Machine Learning Engineer Nanodegree

Capstone Project for AMES House pricing Kaggle Competition

Jean-Christophe PINCE
January 23rd, 2017

I. Definition

Project Overview

Transactions volume on real estate market in the US only is above 350 billion dollars every year (see http://www.colliers.com/-/media/files/marketresearch/unitedstates/2016-capital-flows-reports/2016-1h-cap-flows-market-report.pdf). This is a huge market with many competitors addressing mostly anyone in the world.
Buying a house is a major step in the life of anyone and requires the buyer to think thoroughly the pros and cons of the house he or she plans to buy whatever what his or her budget is. There are a lot of professionals in this market allowing the buyers to consider as many houses as possible given their criteria such as the house location, its size, the number of bedrooms but also the look and feel and many other features.

On the other hand, Kaggle is a web platform for data science sharing and competitions. It offers datasets, forums, evaluation, ranking and tutorials to professional data scientists as well as hobbyists to apply this science they love so much! The problems can be addressed by people individually or in groups and some competitions have a reward associated which can be quite substantial.

I have chosen the competition named ‘House Prices: Advanced Regression Techniques’ for this project. This is what Kaggle calls a playground competition, it is a competition with no financial reward but one offering a challenging problem with a realistic dataset in a competitive environment for fun or learning (which is exactly my case). The dataset used here has been provided by Dean De Cock: ‘The Ames Housing dataset’ (http://www.amstat.org/publications

/jse/v19n3/decock.pdf). This dataset was compiled by Dean for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often cited Boston Housing dataset that we used in the nanodegree exercises as well as in many courses on the web.

My goal is to play with a realistic dataset and provide a solution approaching what the bests can do. I have tried to apply most of the techniques learned throughout this nanodegree all along the project but also go beyond this by using a stacking machine learning methodology well used by the Kaggle community. Here is the link to the competition: https://www.kaggle.com/c/house-prices-advanced-regression-techniques

Problem Statement

Given the size of the market and the extremely careful approach of the buyers, estimating properly the price of a house can make a huge difference for any professional of this market. Being able to estimate the right price is important since if a house is overestimated, it will never get sold and if it is underestimated, the seller and selling agency will loose a lot of money.
Predicting the price of a house is a complex task requiring a lot of experience to estimate the different features such as the house location, the number of rooms, size of the house and many other features which have a less evident impact on the price such as the roof style for example and it is addressable with machine learning.
Zillow and many others already understood that and are using machine learning to estimate the price of the houses on their site; their estimation is an important factor for the house buyer since it gives him or her some confidence that the price is right.

As mentioned earlier, stacking methodologies are particularly suited to this regression problem since it implies a particularly complex problem with a lot of features from which extracting the essence of the information is quite hard. The dataset is quite small and prone to overfit given the number of features and the fact that a lot of them are uncorrelated with each others (as we will see later in the data exploration section). This makes a single algorithm susceptible to use a secondary variable as a major indication of the output.
Using stacking as explained in here (http://mlwave.com/kaggle-ensembling-guide/) with fundamentally different algorithms and different versions of the dataset will help refine our predictions and finally get predictions accurate enough for an industrial solution.

Metrics

The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample and population values) predicted by a model or an estimator and the values actually observed. The RMSD represents the sample standard deviation of the differences between predicted values and observed values. These individual differences are called residuals when the calculations are performed over the data sample that was used for estimation, and are called prediction errors when computed out-of-sample. The RMSD serves to aggregate the magnitudes of the errors in predictions for various times into a

single measure of predictive power. RMSD is a good measure of accuracy, but only to compare forecasting errors of different models for a particular variable and not between variables, as it is scale-dependent (Source Wikipedia).

