Intro to Machine Learning

Tej Gorde

Women in
Data Science
Worldwide

Index

How Models Work

The Real Estate Prediction Challenge 🏠

Imagine this scenario : Your cousin is a successful real estate investor who has made millions by buying and selling houses. He wants to partner with you because of your interest in data science.
The deal? He provides the money, and you create models to predict house prices accurately.

How Do We Predict House Prices? 🤔
When you ask your cousin how he predicts house prices, he says it's "just intuition." But after talking more, you discover that he's actually:

Observed patterns in house prices from past sales
&
Used these patterns to make predictions about new houses

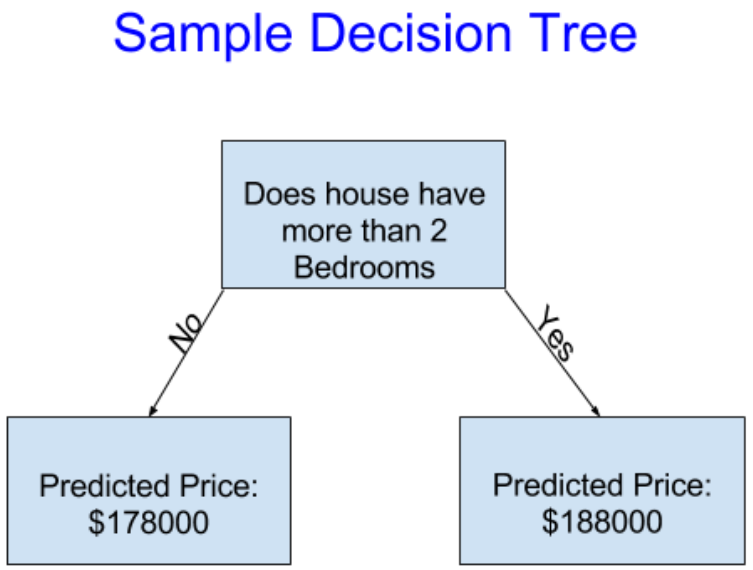
This is exactly how machine learning works!

Understanding Decision Trees 🌳
We'll start with a simple but powerful model called a Decision Tree. While there are more sophisticated models out there, decision trees are:
•Easy to understand
•Great for learning the basics
•Building blocks for advanced machine learning models

Understanding Decision Tress

This model asks just ONE question: "Does the house have more than 2 bedrooms?"
Makes predictions based on TWO possible answers:

1. If NO → Predicts $178,000
2. If YES → Predicts $188,000



Key Concepts Explained 📚

Predicted Prices

•The predicted price ($178,000 or $188,000) is not random
•It's the **historical average price** of all houses in that category
•Example: $178,000 is the average price of all houses with ≤2 bedrooms in our data

