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 here). 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'. 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.

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 correlated 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 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:

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

In our case, in addition to the fact that Kaggle chose it as the metric for the ranking, it is particularly well suited to our problem where our estimations range from $40000 to 755000$. The error on a house in the top of the range will likelly be more important in terms of dollars than an error on the bottom of this range and hence, the mean would be much more affected by those high price houses. Using log makes the sale price range from 10.6 to 13.5 only and an error of 10% is 0.10 and 0.11 respectively on the low and high prices.
This makes this metric a very good one for the problem at hand.

II. Analysis

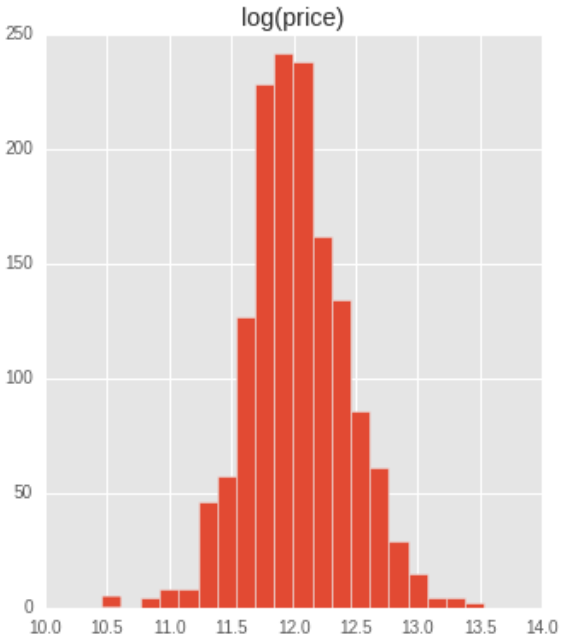
Data Exploration

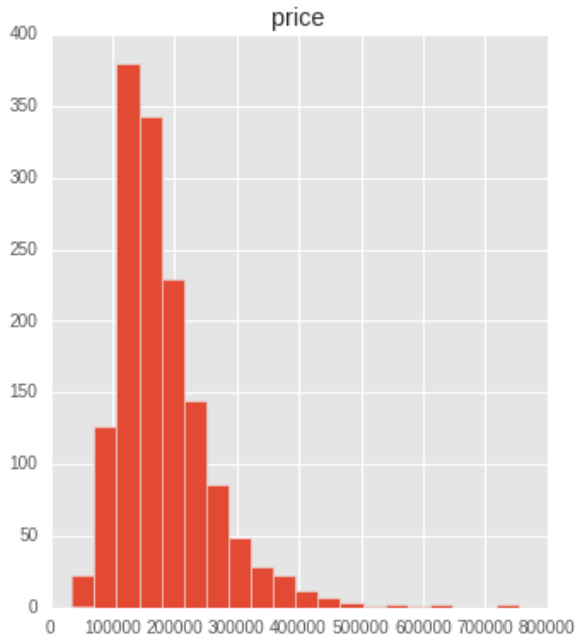
The data provided to support the competition is in the form of two CSV files; a first file containing 1460 samples with sale prices for training and a second file with 1459 records with no sale price for evaluation and ranking by Kaggle.
The datasets are describing the sale of individual residential properties in Ames, Iowa from 2006 to 2010 and comprises 79 features with invaluable 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. There is also a separate text file containing 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,
- The quality of almost everything

There are too much features to make an exhaustive list here; please find the full description here.

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).

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.

Outliers detection

On the outliers side, I tried a selection using 2 different methods:
1. SalePrice inter-quantile range
2. Identification using the price per square foot

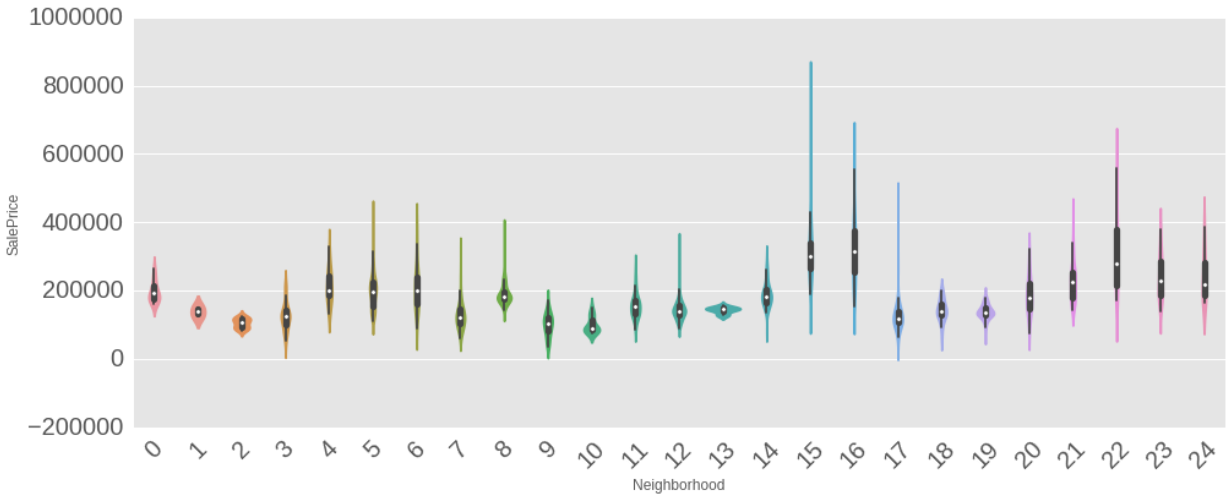
The first method did not yield good results when generalizing to the submission set.
The second method has been mentioned by Deean De Coq in his document on the dataset and is the method yielding the best results. Here is Dean's note on the subject:

> Potential Pitfalls (Outliers): Although all known errors were corrected in the data, no observations have been removed due to unusual values and all final residential sales from the initial data set are included in the data presented with this article. There are five observations that an instructor may wish to remove from the data set before giving it to students (a plot of SALE PRICE versus GR LIV AREA will quickly indicate these points). Three of them are true outliers (Partial Sales that likely don't represent actual market values) and two of them are simply unusual sales (very large houses priced relatively appropriately). I would recommend removing any **houses with more than 4000 square feet** from the data set (which eliminates these five unusual observations) before assigning it to students.

Exploratory Visualization

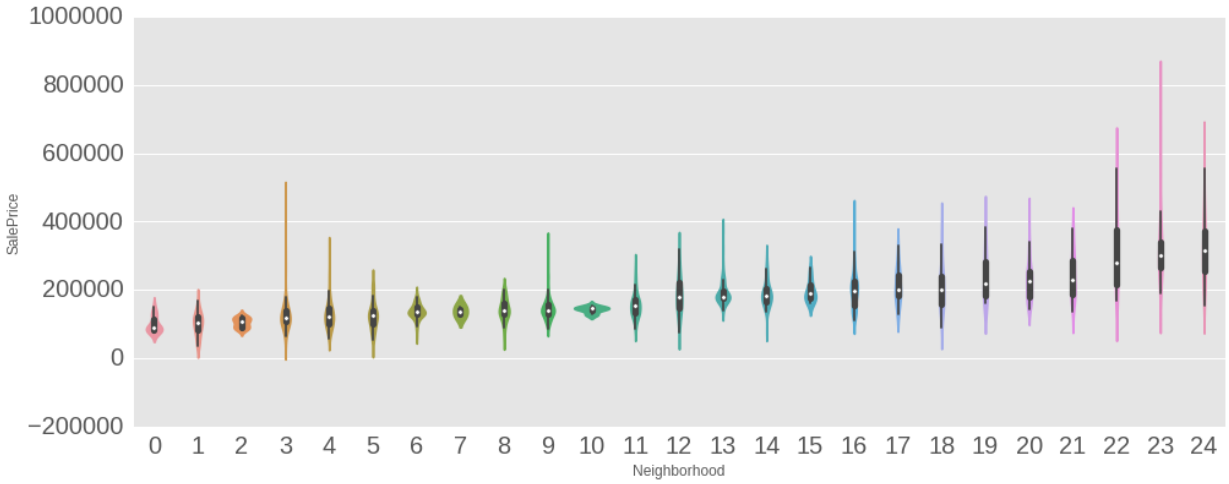
I have visualized a great deal of information (most of it is not even in the notebooks) to get a feel on the information. Given the number of features, a lot of the plots could not be placed in the full context.
I will present here five of the data visualizations:
1. A feature encoded with the label 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)

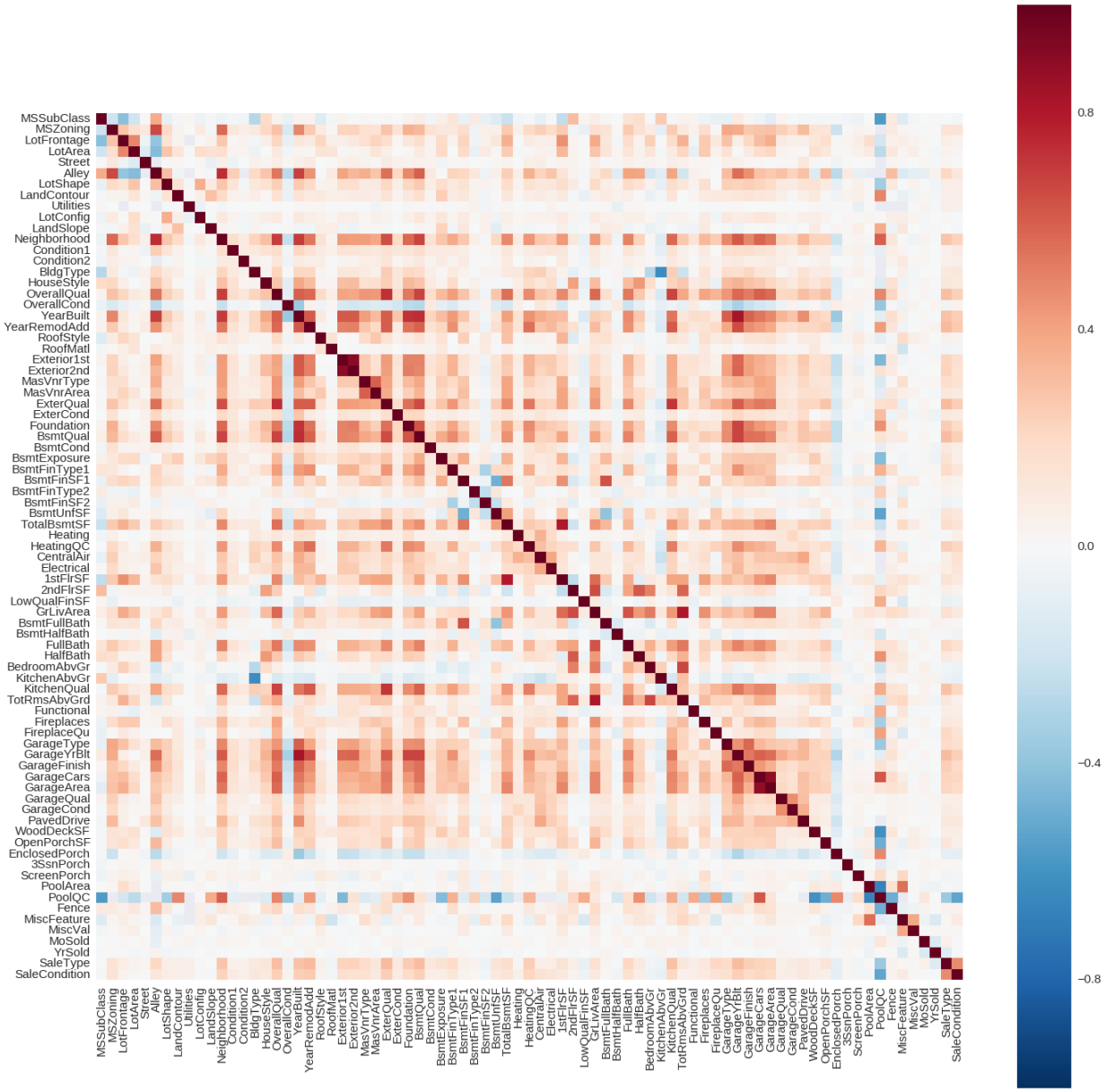


Here, we can clearly observe that the relationship between the neighborhood and the price is tight and much more linear.

