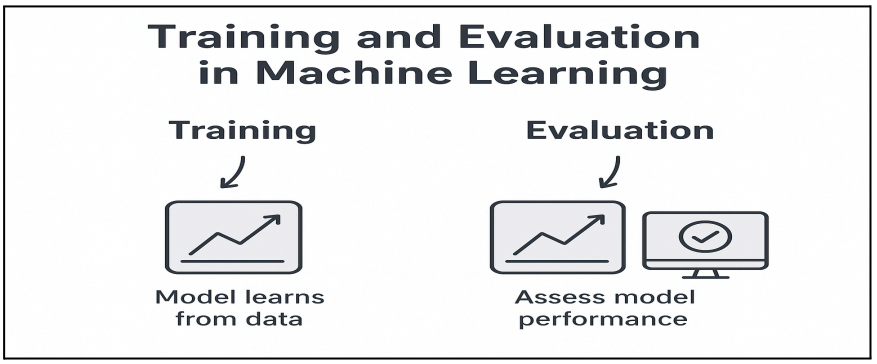
Machine Learning

Introduction to Machine Learning and Regression

Train and evaluate a regression model using scikit-learn:

We're moving from understanding what ML is, to actually doing ML.
We'll teach the computer to learn from data and make predictions, that's called **training and evaluation.**

Let's understand what is training and evaluation(testing) in ML:



Training is the process where a machine learning model learns patterns and relationships from the data we give it.

It looks at many examples of inputs (features) and their correct answers (labels or targets), and tries to find a rule or formula that connects them.

In other words, during training the model is learning how changes in the input affect the output, so it can make good predictions on new, unseen data later.

Think of training this way:

If we show the model house sizes (input) and their prices (output), it learns how price usually changes when the size changes, that's training.

Think of training like studying from examples. The model "studies" the training data to learn how inputs relate to outputs.

In scikit-learn, this happens when we call **.fit(X_train, y_train)**. We will learn more in this lesson.

And, evaluation is the process of checking how well a trained model performs, in other words, how good it is at making predictions on new, unseen data.

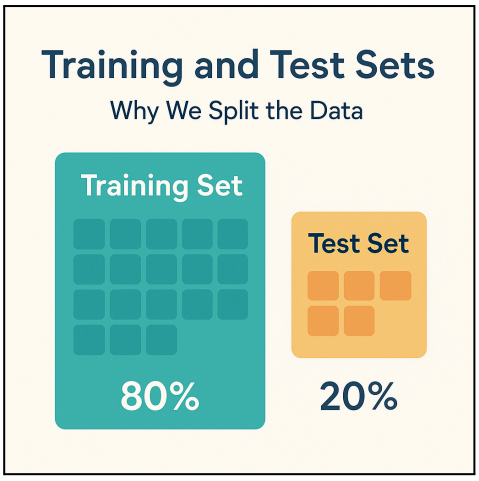
After the model has learned from the training data, we test it using a separate part of the data called the test set. This helps us see if the model has really learned the pattern, or if it just memorized the training examples.

Think of evaluation this way: after studying (training), a student takes an exam (evaluation) to see how well they can apply what they've learned to new questions they haven't seen before.

We measure the model's performance using evaluation metrics, such as:

1. **MSE (Mean Squared Error)**, shows how far the predictions are from the actual values (smaller is better).
2. **R^2 Score**, shows how well the model explains the variation in the data (closer to 1 means a better fit).

Training and Test Sets, why We Split the Data



When we train a model, we want to know how well it can make predictions on new data, not just the data it has already seen.

Think of it this way, when you study for a test, you practice using your notes and homework problems (training). Then, during the real exam (testing), you get new questions you haven't seen before, that's how we see if you truly understand the topic!

To do that, we divide our dataset into two parts:

1. **Training set:** The data the model learns from. The model uses this data to find patterns and relationships.

2. **Test set:** The data we keep aside to check how well the model learned. This simulates how the model will perform on real-world, unseen and new data.

We usually use around 70–80% of the data for training and 20–30% for testing.