The RMSD of predicted values for times t of a regression's dependent variable y is computed for n different predictions as the square root of the mean of the squares of the deviations:

$$RMSD = \sqrt{\frac{\sum(log(pred) - log(y))^2}{n}}$$

II. Analysis

(approx. 2-4 pages)

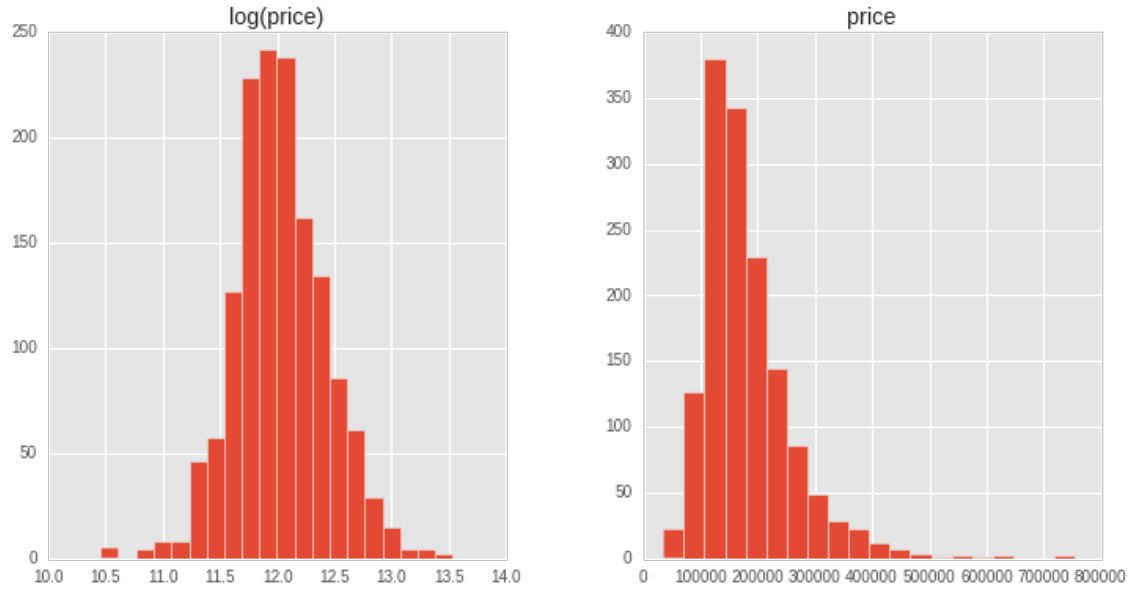
Data Exploration

The data provided to support the competition is in the form of two CSV files; a first file containing 1460 samples with sale prices for training and a second file with 1459 records with no sale price for evaluation and ranking by Kaggle.
The datasets are is describing the sale of individual residential property in Ames, Iowa from 2006 to 2010 and comprises 79 features with invaluable tiny details such as the type of roof or the proximity of the house to arterial street and many others allowing to fine tune the estimation to a great extent. It also comprises a description of the features.
The 79 features are both numerical and categorical and mainly describe with many details:
- The type of property,
- The size of the different parts of the property (house, basement, garage…),
- The number of principal rooms (bedrooms, bathrooms…),
- The important features such a pool, the number of fireplaces…,
- The neighborhood

There are too much features to make an exhaustive list here; please find the full description here (http://www.amstat.org/publications/jse/v19n3/decock/DataDocumentation.txt).

Prediction Output

Before looking into the features which are dense, I will look at the price repartition since some of the algorithms used are not particularly efficient when dealing with skewed data (particularly the non tree based algorithms such as Ridge, Lasso or LinearRegression; the tree based algorithms are more robust to that).



Note that the log of the price is a much better output for the non tree-based algorithms (and is the value used in the error measurement).

Missing values

This dataset has 1460 records in the training set and 1459 records in the submission set with 79 features; on this 79 features, 34 contain 14% of missing values, the rest are full.
This is a major concern because applying mean or median values to those 34 features might introduce a lot of biased data and penalize the predictions.
In order to address this issue, I have written a full per-feature algorithm taking into account other features or the data description. For example, the missing basement descriptions have been reset to default values when there was no basement. Some of the missing features like LotFrontage, MSZoning (representing respectively the Linear feet of street connected to property and Identification of the general zoning classification of the sale) have been set to the median value of the neighborhood.

