**Getting started**

Credits to DanB from Kaggle, am following the tutorials created by him.

Welcome to data analysis

Starting the course considering myself as a layman, will condense as much as I can so that it is easier understanding.

Pre-requisites: Basic programming language with python.

By the end of this series, we should have a basic knowledge and introduction about machine learning and how the models work. Ability to write and build decision tree and random forests.

Using decision trees as a model, it may not be so accurate but easier to start off with.

Our problem- Predict how much a house is worth of!

Example of decision tree

Is the house is having two rooms

Yes- price is $180000

No- price is $160000

The price is obtained from historical average of house prices. We create a pattern by analysing the data. We use the data to split into groups and then decide to which group the house belongs to. This process is called fitting and the data which we use is called training data.

Using this model we predict prices of additional houses.

Next step- Improve the decision tree

Since for handling data and predicting the prices, just one level may not be sufficient. In such case, we take multiple factors into consideration and design deeper trees. For each condition, we travel through the trees by deciding the path based on the condition/characteristics. Finally we predict the price in the last level which is called the leaf node.

Working with our data

We'll also see examples working this problem using data from Melbourne, Australia. We will then write code to build a model predicting prices in the American state of Iowa, where you will be investing.

Before you start coding, pull up descriptions of the data fields for your Iowa Data and the Melbourne Data.

Using Pandas in getting familiar with the data:

We will be using python code over here, so it is advisable to learn a bit or basics about python from the specified link

We are going to use DataFrame which is similar to an excel sheet or a SQL table. Pandas Dataframe is having efficient tools which is useful in handling and analysing data.

**My python code**

import pandas as pd

#saving filepaths for easier access

melbourne\_file = 'C:/Users/Arjun/Desktop/Project december/melb\_data.csv';

#reading data from the file using Pandas

melbourne\_contents = pd.read\_csv(melbourne\_file);

print(melbourne\_contents) import pandas as pd

#saving filepaths for easier access

melbourne\_file = 'C:/Users/Arjun/Desktop/Project december/melb\_data.csv';

#reading data from the file using Pandas

melbourne\_contents = pd.read\_csv(melbourne\_file);

print(melbourne\_contents)

**Doing the same for Iowa dataset and then printing the column names in both the data sets**

#reading the contents from iowa file as a part of the exercise

iowa\_file = 'C:/Users/Arjun/Desktop/Project december/train.csv';

iowa\_contents = pd.read\_csv(iowa\_file);

print(iowa\_contents);

#to print the column names from IOWA and Melbourne dataset

print(melbourne\_contents.columns);

print(iowa\_contents.columns); #reading the contents from iowa file as a part of the exercise

iowa\_file = 'C:/Users/Arjun/Desktop/Project december/train.csv';

iowa\_contents = pd.read\_csv(iowa\_file);

print(iowa\_contents);

#to print the column names from IOWA and Melbourne dataset

print(melbourne\_contents.columns);

print(iowa\_contents.columns);

Selecting columns from the data separately using the below specified code

## add some content here

# store the series of prices separately .

melbourne\_price\_data = melbourne\_contents.Price;

# to get the top few lines of the series use the head()

print(melbourne\_price\_data.head())

#repeating the same step for IOWA dataset, select one of the columns

iowa\_year\_sold = iowa\_contents.YrSold;

#print first few lines of the selected column using print() and head()

print(iowa\_year\_sold.head())# store the series of prices separately .

melbourne\_price\_data = melbourne\_contents.Price;

# to get the top few lines of the series use the head()

print(melbourne\_price\_data.head())

#repeating the same step for IOWA dataset, select one of the columns

iowa\_year\_sold = iowa\_contents.YrSold;

#print first few lines of the selected column using print() and head()

print(iowa\_year\_sold.head())

Now we are going to build a model which will predict the outcome for us.

So the first step is to choose a column from the data which we are going to predict in other words it is called as prediction target.

Since we are going to predict the price, we are going to use the Price column in both the data sets as the prediction targets.

Next step in this process is to select the columns or predictors which the model should use for prediction.

