**Final Report**

**Team Members:**

Cedric Tong

Wilbert Veit

Gonzalo Serrano

Aaron Romero

Avik Gharzarian

**Project Title:**

Sberbank Russian Housing Market

**Project Description:**

Sberbank is Russia’s oldest and largest bank that helps their customers by making predictions about realty prices so renters, developers, and lenders are more confident when they sign a lease or purchase a building. Due to the country’s volatile economy, forecasting prices imposes a unique challenge. Complex interactions between housing features, locations, and pricing makes the predictions more complicated. In addition, an unstable economy means Sberbank and their customers need more than simple regression models to predict prices. The aim of this project is to predict the sale price of each property.

**Data:**

The target variable is called **price\_doc** in the train.csv file.

**train.csv and test.csv**: they hold information regarding transactions. Each Row in these files are indexed by an “id”, which refers to individual transactions. These files also include supplementary details about local area of each person.

**macro.csv:** This file contains details about Russia’s macroeconomy and financial sector; this file can be joined to the test and train csv files on the “timestamp” column.

**BAD\_ADDRESS\_FIX.xlsx**: There was a problem with coordinates for some properties which lead to errors in distance parameters, because they were calculated from the heart of Moscow for such cases, which is wrong. This file fixes the parameters.

**Goals:**

* After looking at our data, one of our goals in this project was to find out a way to shrink the amount of features (our data consisted of 200+ features)
* Also find out a way to exclude the rows and columns that contained NaN values without shrinking our data too much. And trying to get the best accuracies predicting on our target variable.
* Although our data is based around regression models, we wanted to experiment with classifiers as well, and compare accuracies.

**Methods Used:**

Initially looking at our data, we realized that our data contained too many features; our data set contained at least 200+ features and 30k plus rows. Thus, we decided to use the Principal Component Analysis (PCA) algorithm on our data set.

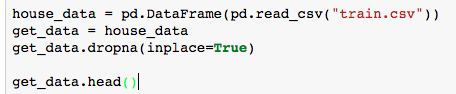
We were looking for ways to use all the data given to us: training, testing, and macro.

**Method 1: Experimenting with the Training dataset:**

For our first method, we wanted to experiment with the training set only. Even though kaggle provided a testing dataset, we wanted to see what kind of accuracies would result from using algorithms on just the training set.

* **Ignoring Rows with NaNs in our Training Set:**

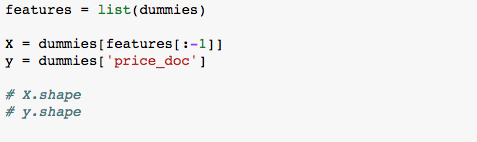
Before we dealt with PCA, we noticed that our data had too many columns and rows that contained NaNs and as we expected, we were getting problems when trying to run One Hot Encoding in our training and testing datasets. So we first experimented with ignoring rows with NaN values by using this:



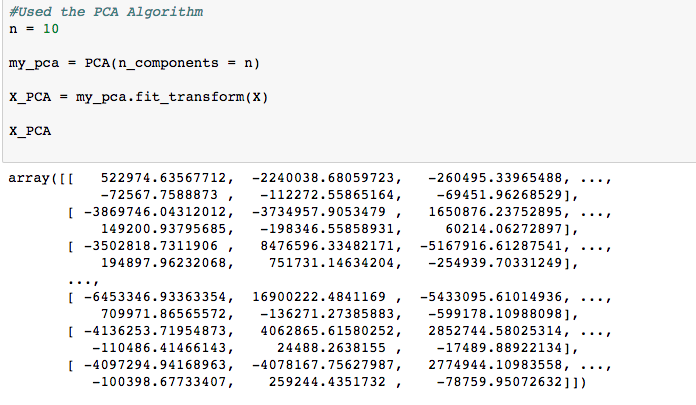
As you can see, the line of code that says: .dropna(inplace=True) would ignore the rows that had any NaN values in them, thus reducing our rows from 30k plus to 6k rows. But, we were still too skeptical of only using 6k rows of data and 200+ features with our algorithms. Therefore, our professor suggested to physically drop 6 columns from our training dataset that contained the most NaN values which were: max\_floor, material, build\_year, num\_room, kitch\_sq and state. Thus, after running the same code above, our dataset resulted to 9k plus rows and 200k plus rows, which was a bit better but not amazing.