Data Correlations

Prior to measuring the data correlations, I have encoded the data so I could measure the correlations of the numerical but also the categorical features.
All along this project, I have tested several encodings:
1. One hot
2. enum-like encoder (using LabelEncoder)
3. output-based encoder (based on https://www.kaggle.com/omonnier/allstate-claims-severity/impact-of-categories-encoding-on-the-model-perf)

During the data correlations analysis, I didn't use the OneHot encoding because the feature set is large and using onehot will multiply it making the data visualization just unusable.
So, I used the enum-like encoding using LabelEncoder and and output based encoder using the median sale price of each value of the category as the rank of that value.
Using the correlation matrix as an indicator of the feature importance, I first used the LabelEncoder version of the encoder. It showed a lot of totally uncorrelated features and some of the features known to be impacting like the neighborhood were not correlated because of the hazardous encoding.
Using the output based encoder did show much more correlation between the features and the order of importance got some more sense (see here after).

In this correlation analysis work, I have used two more methods:
- Boruta
- Lasso

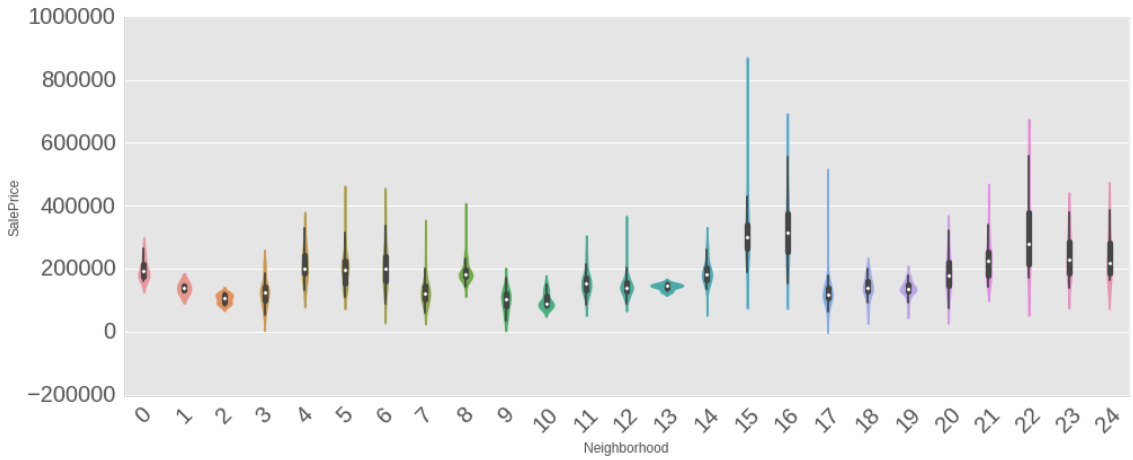
Boruta is an all relevant feature selection wrapper algorithm. It finds relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies. The package BorutaPy allows to apply this algorithm to our dataset and allowed me to extract a total of 15 features out of the 79 original.

Lasso or least absolute shrinkage and selection operator is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces (source Wikipedia). I used it in my data exploration phase to extract a subset of important data the same way as I did with Boruta. Lasso extracted a total of 53 important features out of the 79 originals.