In scikit-learn, we can easily do this with:

from sklearn.model_selection import train_test_split (imports a function from scikit-learn)

X_train, X_test, y_train, y_test = **train_test_split(X, y, test_size=0.2, random_state=42)**

This line splits our data into four parts:

1. X_train: features for training
2. X_test: features for testing
3. Y_train: labels (answers) for training
4. Y_test: labels for testing

test_size=0.2 means 20% of the data will be used for testing, and the rest (80%) for training.

random_state=42 just makes the split repeatable, so we get the same result every time we run the code.

Quick note:

Sometimes, we even split the data into three parts, training, validation, and test. The **validation** set helps us fine-tune and improve the model before doing the final test. We'll explore that later when we learn how to improve model performance.

Let's also quickly recap Regression Model & scikit-learn

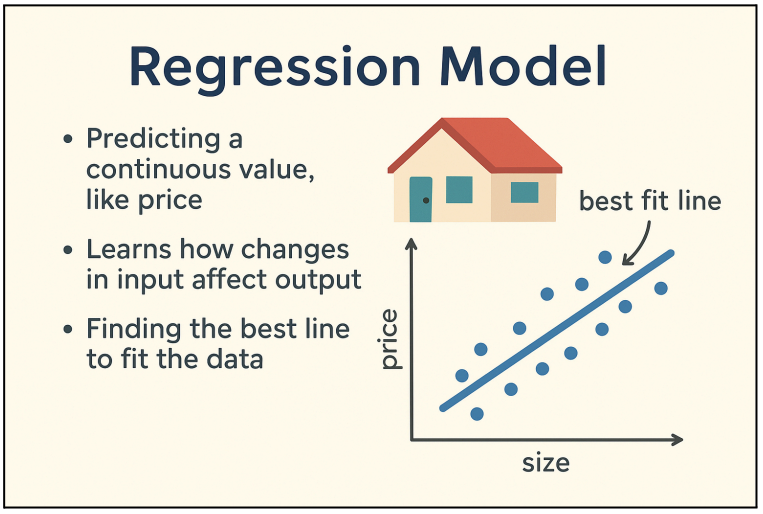
What is a Regression Model?

A regression model is used when we want to predict a number, a value that can go up and down. This kind of value is called a continuous value.

Examples of things we might want to predict:

House price (in dollars), Car mileage (in miles per gallon), Temperature (in degrees), Time something will take (in minutes)

In all these cases, the answer is a number, not a label. That's why we use regression.



Let's understand with House Prices example:

Suppose we want to predict the price of a house.

We know that things like:

1. House size (square feet)
2. Number of bedrooms
3. Neighborhood

usually affect the price.

A regression model looks at many past houses and learns:

"If size increases, price tends to increase."

"If located in area A, the price might be higher."

It learns the relationship between the inputs (features) and the price (target).

Let's also recap: What is scikit-learn?

scikit-learn is a Python library that gives us easy-to-use tools for building machine learning models.
Instead of writing math from scratch, we use scikit-learn to handle the heavy lifting.

Why we use scikit-learn

1. It provides ready-to-use models (like Linear Regression).
2. It provides tools to split data into training and test sets.
3. It provides evaluation metrics to measure how good our model is.
4. It keeps our code simple, clear, and consistent.

Connecting to Our Housing Example
Think about predicting house prices.

If we tried to figure this out ourselves, we'd need to:

1. Study a lot of past house sales
2. Notice patterns (like bigger houses cost more)
3. Build a formula
4. Test whether our formula works on new houses

Scikit-learn helps us do exactly this, but automatically. We just tell scikit-learn:

1. What data to learn from (house size, number of bedrooms…)
2. What value we want to predict (house price) And scikit-learn learns the relationship for us.

The scikit-learn general workflow, in simple:

1. **Choose a model:**

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

2. **Train the model on the training data:**

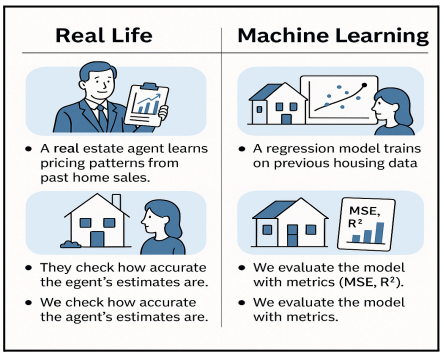
```
model.fit(X_train, y_train)
```

3. **Make predictions on new data:**

```
predictions = model.predict(X_test)
```

