Housing Prices Prediction Project

Rational Statement:

The prediction of property prices for real estate markets is becoming increasingly important and beneficial. Property prices are a good indicator of both the overall market condition and the economic health of a country. Housing price ranges are of great interest for both buyers and sellers.

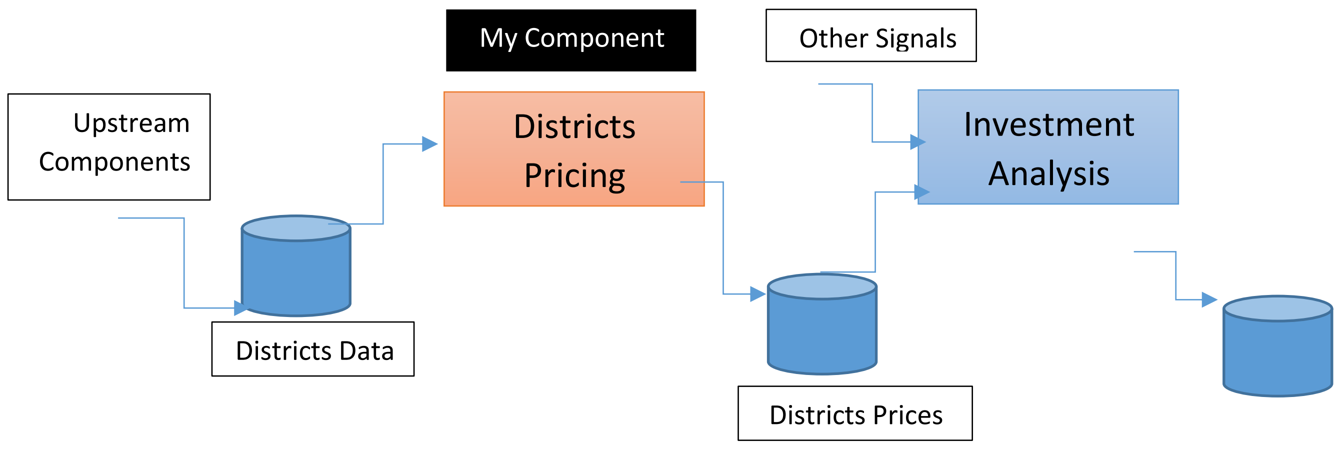
Motivation:

In this project, As a first experience, I want to cover many data science concepts such as data cleaning, feature engineering, dimensionality reduction…etc. as much as possible by approaching every different steps of the machine learning process and trying to understand them deeply. I chose to take Real Estate Prediction as approach. The goal was to predict the price of a given house according to the market prices taking into account different "features" that will be developed in the following sections.

Problem Statement:

The real estate company estimate the houses prices manually by experts who collect data about the district to study it and apply complicated rules to get the median houses prices. This procedure consumes much time, much money and much effort. In addition, there is fairly error percentage.

The company needs a data scientist to build a machine-learning model, which can predict houses prices in districts with low error percentage. This model will go with other signals to feed another ML model that predicts whether is good to invest in the area or not.



<u>Frame the problem:</u>

After discussing with the business manager, we got this information.

- The type of machine learning that will be used is Supervised Machine Learning. Because the training data are labeled and contains the desired class which is the median houses price.

- Because the desired feature that we will predict is value, the task of the machine learning will be Regression.
- The data is not streaming. Therefore, we will use batch learning.
- As the model will predict house price based on many features, the regression type will multivariate regression problem.

<u>Select a performance measure:</u>

As the problem is regression task. Therefore, I will use the most popular function for that which is RMSE (Root Mean Square Error). We use RMSE to measure standard deviation.

$$RMSE = \sqrt{\frac{1}{m}\sum_{1}^{m}(h(x) - y)2}$$

M: number of instances.

X: vector represents all features except the desired feature.

y: the desired solution label. (house price).

h: the hypothesis function.

Data Requirements:

Data Source:

I will use the California Housing Prices dataset(housing.csv) from the following kaggle site:

https://www.kaggle.com/camnugent/california-housing-prices.

The dataset contains 2064020640 observations and 10 attributes (9 predictors and 1 response). Below is a list of the variables with descriptions taken from the original Kaggle site given above.

- longitude: A measure of how far west a house is; a higher value is farther west
- latitude: A measure of how far north a house is; a higher value is farther north
- housing_median_age: Median age of a house within a block; a lower number is a newer building
- total_rooms: Total number of rooms within a block
- total_bedrooms: Total number of bedrooms within a block
- population: Total number of people residing within a block
- households: Total number of households, a group of people residing within a home unit, for a block
- median_income: Median income for households within a block of houses (measured in tens of thousands of US Dollars)
- ocean_proximity: Location of the house w.r.t ocean/sea
- median_house_value: Median house value for households within a block (measured in US Dollars)

Data Assumptions:

To make sure about the understanding of the machine learning task, I need to ask the work team who work on the next machine learning component to understand their needs. As they need the actual values of the houses prices, I will select the regression task in my work.

