Intro to Machine Learning

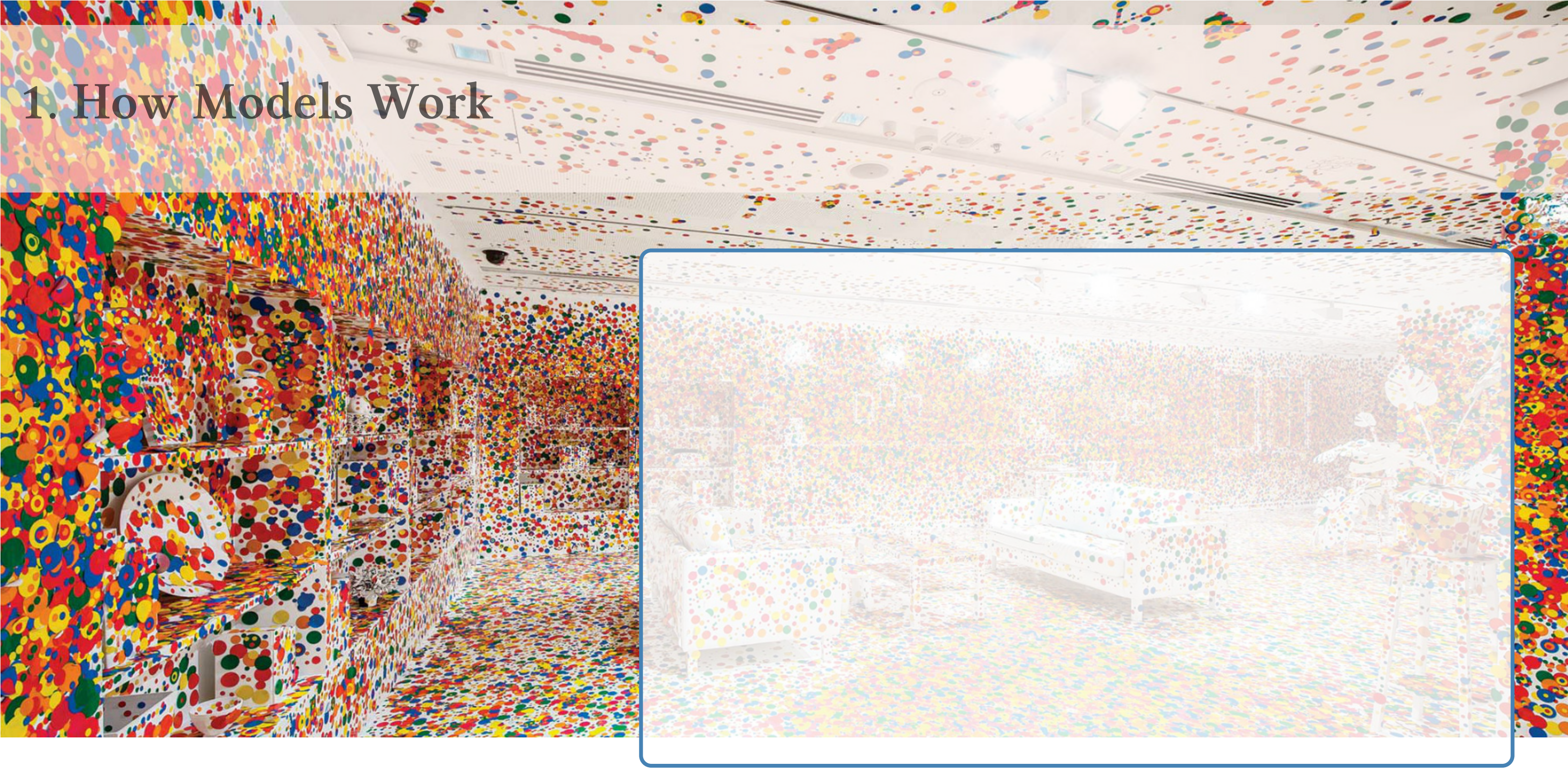
Ismael Sallami Moreno

Computer Engineer

This book has been written by a student of 2 of double degree Computer Engineer and Business Management. Is a fast book to have a little idea about machine learning. I wait you like it. Thanks you!