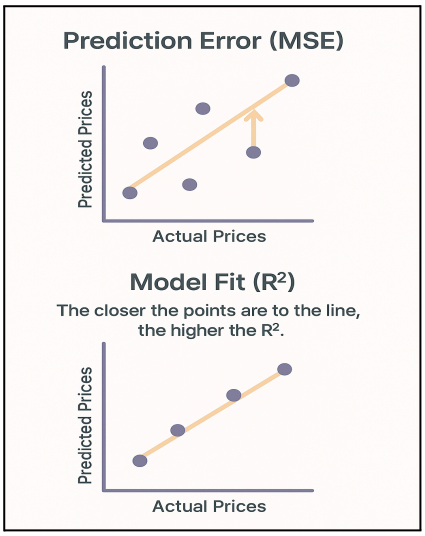
4. **Evaluate how close the predictions are to the real prices:**

```
from sklearn.metrics import mean_squared_error, r2_score
```



Evaluation metrics for regression

When we build a regression model (such as predicting house prices), we need a way to measure how well the model is performing. We compare the model's predicted prices with the actual prices from the dataset.



Here are the two most common evaluation metrics we use:

1. Mean Squared Error (MSE)

MSE tells us how far off the model's predictions are, on average.
It works by taking the squared difference between predicted and actual prices.
Lower MSE = Better model
A high MSE means the model's predictions are very different from the real prices.
If the model keeps making big mistakes, MSE becomes large.

Example:
If the model predicts a house should cost $5,000,000 but the real price is $4,000,000, the error is large, MSE increases.

2. R^2 Score (R-Squared)

R^2 tells us how well the model explains the pattern in the data.

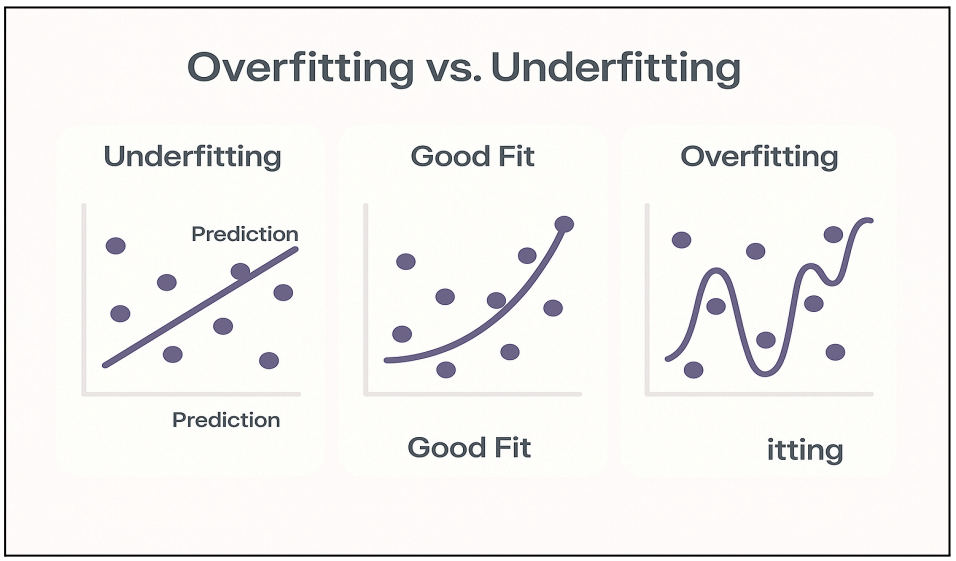
-
- $R^2 = 1 \longrightarrow$ Perfect prediction
- $R^2 = 0 \longrightarrow$ Model learned nothing useful
- Higher R^2 = Better fit

R^2 shows how much of the variation in house prices the model can explain.

Introduction to overfitting and underfitting

When we train a machine learning model, our goal is for it to learn real patterns in the data so it can make good predictions on new houses (not just the ones it has already seen).

However, sometimes the model learns too little or too much.
This leads to underfitting or overfitting.



Underfitting

1. The model didn't learn enough.
2. The model is too simple and misses important patterns in the data.

In the Housing Example:

If the model only uses area to predict price and ignores features like bedrooms, bathrooms, or location, its predictions will be very rough and inaccurate.