**1. Applying PCA to our Training dataset:**

After setting up our dataset correctly, we were able to perform One Hot Encoding to our training dataset, leaving us with 9k rows and 1500 plus columns. Next we set up our Vector X and target y:



To make our lives easier, we set up our feature X by excluding the last column (target column) in the data frame that was generated in the One Hot Encoding process and the our y was created by grabbing the last column called ‘price\_doc’. Therefore, we were ready to use PCA on our features:



As you can see above, we gave the algorithm a value of 10, to reduce our columns from 200+ columns to 10 columns, leaving us with 9k+ rows and 10 columns. Lastly, we get our result from above and we store it in a new dataframe in order to use it for when we split our data.

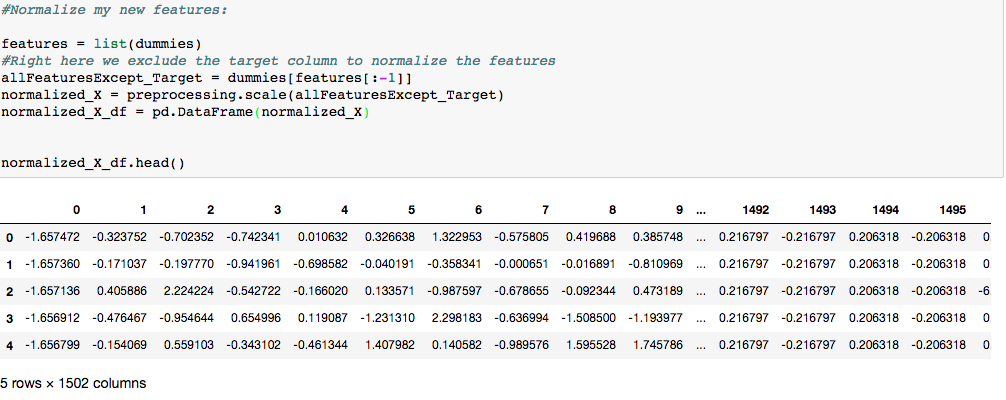
Next, we split our training set into testing and training (Testing = 30% and Training = 70%). The fun now begins because we can start testing algorithms on our dataset to see the accuracies of various algorithms.

* However, after trying regression and classifier algorithms on our data, we were disappointed to get very bad results (Numbers/Percentages are displayed in the “Result” Section). We were left in confusion and annoyance because we were sure we used every algorithm correctly; we even went back to check if set up our X and y correctly and check if PCA was coded correctly as well. So after double checking everything, we thought, maybe PCA is messing with up our accuracies.

**2. Not using PCA in our Training Dataset:**

After getting horrible accuracies for our algorithms, we were motivated to try some other approaches, so we ran our algorithms without using PCA this time. Since our whole project revolved around regression models, we were happy that are linear regression rmse value was a really low error value(our mean value for using cross validation on Linear Regression was really low as well); Logistic Regression would take too long to compile, so we were not able to efficiently test that algorithm unfortunately. As far as classifiers, we got better results from Decision Tree, but we were more puzzled when Decision Tree did much better at predicting the sale price of a house than Random Forest. Thus, we were still not satisfied, so now we tried our algorithms, but this time we normalize our features.