Exploratory Visualization

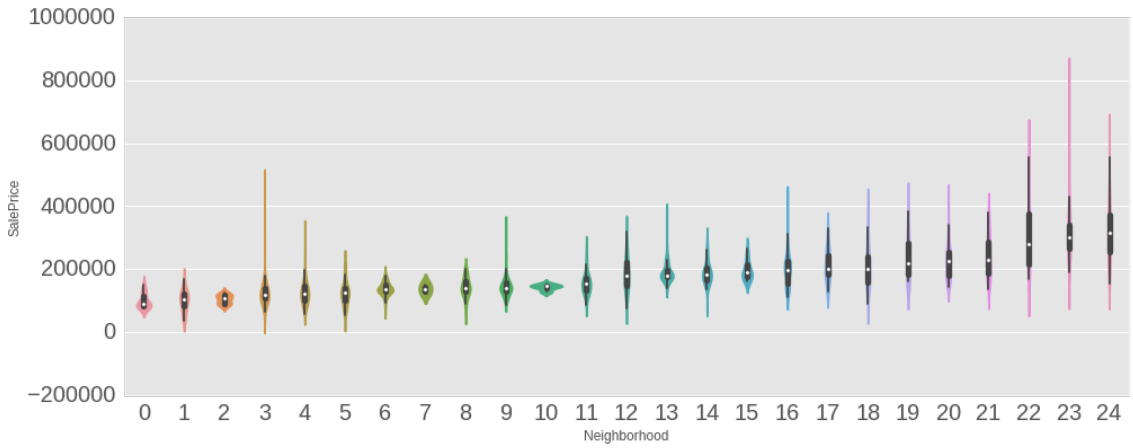
I have visualized a great deal of information (most of it is not even in the notebooks) to get a feel on the information. Given the number of features, a lot of the plots could not be placed in the full context.
I will present here four of the data visualizations:
1. A feature encoded with the label encoder
2. The same feature encoded with the output-based encoder
3. The whole features correlations heatmap
4. A scatter matrix of an extract of the most important features
5. A scatter matrix of a few combination of features

Feature encoded using LabelEncoder (Neighborhood)



