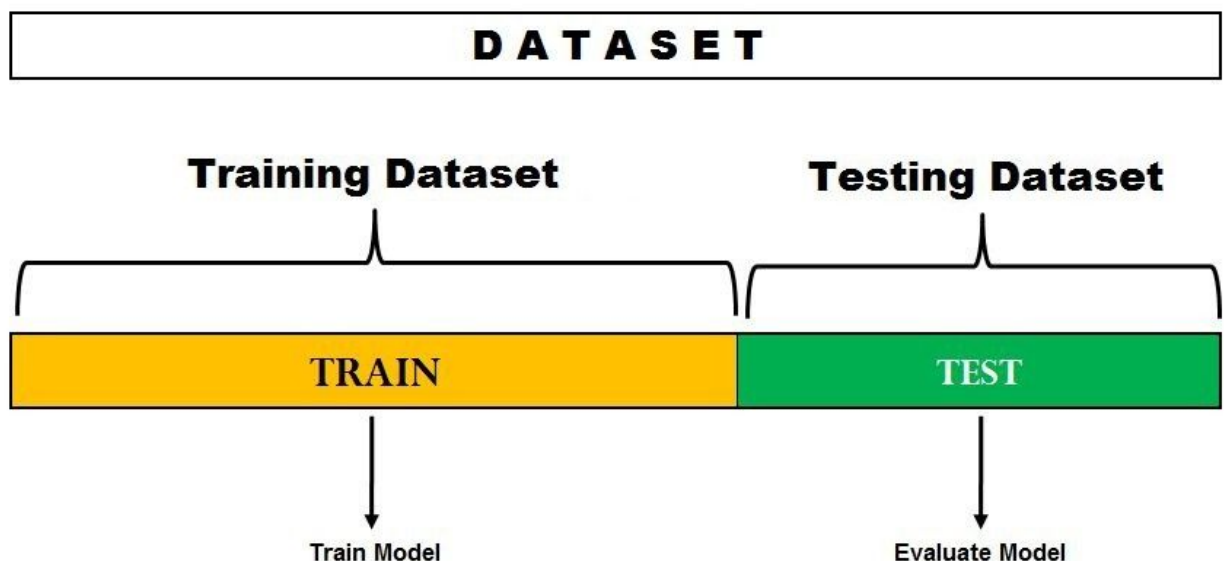
Cross-validation is a popular method for evaluating the performance of machine learning models while minimizing issues like overfitting. It involves partitioning the dataset into multiple subsets, training and evaluating the model on different combinations of these subsets, and then aggregating the results to get a more robust estimate of the model's performance

Types of Cross Validation techniques

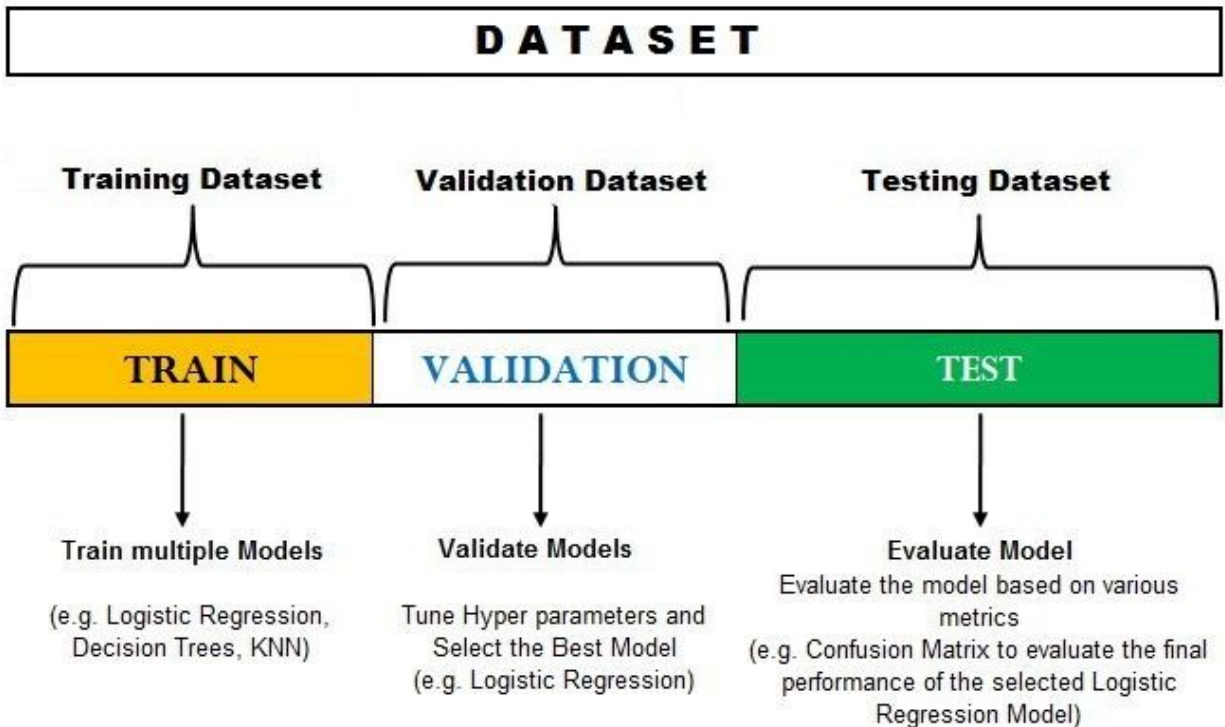
5.1 Holdout Cross-Validation

The dataset is split into two subsets: a training set and a validation (or testing) set. The model is trained on the training set and evaluated on the validation set. This is the simplest form of cross-validation and is prone to variability depending on how the data is split.

As a rule, the proportion of training data has to be larger than the test data.



if we want to hyper-tune your parameters or want to select the best model, you can make a validation set like the one below.



5.1.1 Advantages:

1. Simplicity: Holdout CV is easy to implement and understand.
2. Speed: It is computationally less intensive compared to some other cross-validation techniques.
3. Useful for large datasets: Particularly useful when the dataset is large and creating multiple folds might be time-consuming.

5.1.2 Disadvantages:

1. Variability: The performance estimate can be highly dependent on the specific data split. The choice of which data points to include in the training and validation sets can lead to biased performance estimates.
2. Limited data utilization: A portion of the data is held back for validation, potentially resulting in underutilization of data for training.

5.1.3 Application:

Holdout CV is often used when the dataset is large, time is a constraint, and a quick estimate of a model's performance is needed.

5.1.4 Libraries and Modules Used:

- [sklearn.model_selection.train_test_split](#)

```

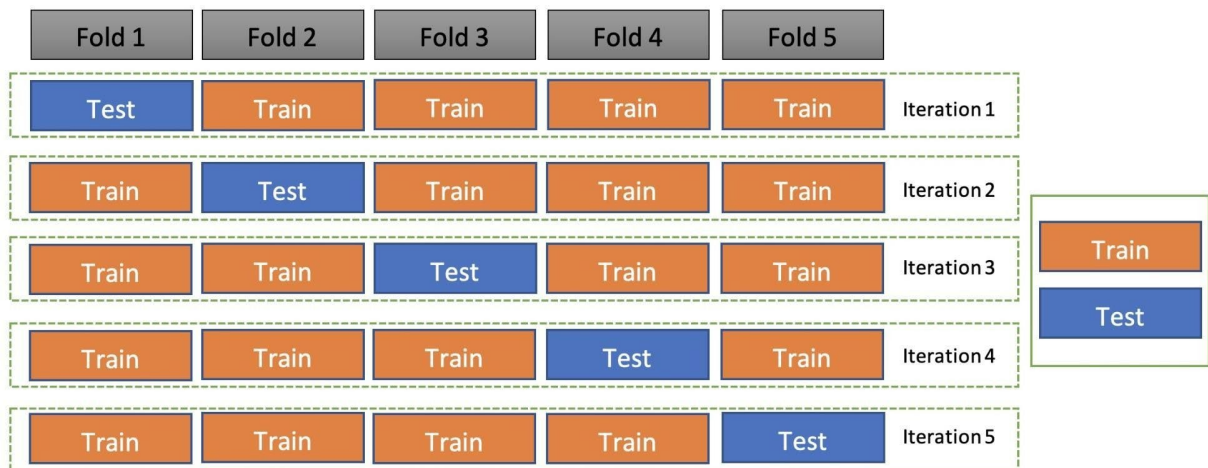
1 from sklearn.model_selection import train_test_split
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.metrics import accuracy_score
4
5 # Load your dataset and split it into features (X) and target (y)
6 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
7
8 # Create and train a model
9 model = LogisticRegression()
10 model.fit(X_train, y_train)
11
12 # Make predictions on the validation set
13 y_pred = model.predict(X_val)
14
15 # Calculate accuracy
16 accuracy = accuracy_score(y_val, y_pred)
17 print("Validation Accuracy:", accuracy)

```

5.2 K-fold cross-validation

In this technique, the whole dataset is partitioned in k parts of equal size and each partition is called a fold. It's known as k-fold since there are k parts where k can be any integer - 3,4,5, etc.