**3. Using PCA with our Normalized Features:**

Since we had a brief introduction of using PCA this semester, we had to do some more research on this algorithm to see if it had any correlation with normalizing data. Thus after doing some research, we learned that PCA creates a new projection of our dataset. In other words, the new axis that is build is based on the standard deviation of our variables; the variables with the highest standard deviation will have a higher weight for the calculation compared to the variables with a small standard deviation in the calculation. Therefore, normalizing our data would make our variables all have the same standard deviation, thus PCA would calculate a more consistent axis.

Before normalizing our data, we had to make sure that we did not include the target column so we excluded it and we used preprocessing.scale(showed above) to get our features normalized, which are then stored in a new Dataframe. Finally, after setting up our new X and y (target column still the same), we split our data into a new Testing and Training (Testing = 30% and Training = 70%) on our normalized data. Lastly, after running our algorithms, we were again confused of why we were getting terrible accuracies, so we started to theorize normalizing our data was not necessary with PCA. But on the other hand, we were able to run Logistic Regression and perform cross validation (accuracies were still not great), but it was better than nothing. Also running cross validation on Linear Regression did not help our situation either.

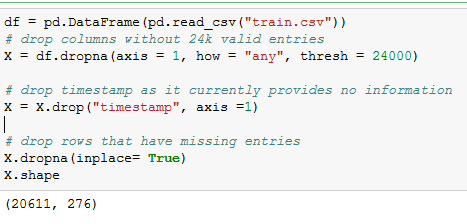
**4. Not Using PCA with our Normalized Features:**

Keep in mind, all these approaches are to test and experiment with just ignoring NaNs in our dataset in every row, later on we run the same approaches but instead we ignore the NaNs in every column. Now, at this point, we’re losing hope so we try one more approach which includes using our normalizing data without using PCA. Same as approach number 3, but unfortunately, logistic regression was still not playing nice this time around. However, the algorithms that have been running quickly, their accuracies still seemed atrocious, except for Decision Tree; it got the same accuracy as it did in approach number 2. So for now, we concluded that are “best results” were from approach number 2 and/or not using PCA in our dataset.

* **Ignoring Columns with NaNs in our Training Set:**

A different approach was to remove the columns that had any missing values to reduce the number of features. We dropped all columns that didn’t have at least 24k valid data entries and then reduce the rows by removing the ones that have missing data. We also dropped the column labeled timestamp because we were not combining the file with the macro.csv so the timestamps didn’t currently provide any information. This resulted with a data set that contained 20k rows and 276 columns.

We initially dropped all columns that had missing entries but this was an issue because we did not know how important the dropped features were. A half way solution to the issue was to come up with a threshold so that not all columns were dropped but we still removed most of the missing entries. 24k was selected because in the previous method we only had 6k out of 30k rows which meant that only 6k rows had no missing data, therefore there was 24k rows that had mixed data.



After this we applied the same one hot encoding method to replace categorical data in our data set. Then we separated the target from the data; that is we removed price\_doc from the features and moved it to df\_target\_y.



We then ran a version of the code that uses normalization and a version of the code that doesn’t normalize the data. The final shape of the data frame was 20611 rows and 417 columns after normalization and one hot encoding. For the versions with PCA I only chose to run with the features condensed to 10.

We then split the the data with 70% being training and the other being 30% for testing. I ran the algorithms we learned on the training and testing dataset and recorded the results in the Results sections.

Some final thoughts are that it seems strange that every time we ran an algorithm with PCA the accuracy would be lower than without PCA since PCA is reducing the features into 10 that contain the most important informations. We initially thought that the issue was just because we weren’t normalizing our data but after we started normalizing our data we found the same pattern.

