TUT206 Nov 15

Review

1. How are **predictors variables** used to **predict** an **outcome variable**?

   E.g.,

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 1_B(k_i) + \beta_3 1_C(k_i) + \beta_4 x_i 1_B(k_i) + \beta_5 1_B(k_i) 1_C(k_i) + \beta_6 x_i^2$$

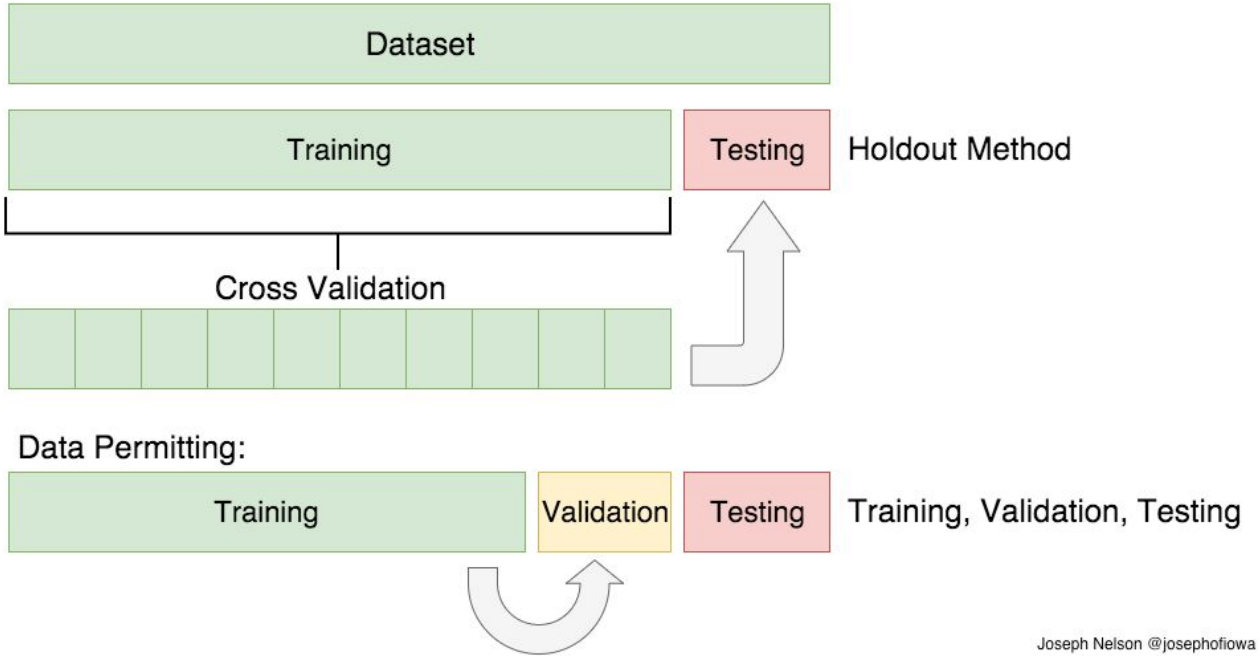
Does it has interaction term?

Review

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 1_B(k_i) + \beta_3 1_C(k_i) + \beta_4 x_i 1_B(k_i) + \beta_5 1_B(k_i) 1_C(k_i) + \beta_6 x_i^2$$