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



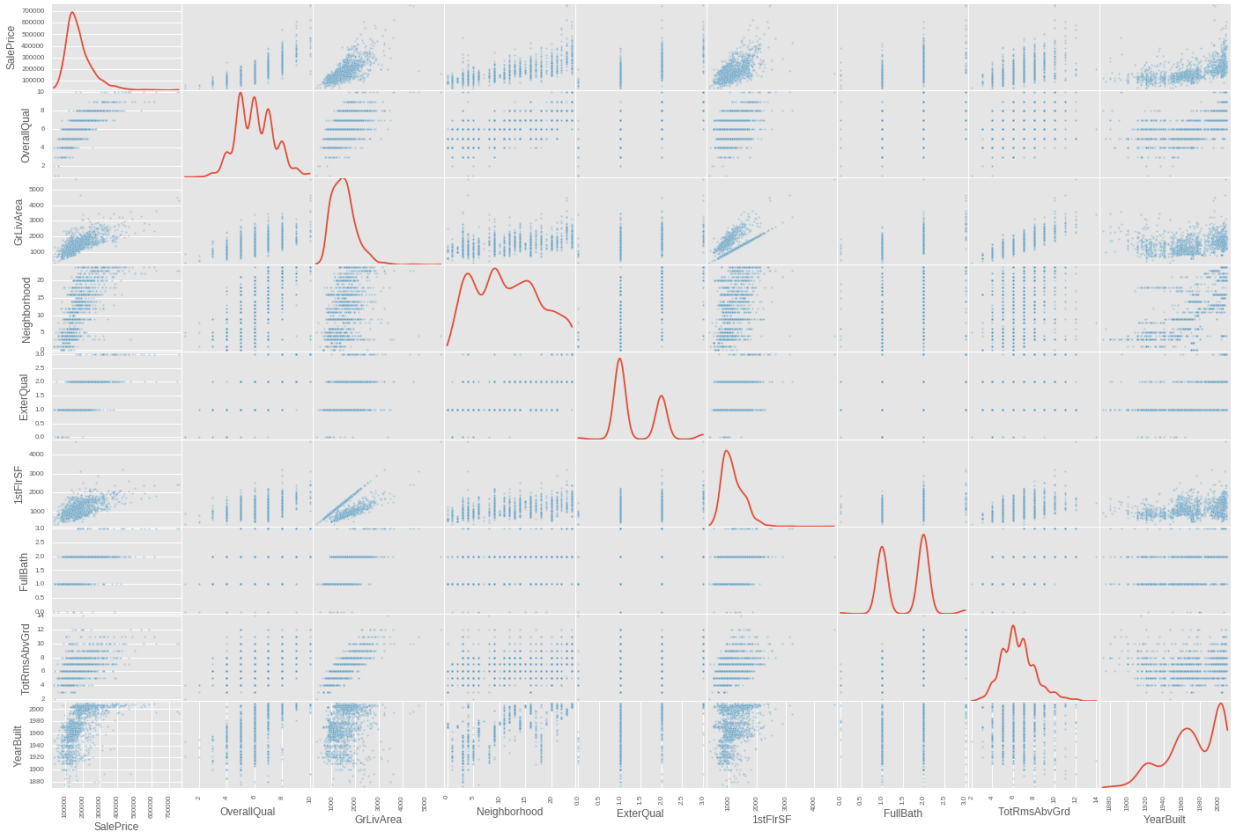
A heatmap shows how much the rows and columns of a matrix are correlated, the more they are related, the darker red is the square corresponding and the lesser they are related, the darker blue is the square.

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 they 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 adding value to the house but not really determining its price… This will be an evident source of overfitting since, for example, any algorithm could associate a 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 axes are the features. The diagonal corresponds to the histogram of each 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 interesting 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 two 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).

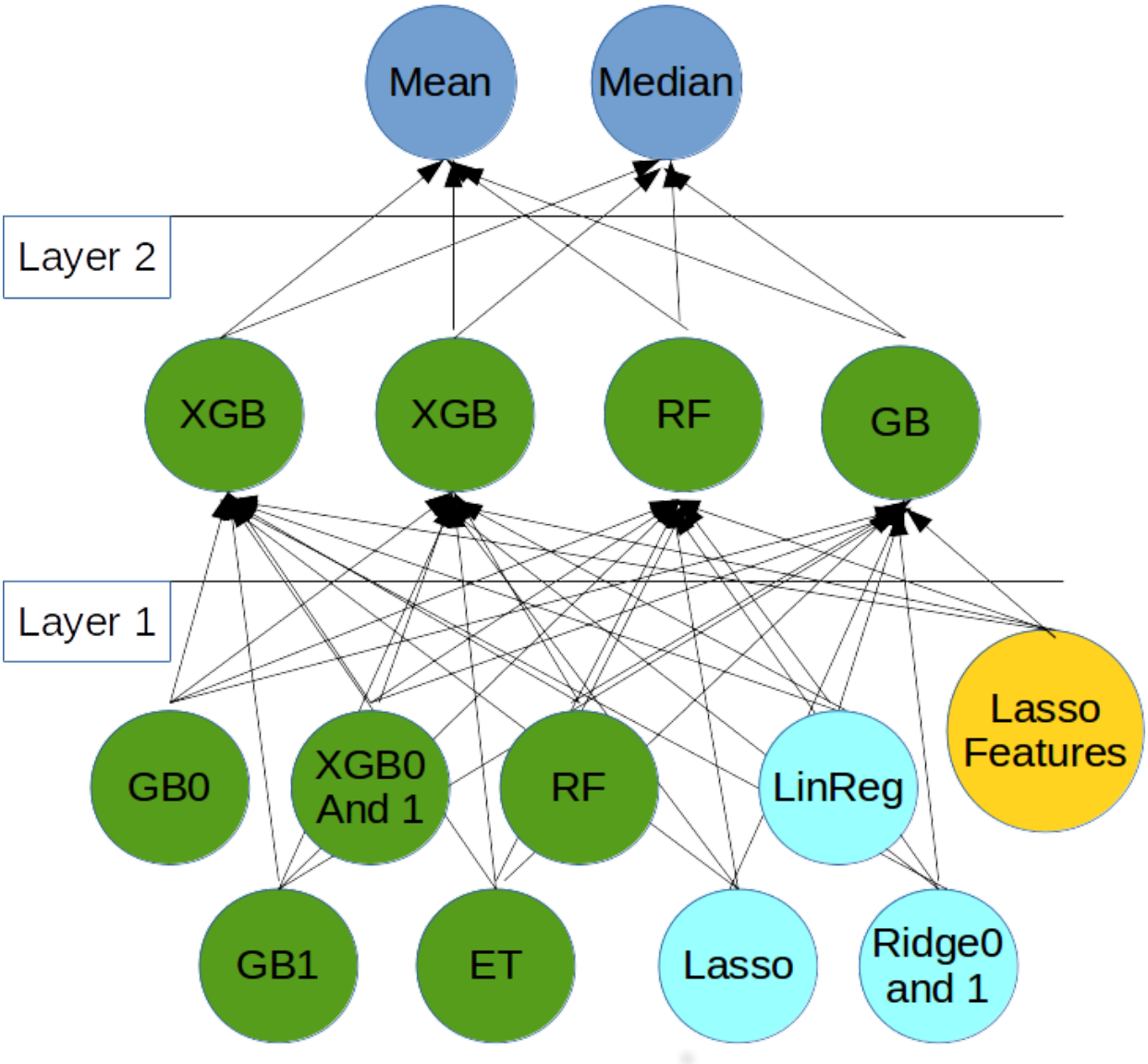
The base algorithms I've used for the layer1 are composed of 6 tree based models and 4 non-tree based:
1. GradientBoostingRegressor (2 models with different parameters, output, and input flavors)
2. XGBRegressor (2 models with different parameters, input flavors)
3. ExtraTreesRegressor

