Report Homework 3

(Anjul Kumar Tyagi – 111482304 , Sanaz Sheikhi - 111733554)¶

➢ Task 1

To choose 10 most desirable houses we developed a scoring function which calculates a score for each of the houses Then we rank the houses based on their scores and choose the top 10 houses with highest scores. In the same way we choose 10 least desirable houses which are the ones with lowest scores.

Scoring function:

To build the scoring function we needed metrics for measuring the degree to which each house is desirable. As in Zillow data set we don't have prices to deal with them as "Standard Goals", we thought about some sort "Proxies". Since the question relates desirability to cost or value notions, we tried to choose among those features (variables) that have cost / value notion in essence. So we mainly, focused on the following variables to use them as proxies.

- **Land Tax Value**: Land Tax Value justifies how good/bad the area where the house is located is. The **more** the land tax, **more** valuable the land is, and hence, **more desirable** house. Because in future probably, the value of the land would increase with a much higher rate than rate of tax.
- **Structure Tax Value**: The tax on the house shows how valuable that house is. So, in our desirability matrix, **more** the structure tax, **more desirable** the house.
- **Tax Amount**: **Higher** the tax, lower is it's desirability.
- **Year built**: Generally, **Newer** houses are more desirable. Although, for luxurious or historical houses which are so vast like a castle oldness is kind of desirability but for usual houses newer houses are more desirable.

In the next step we prioritized our proxies (variables) to assign a weight to each of them. Based on the above assumption and more important based on the feature selection we had done to measure the correlation of features with logerror. The weights are as follows :

Feature	weight
landtaxvaluedollarcnt	0.5
structuretaxvaluedollarcnt	0.3
taxamount	-0.2
yearbuilt	0.1

Table 1: Desiarability Proxies and their weights

In the next step we normalize the value of variables to lay in [0,1] range to avoid some kind of false positive condition in which less importannt variables with bigger values but smaller weights would affect the total score more. Finally we used the following formulla to calculate score of each house :

Score = SUM(Weight_feature*Feature_value)

Analysis of Top 10 Most desirable houses

As it's clear from the top 10 most desirable houses list, the most desirable house has the **Highest Land tax value** among all other houses, this means that this house is located at a very good location compared to other houses. Also, some of the houses are very recently built which can be seen by the parameter **year built.**

However, there are some old houses such as the ones built in **1937 and 1927** but because of their location in a posh area, they are more desirable than other houses.

Tax Amount also plays a role in deciding the top 10 houses as it can be seen that all of these houses have a decent tax amount compared to their location.

➢ Task 2

To define a house “pairwise distance function”, which measures the similarity of two properties we've decided to calculate the distance between two properties based on **important features which affect the logerror.** So, suppose we've top **k** features which affect the prediction of logerror the most, we use these features to cluster the houses based on **how well they can predict the logerror.**
Assuming we've the weights of these **k** features, we can use these weights in our distance function. So, **nearer the houses, closer they're in predicting the logerror.**

We have done a sophisticated feature selection experiment including to predict the top 10 features which can effectively predict the logerror . The following table indicates the result of these experiment:

** Feature selection experiments would be explained inTask 5 **

Feature	**Mean Ranking**	
1	taxvaluedollarcnt	0.47
2	landtaxvaluedollarcnt	0.42
3	structuretaxvaluedollarcnt	0.34
4	finishedsquarefeet12	0.31
5	taxamount	0.29
6	regionidcounty	0.25
7	calculatedfinishedsquarefeet	0.21
8	lotsizesquarefeet	0.19
9	yearbuilt	0.17
10	regionidzip	0.13

Table 2 Feature selection experiment result

We chose thses top 10 features which help the most in predicting the logerrors in addition to l**atitudes and longitudes** for the further work including distance function and advanced prediction model.

We will also store the **weights** for these features to calculate the **Distance Vector Matrix.** For simplicity, we assume the weight of each factor to be the **average ranking** for that vector generated by feature prediction models.

Before calculating the distance function we processed our data by the following operations:

- Filling the Nan's with the means.
- Normalizing the data
- Putting aside 25% of the data (properties, train) for test.

To calculate pairwise distance we used "***Weighted Eucledian Distance***" method. In mathematics, the **Euclidean distance** or **Euclidean metric** is the ordinary straight-line distance between two points in Euclidean space .Assuming we've two features **A and B** with weights and we're calculating pairwise distance between points **1 and 2.**

$$dist = \sqrt{W_1(1_A - 2_A)^2 + W_2(1_B - 2_B)^2}$$

Evaluation:

For Evaluation, we found the pairs of houses with minimum distance between them and analyze their properties. For example after obtaining the indices of the highest and lowest values in the Distance Matrix, we found the indices of the lowest and highest occurring values as:
[xLow, yLow] = [**(0,1)**,(0,2),(499,497)]
[xHigh, yHigh] = [**(112,136)**,(136,253),(136,478)]

As seen from the above values, all the properties such as **taxamount, land tax value** are very close to each other and hence the houses are similar in predicting the logerror values.What is important that these houses **are not** situated close to each other, as it can be seen by their latitudes and longitudes but because the weights for latitudes and longitudes are very small, hence these houses are similar in types. This is a proof that the **Distance function** is working correctly based on the weights.