The violon plot shown here and in the next paragraph shows the quantile repartition per neighborhood and the white dot is the median value of SalePrice for the given neighborhood.
This plot clearly shows that regression function correlating the sale price to the neighborhood should apply a sinus-like function…
This is a major issue for the non tree-based algorithms that can be fixed only by linearizing this relationship or by using a onehot encoding which will add 24 new features to the already 79 existing…

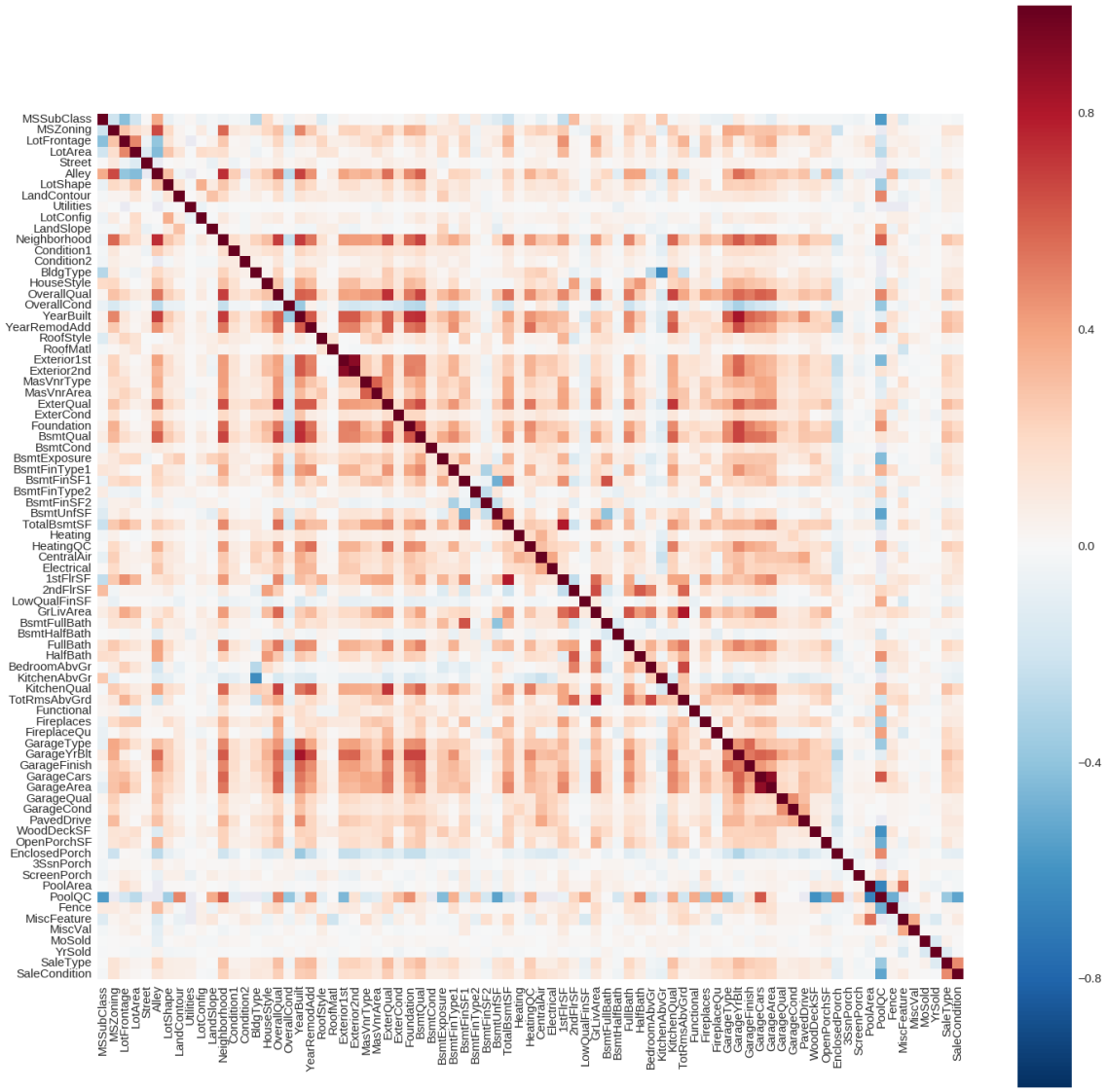
Feature encoded using Output-based Encoder (Neighborhood)