#to filter out the empty rows

filtered\_melbourne\_contents = melbourne\_contents.dropna(axis=0)

#First step in building the model,choose the prediction target

melbourne\_y = filtered\_melbourne\_contents.Price;

iowa\_y = iowa\_contents.SalePrice;

#second step is to select the list of predcitors which we will be using

melbourne\_predictors = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',

'YearBuilt', 'Lattitude', 'Longtitude'];

#We are now going to use X to denote predictors' data

melbourne\_X = filtered\_melbourne\_contents[melbourne\_predictors];

#Doing the same step for IOWA data

iowa\_predictors = ['LotArea',

'YearBuilt','TotalBsmtSF','1stFlrSF','2ndFlrSF','FullBath','BedroomAbvGr',

'TotRmsAbvGrd','GarageArea'];

iowa\_X= iowa\_contents[iowa\_predictors];

Sometimes we might use all the columns to predict except one, depending on the data we decide the predictors. In our case if we use price/SalePrice to predict we might encounter target leakage.

Building the model

The brief information about steps which we follow while we are building a model

**Define**: Defining the model which we are going to use and setting up the parameters as required.

**Fit**: Capture the patterns which the data exhibit and use it for prediction. Easily the core part of a model.

**Predict**: Predicting the data/outcome.

**Evaluate**: Accuracy of the model, in other words how well the model performs.

Scikit-learn package is the one which we will be using in python to build our models.

To evaluate the performance of the model we need to perform validation. There is a problem with our approach since we use the same data for training and prediction. This makes the performance good always. But when we apply the model to real world data, we might end up getting bad results or results with less accuracy. So there is an approach which we can follow which is splitting the data into training and testing. So we train the model with the data allocated for training the model and test it with data allocated for testing. Using this approach we can easily conclude how well the model performs

In our first method, we use the same set of data for training and prediction. We will be using MAE(Mean Absolute Error) measure to find out the efficiency of our model.

We will be evaluating two methods, first method using same data for training and testing, second method using train test split from scikit learn package which Split arrays or matrices into random train and test subsets.

We will be comparing the error measures for both the approaches.

The code snippet goes as follows

from sklearn.tree import DecisionTreeRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error

#Code for loading the data can be found above in the document skipping that part alone

iowa\_predictors = ['LotArea',

'YearBuilt','TotalBsmtSF','1stFlrSF','2ndFlrSF','FullBath','BedroomAbvGr',

'TotRmsAbvGrd','GarageArea'];

iowa\_X= iowa\_contents[iowa\_predictors];

iowa\_y = iowa\_contents.SalePrice;

#defining model and fitting data for IOWA dataset

iowa\_model = DecisionTreeRegressor()

iowa\_model.fit(iowa\_X, iowa\_y)

#first approach

iowa\_model = DecisionTreeRegressor()

iowa\_model.fit(iowa\_X, iowa\_y)

#performing the prediction for IOWA dataset using first approach same data for test and train

print("Making predictions for the specified 5 house");

print(iowa\_X.head());

print("The predictions are");

print(iowa\_model.predict(iowa\_X.head()))

predicted\_home\_prices = iowa\_model.predict(iowa\_X)

#second approach

#performing it with test train split

iowa\_X\_train,iowa\_X\_test,iowa\_y\_train,iowa\_y\_test = train\_test\_split(iowa\_X,iowa\_y);

iowa\_model.fit(iowa\_X\_train,iowa\_y\_train);

iowa\_val\_predictions = iowa\_model.predict(iowa\_X\_test);

print(mean\_absolute\_error(iowa\_y\_test,iowa\_val\_predictions));

print(mean\_absolute\_error(iowa\_y, predicted\_home\_prices));

The output will be like

30601.7835616-MAE by using train test split approach

35.398173516-MAE by using same data for train and test

Which clearly shows that the model performs really well when we use the same data for training and testing, whereas when we use the split approach we can see the real performance of the model.

**Concluding the first part getting started**

There are multiple methods in scikit learn by which we can create different models and evaluate the performance by using other measures. There are two main things which we should keep in mind overfitting and underfitting.

**Overfitting:** Leads to less performance while predicting the data because the model tries to fit itself with the training data available. Providing so much of information leads to overfitting, which causes the model to perform really well with training data and performance worsens when it encounters unknown data.

**Underfitting:** failing to capture relevant patterns, again leading to less accurate predictions. It performs badly even while predicting trained data.