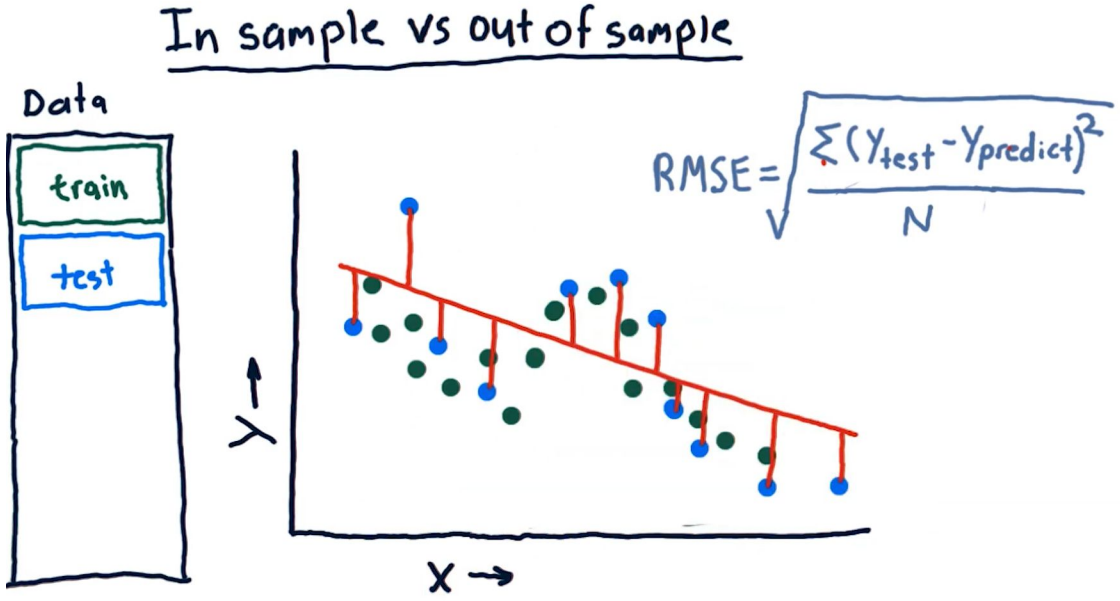
- β_0: Intercept of the model (the expected value of Y_i when all predictors are 0).
- $\beta_1 x_i$: Linear effect of the continuous predictor x_i.
- $\beta_2 1_B(k_i)$: Effect of the binary indicator variable for category "B".
- $\beta_3 1_C(k_i)$: Effect of the binary indicator variable for category "C".
- $\beta_4 x_i 1_B(k_i)$: Interaction effect between x_i and the indicator for "B".
- $\beta_5 1_B(k_i) 1_C(k_i)$: Interaction between indicators for "B" and "C".
- $\beta_6 x_i^2$: Non-linear (quadratic) effect of x_i.

Review: Train-Test "in sample" versus "out of sample" validation



Review: Train-Test "in sample" versus "out of sample" validation

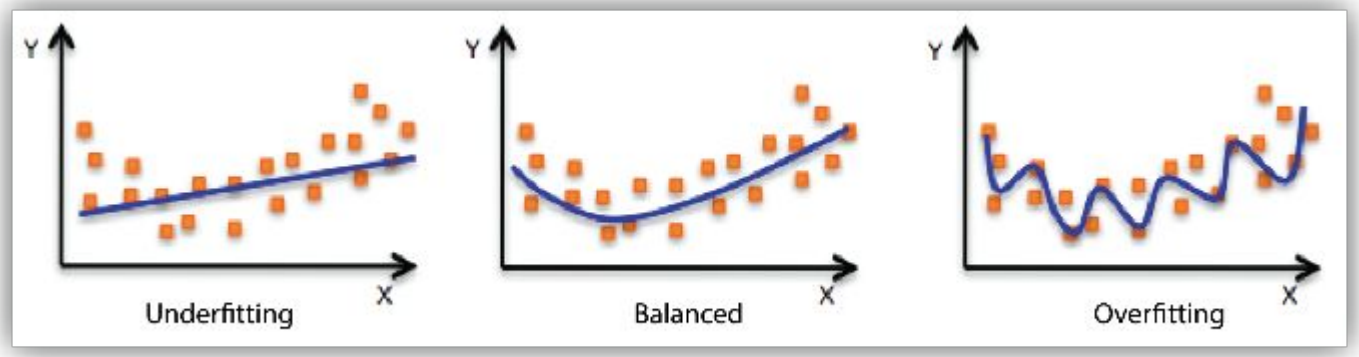
- **In-sample validation**: Evaluates the model using the same data it was trained on. This can overestimate the model's predictive power because it may fit the noise in the data (overfitting).
- **Out-of-sample validation**: Evaluates the model using new, unseen data (test set). This provides a better estimate of how the model will perform on future data.