One fold is used for validation and other K-1 folds are used for training the model. To use every fold as a validation set and other left-outs as a training set, this technique is repeated k times until each fold is used once.



This validation technique is not considered suitable for imbalanced datasets as the model will not get trained properly owing to the proper ratio of each class's data.

5.2.1 Advantages:

1. Reduced Variability: K-Fold CV provides a more stable estimate of a model's performance by reducing the impact of a single data split.

2. Better Data Utilization: K-Fold CV uses each data point for both training and validation, resulting in more effective use of the dataset.
3. Robustness: K-Fold CV works well for a wide range of dataset sizes and is suitable for various machine learning algorithms.

5.2.2 Disadvantages:

1. Computational Complexity: K-Fold CV is more computationally intensive than Holdout CV, as it trains and evaluates the model K times.
2. Potential Overfitting: If the dataset is small, K-Fold CV may still be prone to overfitting due to multiple model evaluations on the same data.

5.2.3 Libraries and Modules Used:

- [sklearn.model_selection.KFold](#)

```
1 from sklearn.model_selection import KFold
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.metrics import accuracy_score
4
5 # Load your dataset and split it into features (X) and target (y)
6 kf = KFold(n_splits=5, shuffle=True, random_state=42)
7
8 accuracy_scores = []
9
10 for train_index, val_index in kf.split(X):
11     X_train, X_val = X[train_index], X[val_index]
12     y_train, y_val = y[train_index], y[val_index]
13
14     # Create and train a model
15     model = LogisticRegression()
16     model.fit(X_train, y_train)
17
18     # Make predictions on the validation set
19     y_pred = model.predict(X_val)
20
21     # Calculate accuracy and store it
22     accuracy = accuracy_score(y_val, y_pred)
23     accuracy_scores.append(accuracy)
24
25 # Calculate the average accuracy over all folds
26 average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
27 print("Average Cross-Validation Accuracy:", average_accuracy)
28
```

5.3 Repeated K-Fold Cross-Validation:

Repeated K-Fold Cross-Validation is an extension of K-Fold Cross-Validation that involves repeating the process of K-Fold CV multiple times with different random splits. It

aims to provide a more stable and reliable estimate of a model's performance by averaging over multiple runs of K-Fold CV.

Each iteration of the repeated K-fold is the implementation of a normal K-fold algorithm

5.3.1 Advantages:

1. **Reduced Variability:** Repeated K-Fold CV helps reduce the impact of random data splits, providing a more stable performance estimate.
2. **Robustness:** By averaging over multiple runs, it captures the overall model performance across different data partitions.

5.3.2 Disadvantages:

1. **Increased Computational Complexity:** Repeated K-Fold CV requires running K-Fold CV multiple times, which can be more computationally intensive.
2. **Time-Consuming:** Depending on the number of repetitions and the size of the dataset, repeated K-Fold CV can take longer to compute.

5.3.3 Application:

Repeated K-Fold CV is often used when a more robust estimate of a model's performance is desired, especially when dealing with variability due to random data splits.

5.3.4 Libraries and Modules Used:

- [sklearn.model_selection.RepeatedKFold](#)

```

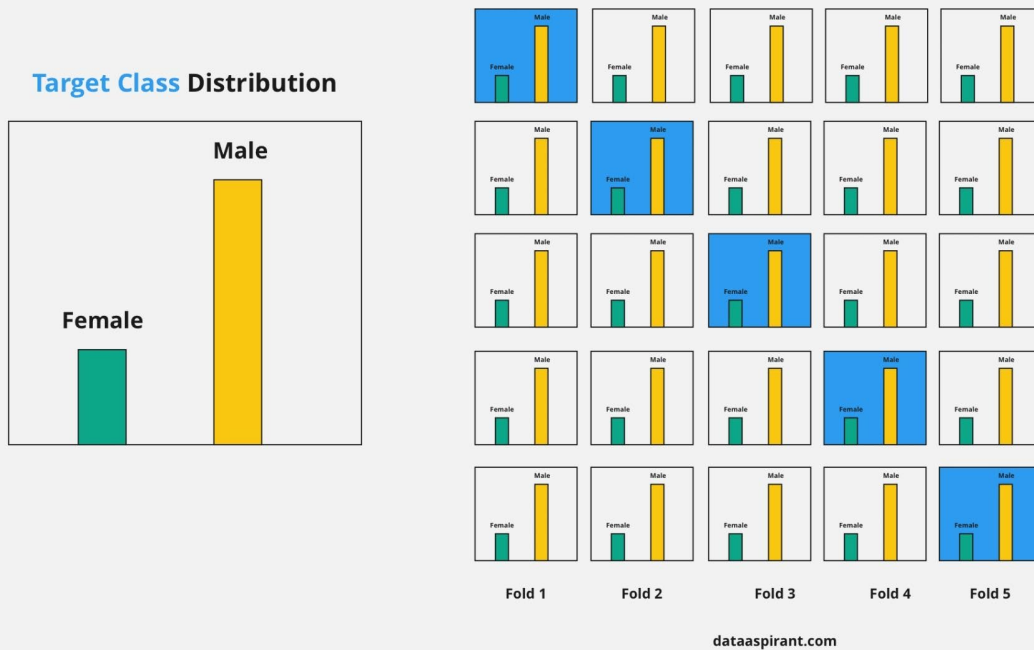
1 from sklearn.model_selection import RepeatedKFold
2 from sklearn.ensemble import RandomForestClassifier
3 from sklearn.metrics import accuracy_score
4
5 # Load your dataset and split it into features (X) and target (y)
6 rkf = RepeatedKFold(n_splits=5, n_repeats=3, random_state=42)
7
8 accuracy_scores = []
9
10 for train_index, val_index in rkf.split(X):
11     X_train, X_val = X[train_index], X[val_index]
12     y_train, y_val = y[train_index], y[val_index]
13
14     # Create and train a model
15     model = RandomForestClassifier()
16     model.fit(X_train, y_train)
17
18     # Make predictions on the validation set
19     y_pred = model.predict(X_val)
20
21     # Calculate accuracy and store it
22     accuracy = accuracy_score(y_val, y_pred)
23     accuracy_scores.append(accuracy)
24
25 # Calculate the average accuracy over all repetitions and folds
26 average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
27 print("Average Repeated K-Fold Cross-Validation Accuracy:", average_accuracy)
28

```

5.4 Stratified k-fold cross-validation

As seen above, k-fold validation can't be used for imbalanced datasets because data is split into k-folds with a uniform probability distribution. Not so with stratified k-fold, which is an enhanced version of the k-fold cross-validation technique. Although it too splits the dataset into k equal folds, each fold has the same ratio of instances of target variables that are in the complete dataset. This enables it to work perfectly for imbalanced datasets, but not for time-series data.

Stratified K-Fold Cross Validation



5.4.1 Advantages:

1. **Handles Imbalance:** Stratified K-Fold ensures that each fold has a representative distribution of classes, which can lead to more accurate performance estimates, especially in cases of imbalanced data.
2. **Better Generalization:** By maintaining class proportions, the performance estimate can better reflect how the model will perform on unseen data.

5.4.2 Disadvantages:

1. **Increased Complexity:** Stratified K-Fold can be slightly more computationally intensive than regular K-Fold, as it requires additional processing to maintain class proportions.

5.4.3 Application:

Stratified K-Fold CV is commonly used when the dataset has imbalanced classes and the goal is to get accurate performance estimates for each class.

5.4.4 Libraries and Modules Used:

- [sklearn.model_selection.StratifiedKFold](#)

```

1 from sklearn.model_selection import StratifiedKFold
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.metrics import accuracy_score
4
5 # Load your dataset and split it into features (X) and target (y)
6 skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
7
8 accuracy_scores = []
9
10 for train_index, val_index in skf.split(X, y):
11     X_train, X_val = X[train_index], X[val_index]
12     y_train, y_val = y[train_index], y[val_index]
13
14     # Create and train a model
15     model = LogisticRegression()
16     model.fit(X_train, y_train)
17
18     # Make predictions on the validation set
19     y_pred = model.predict(X_val)
20
21     # Calculate accuracy and store it
22     accuracy = accuracy_score(y_val, y_pred)
23     accuracy_scores.append(accuracy)
24
25 # Calculate the average accuracy over all folds
26 average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
27 print("Average Stratified Cross-Validation Accuracy:", average_accuracy)
28

```

5.5 Group K-Fold Cross-Validation:

Group K-Fold Cross-Validation is a variation of K-Fold Cross-Validation designed for scenarios where data points are grouped or clustered in a way that they should remain together in the same fold during cross-validation. This is particularly useful when dealing with data that exhibits group-level dependencies or when there are constraints that require keeping certain groups together.