- Task 3

For this part, we used the same **Distance function** used in task 2 and generate clusters using the **DBSCAN** library in python. DBSCAN also suggests the **best possible number of clusters** from given data.
So, now we learnt that the sufficient number of clusters for our data is **11** clusters. Then we used this information to plot the clusters with lattitudes and longitudes using **K-Means** clustering algorithm.
K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable *K*. The algorithm works iteratively to assign each data point to one of *K* groups based on the features that are provided. Data points are clustered based on feature similarity.
Then we ploted the cluster with **lattitudes and longitudes** on the X and the Y axis to see how the location affect the clustering. However, because the weights of lattitudes and longitudes in our model were not significant, we didn't expect the clusters to be properly seperated based on locations.

Discussion and Analysis of the clustering algorithm

First we decided to run DBSCAN to evaluate what is the best number of clusters for the given dataset. We got the result to be **11 clusters** which can significantly represent the dataset.
Looking at the clusters, we've plotted the clusters with Latitudes and Longitudes so that we can clearly evaluate the how the clusters of houses are distributed based on their location. Note that our prediction function was **predicting the logerror** and clusters are made based on prediction of logerror . As a result it is obvious that location **does** affect the prediction of logerros when taken into count. Houses are clubbed into small clusters of same color show that they have the same or close prediction power for the logerror based on the features that we've got by

Even though latitude and longitude didn't have high weights during feature selection, they do affect the prediction of logerrors, as it is clear from the clustering. However, there are certain discrete areas where the clusters aren't very clear, it shows that latitudes and longitudes aren't good at predicting logerros in those areas.
Also, considering that we plotted 11 clusters, only a few of them are dense and dominant. This is a sign that not all clusters are very much dependent on location of the houses. This point makes sense and is coherent with the feature selection results which showed that latitudes and longitudes don't contribute much towards predicting the logerrors.

➢ Task 4

For the external dataset, we've decided to include the **Crime data in Los Angeles from 2010 to present** provided by Data.GOV, along with the original properties datasset to see if prediction model's performance improves .
Link to the crime data: https://catalog.data.gov/dataset/crime-data-from-2010-to-present

The dataset consists of various fields describing what kind of crime has happened in various locations in Los Angeles. Although it includes interesting fields such as **Latitude** and **Longitude,** it doesn't have data on crime rate in each area or district. As a consequence, we had to deduce and calculate crime rate based on existing location data.
The first solution was to use some sort of API to convert coordinates of (latitude,longitude) to city and then count the number of crimes occurred in each city. To reach this end, we used various APIs such as geocoder, geolocator and geopy but as they produced various results on the same coordinates we guessed they may produce different results on coordinates of properties dataset. So, these APIs seem unreliable.
However, another solution was to create blocks of (latitude,longitude) from crime dataset and consider each block as one area (city). Then check each coordinate of (latitude,longitude) against these block to finally find the block where the coordinate lies and increase the number of crimes occurred in that block by one. Checking all cooerdinates of the government dataset against abstracted cities gives us roughly fair distribution of crimes in various areas of Los Angeles.
On the other hand we have the latitude and longitude data of parcelid in properties dataset. W added a column called ""rate" to properties dataset to keep crime rate of the area where each parceled belongs to. So, we checedk whether each parcelid lies on any of the blocks of (latitude,longitude) to assign the number of crimes occurred at that block to the rate related to the mentioned parcelid. In this way we added crime data to our properties dataset as external data.
Choosing the latter solution we divided the latitudes of crime dataset into 100 group the same as longitudes. As a consequence 10000 blocks of coordinate (latitude,longitude) created.

Analysis
For evaluation, we used linear regression on the new dataset and compare it to the results of the linear regression without including the crime data.

Evaluation Results:
It can be seen that after including the crime data, the r^2 value increased but there was only a slight decrease in the **mean squared error.** The increase in the r^2 value shows that the quality of the model has increased with increase in the prediction power for the logerror.
Hence, the insertion of crime data, even though not that significantly helpful, has somewhat increased the prediction power of the model.

	Mean square error	r^2 score
Prediction model without external data	0.0246944107389	0.00398665852805
Prediction model with external data	0.0245493147902	0.00992894127855

Table 3 Results of prediction with / without external data

➢ Task 5

To build a better prediction model we managed to both improve the dataset and training algorithm. For this purpose, we accomplished the following tasks.