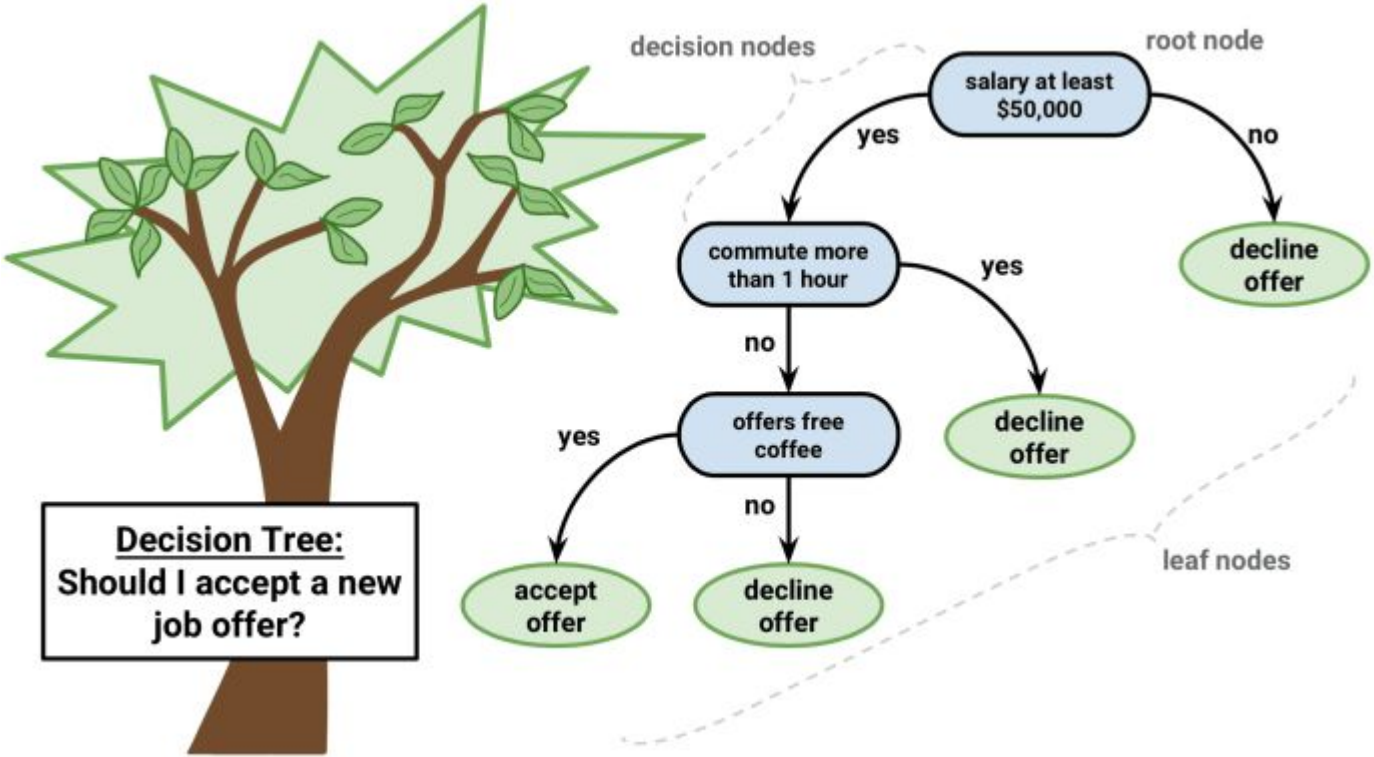
Review: Train-Test "in sample" versus "out of sample" validation

See demo

Review: Train-Test "in sample" versus "out of sample" validation



Demo: what is decision tree?



Demo

What's similar about this code and the multiple linear regression code above?

```
# Fit the linear regression model using statsmodels train_data =
pd.concat([X_train, y_train], axis=1)
formula = 'Q("mean smoothness") ~ Q("mean area") + Q("texture error") +
Q("smoothness error") + Q("mean fractal dimension")'
linear_model = smf.ols(formula, data=train_data).fit()
```

```
# Initialize and fit the Decision Tree model
tree_model = DecisionTreeClassifier(random_state=130)
tree_model.fit(X_train, y_train)
```

Demo

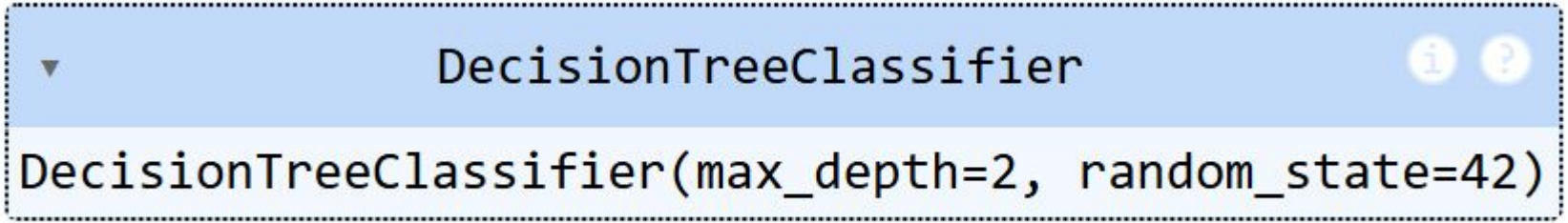
Key Differences:

- Outcome Type:
  - **Linear Regression:** Predicts a **continuous numeric outcome** (e.g., `mean smoothness`).
  - **Decision Tree:** Predicts a **categorical outcome** (e.g., `malignant` or `benign`).
- Evaluation Metric:
  - **Linear Regression:** Uses MSE (lower is better).
  - **Decision Tree:** Uses accuracy (higher is better).
- Model Type:
  - **Linear Regression:** Assumes a **linear relationship** between predictors and the outcome.
  - **Decision Tree:** Splits data into subsets based on **feature thresholds**, capturing non-linear relationships.

Demo

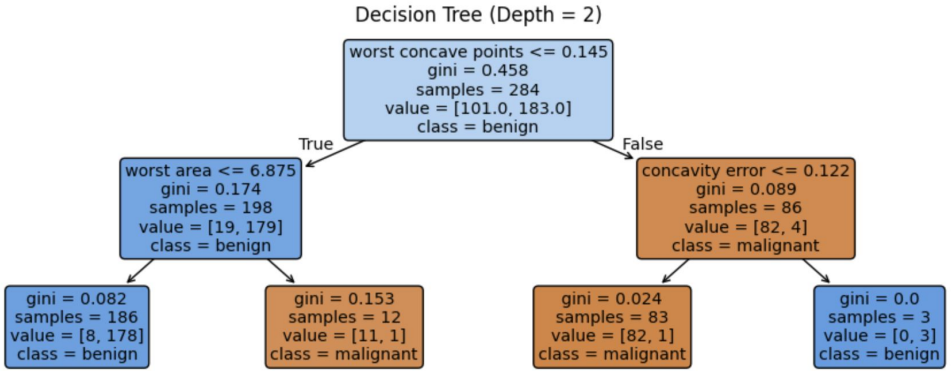
- **Output Type:**
  - Multiple linear regression predicts a **continuous numeric value** (e.g., price, weight).
  - A classification decision tree predicts a **categorical class label** (e.g., "yes" or "no", "A" or "B").
- **Model Form:**
  - Multiple linear regression is a **linear model**, assuming a straight-line relationship between predictors and the outcome.
  - A decision tree is a **non-linear model**, allowing for more complex relationships by making splits in the feature space.

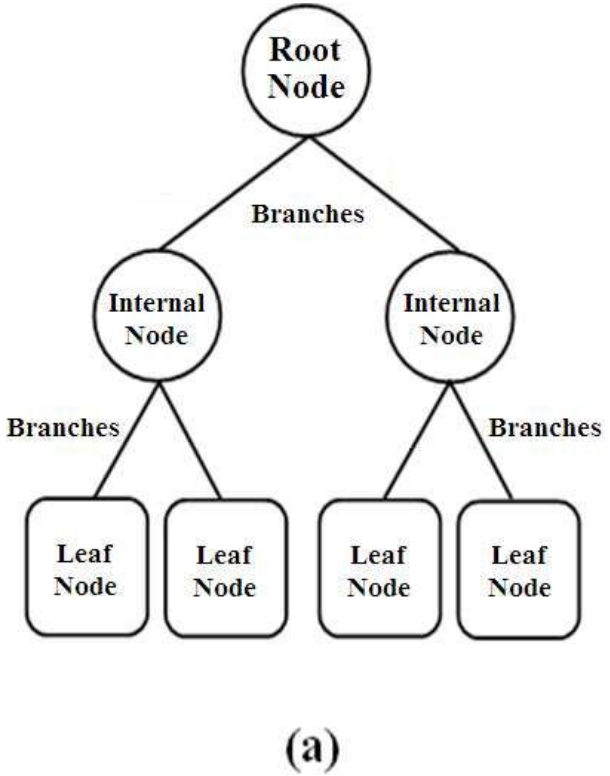
Demo



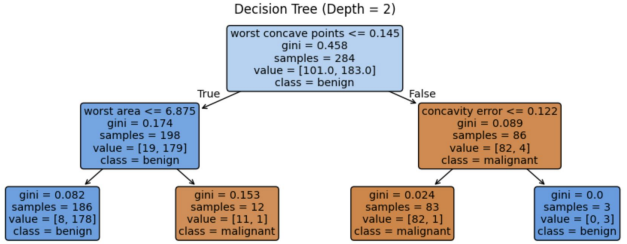
- `DecisionTreeClassifier(max_depth=2, random_state=42)`
  - `max_depth=2` : This parameter sets the maximum depth of the tree to 2. It means the tree can have at most 2 levels of decision splits. A lower depth typically prevents overfitting but might underfit the data if the model is too simple.
  - `random_state=42` : This parameter controls the randomness of the algorithm. By setting a seed value (in this case, 42), we ensure that the decision tree's behavior is reproducible every time we run the code.

