Introduction to Data Analysis **Tomas Ye**
Home project no. 1

Problem Statement

Our goal is to develop a linear regression model for describing and predicting residential house prices in Ames, Iowa. At our disposal, we have a dataset consisting of 2919 observations of 18 different variables. Each variable is some kind of a feature attribute of a residence (for example Lot Area). Around one half of the observations is labelled with a corresponding price. Our task is to take the labelled results and create a reasonable regression model and then use this model to make predictions of housing prices for the unlabelled observations. The coded solution to the problem is included. Short description of the code can be found in the *README.md* file.

Solution

To solve our task, we have chosen this methodology of work. First we will take a quick glance at the data and develop the first model that comes to mind. Then we shall check the describing characteristics of this naive model and see where it can be improved. We will then take a more detailed look into the structure of the data and apply necessary transformations to create a better model. Finally, we will check, whether the new model is really an improvement of the old one. For evaluating the relevance of the models, we choose the following metrics: Root mean squared error (**RMSE**), Mean absolute error (**MAE**) and condition of the model matrix.

Rough data exploration

After loading the data, we split the observation into two parts: labelled and unlabelled (in code *development_data* and *evaluation_data*). For now, we will only use the development set. We observe that 9 out of the 18 variables are categorical with the number of categories ranging between 2 and 5. We also observe that some rows are missing some values.

Naive model

To get a valid feature matrix, we drop the NaN rows, include a dummy column for each category and include a column of ones to represent the constant coefficient. This is a must-do for any model. Later, we will add more transformations but for now, let us stop here. We split the development data randomly into training and testing sets by the ratio $4:1$. We use the training set to train the model and the calculate the **RMSE** and **MSE** on both the training set and the testing set. We observe that the testing errors are always a little less then the testing error but not by that much (around 10What is not so kosher is the smallest eigenvalue of the model matrix. It has magnitude of 10^{-20}. In fact, the model matrix is precisely singular, since adding all dummy columns corresponding to one category gives the column of constants). Therefore, the goal for our new model is to have a more regular model matrix and similar error.

Detailed data exploration

After looking at the data for some time, we notice some patterns. For instance, 5 out of the 9 categorical variables are super one sided in our data, therefore not providing a very general information. Let us drop these categories and merge small categories of the rest variables together to get a binary dependency.

Final model

After running *sm.OLS(...).fit()* on the remaining data, in the summary we see that some regression coefficients have a very high *p-value*. Let us drop these seemingly unimportant columns from our matrix. We repeat the procedure of fitting the model and dropping columns with high *p-value* (backward-stepwise algorithm) until we get to a point where all variables are significant. We are left with 7 variables of which only one is categorical. To solve the reoccurring problem of singularity, we try to transform some columns by certain transformations. After a while of experimenting we notice, that a pretty good result is achieved when we transform 4 out of the 7 variables by $x \mapsto \log_2(x+1)$ and leave the rest unchanged. To add more precision to the model, we add some extra columns by transforming the new 7 variables. After some more time of playing around, we conclude that we get a pretty good model when we include some square roots, some logarithms, most of the quadratics and a lot of the mixed terms (i.e. transformations of type $(x_1, x_2) \mapsto x_1x_2$).

Comparison

By running the scripts *naive_model.py*, *final_model.py* and *comparison.py* we see that the final model has a much better conditioned model matrix and reasonably smaller RMSE. Also, when we look at the final predictions made by both models (by running *naive_predictions.py* and *final_predictions.py*) we notice that the naive model has a much greater tendency to make nonsense predictions like predicting negative price. However, the MAE is pretty much the same, maybe a little smaller for the naive model.

Conclusion

It seems, that the naive model is not that bad when you want just a rough overview of the given situation. The only thing we need to take care of is that the model matrix is prone to be badly conditioned. On the other hand, often times we need more precision than the naive model can give us. For me, finding a model that is at least slightly better than the naive model in all of the important criteria was pretty hard and it took me a lot of time. I hope more advanced machine learning techniques will achieve this with less effort.