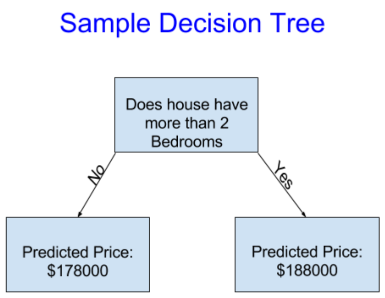
Art. No xxxxx
ISBN xxx--xx--xxxx--xx--x
Edition 0.0

Περιεχόμενα



1. How Models Work

In this field the model works by a simple way. For example, we can use trees. Imagine that you want to sell a house with bathroom and other house without bathroom. In this case you can apply the next tree:

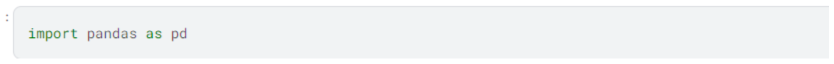


Σχήμα 1.1: In this case compare with the fact that if the house have more than two bathrooms or no, but is the same idea

Based on this we can use different trees, more complex or different methods, The next explanations will be with this type of trees or other more complex.

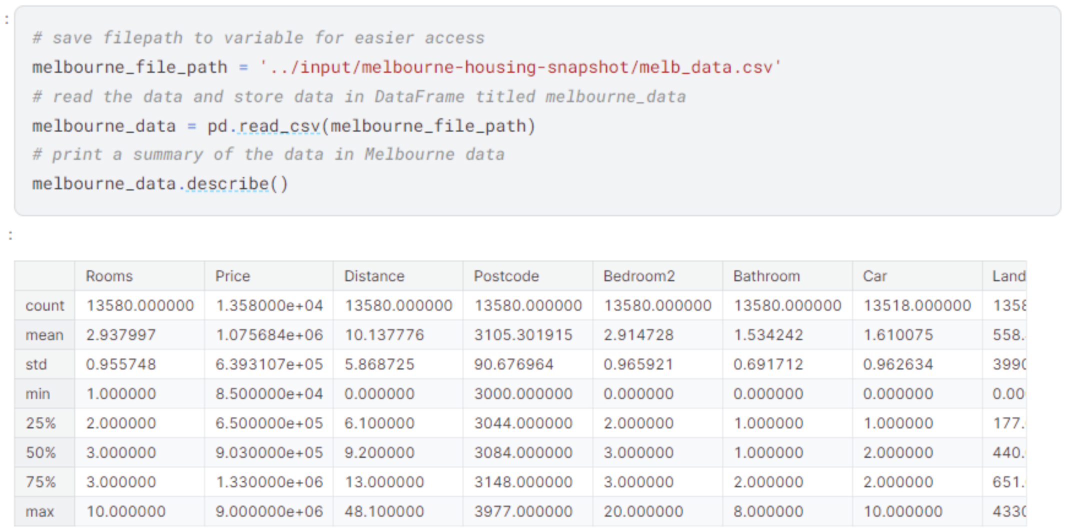
2. Using Pandas to work data

Panda is a Phyton library. It help you in order to you can work with different dataset. We use that of the next way:



```
import pandas as pd
```