5.5.1 Advantages:

1. **Group Integrity:** Group K-Fold ensures that data points within the same group are always together in the same fold, preserving group-level dependencies.
2. **Realistic Performance Estimates:** By maintaining group coherence, Group K-Fold provides performance estimates that better reflect real-world scenarios.

5.5.2 Disadvantages:

1. **Complexity:** Implementing Group K-Fold requires additional logic to handle grouping constraints, potentially making the code more complex.
2. **Uneven Group Sizes:** Unequal group sizes can lead to imbalanced folds, affecting the representativeness of the performance estimate.

5.5.3 Application:

Group K-Fold CV is commonly used in situations where data points are grouped or clustered, such as analyzing medical data from different patients or time series data from multiple subjects.

5.5.4 Libraries and Modules Used

- [`sklearn.model_selection.GroupKFold`](#)

```
1 from sklearn.model_selection import GroupKFold
2 from sklearn.svm import SVC
3 from sklearn.metrics import accuracy_score
4
5 # Load your dataset and split it into features (X), target (y), and groups (groups)
6 gkf = GroupKFold(n_splits=5)
7
8 accuracy_scores = []
9
10 for train_index, val_index in gkf.split(X, y, groups):
11     X_train, X_val = X[train_index], X[val_index]
12     y_train, y_val = y[train_index], y[val_index]
13
14     # Create and train a model
15     model = SVC()
16     model.fit(X_train, y_train)
17
18     # Make predictions on the validation set
19     y_pred = model.predict(X_val)
20
21     # Calculate accuracy and store it
22     accuracy = accuracy_score(y_val, y_pred)
23     accuracy_scores.append(accuracy)
24
25 # Calculate the average accuracy over all folds
26 average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
27 print("Average Group K-Fold Cross-Validation Accuracy:", average_accuracy)
28
```

5.6 Leave-p-out cross-validation

Leave-P-Out Cross-Validation (LPOCV) is a variant of cross-validation where P samples are left out from the dataset for validation, and the model is trained on the remaining N-P samples. This process is repeated for all possible combinations of leaving P samples out. LPOCV provides an exhaustive evaluation of the model's performance by considering every possible subset of the data.

The technique, which has a high computation time, produces good results. However, it's not considered ideal for an imbalanced dataset and is deemed to be a computationally unfeasible method. This is because if the training set has all samples of one class, the

model will not be able to properly generalize and will become biased to either of the classes.

5.6.1 Advantages:

1. Exhaustive Evaluation: LPOCV evaluates the model on every possible subset of the data, providing a comprehensive estimate of its performance.
2. Robustness: LPOCV is less dependent on a single data split, leading to a more reliable performance estimate.

5.6.2 Disadvantages:

1. Computational Complexity: LPOCV can be computationally intensive, especially for large datasets, as it requires training and evaluating the model for multiple subsets of data.
2. Resource Intensive: LPOCV may require substantial memory and processing power for larger values of P .

5.6.3 Application:

LPOCV is useful when a very accurate estimate of a model's performance is desired, but it is computationally expensive and might not be practical for very large datasets.

5.6.4 Libraries and Modules Used:

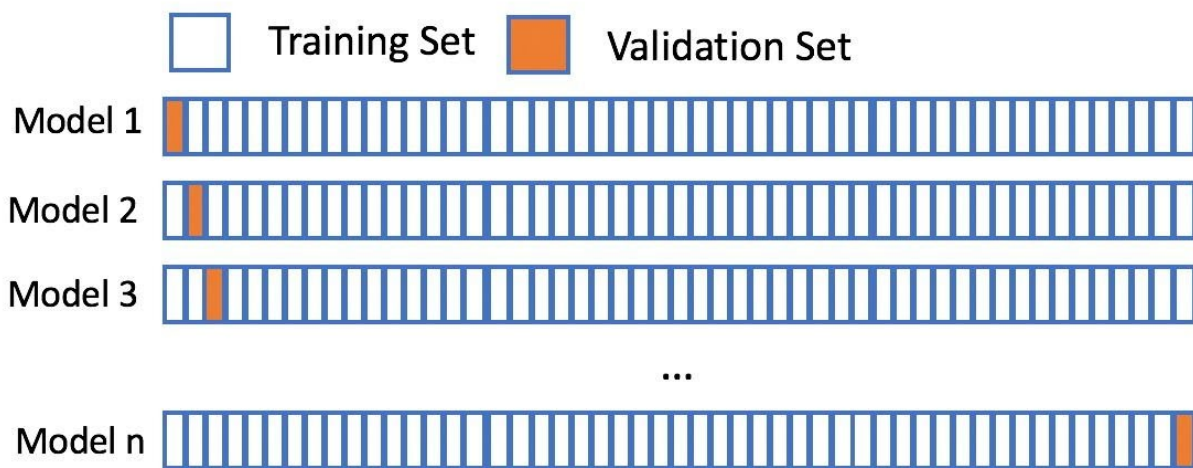
- [sklearn.model_selection.LeavePOut](#)

```
1 from sklearn.model_selection import LeavePOut
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.metrics import accuracy_score
4
5 # Load your dataset and split it into features (X) and target (y)
6 lpo = LeavePOut(p=2) # Change the value of 'p' as needed
7
8 accuracy_scores = []
9
10 for train_index, val_index in lpo.split(X):
11     X_train, X_val = X[train_index], X[val_index]
12     y_train, y_val = y[train_index], y[val_index]
13
14     # Create and train a model
15     model = LogisticRegression()
16     model.fit(X_train, y_train)
17
18     # Make predictions on the validation set
19     y_pred = model.predict(X_val)
20
21     # Calculate accuracy and store it
22     accuracy = accuracy_score(y_val, y_pred)
23     accuracy_scores.append(accuracy)
24
25 # Calculate the average accuracy over all combinations
26 average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
27 print("Average Leave-P-Out Cross-Validation Accuracy:", average_accuracy)
28
```

5.7 Leave-one-out cross-validation

Leave-One-Out Cross-Validation (LOOCV) is a special case of Leave-P-Out Cross-Validation where P is set to 1. In LOOCV, one data point is left out as the validation set, and the model is trained on the remaining $N-1$ data points. This process is repeated for all N data points. LOOCV provides an exhaustive evaluation of the model's performance and is particularly useful when the dataset is small.

Consider there are 1000 instances in your dataset. In each iteration, 1 instance will be used for the validation set and the remaining 999 instances will be used as the training set. The process repeats itself until every instance from the dataset is used as a validation sample.



5.7.1 Advantages:

1. **Exhaustive Evaluation:** LOOCV evaluates the model on every possible subset of data, providing a comprehensive estimate of its performance.
2. **Low Bias:** LOOCV provides an unbiased estimate of the model's performance since it trains on $N-1$ samples, similar to the final model.

5.7.2 Disadvantages:

1. **Computational Intensity:** LOOCV can be computationally expensive, as it requires training and evaluating the model N times, where N is the size of the dataset.
2. **Resource Demands:** LOOCV may require substantial memory and processing power, especially for large datasets.

5.7.3 Application:

LOOCV is commonly used when the dataset is small, and a very accurate estimate of the model's performance is required. It can be useful for evaluating model performance, comparing algorithms, and selecting the best model.

5.7.4 Libraries and Modules Used:

- [sklearn.model_selection.LeaveOneOut](#)

```
1 from sklearn.model_selection import LeaveOneOut
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.metrics import accuracy_score
4
5 # Load your dataset and split it into features (X) and target (y)
6 loo = LeaveOneOut()
7
8 accuracy_scores = []
9
10 for train_index, val_index in loo.split(X):
11     X_train, X_val = X[train_index], X[val_index]
12     y_train, y_val = y[train_index], y[val_index]
13
14     # Create and train a model
15     model = LogisticRegression()
16     model.fit(X_train, y_train)
17
18     # Make a prediction on the validation set
19     y_pred = model.predict(X_val)
20
21     # Calculate accuracy and store it
22     accuracy = accuracy_score(y_val, y_pred)
23     accuracy_scores.append(accuracy)
24
25 # Calculate the average accuracy over all data points
26 average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
27 print("Average Leave-One-Out Cross-Validation Accuracy:", average_accuracy)
28
```

5.8 Monte Carlo (shuffle-split)

Also known as shuffle split cross-validation and repeated random subsampling cross-validation, the Monte Carlo technique involves splitting the whole data into training data and test data. Splitting can be done in the percentage of 70-30% or 60-40% - or anything you prefer. The only condition for each iteration is to keep the train-test split percentage different.