Limitations:

The dataset is about houses data in California in 1990. I am fully aware that housing prices have increased dramatically since 1990, when the data was collected. This model should not be used to predict today's housing prices in California. This is purely an academic endeavor to explore statistical prediction.

The Process:

- Get Data
- Discover and visualize the data to gain insights
    - Create Test Set
    - Sampling [Random – Stratified]
    - Discover & Visualize
    - Scatter Plot

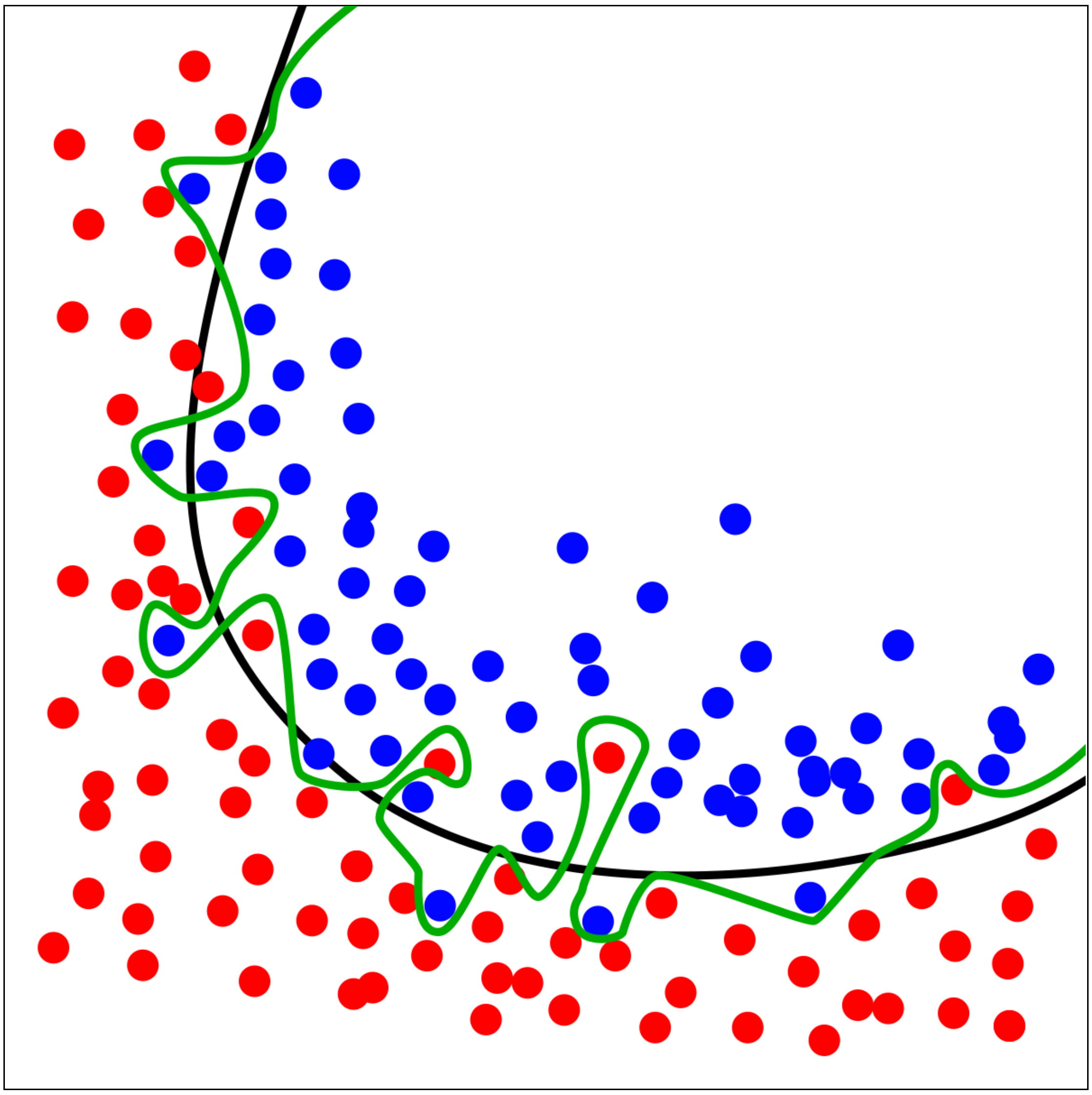
    - Correlation
    - Attribute Combination
- Prepare the data
    - Data Cleaning
        - total_bedroomsattribute has some missing values
    - Handling Text and Categorical Attributes
        - ocean_proximity is categorical field
- Select and train model
    - Linear Regression Model
    - Decision Tree Regression
    - Random Forest Regression
    - Support Vector Machine (SVM)
- Fine tune the model

Creating Test Set:

Isolating Test Set from early will help to know some patterns in the Test Set that may help us to choose the model.

The main objective of separating test set and training set is to avoid the generalization error or overfitting.

