SYSTEM DIAGNOSTICS

HOUSE PRICE PREDICTION

Ilya Slyusarchuk -> s191894

28 January 2022

MAIN PROBLEM

We are trying to design a consistent model that can understand certain features in order to output some specific result. In our case we are trying to output the price of a house based on some data provided by the California government. This dataset is composed by 20640 records of houses, each of them with 8 features (we're not taking into account the target price). By default this dataset provides us the following 8 features

- MedInc median income in block group
- HouseAge median house age in block group
- AveRooms average number of rooms per household
- AveBedrms average number of bedrooms per household
- Population block group population
- AveOccup average number of household members
- Latitude block group latitude
- Longitude block group longitude

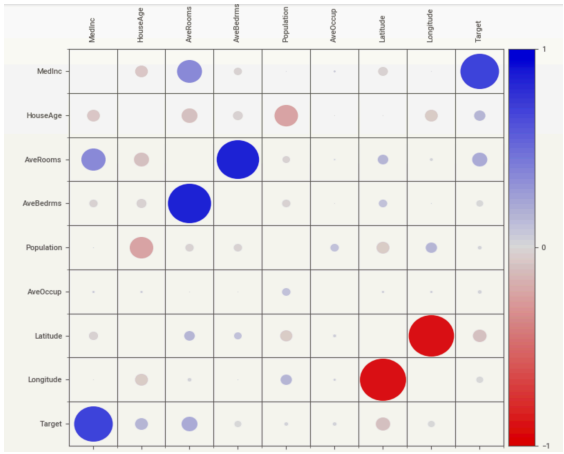
The target variable is the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000).

Our goal is to optimise a model to predict accurately the price of the house based on this features. To do that we are firstly going to execute an exploratory data analysis. This analysis will help us understand our data and will help us to define if a feature is or is not correlated with the target, and also how much correlation there is.

EDA

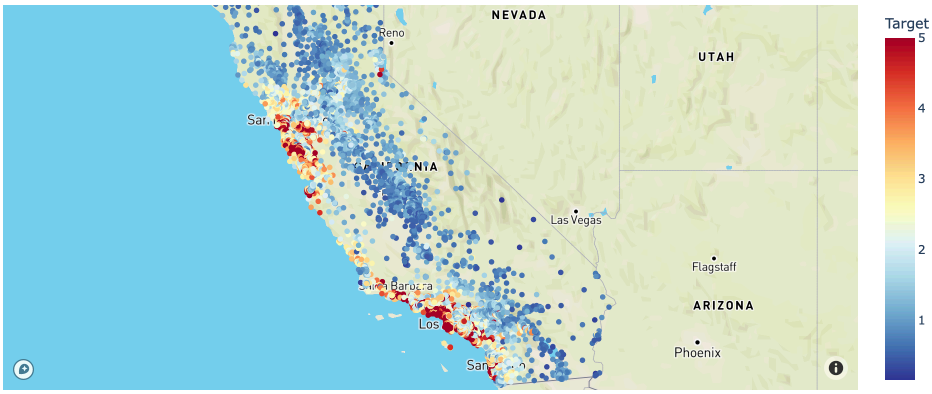
In this case we will be using the Pandas library, alongside with NumPy, Matplotlib, Plotly and SweetViz to store, manage and visualise our data the easiest way possible. First we copy the data to our DataFrame and add the 'Target' column, which contains all the houses' prices. Then we use SweetViz to plot some information about our dataset.

Let's analyse the following table to see the correlations between each variable

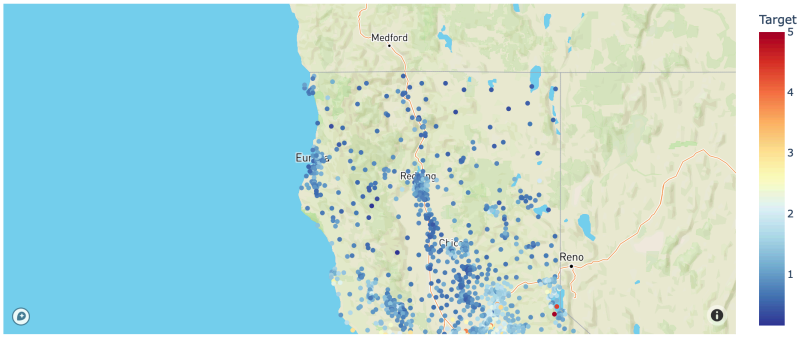


The following image shows us how correlated are the variables between each other. The strongest correlations are clearly the ones between AvgRooms and AvgBedrms, and Target and MedInc. This makes sense since a house with more bedrooms than other is more likely to have a bigger amount of rooms, and also if a family has more yearly income, they tend to buy more expensive houses.

Overall the data seem pretty clear, but we see that the latitude and longitude is not even close to be correlated with the price of the house. This however doesn't make too much sense, since a house in a nice area will normally cost more than the same house in a bad neighbourhood.