Σχήμα 2.1: Import Panda

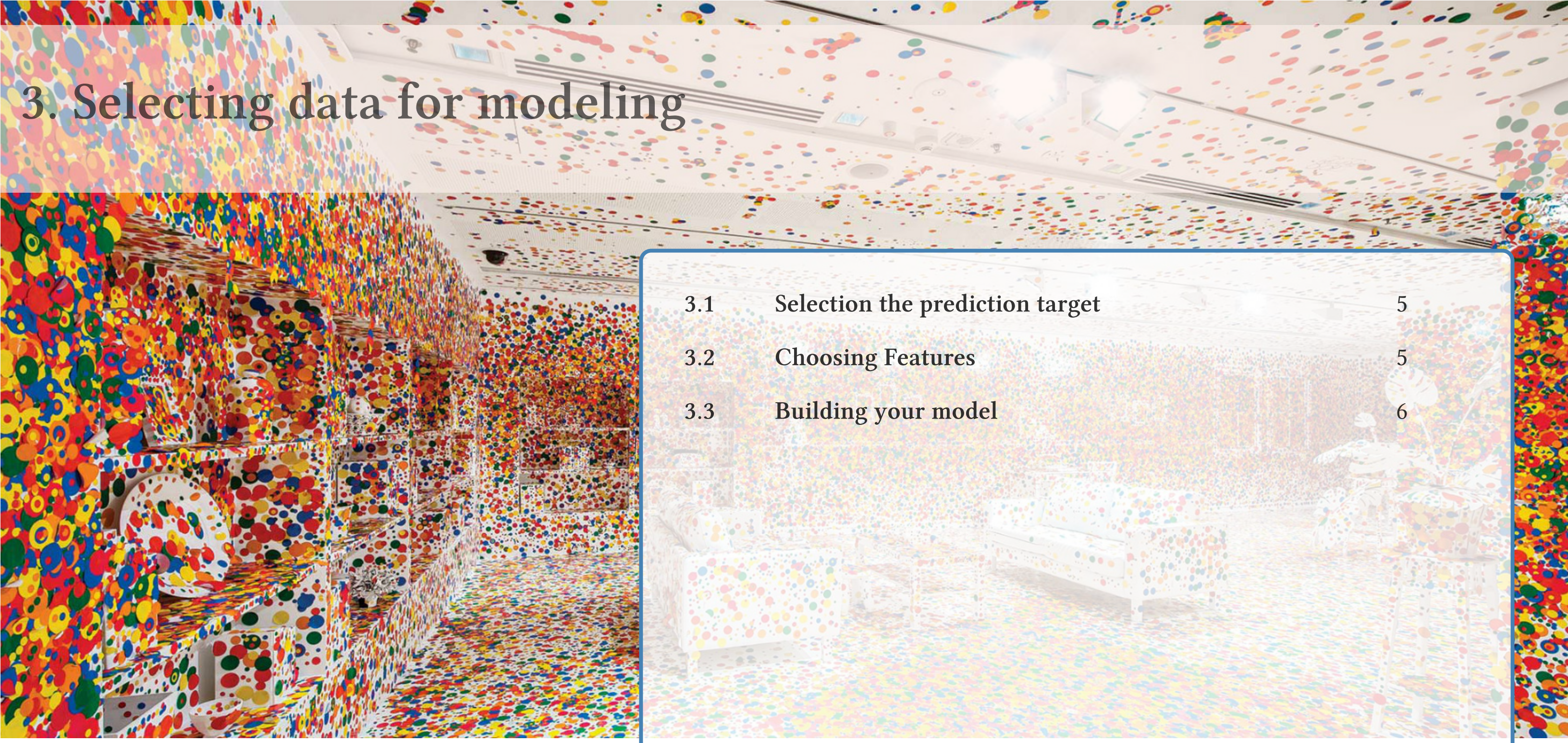


```
# save filepath to variable for easier access
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
# read the data and store data in DataFrame titled melbourne_data
melbourne_data = pd.read_csv(melbourne_file_path)
# print a summary of the data in Melbourne data
melbourne_data.describe()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

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Land
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000	1358
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075	558.
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634	3990
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.00
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	4330

Σχήμα 2.2: Example of use of Pandas

How we can see in the photo, we have to indicate the path of the dataset, then we have to create a "DataFrame" and for last we can paint this as a table.



3. Selecting data for modeling

We can see the columns of our dataset with the next code:

Listing 3.1: COMMANDS TO GET COLUMNS OF OUR DATASET

```
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
```

We can have the occasion when in our dataset we have missing values for various reasons. So we can delete this missing values with the next command:

```
    melbourne_data = melbourne_data.dropna(axis=0)
```

3.1 Selection the prediction target

We will use the dot notation to select the column we want to predict, which is called the prediction target. By convention, the prediction target is called **y**:

```
y = melbourne_data.Price
```

3.2 Choosing Features

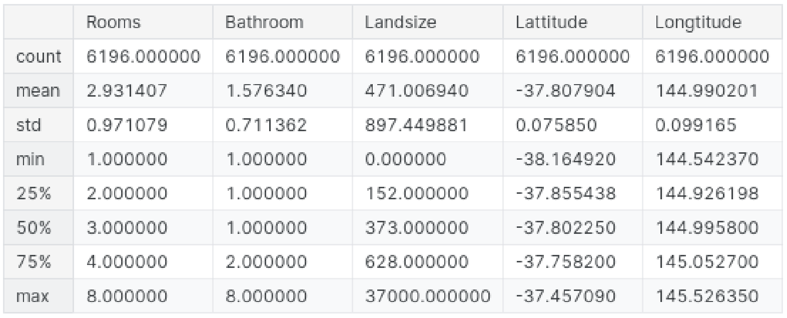
Now, we have to select the features, which we go to train our model. In this case we go to provide a list of names:

```
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude',
'Longtitude']
```

By comnvention, this data is called **x**.