The next step is to fit the model on the train data set in that iteration and calculate the accuracy of the fitted model on the test dataset. Repeat these iterations many times - 100,400,500 or even higher - and take the average of all the test errors to conclude how well your model performs.

It is similar to other cross-validation techniques but involves random sampling to create different splits in each iteration.

For a 100 iteration run, the model training will look like this:



You can see that in each iteration, the split ratio of the training set and test set is different. The average has been taken to get the test errors.

5.8.1 Advantages:

1. Randomness: MC-CV introduces randomness into the data splitting process, reducing the dependency on a single data split and providing a more robust performance estimate.
2. Representative Sampling: MC-CV can better capture the variability in the data and model performance, especially in cases where specific data splits might result in biased estimates.

5.8.2 Disadvantages:

1. Computational Complexity: MC-CV can be computationally intensive as it involves multiple iterations of training and validation.
2. Variability: The performance estimates can still be influenced by random variations in data splits.

5.8.3 Application:

MC-CV is used when the goal is to obtain a more reliable performance estimate that accounts for the inherent variability in data and model training.

5.8.4 Libraries and Modules Used:

```
1 import numpy as np
2 from sklearn.model_selection import train_test_split
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.metrics import accuracy_score
5
6 # Load your dataset and split it into features (X) and target (y)
7 num_iterations = 10 # Change the number of iterations as needed
8
9 accuracy_scores = []
10
11 for _ in range(num_iterations):
12     X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=np.random.randint(100))
13
14     # Create and train a model
15     model = LogisticRegression()
16     model.fit(X_train, y_train)
17
18     # Make predictions on the validation set
19     y_pred = model.predict(X_val)
20
21     # Calculate accuracy and store it
22     accuracy = accuracy_score(y_val, y_pred)
23     accuracy_scores.append(accuracy)
24
25 # Calculate the average accuracy over all iterations
26 average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
27 print("Average Monte Carlo Cross-Validation Accuracy:", average_accuracy)
28
```

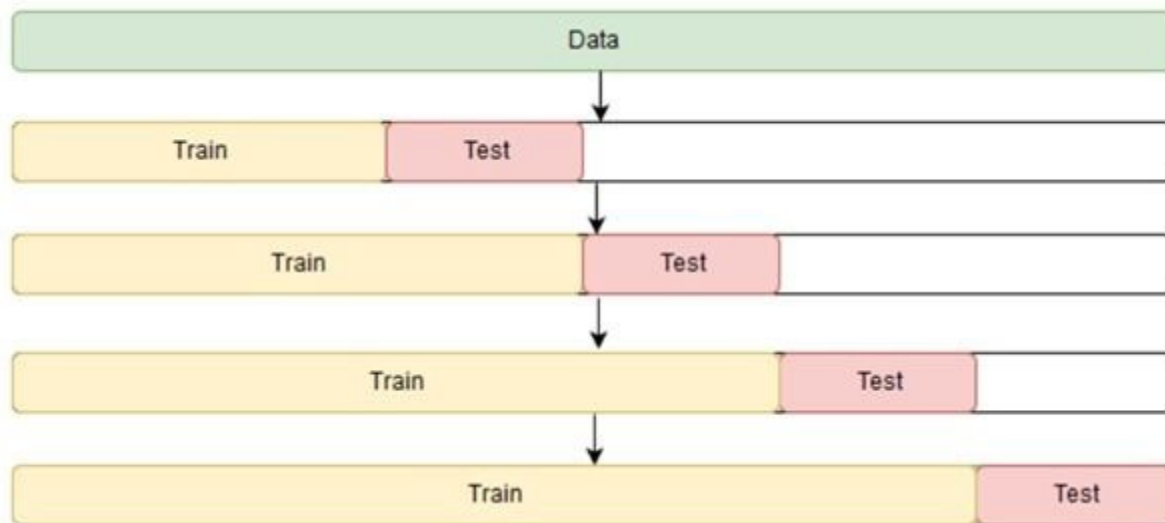
5.9 Time series (rolling cross-validation)

Time series is the type of data collected at different points in time. This kind of data allows one to understand what factors influence certain variables from period to period.

In the case of time series datasets, the cross-validation is not that trivial. You can't choose data instances randomly and assign them the test set or the train set. Hence, this technique is used to perform cross-validation on time series data with time as the important factor.

Since the order of data is very important for time series-related problems, the dataset is split into training and validation sets according to time. Therefore, it's also called the forward chaining method or rolling cross-validation.

Start the training with a small subset of data. Perform forecasting for the later data points and check their accuracy. The forecasted data points are then included as part of the next training dataset and the next data points are forecasted. The process goes on.



5.9.1 Advantages:

1. Temporal Realism: Time Series CV mimics the real-world scenario where the model is tested on unseen future data, making it suitable for time-dependent tasks.
2. Prevents Data Leakage: By maintaining the temporal order, Time Series CV avoids data leakage that might occur with traditional cross-validation.

5.9.2 Disadvantages:

1. Limited Data for Training: In time series data, as you move forward in time, the number of available data points for training decreases, which might affect model performance.
2. Complexity: Time Series CV requires careful implementation to ensure proper temporal order and handling of sequences.

5.9.3 Application:

Time Series CV is commonly used for evaluating and tuning models on time-dependent tasks, such as financial forecasting, weather prediction, and sales forecasting.

5.9.4 Libraries and Modules Used:

- [`sklearn.model_selection.TimeSeriesSplit`](#)

```

1 from sklearn.model_selection import TimeSeriesSplit
2 from sklearn.linear_model import LinearRegression
3 from sklearn.metrics import mean_squared_error
4
5 # Load your time series data
6 # Assuming 'X' contains features and 'y' contains target values
7
8 tscv = TimeSeriesSplit(n_splits=5)
9
10 mse_scores = []
11
12 for train_index, val_index in tscv.split(X):
13     X_train, X_val = X[train_index], X[val_index]
14     y_train, y_val = y[train_index], y[val_index]
15
16     # Create and train a model
17     model = LinearRegression()
18     model.fit(X_train, y_train)
19
20     # Make predictions on the validation set
21     y_pred = model.predict(X_val)
22
23     # Calculate Mean Squared Error (MSE) and store it
24     mse = mean_squared_error(y_val, y_pred)
25     mse_scores.append(mse)
26
27 # Calculate the average MSE over all splits
28 average_mse = sum(mse_scores) / len(mse_scores)
29 print("Average Time Series Cross-Validation MSE:", average_mse)
30

```

5.10 Nested Cross-Validation:

Nested Cross-Validation is a technique that involves using an outer loop of cross-validation to evaluate the model's generalization performance, and an inner loop of cross-validation to perform model selection or hyperparameter tuning. It helps in obtaining a more reliable estimate of a model's performance while making informed decisions about model configuration.

5.10.1 Advantages:

1. **Robust Performance Estimate:** Nested CV provides a more accurate and unbiased estimate of a model's performance by avoiding overfitting during hyperparameter tuning.
2. **Informed Model Selection:** The inner loop helps in selecting the best model configuration, leading to better model generalization.

5.10.2 Disadvantages:

1. Increased Computational Complexity: Nested CV requires running multiple cross-validation loops, which can be computationally intensive.
2. Time-Consuming: The added complexity can lead to longer computation times, especially for large datasets or complex models.

5.10.3 Application:

Nested Cross-Validation is commonly used when model selection and hyperparameter tuning are important and overfitting is a concern. It is especially useful when the dataset is small or when the model has many hyperparameters.