Here, we can clearly observe that the relationship between the neighborhood and the price is tight and much more linear.
One interesting information I see here is that there seem to be 3 plateaus: A first plateau for the neighborhoods 0 to 11, a second for 12-21 and a third for 22-24.
I would assume that this kind of plateau forms might be a good indication that we could merge those values and then help our non tree-based algorithms in extracting the right information. In fact, if we have two neighborhoods which are close enough in terms of impact on the pricing, there's not really any sense in differentiating them. I have generated a scatter plot of the Neighborhood vs SalePrice with those plateaus merged into 3 distinct values but the result is not

satisfactory, the dilution of information is way too much! I assume this is because the plateaus are not distinct enough; I'll keep this idea in mind though for my next projects…

All features heatmap encoded using Output-based Encoder



A heatmap shows how much the rows and columns of a matrix are correlated, the more they are related, the darker red is the square corresponding and the lesser they are related, the darker blue is the square.

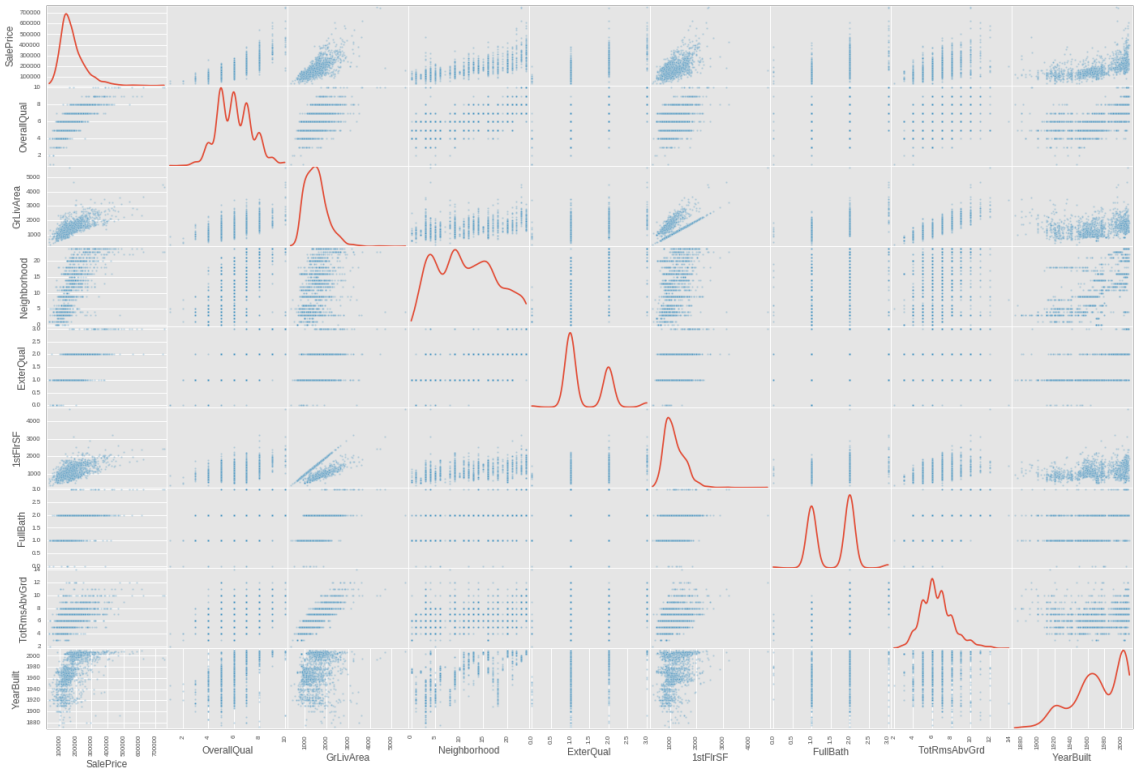
This heatmap shows that a lot of the features are correlated with each others; this is to be expected since a house is a coherent thing. For example, there are almost always more bedrooms in a big house or houses from a same neighborhood might have a particularly nice garden quality.

We can see here that the relation between Exterior1st and Exterior2nd is really high; I tried to merge those two columns but being categories, I first checked the combinations existed in both

training and submission sets and it didn't…

I think that those relationship are interesting since it shows that the houses descriptions are coherent but it might also cause a great deal of difficulty to differentiate the core relationships to the sale price and the secondary relationships which are side effects of the first… This will be an evident source of overfitting since, for example, any algorithm could associate an more important coefficient to GarageYrBlt compared to YearBuilt but this would be a wrong weighting from a expert point of view…

SalePrice highly correlated features scatter plot



The scatter matrix shows every point of a dataframe in nxn mini plots whose axises are the features 2 by 2. The diagonal corresponds to the histogram of the feature.
Here, we can observe on the first row or the first column the correlation between the sale price and the most important features discovered during the data exploration.
The linear relation is quite evident but plotting also the relation between those important feature also shows that some are highly correlated with each others. For example, 1stFlrSF and GrLivArea are tighly related but TotRmsAbvGrd and YearBuilt are clearly unrelated.
In the notebook HousePrices_DataExploration, I have plotted some more associations of features showing that combining those important features might add some intersting features to the dataset.

Algorithms and Techniques

The technique I'm using here is the stacking method. This method is basically using many regressors and make some predictions (layer 1 in our stack), then, we use those predictions as input of a second layer where we have other regressors etc for as many layers as desired.
In order to be as efficient as possible, the algorithms used per layer should extract different information from the datasets and then make different predictions. The more different the predictions are (being still not too far from the truth), the more the next layer will be able to devise a good combination and narrow down the error.
I have spent a great deal of time filling the missing values and though I saw some kagglers describing how they use different filling methods as a hyper parameter of their stacking system, I chose not to do so since I'm not sure to grab the effect of doing so (I have some ideas about adding some noise to the dataset to make it more robust like we do for neural networks when dropping some neurons but I'll need to think more about it and do some tests; in any case, I would think that a sort of Gaussian noise or the like will be a clearer solution).

