Machine Learning: Linear Regression

Linear Regression

In [3]:
```
from IPython.display import YouTubeVideo
```

In machine learning, a **regression** problem is when you need to build a model that's going to predict a continuous, numerical value, like the sale price of an apartment. One of the models that you can use for regression problems is called **linear regression**. In it's simplest form, we fit a model that will predict a single output variable (called a **target vector**) as a linear function of a single input variable (called a **feature matrix**).

Speaking mathematically, if we have input data points x and corresponding measured output y, then we find parameters m and b such that $y \approx m\times x + b$ for our measured data points. We then use the fitted values of m and b to predict values of y for new values of x.

Fitting a Model to Training Data

You'll work on two cases: a model on the raw data set and a model on transformed data. First try to use linear regression to predict `price_aprox_usd` as a multiple of `surface_covered_in_m2` and the addition of a constant for the `mexico-city-real-estate-1.csv` dataset.

In [4]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Import data
columns = ["surface_covered_in_m2", "price_aprox_usd"]
mexico_city1 = pd.read_csv("./data/mexico-city-real-estate-1.csv", usecols=columns)

# Drop rows with missing values
# (or you could use an imputer 🤞)
mexico_city1.dropna(inplace=True)

# Split data into feature matrix
X = mexico_city1[["surface_covered_in_m2"]]
y = mexico_city1["price_aprox_usd"]

# Instantiate predictor
lr = LinearRegression()

# Fit predictor to data
lr.fit(X, y)
```

Out[4]:
```
LinearRegression()
```

Practice

Fit a linear regression model to the `mexico-city-real-estate-2.csv` data set to relate `"price_aprox_usd"` and `"surface_covered_in_m2"`.

In [7]:
```
# Import data
columns = ["price_aprox_usd", "surface_covered_in_m2"]
mexico_city2 = pd.read_csv("./data/mexico-city-real-estate-2.csv", usecols=columns)
# Drop rows with missing values
mexico_city2.dropna(inplace=True)

# Split data into feature matrix
X = mexico_city2[["surface_covered_in_m2"]]
y = mexico_city2["price_aprox_usd"]

# Instantiate predictor
lr = LinearRegression()

# Fit predictor to data
lr.fit(X, y)
```

Out[7]:
```
LinearRegression()
```

In [8]:
```
mexico_city2.head()
```

Out[8]:

	price_aprox_usd	surface_covered_in_m2
0	226578.22	74.0
1	146889.91	111.0
2	176584.01	82.0
3	150830.86	73.0
4	168849.64	63.0

Generating Predictions Using a Trained Model

After fitting the model, we want to use it to make predictions. In most applications, you'll want to predict an unknown quantity from data that's different from the data you've fitted our model on. To test the accuracy of your fitted model, you'll typically use a different set of data with an outcome you already know. Here, we'll use the dataset from `mexico-city-test-features.csv` and `mexico-city-test-labels.csv`. It's also helpful to plot the data and predicted data to see if there are any patterns that suggest fitting a different model.

In [9]:
```
# Import data
mexico_city_features = pd.read_csv(
    "./data/mexico-city-test-features.csv", usecols=["surface_covered_in_m2"]
)
mexico_city_labels = pd.read_csv("./data/mexico-city-test-labels.csv")

# Drop missing values
mexico_city_features.dropna(inplace=True)

# Generate predictions
```

```
price_pred_example = lr.predict(mexico_city_features)

# Print predictions
price_pred_example[:5]
```

Out[9]:
```
array([309549.84644749, 309411.49120101, 310207.03386826, 309428.78560682,
       309515.25763587])
```

In [10]:
```
mexico_city_labels.head()
```

Out[10]:

	price_aprox_usd
0	120652.04
1	46890.97
2	44572.76
3	23392.79
4	78766.29

Practice

Read the data from `mexico-city-real-estate-4.csv` into a DataFrame and then generate a list of price predictions for the properties using your model `lr`.

In [12]:
```
# Import data...
mexico_city4 = pd.read_csv("./data/mexico-city-real-estate-4.csv",
                           usecols=["surface_covered_in_m2"]
                          )
# Drop missing values
mexico_city4.dropna(inplace=True)