```
X = melbourne_data[melbourne_features]
```

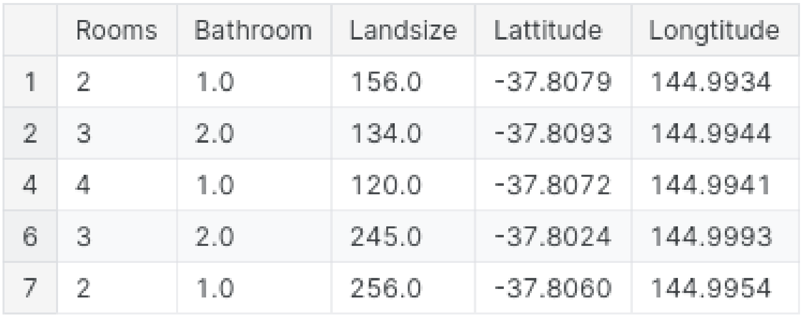
If we introduce the command **X.describe()**:



	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	2.931407	1.576340	471.006940	-37.807904	144.990201
std	0.971079	0.711362	897.449881	0.075850	0.099165
min	1.000000	1.000000	0.000000	-38.164920	144.542370
25%	2.000000	1.000000	152.000000	-37.855438	144.926198
50%	3.000000	1.000000	373.000000	-37.802250	144.995800
75%	4.000000	2.000000	628.000000	-37.758200	145.052700
max	8.000000	8.000000	37000.000000	-37.457090	145.526350

Σχήμα 3.1: Example

If we introduce **X.head()**:



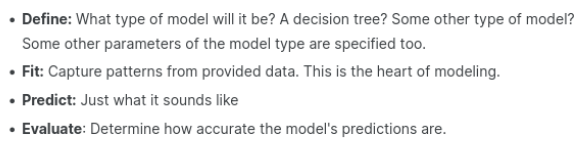
	Rooms	Bathroom	Landsize	Lattitude	Longtitude
1	2	1.0	156.0	-37.8079	144.9934
2	3	2.0	134.0	-37.8093	144.9944
4	4	1.0	120.0	-37.8072	144.9941
6	3	2.0	245.0	-37.8024	144.9993
7	2	1.0	256.0	-37.8060	144.9954

Σχήμα 3.2: Example

3.3 Building your model

We go to use the scikit-learn library to create our models. This library is writtten as sklearn. The steps to building and using a model are:

Example of use:

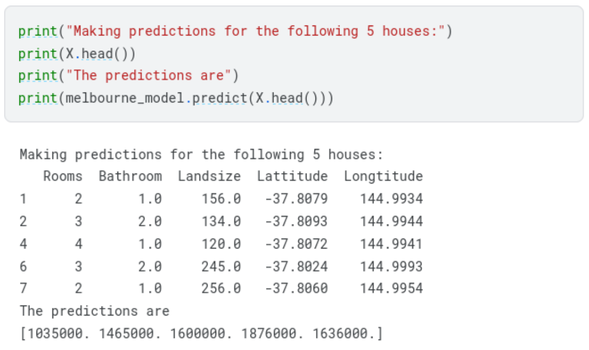


- **Define:** What type of model will it be? A decision tree? Some other type of model? Some other parameters of the model type are specified too.
- **Fit:** Capture patterns from provided data. This is the heart of modeling.
- **Predict:** Just what it sounds like
- **Evaluate**: Determine how accurate the model's predictions are.

Σχήμα 3.3: Example

```
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 due to our model )
#We put random_state=1  in order to you are ensuring that the model is
fitted in the same manner every time the code is executed.

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

If we introduce the next commands we have the next output:



```
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.]
```

Σχήμα 3.4: Example

4. What is model validation?

We ask we that how much is the quality of our model. If we want to know the answer, we have to detect the error of our model. For this we go to use MAE (Mean Absolute Error). We have to use the next formula:

error=actual-predicted

For example if the current house cost 100000$ and yoou model predicted that it costs 130000$, the error is the 30000.

Our total error is the result of the average of the various errors. Next, er have to predict an error about a model. The model is:

```
    # Data Loading Code Hidden Here
import pandas as pd

# Load data
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
# Filter rows with missing price values
filtered_melbourne_data = melbourne_data.dropna(axis=0)
# Choose target and features
y = filtered_melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = filtered_melbourne_data[melbourne_features]

from sklearn.tree import DecisionTreeRegressor
# Define model
melbourne_model = DecisionTreeRegressor()
# Fit model
melbourne_model.fit(X, y)
```