1. Creating several train sets and test sets by combining data of 2016 and 2017.
2. Clearing and normalizing the datasets.
3. Using feature selection algorithms to choose a set of features from dataset
4. Adding the external data (crime rate) to dataset for training the model.
5. Using **Gradient Descent** algorithm rather than simple linear regression.

In the following sections each of the above items would be explained in detail.

○ Creating several train sets and test sets

Because the Kaggle data now includes the properties from **2016** and from **2017,** so we've decided to create four files combining the logerrors and the properties. Although the fields of properties datasets are the same but their values of feartures may have changed. For instance it is possible that records of some houses have changed by because of tax related features or even structural features such as number of rooms have been added for some houses.

On the other hand, upon update of properties dataset it is obvious that train dataset has changed. However, the size of train data set 2017 shows that not all parcelid have updated logerror. All in all we decided to create the following dataframes to train and test our model with all of them:

- Properties of 2016 combined with the logerrors of 2017
- Properties of 2016 combined with the logerrors of 2016
- Properties of 2017 combined with the logerrors of 2017
- Properties of 2017 combined with the logerrors of 2016

25 percent of each data frame has been dedicated to train. It is worth noting that train data selection has been done in such a way that various parcelid are picked up from each data frame.

○ Clearing and normalizing the datasets

- *Replacing the Nan's with Means*

  We preferred replacing the Nan's with the means because while normalizing the data, the values with means become **zero.** Hence, they don't contribute in training and we can train our model only on real data values

- Managing type id field

  As properties dataset includes fields with multiple values (known by type id) we used get_dummies( ) function to create kind of binary string for each parcelid show in that parcelid has which type of a category. But in feature selection part we found out that features with type id have little impact on logerror prediction. So, as we decided not to choose them, we eliminated the binary string creation operation as well as type id fields.

- Normalizing dataset

  We normalized our thew data set using min_max_scalar.fit_transform() function.

○ Feature selection

For finding the features which affect the logerror the most, we ran feature selection methods like **Linear Regression and Ridge Regression.**

- **Linear regression for feature selection.**

  Linear regression recursively goes through all the features one by one and gives weight to that feature. After we've weights for each feature, we got a ranking of the features based on how effective they are in predicting the Logerror.

- **Ridge Regression**

  The ridge regression works for the cases when there is multicollinearity between the features. We obtained ranking of features to predict the top 10 features which can effectively improve prediction of the logerror.

After Computing the ranking for all the features, we create an empty dictionary and use it to store the average ranking for each feature. The following table shows the final ranking of features.

Feature	**Mean Ranking**	
1	taxvaluedollarcnt	0.47
2	landtaxvaluedollarcnt	0.42
3	structuretaxvaluedollarcnt	0.34
4	finishedsquarefeet12	0.31
5	taxamount	0.29
6	regionidcounty	0.25
7	calculatedfinishedsquarefeet	0.21
8	lotsizesquarefeet	0.19
9	yearbuilt	0.17
10	regionidzip	0.13

Table 4 Feature selection experiment results

So here are the top 10 features which help the most in predicting the logerrors. Most features are related to tax values or squarefeet. So, we only worked with these features combined with **latitudes and longitudes** for the further work.

- **Adding the external data**

  Based on task 4 crime data, was considered as external data, strengthened the prediction power of the model. So, we added crime data to set of features to train the model with it.

- **Using** Gradient Descent **algorithm**

  We used **Gradient Descent** algorithm to predict the data, which should perform better than simple linear regression. Because Gradient descent is a particular optimization algorithm to learn the weight coefficients of a linear regression model .It works gradually towards decreasing the **mean absolute error** and goes until the mean absolute error starts increasing. It then reports the final parameters for the **lowest mean squared error**.

Evaluate the model

For evaluation we calculated the mean squared error and r^2 values for our trained model.
The r^2 has greatly increased from the last **linear regression** results, which is a sign that the model has good predicting power and has improved in comparison to HW2 regression model. Below are the **Kaggle submission** results:

- Submission 1 **(Support Vector Regression)** zestimate = 0.1131807
- Submission 2 **(Gradient Descent)** zestimate = 0.0656324

- Task 6

we used python's **permutation_test_score** library included in **sklearn.** Permutation testing is done on the predicted values by permuting the original test values in random order.

The following results obtained from permutation test :

Classification score -1.87045091487 and Pvalue : 0.029702970297
The small P value shows that the prediction model actually works and is not a fluke.

For large number of permutation i.e. 100, the prediction by the model come close to the mean of the random predictions for the small portion of the dataset.

As we increase the size of the dataset, the P value decreases, showing that for a large dataset, P value actually becomes very small.
Permutations > 100 for a **50,000** row dataset produce the model as good as the real dataset with a low P value of around 0.0099.