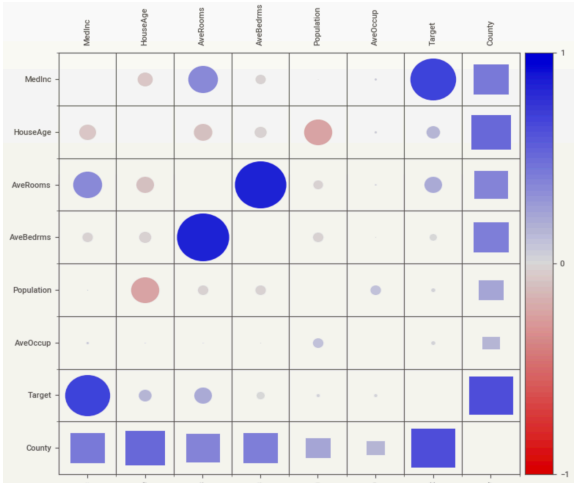
Here is a geoplot, each dot represent a house location, and the colour represents the price of the house. Clearly the location of the house influence the price. South-West and Center-West of California is a more valuable are when talking about houses. San Francisco, San Diego and Los Angeles are the ones with more red dots. So, with this information we can conclude that houses in big cities and near the ocean, are more expensive. We need to specify that not only being near the ocean is a key factor to value a house. Here we have an example of a coaster city that doesn't have expensive houses.



To fix this we are going to scrap our own data, and get some more features that may be helpful at the time of training the model. In our case we will be using a library called GeoPy. This library has a class called Nominatim that has implemented a reverse lookup method that takes as input a latitude and longitude, and spits out data about that coordinate. We will be doing this for each of the coordinates, and will store the county and road (if exists) of that coordinate.

After finishing this process we will explore our data once again.

In this case we have some new correlations. By the way, circles in SweetViz represent symmetrical numerical correlations (Pearson's), and squares represent categorical associations (uncertainty coefficient and correlation ratio). For the sake of understanding the data we are just going to focus on the correlation ratio. Now we can see that the county is strongly correlated with many features such as the income, the target or the average number of rooms and bedrooms. It makes sense, since there are counties that have more expensive houses than others on average, and thus more income.



DEALING WITH MISSING DATA

Now we have to deal with the records that don't have a county/road. Since our geo locator is just an API, it may happen that it doesn't find a specific street or county. To solve this, we have two possible approaches, one is drop the records that have a null value on county or road, and the other one is fulfil that data with a classifier. In this case we used the approach of using a SGD classifier on all the records that are correct, and apply that classifier to our missing data records. By doing this we are creating a dependency between the most correlated features and the county/road. Now the data is pretty much correct and we don't have to drop any row of our dataset.

LAST PRE-PROCESSING STEPS

Last but not least, we encode our county and road column. We do this to feed our model with numbers and not raw strings. In this case we used a LabelEncoder class from sklearn.preprocessing, but a hot-encode approach would have also been a possibility to avoid implicit relations between different counties/roads that are close to each other in terms of encoding.

TRAINING THE MODEL

For training the model, a RandomForestRegressor was used. This provides us a model that have a low bias and a low variance, due to the architecture of this type of model. It's composed of many decision trees, where each of those decision trees extract some rows and features from the dataset, creating his own dataset. This process is called row sampling and feature selection. By doing this we make a decision tree be very good on predicting in that specific training set. This is a problem because we will get a small error on the training dataset, but a big one on data that it hasn't seen. This is where the magic of RandomForest comes into play. When combining different decision trees into one model, this high variance disappears, because now we are getting different results, on different data, and with different features for each tree, and the output is merely the average of all the decision trees. This gives us the possibility to investigate about which feature was more influencing to the target and which decision trees were the ones that activated more in a specific timestamp.

Here are the results on training data that was never seen by the model before.

It's too perfect to be good, but I could not find the error. Here are some performance metrics that gives us a little bit more of information. As we can see, it actually predicts almost identical values as the targets array. They have a correlation of 0.999 and an accuracy of 99.98% which is suspiciously good. If you got an idea of what it may be, communicate with me and I'll try to change it. For now I conclude that it somehow overfits on the training dataset (that was never seen… and doesn't make sense to me).

```
[54] from sklearn.metrics import r2_score

     print('The R2 score between y_test and y_pred is {0}'.format(r2_score(y_test, y_pred)))

     print('The first 5 values inside the test targets: ', end = '')
     print(*y_test[:5], sep = ', ')
     print('The first 5 values inside the prediction targets: ', end = '')
     print(*y_pred[:5], sep = ', ')

     The R2 score between y_test and y_pred is 0.9999995990743472
     The first 5 values inside the test targets: 0.939, 2.098, 4.837, 1.347, 1.222
     The first 5 values inside the prediction targets: 0.9390899999999982, 2.0980100000000035, 4.836590000000005, 1.34701999999

[57] errors = abs(y_pred - y_test)
     mape = 100 * (errors / y_test)

     accuracy = 100 - np.mean(mape)
     print('Accuracy: {0}%'.format(round(accuracy, 2)))

     Accuracy: 99.98%
```