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

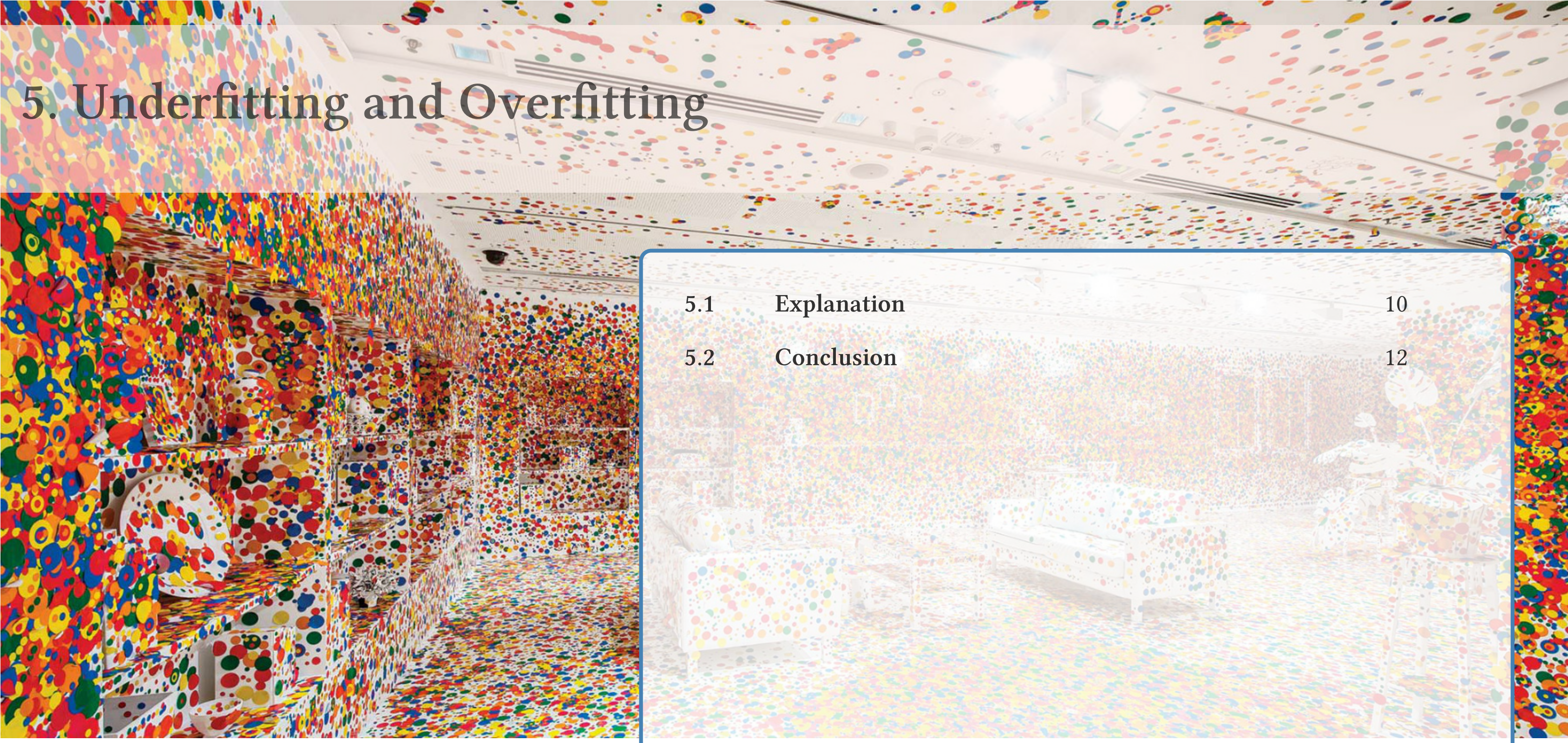
```
from sklearn.metrics import mean_absolute_error

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

4.1 Coding it

The scikit-learn library has a function train_test_split to break up the data into two pieces. We'll use some of that data as training data to fit the model, and we'll use the other data as validation data to calculate mean_absolute_error.

Here is the code:

```
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))
```

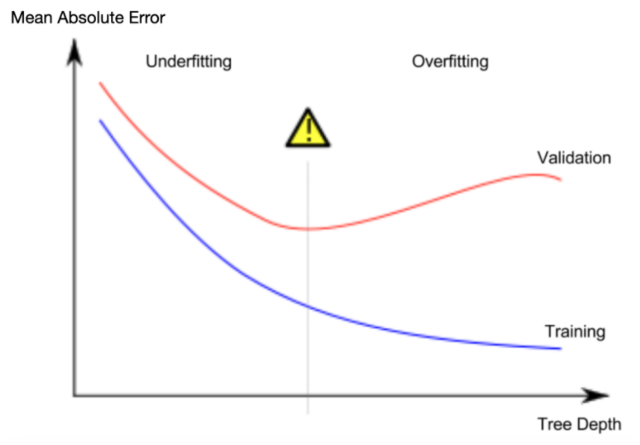


5. Underfitting and Overfitting

5.1 Explanation

When we divide the houses amongst many leaves, we also have fewer houses in each leaf. Leaves with very few houses will make predictions that are quite close to those homes' actual values, but they may make very unreliable predictions for new data (because each prediction is based on only a few houses).This is a phenomenon called **overfitting**.

At an extreme, if a tree divides houses into only 2 or 4, each group still has a wide variety of houses. Resulting predictions may be far off for most houses, even in the training data (and it will be bad in validation too for the same reason). When a model fails to capture important distinctions and patterns in the data, so it performs poorly even in training data, that is called **underfitting**.



Σχήμα 5.1: Comparison between underfitting and overfitting

Example:

We can use a utility function to help compare MAE scores from different values for max_leaf_nodes:

```
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)

The data is loaded into train_X, val_X, train_y and val_y using the code
you've already seen
(and which you've already written).
```