In order to extract different information from the dataset, I have used different encoding algorithms based on 3 flavors:
1. Output-based encoding (referred to ordered in my code)
2. LabelEncoder based
3. Onehot encoding
On top of those 3, I have also played with the normalization and the skewness to produce a total of 9 different flavors of input datasets.
Finally on the input side, I have also added combinations of the Boruta and Lasso important features for certain base algorithms of the layer 1.

In the same spirit of producing different predictions, I have used 4 flavors of the ouput:
1. The SalePrice as it is
2. The natural logarithm of the SalePrice
3. The square root of the SalePrice
4. The inverse of the SalePrice (multiplied by 1000 to avoid precision errors)
3. The SalePrice divided by 1000 (could also affect some precision errors)

In my solution, I have written a framework allowing to easily select a model, its parameters, the output form (log, sqrt…) and the input flavor I want to use for any layer (though the input flavor can be used only for the layer 1).

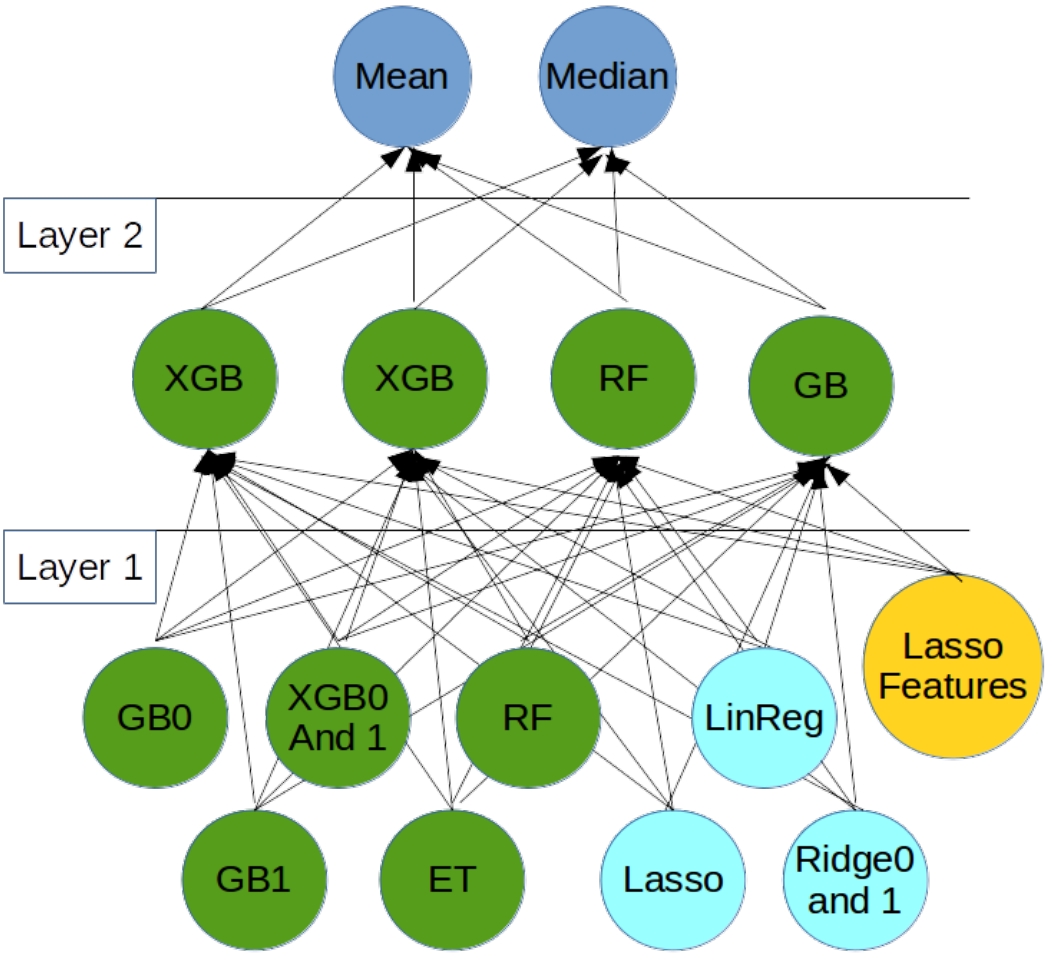
For every base model I've used for every layer, I have used three methods to fine tune the parameters:
1. GridSearchCV
2. BayesianOptimization

