DATA LEAKAGE

The Problem:

Imagine you have a dataset, and you perform a train-test split. You train your model on the training data and then evaluate its performance on the test data, achieving an impressive accuracy of 95%. You're thrilled and deploy the model into production. However, in production, the model fails to predict accurately, and its performance drops to 75%.

You wonder what went wrong. Why did the model perform so well during development but so poorly in production? Seeking advice from your seniors, they suggest that the issue might be due to *data leakage*. Curious, you ask, "What is data leakage?" and start studying it in detail.

What is Data leakage?

Data leakage is a problem where extra information that would not be available during prediction is mistakenly included in the training data while building the model. This leads to artificially high performance during training and testing but poor performance in production, as the model cannot access that extra information when making predictions.

Let's understand this with an analogy:

Imagine you are a college student who studies independently (self-study). For your mid-semester exams (MST), you cheat and score very well. However, during your university exams, you are unable to cheat, and your scores drop significantly.

If we relate this to data leakage:

- **Self-study represents training the model.**
- **Cheating represents adding extra information to the training process.**
- **Taking the university exam represents making predictions.**

When you cheat (add extra information), you score well, but when you can't cheat during the university exam (no extra information during prediction), your scores decrease.

This is the essence of data leakage—it creates an illusion of strong model performance during development, but the performance drops in real-world scenarios where the leaked information is unavailable.

DATA LEAKAGE

Ways in which Data Leakage can occur ?

Target Leakage:

Target leakage occurs when you use a column during model training that will not be available in production. This results in an inflated performance during training and testing but poor performance in production because the model relies on information it won't have access to in real-world scenarios.

Let's understand this with an example:

Suppose we are building a credit card fraud detection model. The dataset includes the following columns:

- Value
- Date
- Website
- Reversed Transactions
- Fraud Detection (Target column)

Here, if *Fraud Detection* is our target column, let's focus on the *Reversed Transactions* column. This column tells us whether a refund has been processed or not. However, a refund is processed only if fraud is detected (i.e., the *Fraud Detection* column is "Yes").

Since *Reversed Transactions* depends directly on the *Fraud Detection* column, including it in the training data would artificially boost the model's performance because it provides direct information about the target.

Now, think about production: In production, we want to use the model to predict fraud. The *Reversed Transactions* column will not exist as input because it is generated after fraud is detected. Without this column in production, the model will perform poorly.

This type of leakage, where a feature is directly dependent on the target column, is called target leakage.

Multicollinearity with Target Column:

Multicollinearity generally refers to the situation where features are highly correlated with each other. However, in the context of data leakage, if any column is highly correlated with the target column, it means that this column can independently explain a significant portion of the variance in the target column.

In such cases, the target column becomes highly dependent on that specific feature, which may indicate data leakage. This is because the feature might be providing information about the target that should not be available during prediction, resulting in artificially high performance during training but poor performance in production.

To avoid this, it's important to carefully analyze the correlation between features and the target column and ensure no unintended dependencies are included in the model.

Duplicated Data:

Let's consider a situation where we have duplicate data in our dataset. If we split this dataset into training and testing data, some duplicate rows might end up in both sets. When we train the model on the training data, it learns from these duplicate rows. Later, during testing, if the test data contains duplicate rows from the training data, the model can easily predict the outcomes for these rows, as it has essentially "seen" this data before. This leads to artificially high performance, causing a data leakage problem.

In real-world terms, imagine you study many questions the night before an exam, and several of those exact questions appear in the exam. Naturally, you would perform well on these familiar questions, which would inflate your score.

Similarly, in the case of machine learning, having duplicate rows in both training and testing data gives the model an unfair advantage, making its performance during evaluation unreliable.

Preprocess Leakage:

Preprocess leakage occurs when data preprocessing steps, such as scaling, encoding, or imputing missing values, are applied to the entire dataset before splitting it into training and testing sets. This results in information from the test set leaking into the training data, causing artificially high performance during model evaluation.

For example:

- If you scale the data (e.g., using MinMaxScaler or StandardScaler) on the entire dataset before splitting, the scaler computes statistics (e.g., min, max, mean, standard deviation) using both training and test data. These statistics provide information about the test data to the model during training, leading to data leakage.
- Similarly, if you impute missing values (e.g., replacing NaN with the mean) before splitting, the computed mean includes data from both training and testing sets, causing leakage.

Analogy:

Think of preprocess leakage as preparing answers for an exam by looking at both the practice questions (training data) and the actual exam questions (test data).

DATA LEAKAGE

During preparation, you unknowingly memorize the answers to the test questions. When you take the exam, your score is higher than it would have been if you hadn’t seen those answers beforehand.

Hyperparameter Tuning:

In hyperparameter tuning, you adjust the model parameters to increase model performance. However, to check the model performance, you use the test data. Based on the test data, you decide the best hyperparameters. But what if unseen data comes, and for that data, your chosen hyperparameters are not the best? In that case, the model's performance will decrease in production.

How to detect?

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 ?

DATA LEAKAGE

1.**Understand the Data and the Task**: **Before starting with any kind of data processing or modelling, 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.

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.

Validation Set:

A validation set is a simple technique for checking data leakage, where you divide your entire data into three parts: training data, validation data, and test data. You train your model on the training data, then check the model's performance using the validation data. Finally, you check the model's performance on the test data, which is likely to be unseen data or real-world data.

DATA LEAKAGE