```
# Data Loading Code Runs At This Point
import pandas as pd

# Load data
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
# Filter rows with missing values
filtered_melbourne_data = melbourne_data.dropna(axis=0)
# Choose target and features
y = filtered_melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = filtered_melbourne_data[melbourne_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and
target
train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

We can use a for-loop to compare the accuracy of models built with different values for max_leaf_nodess.

```
# compare MAE with differing 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("Max leaf nodes: %d  \t\t
    Mean Absolute Error:  %d" %(max_leaf_nodes, my_mae))
```

Output would be:

```
Max leaf nodes: 5 Mean Absolute Error: 347380
Max leaf nodes: 50 Mean Absolute Error: 258171
Max leaf nodes: 500 Mean Absolute Error: 243495
Max leaf nodes: 5000 Mean Absolute Error: 254983
```

Of the options listed, 500 is the optimal number of leaves.

5.2 Conclusion

Some problems can suffer our model:
· Overfitting: capturing spurious patterns that won't recur in the future, leading to less accurate predictions.
· Underfitting: failing to capture relevant patterns, again leading to less accurate predictions.

6. Random Forests

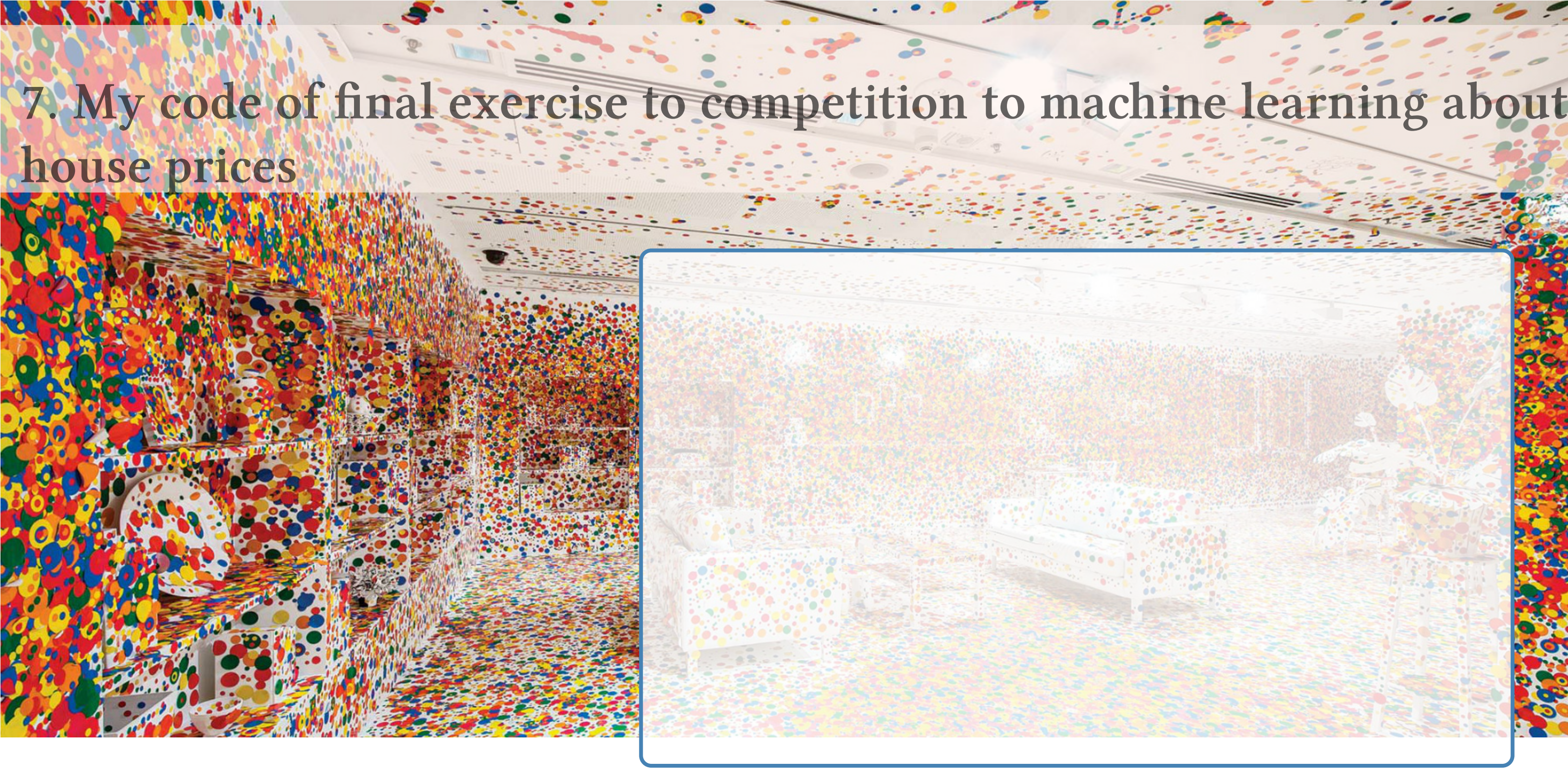
Decision trees leave you with a difficult decision. A deep tree with lots of leaves will overfit because each prediction is coming from historical data from only the few houses at its leaf. But a shallow tree with few leaves will perform poorly because it fails to capture as many distinctions in the raw data.