5.10.4 Libraries and Modules Used:

```
1 from sklearn.model_selection import cross_val_score, GridSearchCV
2 from sklearn.ensemble import RandomForestClassifier
3
4 # Load your dataset and split it into features (X) and target (y)
5 model = RandomForestClassifier()
6
7 # Define parameter grid for hyperparameter tuning
8 param_grid = {
9     'n_estimators': [50, 100, 200],
10    'max_depth': [None, 10, 20],
11    'min_samples_split': [2, 5, 10]
12 }
13
14 # Outer loop of cross-validation for performance estimation
15 outer_cv = 5 # Number of outer CV folds
16 outer_scores = cross_val_score(model, X, y, cv=outer_cv)
17
18 # Inner loop of cross-validation for hyperparameter tuning
19 inner_cv = 3 # Number of inner CV folds
20 grid_search = GridSearchCV(model, param_grid, cv=inner_cv)
21 grid_search.fit(X, y)
22
23 # Print results
24 print("Outer CV Scores:", outer_scores)
25 print("Best Parameters:", grid_search.best_params_)
26 print("Best Score:", grid_search.best_score_)
27
```

5.11 Other Cross validation techniques

- Stratified Shuffle Split Method
- Group Shuffle Split Method
- Leave-One-Group-Out Method

- Leave-P-Group-Out Method
- Blocked Cross-Validation Method

6 Libraries and tools for model evaluation

6.1 Open Source Non-Automated Model Evaluation Tools:

- **scikit-learn:**
 - A widely-used machine learning library for classification, regression, clustering, and more, with manual model evaluation tools.
 - Reference: <https://scikit-learn.org/stable/>
- **TensorFlow Model Analysis (TFMA):**
 - Part of the TensorFlow ecosystem, providing tools for manual evaluation and validation of machine learning models.
 - Reference: <https://www.tensorflow.org/tfx/guide/tfma>
- **Yellowbrick:**
 - A visualization library for manual machine learning model evaluation, providing various diagnostic plots.
 - Reference: <https://www.scikit-yb.org/en/latest/>
- **Caret:**
 - A comprehensive package for model evaluation and selection in R, with manual tools.
 - Reference: <https://topepo.github.io/caret/>
- **MLflow:**
 - An open source platform for managing the end-to-end machine learning lifecycle, including manual model evaluation and tracking.
 - Reference: <https://mlflow.org/>

6.2 Open Source Automated Model Evaluation Tools:

6. **Auto-sklearn:**
 - An automated machine learning toolkit that includes model selection, hyperparameter optimization, and cross-validation.
 - Reference: <https://automl.github.io/auto-sklearn/master/>
7. **TPOT (Tree-based Pipeline Optimization Tool):**
 - A Python library that automates the process of selecting models and optimizing their hyperparameters.
 - Reference: <http://epistasislab.github.io/tpot/>
8. **AutoGluon:**
 - A deep learning toolkit that automates hyperparameter tuning and architecture search for neural networks.
 - Reference: <https://auto.gluon.ai/stable/index.html>
9. **AutoKeras:**
 - An open source AutoML library that automates the selection and tuning of neural network architectures.
 - Reference: <https://autokeras.com/>
10. **Google Cloud AutoML:**
 - A suite of AutoML tools provided by Google Cloud that automate model evaluation, selection, and deployment.

- Reference: <https://cloud.google.com/automl>

6.3 Open Source AI-Powered Model Evaluation Tools:

11. EvalAI:

- An AI-powered platform for evaluating and comparing machine learning algorithms and models.
- Reference: <https://evalai.cloudcv.org/>

12. Lobe:

- An AI-powered platform that enables you to build, train, and deploy custom machine learning models without coding.
- Reference: <https://lobe.ai/>

13. MonkAI:

- An AI-powered tool that simplifies machine learning model building, training, and evaluation.
- Reference: <https://cleverfranke.github.io/monkai/>

14. Diffgram:

- An AI-powered platform for labeling and managing data, aiding in model evaluation and data preparation.
- Reference: <https://diffgram.com/>

15. Spell:

- An AI-powered platform for machine learning experimentation and model evaluation in the cloud.
- Reference: <https://spell.ml/>

6.4 Closed Source / Commercial Non-Automated Model Evaluation Tools:

16. DataRobot:

- An automated machine learning platform that provides model evaluation and selection capabilities.
- Reference: <https://www.datarobot.com/>

17. IBM Watson Studio:

- A cloud-based platform that includes tools for building, training, and evaluating machine learning models.
- Reference: <https://www.ibm.com/cloud/watson-studio>

18. Databricks:

- A unified analytics platform that provides tools for building, training, and evaluating machine learning models at scale.
- Reference: <https://databricks.com/>

6.5 Closed Source / Commercial Automated Model Evaluation Tools:

19. H2O.ai Driverless AI:

- A commercial AutoML platform that automates the end-to-end process of building, deploying, and maintaining machine learning models.
- Reference: <https://www.h2o.ai/products/h2o-driverless-ai/>

20. IBM Watson AutoAI:

- A component of IBM Watson Studio that automates the process of building, evaluating, and deploying machine learning models.
- Reference: <https://www.ibm.com/cloud/watson-studio/autoai>

21. **DarwinAI:**

- A commercial platform that automates the optimization of neural network architectures and hyperparameters.
- Reference: <https://www.darwinai.com/>

22. **Microsoft Azure Machine Learning:**

- A cloud-based platform with tools for model evaluation, deployment, and monitoring.
- Reference: <https://azure.microsoft.com/en-us/services/machine-learning/>

23. **RapidMiner:**

- A data science platform with machine learning capabilities, including model evaluation and deployment.
- Reference: <https://rapidminer.com/>

7 Interview Questions

7.1 Can you explain the difference between precision and recall in the context of evaluation metrics for machine learning models?

Precision and recall are two commonly used evaluation metrics in the context of machine learning models. Precision is a measure of the number of true positive predictions made by the model out of all positive predictions. Precision represents the ability of the model to avoid false positive predictions.

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

On the other hand, recall is a measure of the number of true positive predictions made by the model out of all actual positive instances in the dataset. The recall represents the ability of the model to correctly identify all positive instances.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

Both precision and recall are important evaluation metrics, but the trade-off between them depends on the specific problem being solved and the requirements of the stakeholders. For example, in medical diagnosis, recall might be more important because it is essential to identify all cases of a disease, even if this results in a higher rate of false positives. In contrast, in fraud detection, precision might be more important because it is essential to avoid false accusations, even if this results in a higher rate of false negatives.

7.2 How do you choose an appropriate evaluation metric for a given problem?

Choosing an appropriate evaluation metric for a given problem is a critical aspect of the model development process. In selecting an evaluation metric, it is important to consider

the nature of the problem and the goals of the analysis. Some common factors to consider include:

Problem type: Is it a binary classification problem, multi-class classification problem, regression problem, or something else?

Business objective: What is the end goal of the analysis, and what kind of performance is desired? For example, if the objective is to minimize false negatives, recall would be a more important metric than precision.

Dataset characteristics: Are the classes balanced or imbalanced? Is the dataset large or small?

Data quality: What is the quality of the data and how much noise is present in the dataset?

Based on these factors, a data scientist may choose an evaluation metric such as accuracy, F1-score, AUC-ROC, Precision-Recall, mean squared error, etc. It is important to also keep in mind that often, multiple evaluation metrics are used to get a complete understanding of model performance.

7.3 Can you discuss the use of the F1 score as a measure of model performance?

The F1 score is a commonly used evaluation metric in machine learning that balances precision and recall. Precision measures the proportion of true positive predictions out of all positive predictions made by the model, while recall measures the proportion of true positive predictions out of all actual positive observations. The F1 score is the harmonic

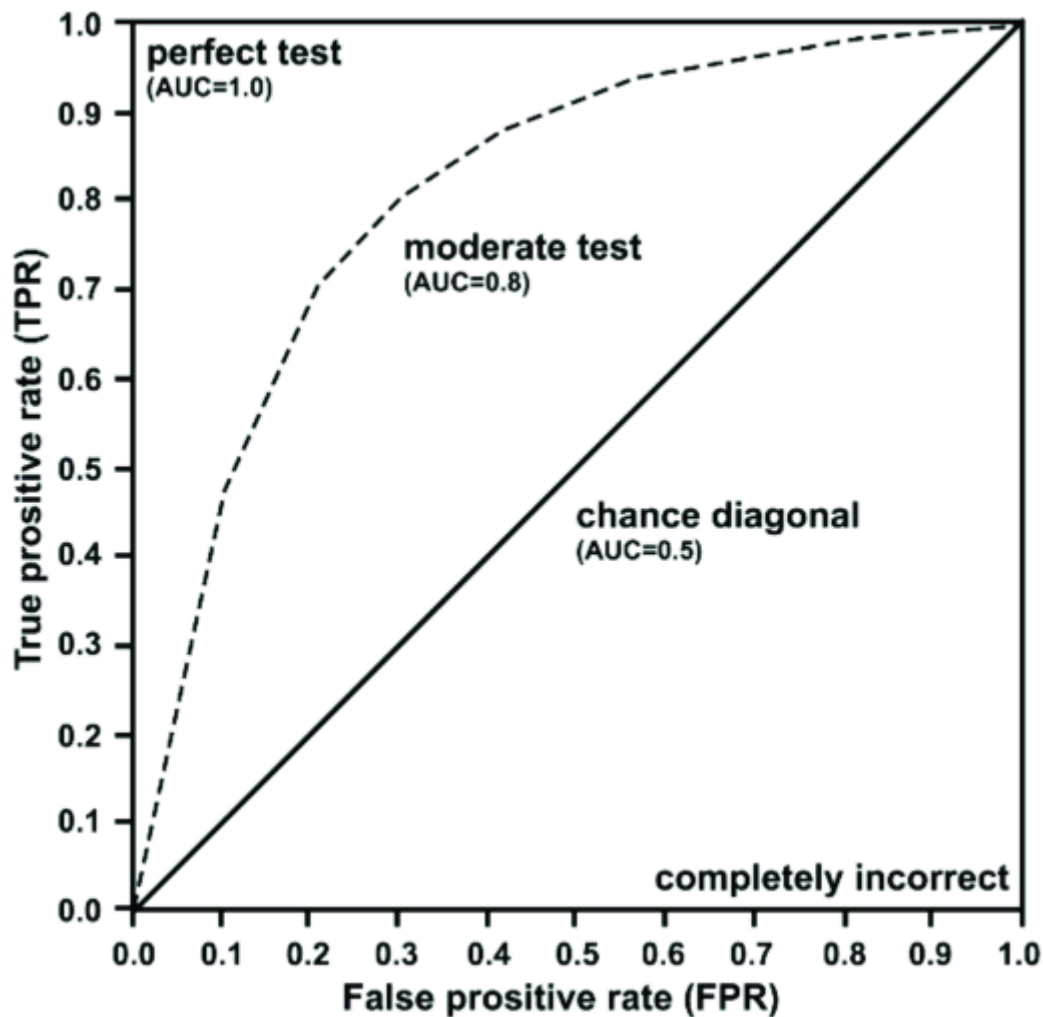
mean of precision and recall and is often used as a single metric to summarize the performance of a binary classifier.