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

The second layer is using tree based model (the non-tree based models I tried didn't behave well enough):
1. XGBRegressor (2 models with different parameters)
2. GradientBoostingRegressor
3. RandomForestRegressor

Finally, I will not count that as a layer but a consolidation of the predictions, I used the mean and median operators to compute the final prediction (taking the best of it for every fine tuning I preformed).

The final setup looks like the following:



Note: I didn't represent the input flavors for the sake of clarity…

For each model, here are the parameters I used:

- **GradientBoostingRegressor**:
    - *n_estimators*: number of estimators or regression trees used,
    - *max_features*: maximal numer of features that any tree can use (I used it to limit overfitting),
    - *max_depth*: maximal depth that any base learner can reach (also a limit to overfitting),
    - *learning_rate*: shrinks the contribution of each tree by learning_rate (balance between overfit and bias),
    - *subsample*: fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. subsample interacts with the parameter n_estimators. Choosing subsample below 1.0 leads to a reduction of variance and an increase in bias.

- **XGBRegressor**:
    - n_estimators: same as above
    - max_depth: same as above
    - learning_rate: same as above
    - subsample: same as above
    - min_child_weight: See here - Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree

- **ExtraTreesRegressor:**
    - max_depth: same as above
    - n_estimators: same as above
    - max_features: same as above
    - min_samples_split: minimum number of samples required to split an internal node (I used it to limit overfitting)

- **RandomForestRegressor:**
    - max_depth: same as above
    - n_estimators: same as above
    - max_features: same as above

- **Ridge:**
    - alpha: Regularization strength improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. (I have favored alpha over pure performance by tuning the input dataset used)

- **Lasso:**

- alpha: Constant that multiplies the L1 term. (I have used this parameter as we use the learning rate with other algorithms)
- max_iter: maximum number of iterations (I had to tune this parameter to allow the algorithm to converge despite the low alpha)

As you can see above, most of my tuning has been to limit the overfitting than can be reached very easily with those algorithms still keeping some performance on the prediction. Overfitting has been my major concern all along this project; I have had results below 0.06 with overfitting leading to leaderboard results above 0.14…

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 3700) 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.

Some recent comments on the forum dedicated to the competition and the scores obtained by the best submissions are raising serious doubts on the competition ranking validity. The dataset used by Kaggle is public and published here.
Moreover, some competitors showed there that given the score computed by Kaggle on just a few submissions, it is possible to get any score (from the perfect to the worst and any in between).

So, I'll place the scores in 5 categories (per range of scoring):
1. From perfect score to 0.11 (first 17 best scores)
2. From 0.11 to 0.12 (next 700 best scores)
3. From 0.12 to 0.13 (next 1000 best scores)
4. Above 0.13 (last 2000 best scores)

Note that a range as wide as 0.01 represents a difference of 7000 dollars on the prediction of most expensive house, 350 dollars on the least expensive and 1700 dollars on the average.
So, I'll consider entering the first range as incredible! The second range will be a good achievement; the third will be an average score and the last will be bad :-(

III. Methodology

Data Preprocessing

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 wrong data and penalize the predictions. I have filled the missing data automatically from mean or median values as observed so many times in public algorithms shared by the Kaggle community and all around the web and I have also spent a big time filling each feature (column) individually based on the feature description coming along with the datasets and the rest of the features of the given records. For example, the missing basement descriptions have been reset to default values when there was no basement and the data was missing because of that in the dataset. 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 records belonging to the same neighborhood assuming that those features might be closer in the same geography.
As mentioned earlier, I used the 4000SF Ground Living Area as an outliers limit.