Even today's most sophisticated modeling techniques face this tension between underfitting and overfitting. But, many models have clever ideas that can lead to better performance. We'll look at the random forest as an example.

The random forest uses many trees, and it makes a prediction by averaging the predictions of each component tree. It generally has much better predictive accuracy than a single decision tree and it works well with default parameters. If you keep modeling, you can learn more models with even better performance, but many of those are sensitive to getting the right parameters.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

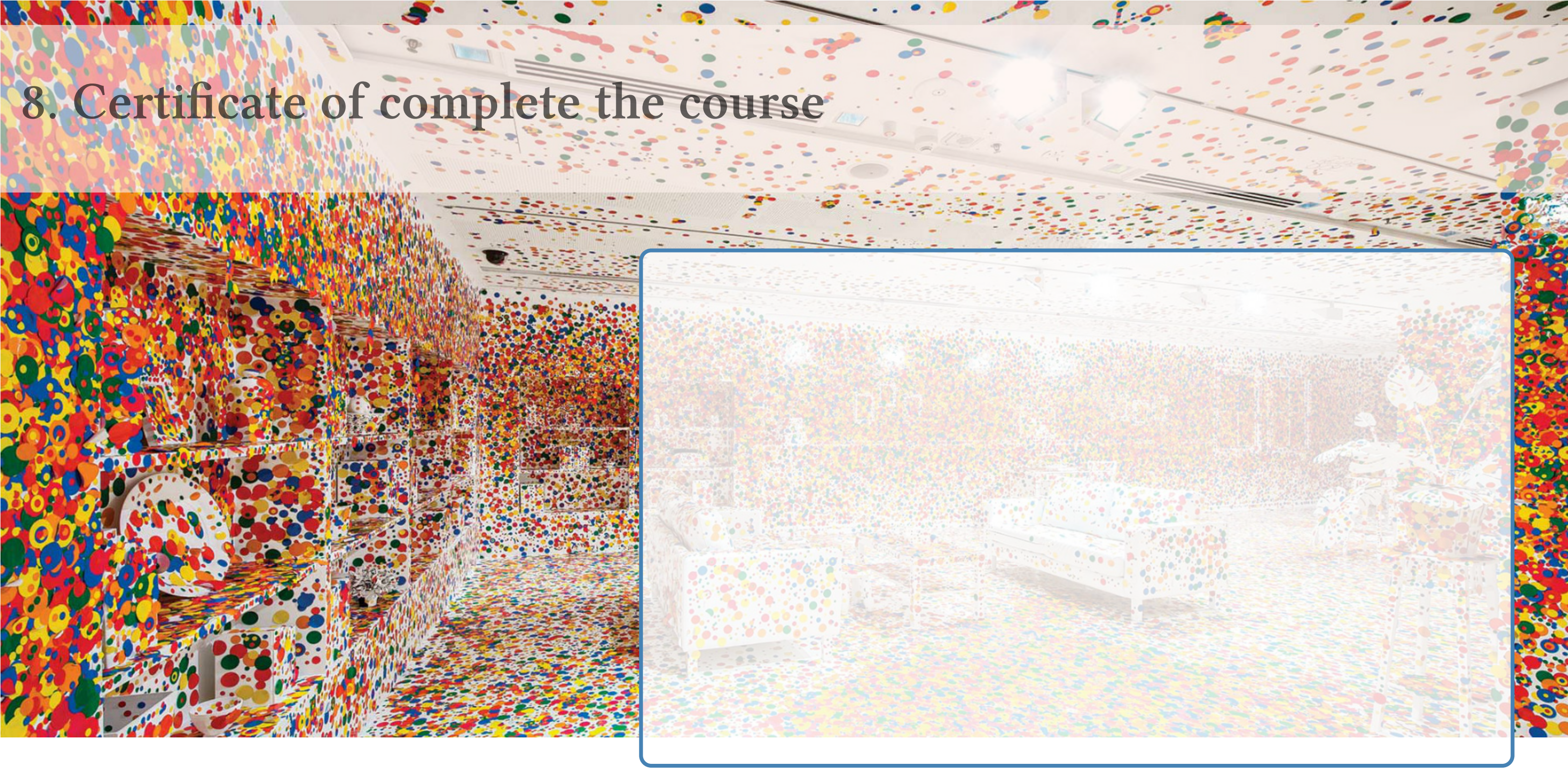
7. My code of final exercise to competition to machine learning about house prices

