

# **Machine learning**

**basics**

Kunal Khurana

2023-12-24

# Table of contents

<b>Intro to machine learning</b>	<b>3</b>
ML models (basic steps) . . . . .	3
Inference . . . . .	4
Step 3 (Specify and fit the model) . . . . .	6
Step 4 (Make predictions) . . . . .	6
<b>Model Validation</b>	<b>8</b>
<b>Underfitting and Overfitting</b>	<b>9</b>

# Intro to machine learning

## ML models (basic steps)

- Step- 1 Figure out which column would you use to make a prediction.  
make prediction target as y
- Step 2 Assign different features you'd use to make predictions  
assign 'features' to variable X
- Step 3 - Specify and Fit model
- Step 4 - Make predictions
- example of decision tree model (simple basic machine learning model; steps-)  
capturing data (training or fitting the model)  
predicting (based on what a model is fed)  
evaluation (how accurate the predictions are)
- more factors can be fed into the decision tree that has more 'splits'
- these trees are called 'deeper' trees
- the point where we make a prediction is called 'leaf'

```
import pandas as pd
file = "melb_data.csv"
data = pd.read_csv(file)
print(data.describe())
```

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610079
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000

## Inference

- count- how many rows have non-missing values
- mean- average
- std- measures the numerical spread of values

```
print(data.columns)
```

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

```
# dropping missing values
data = data.dropna(axis=0)
print(data.columns)
print(data.describe())
```

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

	Rooms	Price	Distance	Postcode	Bedroom2	\
count	6196.000000	6.196000e+03	6196.000000	6196.000000	6196.000000	
mean	2.931407	1.068828e+06	9.751097	3101.947708	2.902034	
std	0.971079	6.751564e+05	5.612065	86.421604	0.970055	
min	1.000000	1.310000e+05	0.000000	3000.000000	0.000000	
25%	2.000000	6.200000e+05	5.900000	3044.000000	2.000000	
50%	3.000000	8.800000e+05	9.000000	3081.000000	3.000000	
75%	4.000000	1.325000e+06	12.400000	3147.000000	3.000000	

max	8.000000	9.000000e+06	47.400000	3977.000000	9.000000
-----	----------	--------------	-----------	-------------	----------

  

	Bathroom	Car	Landsize	BuildingArea	YearBuilt \
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	1.576340	1.573596	471.006940	141.568645	1964.081988
std	0.711362	0.929947	897.449881	90.834824	38.105673
min	1.000000	0.000000	0.000000	0.000000	1196.000000
25%	1.000000	1.000000	152.000000	91.000000	1940.000000
50%	1.000000	1.000000	373.000000	124.000000	1970.000000
75%	2.000000	2.000000	628.000000	170.000000	2000.000000
max	8.000000	10.000000	37000.000000	3112.000000	2018.000000

  

	Lattitude	Longtitude	Propertycount
count	6196.000000	6196.000000	6196.000000
mean	-37.807904	144.990201	7435.489509
std	0.075850	0.099165	4337.698917
min	-38.164920	144.542370	389.000000
25%	-37.855438	144.926198	4383.750000
50%	-37.802250	144.995800	6567.000000
75%	-37.758200	145.052700	10175.000000
max	-37.457090	145.526350	21650.000000

```
# using *dot notation* to predict the prediction target, y

y = data.Price

# building a model with few features
# features are the columns that are used to make predictions

data_features = ['Rooms', 'Bedroom2', 'Landsize']

# saving these in variable x

x = data[data_features]

print(x.describe())
```

	Rooms	Bedroom2	Landsize
count	6196.000000	6196.000000	6196.000000
mean	2.931407	2.902034	471.006940

std	0.971079	0.970055	897.449881
min	1.000000	0.000000	0.000000
25%	2.000000	2.000000	152.000000
50%	3.000000	3.000000	373.000000
75%	4.000000	3.000000	628.000000
max	8.000000	9.000000	37000.000000

```
print(x.head())
```

	Rooms	Bedroom2	Landsize
1	2	2.0	156.0
2	3	3.0	134.0
4	4	3.0	120.0
6	3	4.0	245.0
7	2	2.0	256.0

### Step 3 (Specify and fit the model)

```
from sklearn.tree import DecisionTreeRegressor

data_model = DecisionTreeRegressor(random_state = 1) # ensures same results

# fit
data_model.fit(x,y)
```

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

### Step 4 (Make predictions)

```
print("Making predicitions for the following houses:")
print(x.head())
print("The predictions are: ")
print(data_model.predict(x.head()))
```

Making predicitions for the following houses:

	Rooms	Bedroom2	Landsize
1	2	2.0	156.0
2	3	3.0	134.0
4	4	3.0	120.0
6	3	4.0	245.0
7	2	2.0	256.0

The predictions are:

[ 993200.	1035333.33333333	1600000.	1876000.
1636000.	]		

# Model Validation

- mean absolute error is used
- function from scikit-learn library (train\_test\_split) is used

```
from sklearn.metrics import mean_absolute_error
```

```
predicted_home_prices = data_model.predict(x)  
mean_absolute_error(y, predicted_home_prices)
```

213886.172706383

```
from sklearn.model_selection import train_test_split
```

```
train_x, val_x, train_y, val_y = train_test_split(x, y, random_state = 0)
```

```
# define model  
data_model = DecisionTreeRegressor()
```

```
# fit model  
data_model.fit(train_x, train_y)
```

```
# get predicted prices  
val_predictions = data_model.predict(val_x)  
print(mean_absolute_error(val_y, val_predictions))
```

458475.58063432045

- This means that the MAE was more than double

```
# check model attributes  
print(dir(DecisionTreeRegressor))
```

```
['__abstractmethods__', '__annotations__', '__class__', '__delattr__', '__dict__', '__dir__',
```



# Underfitting and Overfitting

- Overfitting- predictions made with less data for each tree >(or could be the DecisionTreeRegressor contains many Leaves  $>2^{10}$ )
- Underfitting- predictions made with huge data >(or could be the DecisionTreeRegressor doesn't contain many leaves  $<2^2$ )
- we use **max\_leaf\_nodes** argument to control overfitting and underfitting

```
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor

def get_mae(max_leaf_nodes, train_x, val_x, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes = max_leaf_nodes, random_state = 0)
    model.fit(train_x, train_y)
    preds_val = model.predict(val_x)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)

# comparing MAE with different values of max_leaf_nodes

for max_leaf_nodes in [5, 50, 500, 5000]:
    my_mae = get_mae(max_leaf_nodes, train_x, val_x, train_y, val_y)
    print(f"Max leaf nodes: {max_leaf_nodes} \t\t Mean Absolute Error: {max_leaf_nodes, my
```

Max leaf nodes: 5	Mean Absolute Error: (5, 402424.0910391075)
Max leaf nodes: 50	Mean Absolute Error: (50, 386677.9612192755)
Max leaf nodes: 500	Mean Absolute Error: (500, 423585.7986927151)
Max leaf nodes: 5000	Mean Absolute Error: (5000, 461015.283668536)