Data Leakage in Machine Learning

As data scientists, we often receive a dataset and follow a standard machine learning workflow. Suppose we apply the hold-out validation approach, where we split the data into a training set and a test set. We train our model on the training data and evaluate it on the test data.
During this phase, our model achieves 95% accuracy, which seems promising. Excited by the results, we deploy the model to production, where actual users interact with it through a website or application. However, after deployment, we track the performance in real-world scenarios and notice that the accuracy has dropped drastically to 70%.
What Went Wrong?
The sudden drop in performance suggests a serious issue—likely data leakage. This means that, during model training, the model had access to information that it wouldn't have in a real-world scenario. Because of this, the model performed well during validation but struggled when faced with unseen, real-world data.
Data leakage can occur due to various reasons, such as:

- Features derived from target variable (e.g., using future information that wouldn't be available at prediction time).
- Improper data splitting (e.g., test data containing information leaked from the training set).
- Preprocessing steps applied incorrectly (e.g., normalizing data using statistics from the entire dataset instead of computing them separately for training and test sets).

What is Data Leakage?

Data Leakage, in the context of machine learning and data science, refers to a problem where information from outside the training dataset is used to create the model. This additional information can come in various forms, but the common characteristic is that it is information that the model wouldn't have access to when used for prediction in a real-world scenario.
This can lead to overly optimistic performance estimates during training and validation, as the model has access to extra information. However, when the model is deployed in a production environment, that additional information is no longer available, leading to a significant drop in model performance. This discrepancy is typically the result of mistakes in experiment design.

Ways in Which Data Leakage Can Occur

1. Target Leakage

Target leakage occurs when the model has access to information during training that would not be available at prediction time. This gives the model an unfair advantage, leading to overestimated performance during training but poor performance in real-world scenarios.

Example:
Imagine building a loan default prediction model. If we include a feature like "Has the customer paid their next EMI?", this is a clear leak because the answer is only known after the loan period, not at the time of prediction.
Incorrect:

Customer	Income	Credit Score	Loan Amount	Has Defaulted?	Paid Next EMI?
A	50K	750	10K	No	Yes
B	30K	600	20K	Yes	No

Here, the column "Paid Next EMI?" is known only after a customer has already defaulted or not, so it leaks future information into the training process.
Fix: Remove any features that would not be available at prediction time.

2. Multicollinearity with Target Column

Multicollinearity occurs when one or more predictor variables are highly correlated with the target variable. While this isn’t always direct data leakage, it can lead to overfitting, making the model perform poorly on unseen data.

Example:
Let’s say we are predicting house prices and one of our features is "Property Tax Paid". Since property tax is often directly proportional to the house price, including it would result in an artificially high correlation, leading the model to rely too much on it.
Incorrect:

House ID	Area (sq ft)	Bedrooms	Property Tax Paid	House Price
1	1500	3	$3,000	$300,000
2	2000	4	$4,500	$450,000

Since property tax is derived from the target variable (house price), including it makes the model less generalizable to new houses where tax values may change.
Fix: Remove features that are direct transformations of the target variable.

3. Duplicated Data

Data leakage occurs when duplicate records exist in both the training and test sets. Since the model has already seen these exact examples during training, it results in unrealistically high accuracy that won't generalize to real-world data.

Example:
Let's say we are building a spam detection model. If some emails appear in both the training and test sets, the model will simply memorize them rather than learning general patterns.
Incorrect:

Email ID	Text Content	Spam/Not Spam
1	"Win a free iPhone now!"	Spam
2	"Your invoice is attached."	Not Spam
1	"Win a free iPhone now!"	Spam

Fix: Ensure that the training and test sets contain unique samples.

4. Preprocessing Leakage (Train-Test Contamination & Cross-Validation Issues)

Preprocessing leakage happens when data transformations (e.g., normalization, feature scaling, imputation) are performed on the entire dataset before splitting into training and test sets. This allows test data to influence training, resulting in overly optimistic performance estimates.

Example:
Suppose we normalize our dataset before splitting it into training and test sets.
Incorrect:

1. Compute mean and standard deviation on the entire dataset.
2. Normalize both training and test sets using these values.
3. Train the model.

Fix:

1. Split the dataset first (training and test sets).
2. Compute statistics (mean, standard deviation, etc.) only on the training set.
3. Apply those transformations separately to the test set.