# Generate predictions
price_pred = lr.predict(mexico_city4)

# Print predictions
price_pred[:5]
```

Out[12]:
```
array([309480.66882425, 309411.49120101, 310449.15554959, 309411.49120101,
       310103.2674334 ])
```

Ridge Regression

Sometimes,the values for coefficients and the intercept - both positive and negative - are very large. When you see this in a linear model — especially a high-dimensional model — what's happening is that the model is **overfitting** to the training data and then can't generalize to the test data. Some people call this the **curse of dimensionality**. ☠️

The way to solve this problem is to use **regularization**, a group of techniques that prevent overfitting. In this case, we'll change the predictor from `LinearRegression` to `Ridge`, which is a linear regressor with an added tool for keeping model coefficients from getting too big.

Here's a good explanation of what a ridge regression is and why it's important:

In [13]:
```
YouTubeVideo("Q81RR3yKn30")
```

Out[13]:

An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Generalization

Notice that we tested the model with a dataset that's *different* from the one we used to train the model. Machine learning models are useful if they allow you to make predictions about data other than what you used to train your model. We call this concept **generalization.** By testing your model with different data than you used to train it, you're checking to see if your model can generalize. Most machine learning models do not generalize to all possible types of input data, so they should be used with care. On the other hand, machine learning models that don't generalize to make predictions for at least a restricted set of data aren't very useful.

Calculating the Mean Absolute Error for a List of Predictions

Plots are great for displaying information, but a value that tells you the typical error in a prediction is helpful too. This value is called the **mean absolute error**, and it's defined as the average value of the magnitude of the error in the predictions. The closer the MAE is to `0`, the better our model fits the data. scikit-learn will do this for you if you pass it the price predictions from your regression model and the actual prices from the test data set. Let's see how our `lr` model did by comparing its predictions to the true values in `mexico_city_labels`.

In [15]:
```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(price_pred_example, mexico_city_labels)
```

Out[15]:
```
226209.01327442465
```

Access an Attribute of a Trained Model

After training a model that fits a straight line to your data, you can now obtain the parameters that fit your line. We're particularly interested in the slope `regr_lr.coef_` and the axis intercept `regr_lr.intercept_`

In [16]:
```
print(lr.coef_)
```

```
[3.45888116]
```

In [17]:
```
print(lr.intercept_)
```

```
309238.5471429155
```

Multicollinearity

When you're creating a linear model that uses many features to make predictions, some of those features can be highly correlated with each other. This isn't a problem that's going to break your model; it will still make predictions and it might have good performance metrics. But it is an issue if you want to interpret the coefficients for your model because it becomes hard to tell which features are truly important.

Let's look at `mexico-city-real-estate-1.csv` for an example. First we'll import the data.

In [18]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Import data
columns = [
    "price",
    "price_aprox_local_currency",
    "price_aprox_usd",
    "surface_total_in_m2",
    "surface_covered_in_m2",
    "price_per_m2",
]
mexico_city1 = pd.read_csv("./data/mexico-city-real-estate-1.csv", usecols=columns)

# Drop missing values
mexico_city1.dropna(inplace=True)

mexico_city1.head()
```

Out[18]:

	price	price_aprox_local_currency	price_aprox_usd	surface_total_in_m2	surface_covered_in_m2	pr
2	2700000.0	2748947.10	146154.51	61.0	61.0	4
3	6347000.0	6462061.92	343571.36	176.0	128.0	4
4	6870000.0	6994543.16	371882.03	180.0	136.0	5
5	6500000.0	6617835.61	351853.45	300.0	300.0	2
6	670000.0	682146.11	36267.97	65.0	65.0	1

Now let's find the correlations between the columns.

In [19]:
```
mexico_city1.corr()
```

Out[19]:

	price	price_aprox_local_currency	price_aprox_usd	surface_total_in_m2	su
price	1.000000	0.333655	0.333655	0.112588	
price_aprox_local_currency	0.333655	1.000000	1.000000	0.118123	
price_aprox_usd	0.333655	1.000000	1.000000	0.118123	
surface_total_in_m2	0.112588	0.118123	0.118123	1.000000	
surface_covered_in_m2	0.371073	0.598506	0.598506	0.125032	
price_per_m2	0.380879	-0.068775	-0.068775	0.003488	

Let's see what happens when we fit a linear regression model for `surface_covered_in_m2` as a function of `price_aprox_usd` and `price_aprox_local_currency`.

In [20]:
```
lr = LinearRegression()
lr.fit(
    mexico_city1[["price_aprox_usd", "price_aprox_local_currency"]],
    mexico_city1["surface_covered_in_m2"],
)
```

Out[20]:
```
LinearRegression()
```

Let's take a look at the coefficients of the model:

In [21]:
```
print(lr.coef_)
```

```
[6593.61513754 -350.56569568]
```

Ask yourself: Does it make sense that increasing the price of a property by one US dollar would translate to a 6593 m^2 increase in size? Perhaps, though it seems unlikely. And does it make sense that increasing the price by one Mexican peso would translate to a 350 m^2 *decrease* in size? Definitely not. So while this model may perform well when we evaluate it using metrics like mean absolute error, we can't use it to determine which features actually our target.

References & Further Reading

- A primer on linear regression
- More on resampling from the pandas documentation
- More information on rolling averages
- More on absolute and mean absolute errors
- A discussion of the various uses of model fitting in machine learning
- Wikipedia Page on Multicollinearity
- Online Article on Multicollinearity

- Wikipedia Article on Generalization
- Online Tutorial on Regression with scikit-learn
- Official scikit-learn Documentation on Linear Models
- Wikipedia Article on Logarithm Function