As suggested by many when talking about stacking and explained before, after the missing values filling, my preprocessing consisted in feeding my layer 1 algorithms with the best possible flavors of the dataset.
So, I have generated 9 distinct flavors of the inputs:
1. OneHot referred as ‘onehot’ in the code
2. OneHot with log of features whose skewness is above 0.75
3. OneHot with Robust Scaling transformation
4. Label Encoded referred as ‘int’ in the code
5. Label Encoded with log of features whose skewness is above 0.75
6. Label Encoded with Robust Scaling transformation
7. Output based encoded referred as ‘ordered’ in the code
8. Output based encoded with log of features whose skewness is above 0.7
9. Output based encoded with Robust Scaling transformation

In addition to those transformation, I have used the features of importance discovered using Boruta and Lasso to compute new features based on combinations (products) of those features of importance.
This gives us 27 different flavors possible for the inputs (each of the nine listed above vanilla or with Boruta combinations or with Lasso combinations).

Finally, and as suggested in several stacking feedback, I have also added some tuning on the SalePrice with four transformations (not counting the no transformation):
1. No transformation
2. Log transformation
3. Inverse transformation (multiplied by 1000 so as to to have precision errors)
4. Square root transformation
5. Division by 1000 transformation

Implementation

I have implemented my final version in 4 steps:
1. Hyper parameterizing of every layer 1 models based on the 27 input flavors and the 5 outputs
2. Caching of the layer 1 final results
3. Hyper parameterizing of every layer 2 models based on the 5 outputs
4. Computation of the mean and median predictions for the layer 2 and submission of the best

Hyper parameterizing of every layer 1 models

In this step, I have first chosen manually a set of hyper parameters to create a model allowing me to decide which input/output flavors are best for the given model.
Once I was satisfied that the inputs/output were good, I used two methods to fine tune the models' parameters:
1. GridSearchCV for the non tree algorithms
2. BayesianOptimisation for the tree based algorithms

My tests showed that the Bayesian optimization method was not entirely satisfactory for the Lasso and Ridge models since it didn't scan properly the number of maximal iterations… But the tuning of all the tree based algorithms was impressive.
I ddin't check the Leaderboard score for every final Layer 1 models independently but checked the best ones individually all along the project.

Caching of the layer 1 final results

In order to fasten my development and tuning time of the second layer, I have added a caching mechanism allowing me to retrieve the l1 training and submission results instead of recomputing them at every run.
This saved me hours of unnecessary computation!

Hyper parameterizing of every layer 2 models

In this step, as in the layer 1 step, I have first chosen manually a set of hyper parameters to create a model allowing me to decide which output flavor is best for the given model. I have also checked if re-injecting the Lasso or Boruta parameters was an improvement.
My tests showed that re-injecting the Lasso important features was improving significantly the results so I introduced those features for all the data.
Once I was satisfied that the inputs/output were good, I used the BayesianOptimisation method to fine tune the models' parameters.

I finally checked my models against the Leaderboard estimated score for every final model and was nicely surprised to find out that there was no overfitting…

Computation of the mean and median predictions

Finally, I compute the mean and median values of the predictions and use the one with the best score for the final best submission.

Refinement

In the table below are shown the first layer results obtained after refinement of the hyper parameters of the system.
All the initial results have been obtained with default model's parameters, the 'int' input

flavor and no transformation on the output.