5. Hyperparameter Tuning Leakage

Hyperparameter tuning leakage occurs when we use the test set to select the best hyperparameters, instead of using a separate validation set. This leads to overfitting to the test set, meaning the model may not generalize to new data.
Example:
Incorrect Approach:

1. Split data into training and test sets.

2. Train models with different hyperparameters.
3. Select the best hyperparameters based on test set performance.

Fix:

1. Split the data into training, validation, and test sets.
2. Use cross-validation on the training set to select the best hyperparameters.
3. Use the test set only once for final evaluation.

Key Takeaways

- Avoid using future information in predictors (Target Leakage).
- Be careful with highly correlated features (Multicollinearity).
- Ensure training and test sets have unique records (Duplicate Data).
- Always apply preprocessing separately for training and test sets (Preprocessing Leakage).
- Use a proper validation set for hyperparameter tuning.

How to detect data leakage ?

1. Review Your Features: Carefully review all the features being used to train your model. Do they include any data that wouldn't be available at the time of prediction, or any data that directly or indirectly reveals the target? Such features are common sources of data leakage.
2. Unexpectedly High Performance: If your model's performance on the validation or test set is surprisingly good, this could be a sign of data leakage. Most predictive modelling tasks are challenging, and exceptionally high performance could mean that your model has access to information it shouldn't.
3. Inconsistent Performance Between Training and Unseen Data: If your model performs significantly better on the training and validation data compared to new, unseen data, this might indicate that there's data leakage.
4. Model Interpretability: Interpretable models, or techniques like feature importance, can help understand what the model is learning. If the model places too much importance on a feature that doesn't seem directly related to the output, it could be a sign of leakage.

How to remove data leakage?

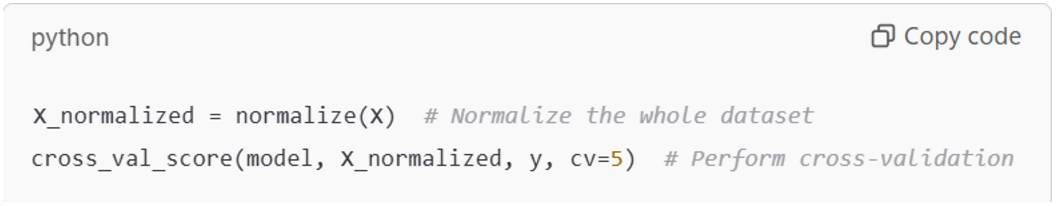
1. Understand the Data and the Task: Before starting with any kind of data processing or modeling, it's important to understand the problem, the data, and how the data was collected. You should understand what each feature in your data represents, and whether it would be available at the time of prediction.

2. Careful Feature Selection: Review all the features used in your model. If any feature includes information that wouldn't be available at the time of prediction, or that directly or indirectly gives away the target variable, it should be removed or modified.

3. Proper Data Splitting: Always split your data into training, validation, and testing sets at an early stage of your pipeline, before doing any pre-processing or feature extraction.

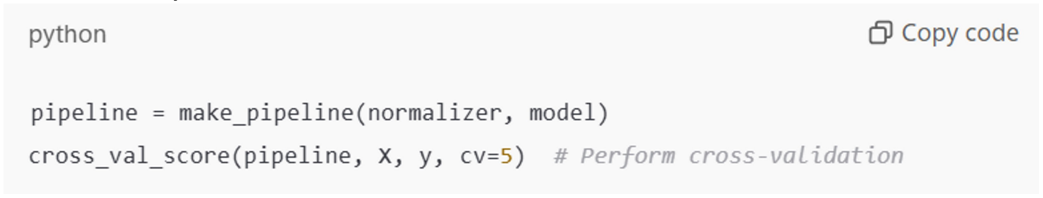
4. Pre-processing Inside the Cross-Validation Loop: If you're using techniques like cross-validation, make sure to do any pre-processing inside the cross-validation loop. This ensures that the pre-processing is done separately on each fold of the data, which prevents information from the validation set leaking into the training set.

   Incorrect way:



```python
X_normalized = normalize(X)  # Normalize the whole dataset
cross_val_score(model, X_normalized, y, cv=5)  # Perform cross-validation
```

   Correct way :



```python
pipeline = make_pipeline(normalizer, model)
cross_val_score(pipeline, X, y, cv=5)  # Perform cross-validation
```

5. Avoid Overlapping Data: If the same individuals, or the same time periods, appear in both your training and test sets, this can cause data leakage. It's important to ensure that the training and test sets represent separate, non-overlapping instances.