$$F1 = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

In cases where a model must make a trade-off between precision and recall, the F1 score provides a more nuanced evaluation of performance than using precision or recall alone. For example, in cases where false positive predictions are more costly than false negatives, it may be more important to optimize for precision, while in cases where false negatives are more costly, the recall may be prioritized. The F1 score can be used to assess the performance of a model in these scenarios and to make informed decisions about how to adjust its threshold or other parameters to optimize performance.

7.4 Can you explain the use of the Receiver Operating Characteristic (ROC) curve in model evaluation?

The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model that plots the True Positive Rate (TPR) against the False Positive Rate (FPR). It helps in evaluating the trade-off between the sensitivity (True Positives) and specificity (True Negatives) of the model and is widely used for evaluating models that make predictions based on binary classification outcomes, such as Yes or No, Pass or Fail, and so on.



ROC Curve

The ROC curve is used to measure the performance of a model by comparing the model's prediction with the actual outcome. A good model will have a large area under the ROC curve, which means that it is able to accurately distinguish between positive and negative classes. In practice, ROC AUC (Area Under the Curve) is used to compare the performance of different models and is a good way to evaluate the performance of models when the outcome classes are imbalanced.

7.5 How do you determine the optimal threshold for a binary classification model?

The optimal threshold for a binary classification model is determined by finding the threshold that balances the trade-off between precision and recall. This can be done by using evaluation metrics such as the F1 score, which balances precision and recall, or by using the ROC curve, which plots the true positive rate against the false positive rate for various thresholds. The optimal threshold is typically chosen as the point on the ROC curve that is closest to the top left corner, as this maximizes the true positive rate while minimizing the false positive rate. In practice, the optimal threshold may also depend on the specific goals of the problem and the cost associated with false positives and false negatives.

7.6 Can you discuss the trade-off between precision and recall in model evaluation?

The trade-off between precision and recall in model evaluation refers to the trade-off between correctly identifying positive instances (recall) and correctly identifying only the positive instances (precision). A high precision means that the number of false positives is low, while a high recall means that the number of false negatives is low. However, it is often not possible to maximize both precision and recall simultaneously for a given model. To make this trade-off, you need to consider the specific goals and requirements of your problem and choose the evaluation metric that aligns with them.

7.7 How do you evaluate the performance of a clustering model?

The performance of a clustering model can be evaluated using a number of metrics. Some common metrics include:

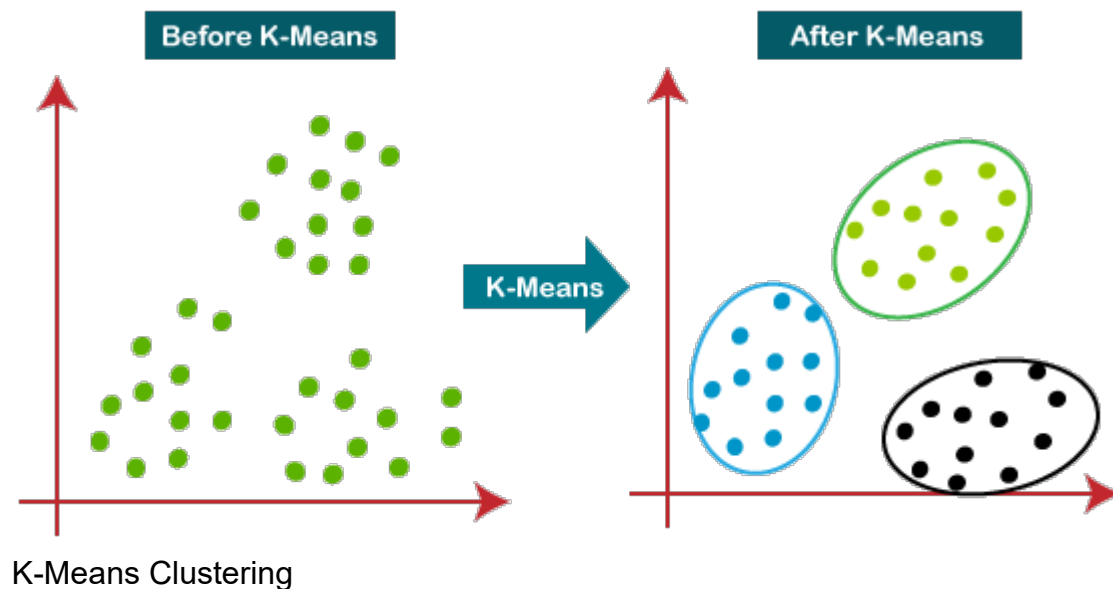
Silhouette score: It measures the similarity of observation to its own cluster compared to other clusters. The score ranges from -1 to 1, with values closer to 1 indicating a stronger clustering structure.

Calinski-Harabasz Index: It measures the ratio of the between-cluster variance to the within-cluster variance. Higher values indicate a better clustering solution.

Davies-Bouldin Index: It measures the average similarity between each cluster and its most similar cluster. Lower values indicate a better clustering solution.

Adjusted Rand Index: It measures the similarity between the true class labels and the predicted cluster labels, adjusted for the chance. Higher values indicate a better clustering solution.

Confusion matrix: It can be used to evaluate the accuracy of clustering models by comparing the predicted clusters to the true classes.



It's important to note that choosing the appropriate evaluation metric depends on the specific problem and the goals of the clustering analysis.

7.8 Can you explain the difference between accuracy, precision, recall, and F1-score in the context of multi-class classification problems?

Here is the comparison between accuracy, precision, recall, and F1-score in the context of multi-class classification problems in tabular format:

Metric	Definition	Use Case
Accuracy	The proportion of correctly classified instances (both true positive and true negative) over all instances.	Measures the overall performance of a classifier
Precision	The proportion of correctly classified positive instances over all instances that are classified as positive.	Measures the ability of the classifier to avoid false positives
Recall	The proportion of correctly classified positive instances over all actual positive instances.	Measures the ability of the classifier to identify all actual positive instances
F1-Score	The harmonic mean of precision and recall, providing a balanced measure of both precision and recall.	A good indicator of the performance of a classifier when the number of positive and negative instances is unbalanced

Comparison between Accuracy, Precision, Recall, and F1-Score

7.9 How do you evaluate the performance of a recommendation system?

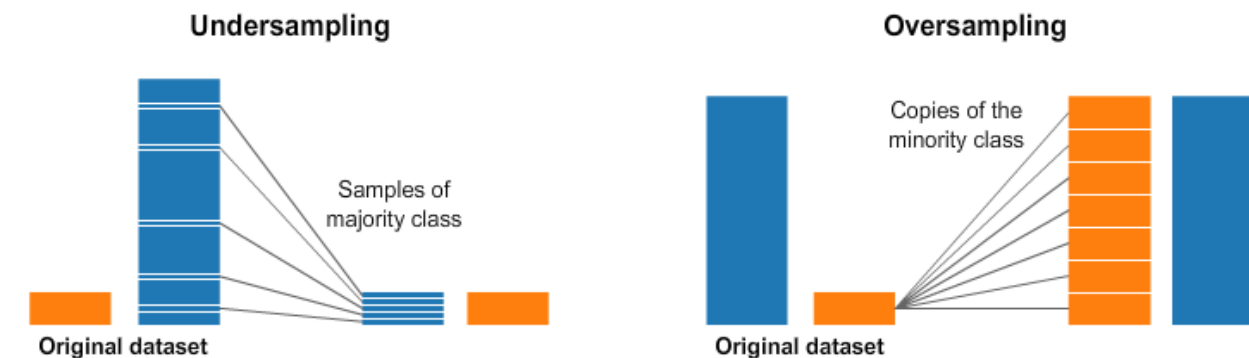
Evaluating the performance of a recommendation system involves measuring the effectiveness and efficiency of the system in recommending relevant items to users. Some common metrics used to evaluate the performance of recommendation systems include:

1. **Precision:** The proportion of recommended items that are relevant to the user.

