Train Test Split in Python

What is train_test_split in Machine Learning

In Scikit-learn, train_test split is a function used to create training and testing data to be usert to measure a machine learning model's performance.

Why Use Train Test Split in Machine Learning?

In machine leaming, we often build or train models on a single dataset To evaluate if a machine learning model is doing as expected, we need to train the model on one portion of the dataset, and compare how accurately the predictions map to the real-world data

To evaluate the accuracy of machine learning models, data scientists need to split datasets in two portions called

-training data (train the model)

-testing set (test the model)

How to Use Train Test Split

1. Split a dataset into a training and testing set
2. Provide the testing size with the test_size parameter
3. Train a model on the training set
4. Make predictions on the training set
5. Compute the accuracy with a metrics such as the accuracy or accuracy_score

In [1]:
```
import pandas as pd
```

In [2]:
```
housing = pd.read_csv("housing.csv")
```

In [3]:
```
housing.head()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	household
0	-122.23	37.88	41	880	129.0	322	12
1	-122.22	37.86	21	7099	1106.0	2401	113
2	-122.24	37.85	52	1467	190.0	496	17
3	-122.25	37.85	52	1274	235.0	558	21
4	-122.25	37.85	52	1627	280.0	565	25

In [4]:
```
y= housing.median_income
```

In [5]:
```
x=housing.drop('median_income',axis=1)
```

In [6]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

In [7]:
```
#printing shapes of testing and training sets :
print("shape of original dataset :", housing.shape)
print("shape of input - training set", x_train.shape)
print("shape of output - training set", y_train.shape)
print("shape of input - testing set", x_test.shape)
print("shape of output - testing set", y_test.shape)
```

```
shape of original dataset : (20640, 10)
shape of input - training set (16512, 9)
shape of output - training set (16512,)
shape of input - testing set (4128, 9)
shape of output - testing set (4128,)
```

Diagnose and Address Underfitting and Overfitting

Diagnosing Underfitting:

Underfitting occurs when a model is too simplistic to capture the underlying patterns in the data. It performs poorly on both the training and test sets. Signs of underfitting include:

- Low training and test performance (low accuracy, high error).
- Consistently poor performance across different datasets or folds in cross-validation.
- Model doesn't seem to learn from the training data.

Addressing Underfitting:

- **Increase Model Complexity:** Consider using a more complex model with more parameters, such as using deeper neural networks, higher-degree polynomial regression, or more complex algorithms.
- **Feature Engineering:** Add more relevant features to the dataset to provide the model with more information.
- **Fine-tuning Hyperparameters:** Adjust hyperparameters like learning rate, regularization strength, or the number of hidden units/layers in a neural network.
- **Reduce Regularization:** If you're using regularization techniques, consider reducing the strength of regularization or using a different type.

Reasons for Underfitting:

- High bias and low variance.
- The size of the training dataset used is not enough.
- The model is too simple.
- Training data is not cleaned and also contains noise in it.

Techniques to Reduce Underfitting:

- Increase model complexity.
- Increase the number of features, performing feature engineering.
- Remove noise from the data.
- Increase the number of epochs or increase the duration of training to get better results.

Diagnosing Overfitting:

Overfitting occurs when a model becomes too flexible and fits the training data noise and outliers. It performs very well on the training set but poorly on the test set. Signs of overfitting include:

- High training performance but significantly lower test performance.
- Large differences between training and test performance.
- Model captures noise and fluctuations in the training data.

Addressing Overfitting:

- **Regularization:** Apply regularization techniques to penalize overly complex models. Common methods include L1 regularization (Lasso), L2 regularization (Ridge), and dropout in neural networks.
- **Feature Selection:** Remove irrelevant or noisy features that might be contributing to overfitting.
- **More Data:** Increase the size of your training dataset to provide the model with more examples to learn from.
- **Early Stopping:** Monitor the performance on the validation set during training and stop training when performance starts to degrade.
- **Simpler Model:** Consider using a simpler model architecture with fewer parameters. Ensemble Methods: Combine predictions from multiple models to reduce overfitting.

Reasons for Overfitting:

- High variance and low bias.
- The model is too complex.
- The size of the training data.

Techniques to Reduce Overfitting:

- Increase training data.
- Reduce model complexity.

- Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
- Ridge Regularization and Lasso Regularization.
- Use dropout for neural networks to tackle overfitting

In [ ]: