

Interpret model predictions using Permutation Feature Importance

06/15/2021 • 4 minutes to read • 

In this article

[Load the data](#)

[Train the model](#)

[Explain the model with Permutation Feature Importance \(PFI\)](#)

[Next steps](#)

Using Permutation Feature Importance (PFI), learn how to interpret ML.NET machine learning model predictions. PFI gives the relative contribution each feature makes to a prediction.

Machine learning models are often thought of as opaque boxes that take inputs and generate an output. The intermediate steps or interactions among the features that influence the output are rarely understood. As machine learning is introduced into more aspects of everyday life such as healthcare, it's of utmost importance to understand why a machine learning model makes the decisions it does. For example, if diagnoses are made by a machine learning model, healthcare professionals need a way to look into the factors that went into making that diagnoses. Providing the right diagnosis could make a great difference on whether a patient has a speedy recovery or not. Therefore the higher the level of explainability in a model, the greater confidence healthcare professionals have to accept or reject the decisions made by the model.

Various techniques are used to explain models, one of which is PFI. PFI is a technique used to explain classification and regression models that is inspired by [Breiman's *Random Forests* paper](#) (see section 10). At a high level, the way it works is by randomly shuffling data one feature at a time for the entire dataset and calculating how much the performance metric of interest decreases. The larger the change, the more important that feature is.

Additionally, by highlighting the most important features, model builders can focus on using a subset of more meaningful features which can potentially reduce noise and training time.


Load the data

Load the data

The features in the dataset being used for this sample are in columns 1-12. The goal is to predict `Price`.

Column	Feature	Description
1	CrimeRate	Per capita crime rate
2	ResidentialZones	Residential zones in town
3	CommercialZones	Non-residential zones in town
4	NearWater	Proximity to body of water
5	ToxicWasteLevels	Toxicity levels (PPM)
6	AverageRoomNumber	Average number of rooms in house
7	HomeAge	Age of home
8	BusinessCenterDistance	Distance to nearest business district
9	HighwayAccess	Proximity to highways
10	TaxRate	Property tax rate
11	StudentTeacherRatio	Ratio of students to teachers
12	PercentPopulationBelowPoverty	Percent of population living below poverty
13	Price	Price of the home

A sample of the dataset is shown below:

text	 Copy
1,24,13,1,0.59,3,96,11,23,608,14,13,32 4,80,18,1,0.37,5,14,7,4,346,19,13,41 2,98,16,1,0.25,10,5,1,8,689,13,36,12	

The data in this sample can be modeled by a class like `HousingPriceData` and loaded into

an `IDataView`.

C#

 Copy

```
class HousingPriceData
{
    [LoadColumn(0)]
    public float CrimeRate { get; set; }

    [LoadColumn(1)]
    public float ResidentialZones { get; set; }

    [LoadColumn(2)]
    public float CommercialZones { get; set; }

    [LoadColumn(3)]
    public float NearWater { get; set; }

    [LoadColumn(4)]
    public float ToxicWasteLevels { get; set; }

    [LoadColumn(5)]
    public float AverageRoomNumber { get; set; }

    [LoadColumn(6)]
    public float HomeAge { get; set; }

    [LoadColumn(7)]
    public float BusinessCenterDistance { get; set; }

    [LoadColumn(8)]
    public float HighwayAccess { get; set; }

    [LoadColumn(9)]
    public float TaxRate { get; set; }

    [LoadColumn(10)]
    public float StudentTeacherRatio { get; set; }

    [LoadColumn(11)]
    public float PercentPopulationBelowPoverty { get; set; }

    [LoadColumn(12)]
    [ColumnName("Label")]
    public float Price { get; set; }
}
```

Train the model

The code sample below illustrates the process of training a linear regression model to

predict house prices.

C#

 Copy

```
// 1. Get the column name of input features.
string[] featureColumnNames =
    data.Schema
        .Select(column => column.Name)
        .Where(columnName => columnName != "Label").ToArray();

// 2. Define estimator with data pre-processing steps
IEstimator<ITransformer> dataPrepEstimator =
    mlContext.Transforms.Concatenate("Features", featureColumnNames)
        .Append(mlContext.Transforms.NormalizeMinMax("Features"));

// 3. Create transformer using the data pre-processing estimator
ITransformer dataPrepTransformer = dataPrepEstimator.Fit(data);

// 4. Pre-process the training data
IDataView preprocessedTrainData = dataPrepTransformer.Transform(data);

// 5. Define Stochastic Dual Coordinate Ascent machine learning estimator
var sdcaEstimator = mlContext.Regression.Trainers.Sdca();

// 6. Train machine learning model
var sdcaModel = sdcaEstimator.Fit(preprocessedTrainData);
```

Explain the model with Permutation Feature Importance (PFI)

In ML.NET use the [PermutationFeatureImportance](#) method for your respective task.

C#

 Copy

```
var permutationFeatureImportance =
    mlContext
        .Regression
        .PermutationFeatureImportance(sdcaModel, preprocessedTrainData,
            permutationCount:3);
```

Note


For pipelines that combine the preprocessing transforms and trainer, assuming that the trainer is at the end of the pipeline, you'll need to extract it using the

`LastTransformer` property.

The result of using [PermutationFeatureImportance](#) on the training dataset is an [ImmutableArray](#) of [RegressionMetricsStatistics](#) objects. [RegressionMetricsStatistics](#) provides summary statistics like mean and standard deviation for multiple observations of [RegressionMetrics](#) equal to the number of permutations specified by the `permutationCount` parameter.

The metric used to measure feature importance depends on the machine learning task used to solve your problem. For example, regression tasks may use a common evaluation metric such as R-squared to measure importance. For more information on model evaluation metrics, see [evaluate your ML.NET model with metrics](#).

The importance, or in this case, the absolute average decrease in R-squared metric calculated by [PermutationFeatureImportance](#) can then be ordered from most important to least important.

C# 

```
// Order features by importance
var featureImportanceMetrics =
    permutationFeatureImportance
        .Select((metric, index) => new { index, metric.RSquared })
        .OrderByDescending(myFeatures => Math.Abs(myFeatures.RSquared.Mean));

Console.WriteLine("Feature\tPFI");

foreach (var feature in featureImportanceMetrics)
{
    Console.WriteLine($"{
featureColumnNames[feature.index], -20}\t{feature.RSquared.Mean:F6}");
}
```

Printing the values for each of the features in `featureImportanceMetrics` would generate output similar to that below. Keep in mind that you should expect to see different results because these values vary based on the data that they are given.

Feature	Change to R-Squared
HighwayAccess	-0.042731
StudentTeacherRatio	-0.012730

Feature	Change to Required
BusinessCenterDistance	-0.010491
TaxRate	-0.008545
AverageRoomNumber	-0.003949
CrimeRate	-0.003665
CommercialZones	0.002749
HomeAge	-0.002426
ResidentialZones	-0.002319
NearWater	0.000203
PercentPopulationLivingBelowPoverty	0.000031
ToxicWasteLevels	-0.000019

Taking a look at the five most important features for this dataset, the price of a house predicted by this model is influenced by its proximity to highways, student teacher ratio of schools in the area, proximity to major employment centers, property tax rate and average number of rooms in the home.

Next steps

- [Make predictions with a trained model](#)
- [Retrain a model](#)
- [Deploy a model in an ASP.NET Core Web API](#)

Is this page helpful?

 Yes  No

Recommended content

Inspect intermediate data during ML.NET processing - ML.NET

Learn how to inspect intermediate data during ML.NET machine learning pipeline loading, processing and model training steps in ML.NET.

Train a machine learning model using cross validation - ML.NET

Learn how to use cross validation to build more robust machine learning models in ML.NET. Cross-validation is a training and model evaluation technique that splits the data into several partitions and trains multiple algorithms on these partitions.

ML.NET metrics - ML.NET

Understand the metrics that are used to evaluate the performance of an ML.NET model

Train and evaluate a model - ML.NET

Learn how to build machine learning models, collect metrics, and measure performance with ML.NET. A machine learning model identifies patterns within training data to make predictions using new data.

Show more ▾