Demo


Decision Tree (Depth = 2)
worst concave points <= 0.145
gini = 0.458
samples = 284
value = [101.0, 183.0]
class = benign
True
False
worst area <= 6.875
gini = 0.174
samples = 198
value = [19, 179]
class = benign
concavity error <= 0.122
gini = 0.089
samples = 86
value = [82, 4]
class = malignant
gini = 0.082
samples = 186
value = [8, 178]
class = benign
gini = 0.153
samples = 12
value = [11, 1]
class = malignant
gini = 0.024
samples = 83
value = [82, 1]
class = malignant
gini = 0.0
samples = 3
value = [0, 3]
class = benign


Root Node
Branches
Internal Node
Internal Node
Branches
Branches
Leaf Node
Leaf Node
Leaf Node
Leaf Node
(a)

Demo



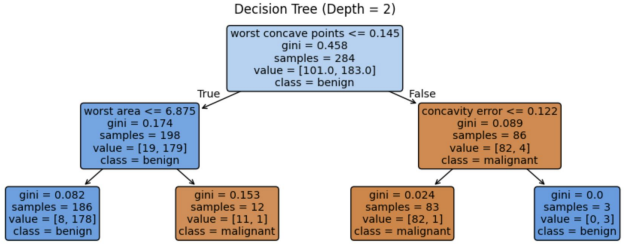
Nodes (Rectangles):

- Each rectangle is a **decision node** or a **leaf node**.
- **Decision nodes** contain a condition (e.g., `worst concave points <= 0.145`).
- **Leaf nodes** are terminal nodes where a final decision is made (e.g., `class = benign` or `class = malignant`).

Splitting Condition:

- For example, the root node checks if `worst concave points <= 0.145`.
- If this condition is `True`, the model follows the left branch. If `False`, it follows the right branch.

Demo



Gini Index (gini):

- **Gini index** measures the impurity of a node. It ranges from 0 (pure node) to 0.5 (maximally impure).
- In a pure node (e.g., `gini = 0.0`), all samples belong to one class.
- The lower the Gini index, the better the split. For instance, the leaf node with `gini = 0.0` and `samples = 3` is perfectly pure (all samples are benign).

Samples:

- This indicates the number of samples (data points) in that node.
- For example, `samples = 284` in the root node means the entire dataset used to build this tree had 284 samples.

Value:

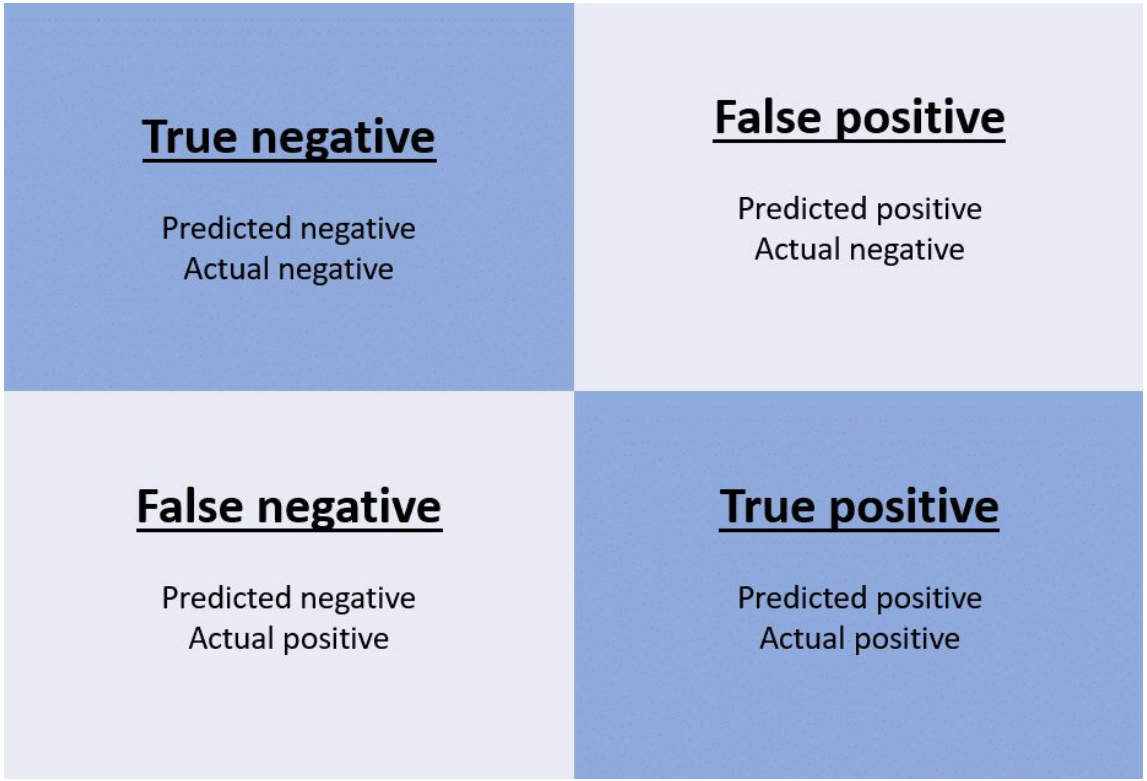
- The `value` indicates the distribution of samples across the classes (e.g., `[101.0, 183.0]`).
- `[101.0, 183.0]` means there are 101 samples of one class (e.g., malignant) and 183 samples of another class (e.g., benign).

Demo

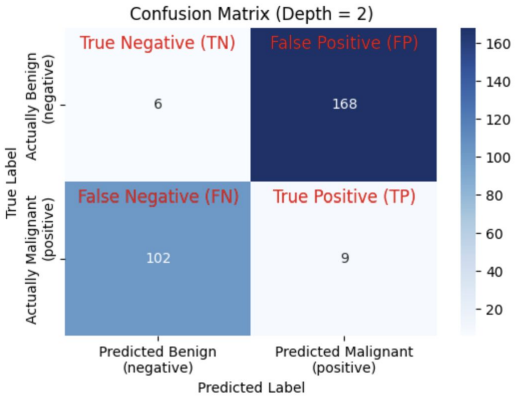
```
# Generate the confusion matrix
conf_matrix =
confusion_matrix(cancer_data.target[testing_indices],
y_pred_depth_2)
```

- `testing_indices` selects only the test subset of the data, ensuring that we compare predictions against the correct test labels.

Demo


True negative
Predicted negative
Actual negative
False positive
Predicted positive
Actual negative
False negative
Predicted negative
Actual positive
True positive
Predicted positive
Actual positive

Demo



Analysis:

- **High False Positive (FP) Rate:**
  - The model has a high number of false positives (168), indicating it often predicts malignant cases even when they are actually benign.
  - This could lead to unnecessary medical interventions and stress for patients.
- **High False Negative (FN) Rate:**
  - The model has a high number of false negatives (102), which is concerning because it fails to identify many malignant cases.
  - In medical applications, false negatives are critical because they represent missed cancer diagnoses, potentially delaying treatment.

Discussion

1. [7 minutes] Purpose of Train-Test Validation Framework

The **train-test** validation framework helps to assess the model's ability to generalize. It allows us to evaluate how well the model performs on unseen data:

- **Train Set**: Used to fit the model.
- **Test Set**: Used to evaluate the model's performance.

Goal: Ensure that the model doesn't just perform well on the training data (in-sample), but also on new, unseen data (out-of-sample). This helps identify issues like **overfitting**.

Discussion Prompt:

- Why do we need both a train and test set?
- What problems might arise if we only evaluate on the training data?

Discussion

Comparison and Key Differences

Concept	Type I Error (False Positive)	False Positive (FP) Prediction
Context	Hypothesis testing	Classification problems
Definition	Incorrectly reject null hypothesis	Incorrectly predict a positive outcome
Example	Declaring a drug effective when it's not	Predicting spam for a non-spam email

Concept	Type II Error (False Negative)	False Negative (FN) Prediction
Context	Hypothesis testing	Classification problems
Definition	Failing to reject a false null hypothesis	Incorrectly predict a negative outcome
Example	Failing to detect an effective drug	Classifying a spam email as non-spam

Thanks!