Signs of Underfitting:

1. Low accuracy on training data
2. Low accuracy on test data

Simple way to think of it:

Like a student who didn't study enough, they perform poorly on homework and on the test.

Overfitting

1. The model learned too much, it memorized the training data.
2. The model becomes too complex and tries to remember every detail, even noise or random patterns.

In the Housing Example:

If the model tries to fit every small fluctuation in price (like whether the house has a specific shade of paint), it will perform great on training data but fail when predicting a new house.

Signs of Overfitting:

1. High accuracy on training data
2. Low accuracy on test data

Simple way to think of it:

Like a student who memorized answers instead of understanding — they do well on practice, but poorly on a new test.

Goal: Find the Balance

We want a model that:

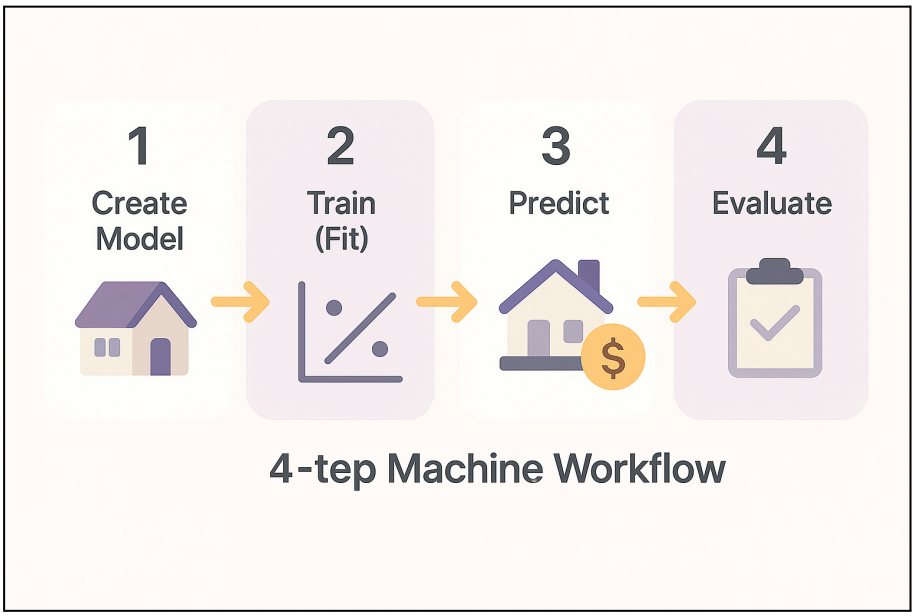
1. Learns the important patterns
2. Ignores noise and random details
3. Performs well on both training and test data

This is called generalization, being able to make good predictions on new, unseen houses.

Next Step

Now we can move into the **actual training step**:

Creating the model → Fitting → Predicting → Evaluating



Hands-on activity:

Train a Regression Model to Predict House Prices

In this activity, we will use real housing data to build a Linear Regression model that can predict the price of a house based on its features (like size, bedrooms, etc.).

We will train the model, test how well it performs, and visualize the results to understand how accurate our predictions are.

Before we start training a machine learning model, we need to understand the data we'll be working with and the process we will be following:

We will use a Housing Dataset (included in the resources folder).

Also can be downloaded directly from Kaggle.com:
https://www.kaggle.com/datasets/yasserh/housing-prices-dataset?resource=download

This dataset contains information about houses (like size, number of bedrooms, amenities, etc.) along with their selling prices.

Steps We Will Follow

1. Explore the dataset
   Look at the data and understand what information we have.
2. Select features (X) and target (y)
   Decide which columns will be used to predict the house price.
3. Split the data into training and test sets
   Train the model on one portion of the data and test it on the rest.
4. Train a Linear Regression model using scikit-learn
   Teach the model to learn the relationship between features and price.
5. Make predictions on the test data
   See what prices the model predicts for unseen houses.
6. Evaluate and visualize predictions
    1. Plot predicted vs actual prices
    2. Look at residuals (errors)
    3. Use metrics like MSE and R^2 to measure performance
7. Compare training vs testing performance
   Check for overfitting or underfitting.
8. Discuss what we learned
   Interpret the results and reflect on model performance.

Now let's complete this activity inside a Jupyter Notebook.
Please open the notebook titled: Regression Model to Predict House Prices.ipynb

Thank you,

Shubham Agarwal