2. **Recall:** The proportion of relevant items that are recommended by the system.
3. **F1-Score:** The harmonic mean of precision and recall.
4. **Mean Average Precision (MAP):** A measure of the average precision of a recommendation system's overall users.
5. **Normalized Discounted Cumulative Gain (NDCG):** A measure of the rank-weighted relevance of the recommended items.
6. **Root Mean Square Error (RMSE):** A measure of the difference between the predicted ratings and the actual ratings for a set of items.
- 7.

7.10 How do you handle imbalanced datasets when evaluating a model's performance?

To handle imbalanced datasets in model evaluation, there are several techniques that can be used:



Imbalance Dataset solving Methods

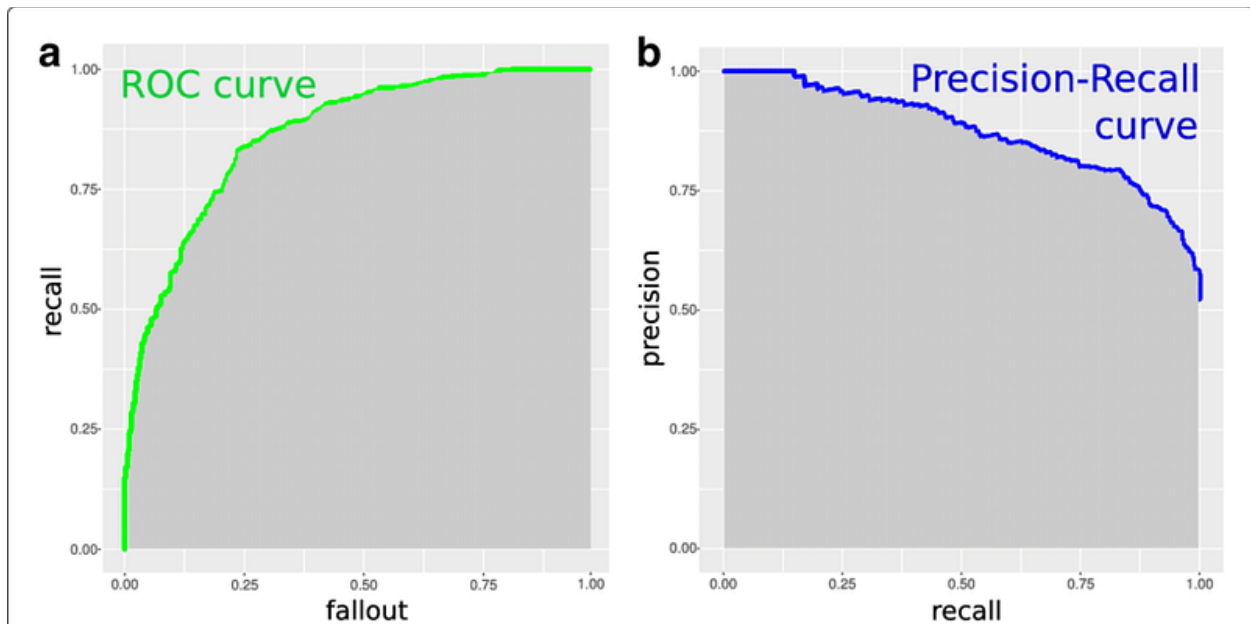
1. **Resampling the dataset:** This involves either oversampling the minority class or undersampling the majority class to balance the class distribution.

2. **Using different evaluation metrics:** Metrics such as precision, recall, F1-score, and area under the ROC curve (AUC-ROC) are sensitive to class imbalance and can provide a better understanding of the model's performance on an imbalanced dataset.
3. **Using cost-sensitive learning:** This involves assigning a cost to different types of misclassification, such as assigning a higher cost to false negatives than false positives, to make the model more sensitive to the minority class.
4. **Using ensemble methods:** Techniques such as bagging, boosting, and stacking can be used to improve model performance on imbalanced datasets by combining the results from multiple models.
5. **Using hybrid methods:** A combination of the above techniques can be used to handle imbalanced datasets in model evaluation.

7.11 Compare ROC and Precision-Recall Curve.

ROC curve is derived from signal detection theory. A ROC curve for a given model represents the trade-off between the true positive rate (TPR) and the false positive rate (FPR). TPR or Recall is the ratio of positive tuples that are correctly labeled and FPR or (1-Specificity) is the ratio of negative tuples that mislabelled as positive.

The precision-recall curve shows the tradeoff between precision and recall. ROC curves recommended for balanced class problems while Precision recall curves recommended for the imbalance class problems.



https://www.researchgate.net/figure/a-Example-of-Precision-Recall-curve-with-the-precision-score-on-the-y-axis-and-the-fig1_321672019

7.12 What are false positives and false negatives?

False positives are the negative tuples that were incorrectly labeled as positive but it is actually negative. False negatives are the positive tuples were incorrectly labeled as negative but actually, it is positive.

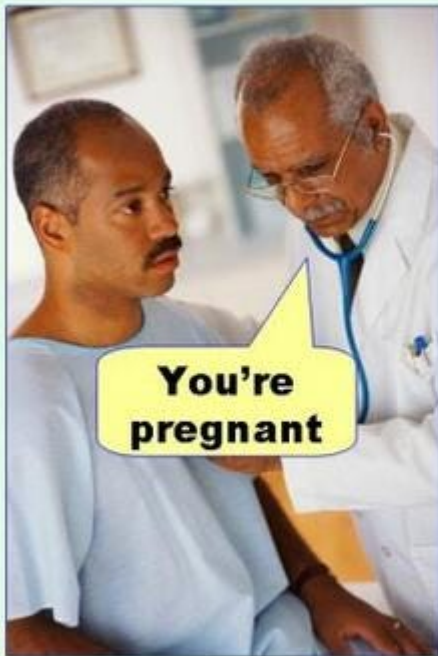
In the case of cancer prediction, false negatives are more important than false positives because it is worse saying someone does not have cancer but actually he has.

7.13 What is the difference between Type 1 and Type 2 errors?

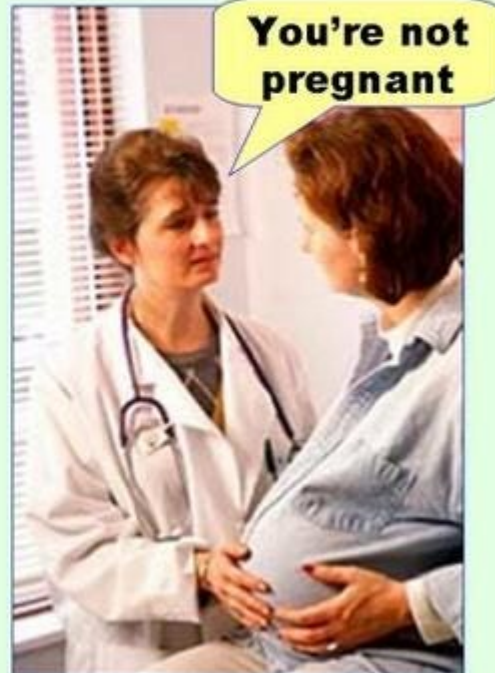
Type 1 error is classified as a false positive. Here, incorrectly labeled as positive but it is actually negative. For example, False fire alarm. The alarm rings but there is no fire.

Type 2 error is classified as a false negative. Here, positive tuples were incorrectly labeled as negative but actually, it is positive.

Type I error
(false positive)



Type II error
(false negative)



<https://chemicalstatistician.wordpress.com/2014/05/12/applied-statistics-lesson-of-the-day-type-i-error-false-positive-and-type-2-error-false-negative/>

7.14 How can you handle imbalanced classes in a logistic regression model?

A. There are several ways to handle imbalanced classes in a logistic regression model. Some approaches include:

Undersampling the majority class: This involves randomly selecting a subset of the majority class samples to use in training the model. This can help to balance the class distribution, but it may also throw away valuable information.

Oversampling the minority class: This involves generating synthetic samples of the minority class to add to the training set. One popular method for generating synthetic samples is called SMOTE (Synthetic Minority Oversampling Technique).

Adjusting the class weights: Many machine learning algorithms allow you to adjust the weighting of each class. In logistic regression, you can do this by setting the `class_weight` parameter to “balanced”. This will automatically weight the classes inversely proportional to their frequency, so that the model pays more attention to the minority class.