Layer	Model	Initial result	Final parameters	Final result
1	XGBRegressor	0.1287	Input: ordered_skewed Output: log 'max_depth': 3, 'learning_rate': 0.05, 'min_child_weight': 5, 'subsample': 0.7, 'n_estimators': 500	0.1174
	XGBRegressor	0.1287	Input: ordered_skewed Output: log 'max_depth': 3, 'learning_rate': 0.03, 'min_child_weight': 3, 'subsample': 0.7, + BORUTA_FEATURES	0.1210
	GradientBoostingRegressor	0.1281	Input: ordered_skewed Output: div1000 'max_depth': 4, 'max_features': 5, 'learning_rate': 0.03, 'n_estimators': 500, 'subsample': 0.8, + LASSO_FEATURES	0.1174
	GradientBoostingRegressor	0.1281	Input: ordered_scaled Output: None 'learning_rate': 0.05, 'max_depth': 4, 'max_features': 15, 'n_estimators': 500, 'subsample': 0.8	0.1164
	ExtraTreesRegressor	0.1502	Input: ordered Output: div1000 'max_depth': 15, 'max_features': 0.68, 'min_samples_split': 5, 'n_estimators': 760 + LASSO_FEATURES	0.1237
	RandomForestRegressor	0.1487	Input: ordered Output: sqrt 'max_depth': 17, 'max_features': 0.4, 'n_estimators': 350 + BORUTA_FEATURES	0.1293
	Ridge	0.1493	Input: ordered_skewed Output: log 'alpha': 0.005	0.1179
	Ridge	0.1493	Input: ordered_skewed Output: sqrt 'alpha': 0.005 + BORUTA_FEATURES	0.1327
	Lasso	0.1495	Input: ordered_scaled Output: log 'alpha': 0.0006 + BORUTA_FEATURES	0.1191
	LinearRegression	0.1496	Input: ordered_scaled Output: log	0.1188

In the table below are shown the second layer refinement results. As above, the initial results are obtained with the default parameters and no tranformation.

Layer	Model	Initial result	Final parameters	Final result
2	XGBRegressor	0.1192	Output: log 'learning_rate': 0.005, 'max_depth': 4, 'min_child_weight': 6, 'n_estimators': 1500, 'subsample': 0.7	0.1178
	XGBRegressor	0.1192	Output: log 'max_depth': 2, 'learning_rate': 0.005, 'min_child_weight': 4, 'subsample': 0.7, 'n_estimators': 2000	0.1174
	GradientBoostingRegressor	0.1176	Output: log 'n_estimators': 500, 'max_features': 5, 'max_depth': 15, 'learning_rate': 0.01, 'subsample': 0.9	0.1149
	RandomForestRegressor	0.1230	Output: log 'max_depth': 17, 'n_estimators': 350, 'max_features': 0.4	0.1156

IV. Results

Model Evaluation and Validation

Here is the table representing the different scores on the training set and on the leaderboard for the second layer:

Layer	Algorithm	CV Score	LB Score
Layer 1	GradientBoostingRegressor 0	0.1164	NA
	GradientBoostingRegressor 1	0.1174	NA
	XGBRegressor 0	0.1174	NA
	XGBRegressor 1	0.1210	NA
	ExtraTreesRegressor	0.1237	NA
	RandomForestRegressor	0.1293	NA
	Ridge 0	0.1179	NA
	Ridge 1	0.1327	NA
	Lasso	0.1191	NA
	LinearRegression	0.1188	NA
Layer 2	XGBRegressor 0	0.1178	0.1211
	XGBRegressor 1	0.1174	0.1221

	GradientBoostingRegressor	0.1149	0.1210
	RandomForestRegressor	0.1156	**0.1203**
Final	median	0.1157	0.1205
	mean	0.1156	**0.1205**

The difference between the training error and the submission error is really small (0.1156 vs 0.1205) meaning that we are not overfitting the training set too much; the tests I conducted during the system setup showed that the test score improved with the training score (always with a delta but trends were good).

During my tests, I used several policies of outliers detection, introducing then small changes in the inputs; those changes showed that the current rule (GrLivArea inferior to 4000) is the best but the difference is really marginal (below 0.05). So, I tend to think that the stacking is quite robust.

Regarding the question whether the model can be trusted or not, I will address this question in the conclusion…

Justification

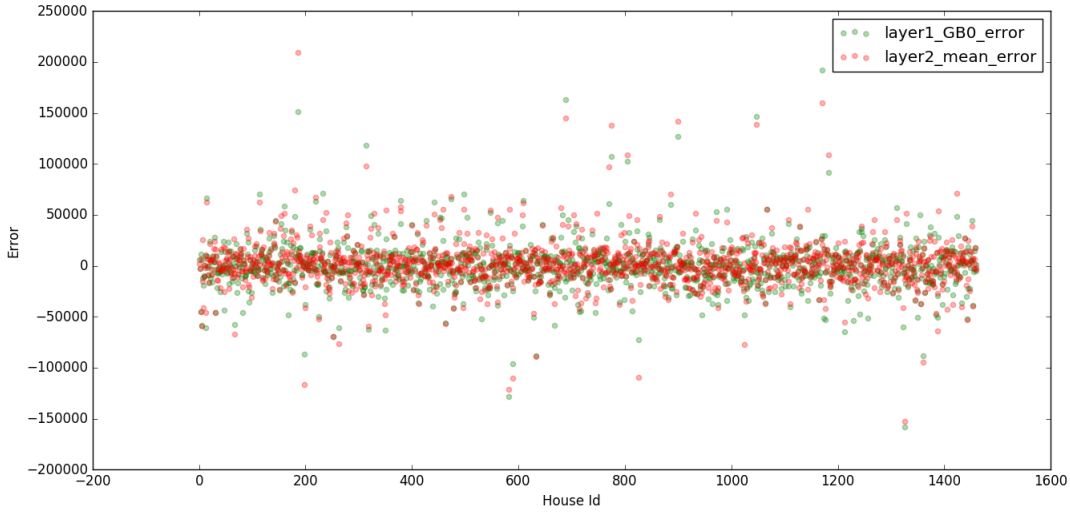
The score of the median prediction of the second layer is 0.1205 on the submission set and is 0.1203 for the second layer's RandomForest model which is within the third range targeted…