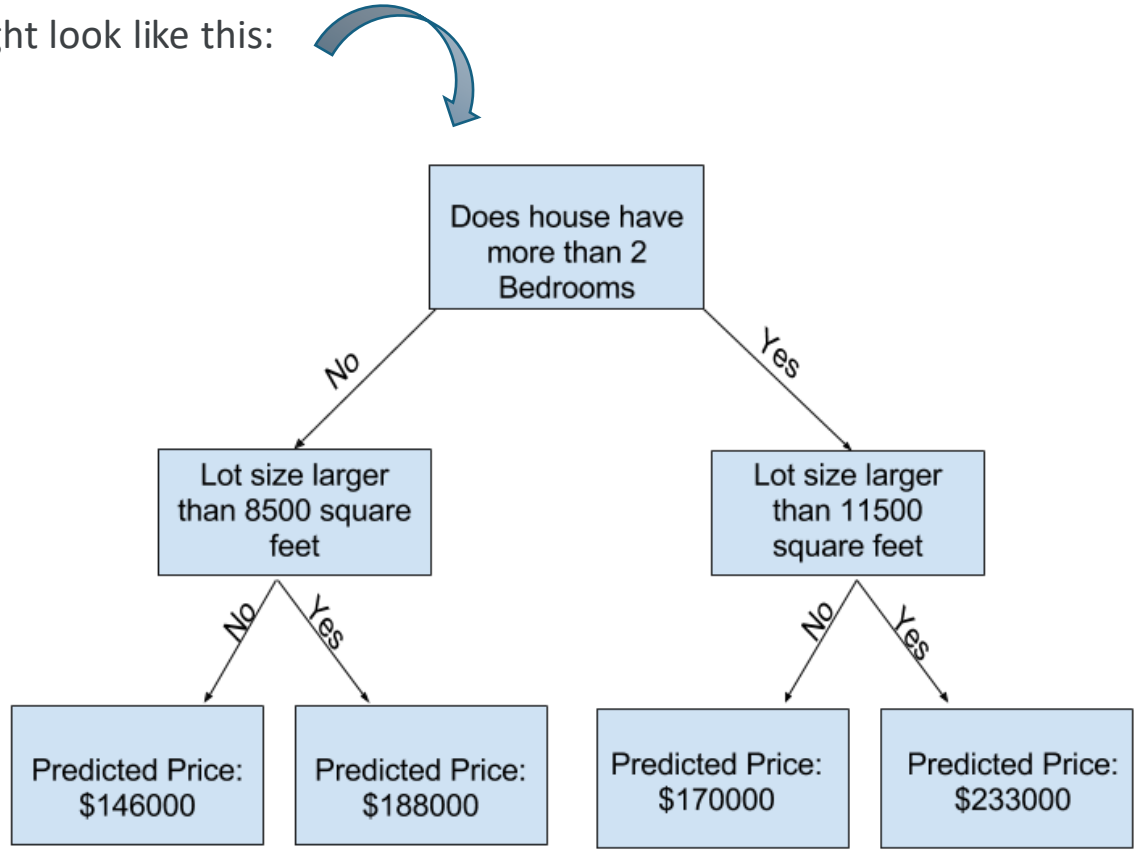
Important Terms to Remember for Modeling

•**Fitting** or **Training**: The process of teaching our model to recognize patterns in data
•**Training Data**: The historical data we use to teach our model
•**Prediction**: Using our trained model to estimate prices for new houses

Improving Decision Tree

We can capture more factors of a house using a tree that has more "splits." These are called "deeper" trees.

A decision tree that also considers the total size of each house's lot might look like this:



Leaf
- The end points at the bottom of the tree
- Where we make our final price predictions
- In this tree, we have 4 leaves ($146K, $188K, $170K, $233K)

Split
- The decision points in the tree
- Where we ask questions about features
- This tree has 3 splits (1 bedroom split, 2 lot size splits)

Considers multiple factors
Provides more specific predictions
Better reflects real market conditions

Basic Data Exploration

Pandas is the primary tool data scientists use for exploring and manipulating data. Most people abbreviate pandas in their code as pd.

```
import pandas as pd
```

```
# save filepath to variable for easier access
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
# read the data and store data in DataFrame titled melbourne_data
melbourne_data = pd.read_csv(melbourne_file_path)
# print a summary of the data in Melbourne data
melbourne_data.describe()
```

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	Buildin
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000	13580.000000	7130.0
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075	558.416127	151.96
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634	3990.669241	541.01
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	93.000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	126.00
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	174.00
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	433014.000000	44515.