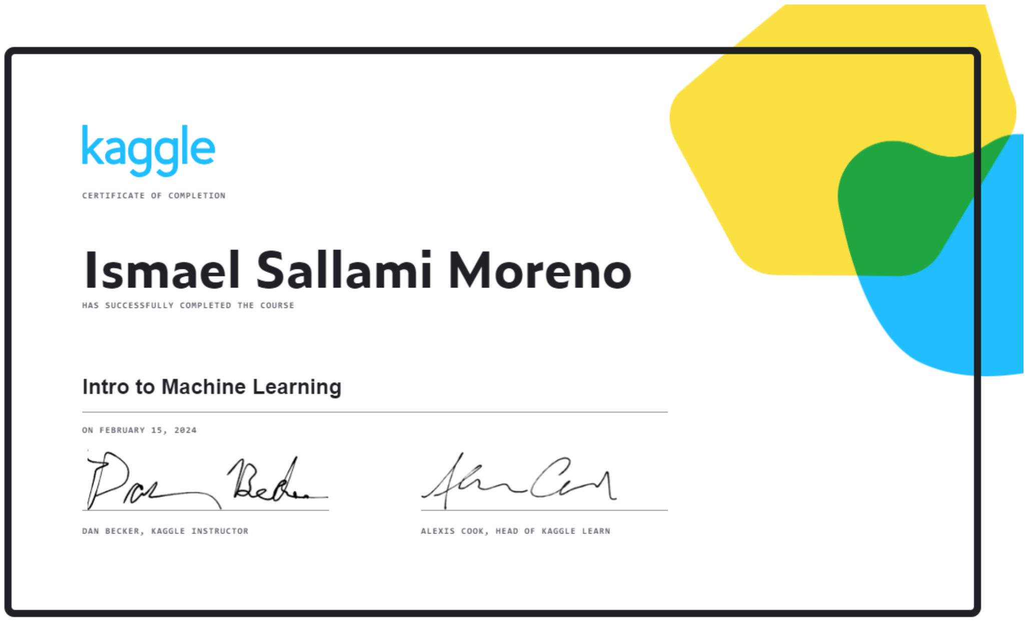


```
# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex7 import *

# Set up filepaths
import os
if not os.path.exists("../input/train.csv"):
    os.symlink("../input/home-data-for-ml-course/train.csv",
    "../input/train.csv")
    os.symlink("../input/home-data-for-ml-course/test.csv",
    "../input/test.csv")

# Import helpful libraries
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

# Load the data, and separate the target
iowa_file_path = '../input/train.csv'
home_data = pd.read_csv(iowa_file_path)
y = home_data.SalePrice

# Create X (After completing the exercise, you can return to modify this
line!)
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath',
'BedroomAbvGr', 'TotRmsAbvGrd']

# Select columns corresponding to features, and preview the data
X = home_data[features]
X.head()

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)
```

```
# Define a random forest model
rf_model = RandomForestRegressor(random_state=1)
rf_model.fit(train_X, train_y)
rf_val_predictions = rf_model.predict(val_X)
rf_val_mae = mean_absolute_error(rf_val_predictions, val_y)

print("Validation MAE for Random Forest Model: {:,.0f}".format(rf_val_mae))

# To improve accuracy, create a new Random Forest model which you will
train on all training data
rf_model_on_full_data = RandomForestRegressor(random_state=1)

# fit rf_model_on_full_data on all data from the training data
rf_model_on_full_data.fit(X,y)

# path to file you will use for predictions
test_data_path = '../input/test.csv'

# read test data file using pandas
test_data = pd.read_csv(test_data_path)

# create test_X which comes from test_data but includes only the columns
you used for prediction.
# The list of columns is stored in a variable called features
test_X = test_data[features]

# make predictions which we will submit.
test_preds = rf_model_on_full_data.predict(test_X)

# Run the code to save predictions in the format used for competition
scoring

output = pd.DataFrame({'Id': test_data.Id,
                       'SalePrice': test_preds})
output.to_csv('submission.csv', index=False)
```

8. Certificate of complete the course





kaggle

CERTIFICATE OF COMPLETION

Ismael Sallami Moreno

HAS SUCCESSFULLY COMPLETED THE COURSE

Intro to Machine Learning

ON FEBRUARY 15, 2024

[illegible] [illegible]

DAN BECKER, KAGGLE INSTRUCTOR

ALEXIS COOK, HEAD OF KAGGLE LEARN

Σχήμα 8.1: Ismael Sallami Moreno - Intro to Machine Learning.png