**Method 2: Experimenting with Training and Testing dataset**

Based on the experiments done on the training dataset, we stepped into using both of the files: Train.csv and Test.csv. Furthermore we figured out the best possible way to approach the problem was to drop the mentioned 6 columns from the dataset, which will help us to have more rows (9601) when removing the rows with NaN values, to compare the removing of rows with NaN values with all rows included (6000). (Figure 2.1)



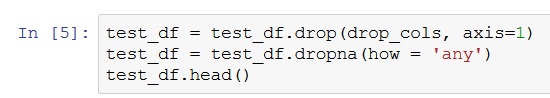


Figure 2.1

In the second step we acknowledged the non-numerical columns by investigating the data set; and applied one-hot-encoding specifically to those columns. And let the last column of the training dataset be the label vector, ”price\_doc” (Figure 2.2)

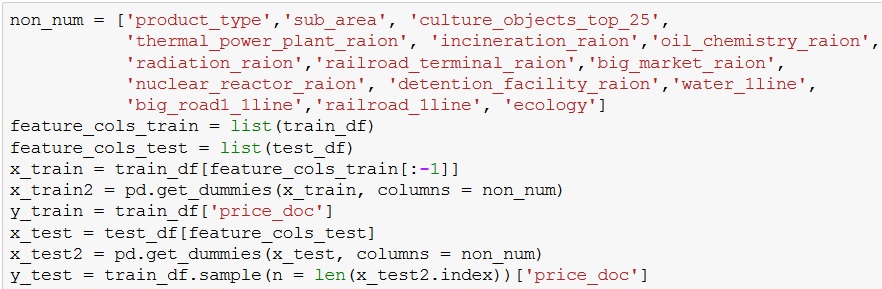


Figure 2.2

Since we already knew processing all the 284 columns which became 379 columns after one hot encoding might take hours or even days. We applied the PCA to reduce the feature to n = 10 or 20. But then we decided to use maximum of 10 features since our computers would take hours to compile 20 features. The first try to compile 20 features on PCA took 1:45 hours.(Figure 2.3)

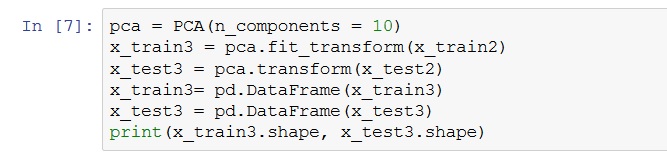


Figure 2.3

Furthermore we normalized the data to test and compare the result of the accuracy with normalized and non-normalized data.(Figure 2.4)

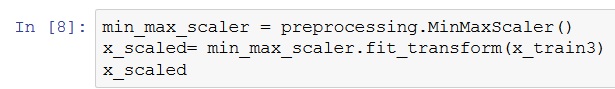


Figure 2.4

Once our dataframe, training and testing sets are set up; We used the methods below:

1. Logistic Regression: This method take the longest time of all. We tried the Logistic Regression with normalized and non-normalized data, and with or without 10 fold cross validation, or with both cross validation and normalized data to see which one is giving better accuracy.
2. Linear Regression:with normalized and non-normalized data, and with or without 10 fold cross validation,or with both cross validation and normalized data.
3. Decision Tree: :with normalized and non-normalized data, and with or without 10 fold cross validation,or with both cross validation and normalized data.
4. KNN: for KNN classifier we used a for loop to see the result of the accuracy in the range of K from 1 to 10 so we can pick the best accuracy possible in that range. KNN classifier is tested with normalized and non-normalized data, and with or without 10 fold cross validation,or with both cross validation and normalized data.

**Method 3: Combining the Testing and Training dataset with Macro dataset**

For this method, before we decided to combine all the datasets, we updated all the data sets with the dataset fix. Afterwards, we combined them with macro and researching the top features of the macro data. To do this, we took the results from one of the discussions on Kaggle about our project at: <https://www.kaggle.com/robertoruiz/dealing-with-multicollinearity>. Afterwards, after combining the macro data with only the features listed from the discussions, we were running into trouble still. After looking into the data, we decided to drop three of the features that were not being filled all the way.

By dropping the features, our dataset was almost complete. We then started running into the NaN issue, so we decided to drop all the rows that contained NaN to simplify the issue. With this in mind, our total rows count fell short to about 9600. We began to test the data by splitting into testing and training again.