Using a different evaluation metric: In imbalanced classification tasks, it is often more informative to use evaluation metrics that are sensitive to class imbalance, such as precision, recall, and the F1 score.

Using a different algorithm: Some algorithms, such as decision trees and Random Forests, are more robust to imbalanced classes and may perform better on imbalanced datasets.

7.15 How do you evaluate a model's performance for a multi-class classification problem?

A. One approach for evaluating a multi-class classification model is to calculate a separate evaluation metric for each class, and then calculate a macro or micro average. The macro average gives equal weight to all the classes, while the micro average gives more weight to the classes with more observations. Additionally, some commonly used metrics for multi-class classification problems such as confusion matrix, precision, recall, F1 score, Accuracy and ROC-AUC can also be used.

7.16 Is classification accuracy used as a metric in imbalanced datasets? Why or Why not?

Accuracy should never be used as a metric in imbalanced datasets.

Classification accuracy measures how many positive and negative data points were correctly classified. Sounds simple, but it is one of the most commonly used metrics.

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

Classification accuracy is a good metric when the dataset is balanced or slightly skewed. However, as the skewness increases, i.e., there are many more positive data points compared to negative ones (or vice-versa), accuracy tends to favor the majority class over the minority classes.

When the data is imbalanced, the model will get a super high accuracy even if it simply classifies every data point into the majority class. Even though the model is not useful, accuracy will not be able to tell.

7.17 When is precision considered an important metric?

Precision answers the following question:

What proportion of observations labeled as positive are actually positive (True positive)?

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

- A **precision score towards 1** implies that the classifier has a very strict criteria in predicting a sample as positive. In other words, if a model labels a data point as positive, it is highly likely to be positive.
- A **low precision score** means the model predicts high number of False positives which could be due to severe data skewness or could be a result of the threshold value chosen.
- Precision is dependent on both positive and negative samples.
- Precision is an important metric when we want to reduce the number of False positives predicted (Type I error) by the model.

- The problem with precision is that it does not give any information about False Negatives, i.e., how many positive data points did the model fail to classify as positive.

7.18 When is recall considered an important metric?

Recall also known as "Sensitivity" or "True positive rate" answers the following question:

What proportion of positive class observations are labeled correctly as positive (True positives)?

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


- A **recall score towards 1** implies that the classifier is very good at predicting positive classes correctly. In other words, the model labels most positive data points correctly as positive.
- A **low recall score** means the model predicts high number of false negatives which could be due to severe data skewness or could be a result of the threshold value chosen.
- Recall is dependent *only* on positive samples.
- Recall is an important metric when we want to reduce the number of False negatives (Type II error) predicted by the model.
- The problem with recall is that it does not give any information about False positives.


7.19 How does threshold impact Precision and Recall?

In order to label the predictions using a model, we set a threshold value above which all the model labels the data points as *positive* and below that threshold the data points are labeled as *negative*.

All the positive samples above the threshold value are the **True Positives** while the negative samples are labeled as **False Positives**. Similarly, below the threshold value,

all the negative samples will be **True Negatives** and all the positive values will be **False Negatives**.

As **threshold**  - More samples are predicted as negative. In other words, TN and FN increase, which in turn will increase the precision score and reduce the recall score.

As **threshold**  - More samples are predicted as positive. In other words, TP and FP increase, which in turn will increase the recall score and reduce the precision score.

7.20 Say you are building a ML classifier to detect fraudulent insurance claims. Which is a better metric to evaluate your model - Precision or Recall?

Remember: These type of questions are asked not to analyze whether you know the definitions of precision and recall, but to understand how well you understand and relate the metric to the business use case.

Let's say the ML classifier labels fraudulent claims into the positive class and non-fraudulent classes into the negative class. Unable to flag fraudulent claims as positive can translate to huge loss for insurance companies. This implies that reducing the number of False Negatives, i.e., fraudulent claims classified as non-fraudulent, would be helpful in saving the company from huge loss due to claims payout. This means you want a model with high recall.

Now, is the precision important in this case? Well, the answer is yes!

Insurance companies also care about keeping their precision high. Otherwise, if the precision is low and the model predicts more number of False Positives, they have to spend money to manually investigate those claims.

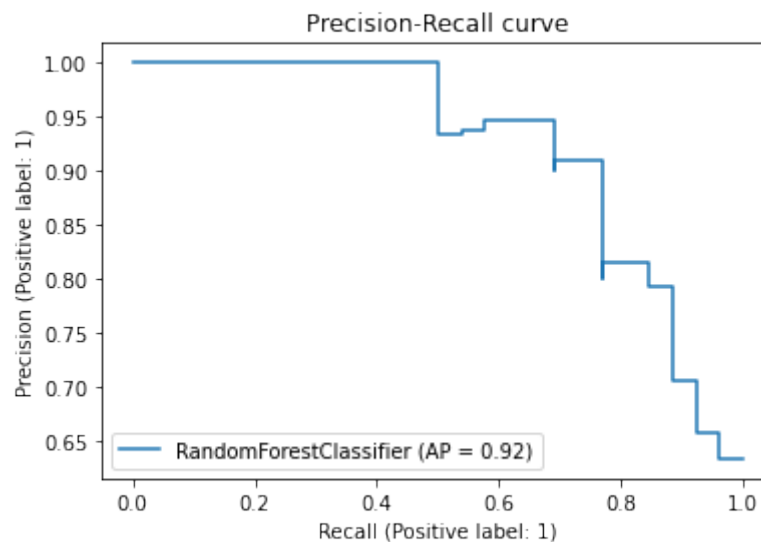
This brings us to a perfect segue to understanding precision-recall trade off.

7.21 What is the intuition behind Precision-Recall curve?

Precision recall curve is used to understand the trade-off between precision and recall. If the precision increases, the recall decreases and vice-versa.

When a classifier has a high recall but low precision, then most of the positive data points are classified correctly but it has many false positives (Type I error). If a classifier has high precision but low recall, then only a few positive data points are correctly classified as positive. We have an ideal classifier when we have high precision and high recall, i.e., when most of the samples are correctly classified.

Below is an example of a Precision-Recall curve at different thresholds. As you can see, changing the threshold value leads to sharp drop in precision even though the gain in recall is not that significant.



7.22 Why use F1-score?

It may happen that you care to reduce *both* False positives (FP) and False negatives (FN). In other words, you want to maximize both precision and recall (ideal world, huh!). As you saw above, there is a trade-off between precision and recall and it is not possible to maximize both of them simultaneously. What to do in this case? Drum roll - we have **F1-score** 🥁 to help us in this situation. F1 measures

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2tp}{2tp + fp + fn}$$

7.23 What does "1" indicate in F1?

One means that equal weight is given to precision and recall in the denominator of the F1 score formula. In general, the formula for F_β is:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$

This general formula considers recall to be β times as important as precision.

7.24 Can you rely solely on F1-score without checking other metrics?

High F1 score implies that there is a good balance between precision and recall scores. Low F1 score implies that the classifier is having a hard time in correctly predicting the labels for the samples. Now, what it doesn't tell is how to improve the classifier as we have no information on whether to focus on false positives or false negatives. Hence, F1 score is usually used in combination with other metrics like precision and recall to better understand the model performance.

7.25 What are Macro metrics?

In case of multi-class classification, instead of calculating different metrics as per class, we use several averaging techniques to report the overall performance of the classifier. Let's consider an example of movie genre prediction. Below are the per-class metrics.

Label	TP	FP	FN	Recall	Precision	F1 Score
Comedy	3	1	2	0.6	0.75	0.67
Action	1	4	1	0.5	0.2	0.28
Drama	3	1	1	0.75	0.75	0.75

Macro metrics refer to the average of metrics across all the classes. This metric gives equal weightage to all the classes.

Macro-Average Recall score	Macro-Average Precision score	Macro-Average F1 score
0.61	0.56	0.56

7.26 What are Micro metrics?

Micro metrics aggregates the TP, FP, FN predictions across all the classes to compute the average metric. In the above example:

Total TP - 7; Total FP - 6, Total FN - 4

Micro-Average Recall score	Micro-Average Precision score	Micro-Average F1 score
0.636	0.538	0.58

7.27 When is micro metric preferred over macro?

In a multi-class classification, micro-average is preferable over macro when the individual classes are severely imbalanced. For example, in the example above both Comedy and Drama classes contribute predominantly to the macro-average precision score of 0.56. However, since the 'Action' class has high number of FP, it brings down the micro-average

precision score to 0.538. In severely imbalanced classes, micro metrics is able to capture more relevant information than macro metrics.

You can also use weighted metrics in case of imbalanced datasets. In weighted metrics, the output is calculated by taking the contribution from each class and multiplying that with the weight/support (the number of true samples for each label) of that class.