So, we want to reduce the error percentage as much as possible for both training set and test set. That depends on the model which we will use.



We have training set in the graph. We can see that the model (green curve) performs very well and there is no error in distinguishing data. However, if we test this model and apply it on the test set, the generalization error will be high. So, we need to reduce the degree of the model to get the least error percentage in training set and test set. Therefore, we use regularization to make the model like the black curve which achieve the least error percentage in both training set and test set.

So, in order to separate the test set and the training set. I will shuffle the whole data set before separating them because I want to choose the test set randomly. However, shuffling the data set will cause a little problem because it will shuffle the data set every time I run the notebook. So, the training set and test set will change every time. As a result, the ML algorithm will train on the data in the test set as well which is not supposed to see it because I need to train the algorithm on the training set only and then assess it on the test set.

To solve this problem, I need to keep the test set away from the ML algorithm in the training step by steadying the random data from the first permutation at the first running the notebook by using np.random.seed(x) method. This way will solve the problem as long as our data set is batch data set not online data set; or, we can use the sklearn library to do that implicitly by using the function train_test_split()

Taking the test set randomly is okay if the data set rows are large and the attributes are big. Otherwise, we may get sampling bias. Therefore, the test set should represent the entries properly. In this project, I will try to avoid the sampling bias that may happen. So, I will use the stratified sampling when specifying the test set.

Let me assume that the company have told me that the median_income attribute has a big impact on the target field (median_housing_price). So, I will take the test set based on median_income field as the test set must represent different categories of median_income.

Since the median_income is continuous numerical attribute, I need to convert it into categorical attribute by adding a new field (income_category).

How can we convert the continuous numerical field to categorical field?

As we can see from the median_income histogram, most of the houses for the people who have income between (2 and 5) thousands. We can put the values of median_income into categories (strata). The strata contains a set of categories called stratums.

- Stratums should not be wide range of values. So, we can reduce the categories by dividing the median_houses by e.g. 1.5
- The stratums must be discrete
- Each stratum must have an enough number of entries.

Therefore,

- In order to reduce the stratums (categories), we divide the median_houses values by e.g. 1.5, if we divide it by 2, the categories will be very little whereas if we divide it by 1, we will get the same number of categories. That's why we chose 1.5
- In order to discrete the categories, we round the decimal values of categories to the nearest integer number.

- In order to make each stratum has enough entries; we will merge all values more than 5 into the category of 5 because those values have low entries.

Prepare the Data for ML Algorithms

Data Cleaning

Most Machine Learning algorithms cannot work with missing features, so let's take care of them.

- You noticed earlier that the total_bedroomsattribute has some missing values, so let's fix this.

We have three options:

- Get rid of the corresponding districts.
- Get rid of the whole attribute.
- Set the values to some value (zero, the mean, the median, etc.).

We can accomplish these easily using DataFrame'sdropna(), drop(), and fillna()methods

Scikit-Learnprovides a handy class to take care of missing values: Imputer

Handling Text and Categorical Attributes

- Most Machine Learning algorithms prefer to work with numbers anyway, so let’s convert these text labels to numbers.
- Scikit-Learn provides a transformerfor this task called LabelEncoder
- One issue with this representation is that ML algorithms will assume that two nearby values are more similar than two distant values
- To fix this issue, a common solution is to create one binary attribute per category: one attribute equal to 1 (and 0 otherwise)
- This is called *one-hot encoding*
- Scikit-Learnprovides a OneHotEncoderencoder to convert integer categorical values into one-hot vectors
- We can apply both transformations (from text categories to integer categories, then from integer categories to one-hot vectors) in one shot using the LabelBinarizerclass

Feature Scaling

- One of the most important transformations we need to apply to your data is *feature scaling*.
- With few exceptions, Machine Learning algorithms don’t perform well when the input numerical attributes have very different scales
- There are two common ways to get all attributes to have the same scale: *min-max scaling*and *standardization*
- Min-max scaling (many people call this *normalization*) is quite simple: values are shifted

and rescaled so that they end up ranging from 0 to 1

- Scikit-Learnprovides a transformer called MinMaxScalerfor this. It has a feature_rangehyperparameter that lets change the range if you don’t want 0–1 for some reason
- Standardizationis quite different: first it subtracts the mean value (so standardized values always have a zero mean), and then it divides by the standard deviationso that the resulting distribution has unit variance.
- Unlike min-max scaling, standardization does not bound values to a specific range, which may be a problem for some algorithms (NN often expect an input ranging from 0 to 1)
- However standardization is much less affected by outliers
- Scikit-Learnprovides a transformer called StandardScalerfor standardization

<u>Select the model:</u>

I have test Linear Regressor model and Decission Tree model and Random Forest Tree model and SVM. I found the Random Forest Model performs the best with error percentage 18,603

<u>Fine Tune the model:</u>

After we selected the best medel, I did fine tune to get the best combination of parameters and its values by using Grid Search I found the best parameters combinations happen when the max_features=8 and the n_estimators=30,, we get 49682.27