The training score of the best model of the first layer is 0.1164 for the GradientBoostingRegressor model. It is 0.1156 for the final prediction. Kaggle limits to 5 submissions a day so I didn't submit all the final first layer models in addition to the second layer but my prior submissions of single models showed that the leaderboard score never went below 0.13.
This is a major improvement that I never approached when playing with my single models!

V. Conclusion

Visualisation

Below is a plot showing the error between the predictions and the training data for the best model (first Gradient Boosting) of the layer 1 and the mean of the layer2.



We can note that the red points are a bit denser but the houses that were badly predicted still are badly predicted; I assume that this means two things:
1 - the system still didn't fully capture the information driving the price of a house
2 - the dataset still has some outliers (a house selling price might vary upon factors that were not in the datasets or even with no real reasons)

Two people reach leaderboard scores around 0.3 and two others around 0.7; if their submission are genuine, it means that they captured significantly more information from this dataset than myself.

Reflection

This project has been a very good first step into the real world of machine learning, I really broke my teeth on the data preprocessing even though I clearly found out that other Kagglers didn't show much effort in the other competitions I checked… The data preprocessing was a huge part that I addressed particularly thoroughly!

My layer 1 models have been a particular challenge to parametrize since I created so many variants of the input data and the output and I spent so many time playing with GridSearchCV until I found about the Bayesian Optimization.

The layer 2 models were much faster to train since they're playing with very few data but the cons of so few data have been the overfitting risk!

All the steps were really interesting but building it all has been my favorite part, making everything working together to improve the first layer predictions was great!!

On the bad side, however, an error of 0.12 represents an error of over $100000 on the $755000 house which is huge and probably not so good for a production system… It really saddens me to work so hard for so bad a result ;-(

Before crying to death, I compared to Zillow predictions accuracy; see here.

Extract:

> The Zestimate's accuracy depends on location and availability of data in an area. Some counties have deeply detailed information on homes such as number of bedrooms, bathrooms and square footage and others do not. The more data available, the more accurate the Zestimate.

Zillow accuracy is on average 45% to 65% of error within 5% of the house final price, 70% to 85% within 10% of the house final price and 86% to 95% within 20%; their median error published is between 3.1% and 5.7% (data published on November 01, 2016).

I computed the same metrics on my training predictions (which is slightly better but not so much) and got 45% within 5% error, 80% within 10% and 95% within 20% with a median value of 4.8%. This is in the same order as Zillow but Zillow has much less information to deal with in input but much more data on the other hand to avoid overfitting.

Improvement

I spent some time making variations by adding more models into my layers and selecting only the best ones. This method allow me to score a 0.11688 with around 30 models and keeping only the ones below 0.16 in the layer1 and the 3 best of layer 2. :-P
I have played with the threshold on layer 1 worst score and layer 2 number of best predictors and the score varies a little bit but keeping the very best is not the way to the best score.

I have not documented this effort here given the amount of information I gathered already and the fact that it was not the original plan…

Some people on the leaderboard made so much better (a lot of people scored between 0.9 and 0.117) that I think I'm missing some steps in the data processing; I don't think they did just add models to their stacking and we don't know who is using stacking and who is using single models…

I also tried some other steps in the data preprocessing that I didn't document and that I would like to exercise on another project like reducing the dimensionality of the inputs or applying a 2 steps prediction; for example, identifying core features to predict an order of pricing and evaluating the second level features to refine the price.

I got a lot of ideas during this project but a lot of them didn't bring the expected value; I know I applied some wrongly at the beginning of the project and some just weren't good ideas or too aggressive for the small dataset we are playing with here.

VI. Thanks

I would like to thank Udacity and the reviewers for the quality of the courses and of the reviews. **All** the reviews!

It's awesome to see the quality of the feedback I got; it was always positive, constructive, instructive and helpful.

Special thanks as well the Kaggle for supporting competitions, providing tons of data and inciting so many people to do their best in a so good and so opened environment.

My sincere thanks to everyone!!!