.describe() is a function form Pandas

- Generates automatic statistical summaries
- Works on numerical columns by default
- Gives you a quick overview of your data

Key Statistics It Shows

count: Number of non-missing values
mean: Average value
std: Standard deviation
min: Minimum value
25%: First quartile
50%: Median
75%: Third quartile
max: Maximum value

Common Use Cases:

Quick data exploration
Checking data quality
Finding outliers
Understanding data distribution

Tej Gorde

First Machine Learning Model

We will start by **selecting data for modeling**. When you have a dataset that's too big and confusing, you need a way to make it simpler.

```
import pandas as pd

melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
melbourne_data.columns
```

```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',
       'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

This shows you all the possible things you can look at (like number of rooms, price, location, etc.)

Start Small:

•Pick just a few important things to look at first

•Use common sense - like picking room count to predict house prices, or the year it’s built in etc.

•Don't try to use everything at once

Keep It Simple: Instead of looking at 20+ different things about each house, maybe start with just 3-4 that you think matter most, like:

•Number of rooms

•Location

•Size of the house

•Number of bathrooms

Think of it like this: If you were buying a house, what are the first 3-4 things you'd want to know? Start with those!

Later on, you'll learn fancy mathematical ways to pick the best features, but starting with your intuition is often a good first step.

Handling Missing Values 🧹

The code uses dropna() to handle missing values:

```
# The Melbourne data has some missing values (some houses for which some variables weren't recorde
d.)
# We'll learn to handle missing values in a later tutorial.
# Your Iowa data doesn't have missing values in the columns you use.
# So we will take the simplest option for now, and drop houses from our data.
# Don't worry about this much for now, though the code is:

# dropna drops missing values (think of na as "not available")
melbourne_data = melbourne_data.dropna(axis=0)
```

Key points about dropna():

axis=0 means drop rows with any missing values
Think of "na" as "not available"
This is the simplest (though not always best) way to handle missing data

Best Practices for Feature Selection 📝

Start Simple

1. Begin with a few meaningful variables
2. Choose features that intuitively affect the target
3. Add complexity gradually

Consider Data Quality

1. Check for missing values
2. Look at data types
3. Consider the reliability of each feature

Avoid Redundancy

1. Don't select multiple columns that represent the same information
2. Example: 'Rooms' and 'Bedroom2' might be redundant

Think About Relevance

1. Select features that likely influence your target
2. Example for house prices:
    1. ✅ Rooms, Location, Size
    2. ❌ House Color, Street Name

Building your Model

You will use the **scikit-learn** library to create your models.

The steps to building and using a model are:

• **Define:** What type of model will it be? A decision tree? Some other type of model? Some other parameters of the model type are specified too.

• **Fit:** Capture patterns from provided data. This is the heart of modeling. It feed your data in to the model

• **Predict:** Just what it sounds like to predict our values. Use trained model on new data

• **Evaluate**: Determine how accurate the model's predictions are. Compare predictions to actual values

```
from sklearn.tree import DecisionTreeRegressor

# Define model. Specify a number for random_state to ensure same results each run
melbourne_model = DecisionTreeRegressor(random_state=1)

# Fit model
melbourne_model.fit(X, y)
```

```
DecisionTreeRegressor(random_state=1)
```

Many machine learning models allow some randomness in model training. Specifying a number for random_state ensures you get the same results in each run. This is considered a good practice. You use any number, and model quality won't depend meaningfully on exactly what value you choose.

Building your Model

Predict

```
print("Making predictions for the following 5 houses:")
print(X.head())
print("The predictions are")
print(melbourne_model.predict(X.head()))
```

```
Making predictions for the following 5 houses:
   Rooms  Bathroom  Landsize  Lattitude  Longtitude
1      2       1.0     156.0   -37.8079    144.9934
2      3       2.0     134.0   -37.8093    144.9944
4      4       1.0     120.0   -37.8072    144.9941
6      3       2.0     245.0   -37.8024    144.9993
7      2       1.0     256.0   -37.8060    144.9954
The predictions are
[1035000. 1465000. 1600000. 1876000. 1636000.]
```

This means:

- First house (2 rooms): Predicted price $1,035,000
- Second house (3 rooms): Predicted price $1,465,000
- And so on...

We train on all data (fit)Then can predict for any houses. Usually predict for new houses. Here we predicted existing ones as example

This is a basic model. Real-world applications might need more sophisticated approaches, but these same four steps (Define, Fit, Predict, Evaluate) apply to all machine learning models!

Model Validation

You've built a model. But how good is it?

We will perform Model validation to understand our model accuracy

It's tempting to test your model by making predictions on your training data (the data you used to build the model) and comparing those predictions to the actual values, that's is a major mistake. This approach can be misleading because your model has already seen this data during training.

Think of it like memorizing answers to a test instead of understanding the material. Sure, you'll do well on questions you've seen before, but you'll struggle with new questions. The same applies to our model - we want to know how it performs on houses it hasn't "seen" before.

When evaluating model quality, we need a systematic way to measure accuracy. If you have predictions and actual values for 10,000 houses, it would be overwhelming to look at each one individually. This is why we use metrics like Mean Absolute Error (MAE). Here's how MAE works...

The prediction error for each house is:

```
error=actual-predicted
```

For example, if a house actually costs $150,000 and your model predicted $100,000, the error is $50,000.

MAE

Once we have a model, here is how we calculate the mean absolute error:

```
from sklearn.metrics import mean_absolute_error

predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

```
434.71594577146544
```

The resulting number tells you, on average, how far off your predictions are. For example, if the MAE is 434.715 (as shown in the output), this means your model's predictions are typically off by about $434.715. In real estate terms, this helps you understand if your model is making reasonable predictions or if it needs improvement.

This metric is particularly useful because:
1. It's easy to understand - it's in the same units as your prediction (dollars, in this case)
2. It treats all sizes of errors proportionally
3. It gives you a clear benchmark for model improvement

The Problem with "In-Sample" Scores¶

The Door Color Example:
Let's say all expensive houses in your data happened to have green doors
Your model learns: "Green door = Expensive house"
This works great on your training data
But in real life, door color doesn't affect house prices!

So the solution is to split the data into TRAINING and TESTING / VALIDATION data

```
from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and target
# The split is based on a random number generator. Supplying a numeric value to
# the random_state argument guarantees we get the same split every time we
# run this script.
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)
# Define model
melbourne_model = DecisionTreeRegressor()
# Fit model
melbourne_model.fit(train_X, train_y)

# get predicted prices on validation data
val_predictions = melbourne_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))
```

Never use validation data for training!

1.Training Data (train_X, train_y):
•Used to teach/train the model
•Typically about 70-80% of your data
•train_X: Features (rooms, size, etc.)
•train_y: Target (prices)

2.Validation Data (val_X, val_y):
•Used to test the model
•Typically about 20-30% of your data
•val_X: Features for testing
•val_y: Actual prices to compare with predictions

Validation

```
# get predicted prices on validation data
val_predictions = melbourne_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))
```

```
265806.91478373145
```

The mean absolute error for the in-sample data aka Training Data was about 500 dollars. That means our model is almost perfect! Like getting 99% on a practice test.

Out-of-sample aka Testing data is more than 250,000 dollars. That means our mode is poor. Yikes! Like failing the test badly

To Put This in Perspective:

•Average house price: $1.1 million
•Error on new houses: $250,000
•This means we're off by about 25% (quarter) of a house's value
•That's like guessing a $400,000 house could be anywhere from $150,000 to $650,000!

Why This is Bad:
•Imagine telling a home buyer: "This house costs somewhere between $850,000 and $1,350,000"
•That's too uncertain to be useful
•No one would trust predictions with such large errors

Overfitting and Underfitting

Tree Depth & Groups:

•With 1 split: 2 groups
•With 2 splits: 4 groups
•With 4 splits: 8 groups
•With 10 splits: 1024 groups!