GridSearchCV is well know and used all along the course.
BayesianOptimization is another method I found when looking for an advanced method to fine tune tree-based models. Bayesian optimization is a sequential design strategy for global optimization of black-box functions that doesn't require derivatives (source Wikipedia).

The base algorithms I've used are for the layer1 are composed of 6 tree based models and 4 non-tree based:
1. GradientBoostingRegressor (2 models with different parameters, output, and input flavors)

2. XGBRegressor (2 models with different parameters, input flavors and objective functions)
3. ExtraTreesRegressor
4. RandomForestRegressor
5. Ridge (2 models with different parameters, output, and input flavors)
6. Lasso
7. LinearRegression

The second layer is using are all tree based (the non-tree based models I tried didn’t behave well enough):
1. XGBRegressor (2 models with different parameters and objective functions)
2. GradientBoostingRegressor
3. RandomForestRegressor



Benchmark

As a benchmark, I'll use the competition leaderboard which will allow me to compare my solution to the thousands of solutions adopted by the community. Given the number of

participants to the competition (roughly 3500) and the density of the scores, I will not take the ranking as a benchmark but the proximity to the best solutions.
As described before, the metric used to measure the result is the RMSE or RMSD on the logs of the price and the estimation; taking the logs allowing to not get a huge penalty on the most expensive houses prediction error.

In this section, you will need to provide a clearly defined benchmark result or threshold for comparing across performances obtained by your solution. The reasoning behind the benchmark (in the case where it is not an established result) should be discussed. Questions to ask yourself when writing this section:
- Has some result or value been provided that acts as a benchmark for measuring performance?
- Is it clear how this result or value was obtained (whether by data or by hypothesis)?

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

In this section, all of your preprocessing steps will need to be clearly documented, if any were necessary. From the previous section, any of the abnormalities or characteristics that you identified about the dataset will be addressed and corrected here. Questions to ask yourself when writing this section:
- If the algorithms chosen require preprocessing steps like feature selection or feature transformations, have they been properly documented?
*- Based on the **Data Exploration** section, if there were abnormalities or characteristics that needed to be addressed, have they been properly corrected?*
- If no preprocessing is needed, has it been made clear why?

Implementation

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:
- Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?
- Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?
- Was there any part of the coding process (e.g., writing complicated functions) that should be

documented?

Refinement

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:
- Has an initial solution been found and clearly reported?
- Is the process of improvement clearly documented, such as what techniques were used?
- Are intermediate and final solutions clearly reported as the process is improved?

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:
- Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?
- Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?
- Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?
- Can results found from the model be trusted?

Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- Are the final results found stronger than the benchmark result reported earlier?
- Have you thoroughly analyzed and discussed the final solution?
- Is the final solution significant enough to have solved the problem?

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:
- Have you visualized a relevant or important quality about the problem, dataset, input data, or results?
- Is the visualization thoroughly analyzed and discussed?
- If a plot is provided, are the axes, title, and datum clearly defined?

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:
- Have you thoroughly summarized the entire process you used for this project?
- Were there any interesting aspects of the project?
- Were there any difficult aspects of the project?
- Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this

section:
- Are there further improvements that could be made on the algorithms or techniques you used in this project?
- Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?
- If you used your final solution as the new benchmark, do you think an even better solution exists?

Before submitting, ask yourself...

- Does the project report you've written follow a well-organized structure similar to that of the project template?
- Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?
- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?
- Does the code execute without error and produce results similar to those reported?

> Written with StackEdit.