Before we ran all the algorithms, we decided to normalize both training and testing features in hopes of achieving higher scores and accuracies.

First, we tested with all 9600 rows and 300 columns against algorithms learned in class. After seeing the results and the runtime of these algorithms, we decided to use Logistic Regression at the end. It was mainly due to the fact it took too long and it felt inefficient to spend all our time on one algorithm when there were many others to use and test on.

Secondly, we reduced the columns using PCA to 10. With that, re-running the algorithms did not give us results that differed too much from what we got with running without reducing columns to a size 10. Again, we did not use Logistic Regression here as well because of how much time it takes.

Overall, the results were extremely similar between 300 and 10 columns. There was no significant difference in results, however in terms of run time, reducing to 10 columns did prove to be faster, as expected.

**Results:**

\*C.V= Cross Validation

\*N.F= Normalize Features

**Method 1:**

* Ignoring Rows with NaNs in our Training Set:

-Again, since our project was based on Regression models, we only did cross validation on Linear and Logistic Regression to analyze their accuracies.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | With PCA | Without PCA | C.V with PCA | C.V w/o PCA | N.F with PCA | N.F w/o PCA |
| Linear Regression | 65.26529 | 6.86285533807e-09 | 65.6953 | 5.85117398415e-08 | 6069998.47582 | 63564.17731 |
| Logistic Regression | N/A | N/A | N/A | N/A | 0.03679 | N/A |
| Decision Tree | 0.23013 | 0.79069 | N/A | N/A | 0.01666 | 0.79035 |
| Random Forest | 0.19299 | 0.08018 | N/A | N/A | 0.02395 | 0.08018 |
| KNN | 0.14301 | 0.14405 | N/A | N/A | 0.027768 | 0.02673 |

|  |  |  |
| --- | --- | --- |
|  | C.V/N.F with PCA | C.V/N.F w/o PCA |
| Linear Regression | 5659773.23923 | 357282.29312 |
| Logistic Regression | 7769243.23318 | N/A |

**Ignoring Columns with NaNs in our Training Set:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | With PCA | Without PCA | C.V with PCA | C.V w/o PCA | N.F with PCA | N.F w/o PCA |
| Linear Regression | 0.028495 | 0.02952 | N/A | N/A | 5969998.47582 | 3942748.65985 |
| Logistic Regression | 0.02965 | N/A | N/A | N/A | 0.03679 | N/A |
| Decision Tree | 0.02956 | 0.06595 | N/A | N/A | 0.026548 | 0.02701 |
| Random Forest | N/A | N/A | N/A | N/A | 0.027167 | 0.03784 |
| KNN | 0.02862 | 0.02955 | N/A | N/A | 0.027768 | 0.03072 |

**Method 2:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PCA | PCA+CV | PCA+NF | PCA+NF+CV |
| Linear Regression | 6589153.84068 | 5846308.66557 | 2.53461e+13 | 5846308.66557 |
| Logistic Regression | 0.04478 | 0.04744 | 0.00393 | 0.04441 |
| Decision Tree | 0.00746 | **N/A** | 0.01767 | **N/A** |
| KNN | k=7 (best result)  0.01924587588 | k=10(best result)  0.031277878909 | k=10 (best)  0.02042419481 | k=10 (best)  0.02988089687 |

**Method 3:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Without PCA | With PCA | CV w/o PCA | CV with PCA |
| Linear Regression | 7821783.75810 | 6623476.66674 | 18720741.03859 | 6018836.02765 |
| Logistic Regression | 0.04257 | 0.04011 | 0.04878 | 0.04916 |
| Decision Tree | 0.00941 | 0.01310 | 0.02240 | 0.01632 |
| Random Forest | 0.01228 | 0.01924 | 0.02433 | 0.01847 |
| KNN | 0.01883 | 0.02251 | 0.01901 | 0.01927 |