The Problem:

1.Deep Trees (Overfitting)

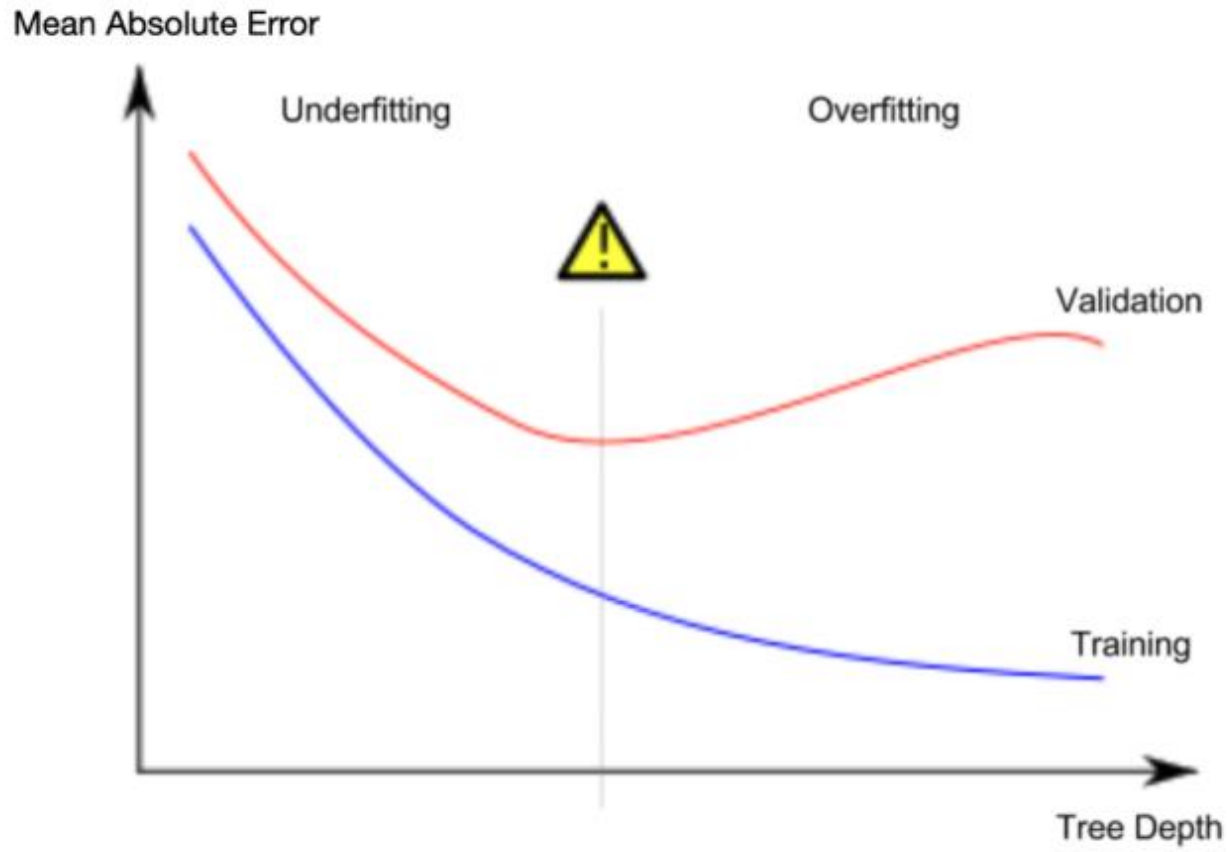
1. Too many splits = too many small groups
2. Like memorizing instead of learning
3. Perfect on training data, bad on new houses

2.Shallow Trees (Underfitting)

1. Too few splits (like 2-4 groups)
2. Too simple, misses important patterns
3. Bad on both training and new houses

The Solution:

•Find the "sweet spot" between too deep and too shallow
•Use validation data to find the best depth
•Look for where validation accuracy is best –
let's look at this in next slide



Tej Gorde

YOUR TURN

Machine learning competitions are a great way to improve your data science skills and measure your progress.

In the next exercise, you will create and submit predictions for the House Prices Competition for Kaggle Learn Users.

Your Turn

Use what you've learned in the course to **create a submission** to a Kaggle competition!

https://www.kaggle.com/code/alexisbcook/machine-learning-competitions