Machine Learning Engineer Nanodegree
Capstone Project

Hugo Saito

April 12st, 2018

I. DEFINITION

Project Overview

Nowadays, Machine Learning(ML) techniques are applied in many different fields. Fast-forwards in the last decade placed Machine Learning as the code of many high-tech products: ranking web search (Google), recommending videos (YouTube), driving car (Tesla), predicting stock prices, etc. Almost all field could (or are) use ML techniques to improve some desired goal, among these a potential field is the housing market.

The real estate agents have the challenging work of evaluate the price of house, but it is so difficult because each house is different and have many features to be evaluated. In addition, all humans suffer influence by sentimental state, family problems, stress levels and diverse others type of problems that probably will influence the result of the agents' evaluations. Thus, housing market is a good field to apply Machine Learning techniques for predicting house price.

Based on that, this project used the dataset called Ames Housing to create a machine learning model that treat with predicting house price problem.

The Ames Housing Data Set was synthesized by the professor Dean De Cock for an end of semester project for an undergraduate regression course and contains information from the individual houses sold in Ames, Iowa, United States from 2006 to 2010. It's a modernized and expanded version of the Boston House Data Set (70's). The features are information that a home buyer or a real estate agent need to know to evaluate a property. This dataset should be used as a teacher for learner algorithm, because the model will use their labels points to extract the patter, seeking to become an effective learner.

The data set was obtained from Kaggle Competition [1] and consists of 2930 samples and 82 features of distinct types: 23 nominals, 23 ordinals, 14 discrete and 20 continuous, where one of the continuous variable is the target variable (*SalePrice*) for this project.

Problem Statement

The housing market is highly competitive. A property can be evaluated by different real estate agents and certainly all prices provided will be different from each other, because the complexity of many characteristics makes the agents being prove to errors. So, use machine learning techniques looking to predict the correct sale price can be an effective way to find the best price for each house. The *correct price* means that based on the quantity and quality of many attributes of the properties, a learner algorithm will seek the fairest price for each house.

The goal is to create a learner algorithm using supervised learning methods, where the learner will use a label data set to extract the pattern and correlation between all house's features looking to find an efficient way to determine the price of a new house (new data point) based on their characteristics, where the price is the target variable. It's easy to observe the performance of the learner if I use just the features of some points that doesn't were used in the learning phase as a new point and compare the true result with the learner prediction.

About strategy, firstly, the Ames Housing Data Set will be explored (simple visualizations, correlations) to gain a basic insight. So, I will prepare the data to be used by machine learning algorithms executing Data Cleaning, Feature Selection, Feature Engineering, Feature Scaling and Dimensionality Reduction when these steps are necessary.

After that, I will apply many supervised models seeking identify a short-list with the most promising and will fine-tune these models for improve their results into testing set, using an appropriate evaluation metric as comparator. Finally, I will apply Ensemble Learning method to verify if I the aggregate model result is better than the individual predictors.

Metric

A typical performance metric for regression problems is Root Mean Squared Error (RMSE) because it's gives a sense of how much error the algorithm (learner) makes on average with higher weight for large errors in its predictions. In other words, obtains a measure of the spread of the predicted values around that average.

The Kaggle's competition specified in the description (section about Evaluation) that the submissions will be evaluated on RMSE between the logarithm of the predicted value and the logarithm of the observed sales price. [4] The logarithm means that positive or negative error will affect the result equally.

So, I will use this metric to evaluate the performance of the models utilized.

The math behind RMSE:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_{true} - y_{pred})^2}{n}}$$

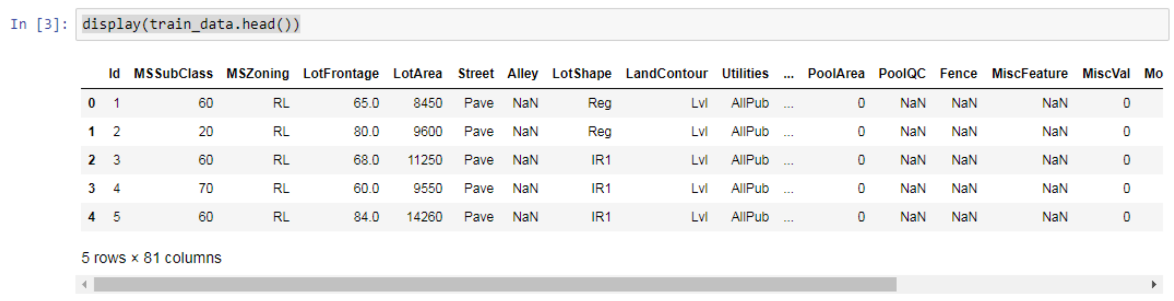
where, y_{true} is the true value of target variable and y_{pred} is the prediction from the model and n is the number of sample that were tested.

II. ANALYSIS

Data Exploration

The dataset consists of 2919 samples and 8 features of distinct types: 23 nominals, 23 ordinals, 13 discrete and 20 continuous. One of the continuous variable is the target variable (SalePrice) for this project. As the number of variables are large, the brief version of data description can be found *HERE*[2] and if you want the full description, click *HERE*[4].

The half of all sample is used as training dataset and the remain is to testing dataset. The image bellow shows 5 examples of training dataset:



```
In [3]: display(train_data.head())
```

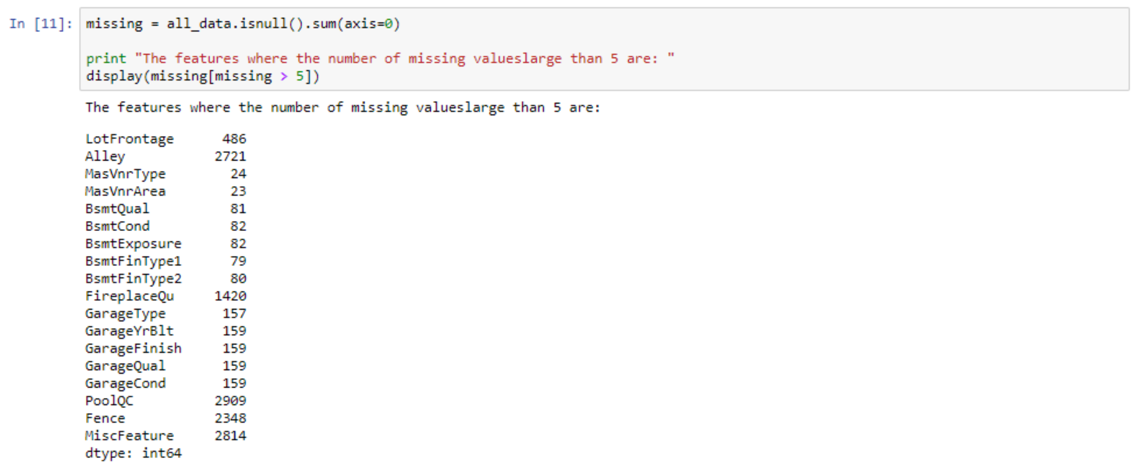
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	Mo
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

5 rows × 81 columns

Figure 1 - Train dataset visualization

After read dataset description and see the type of each feature, notice that the 'MS Subclass' is a categorical feature with numeric (int64) type. Therefore, it's wrong and need to be fixed because outlier detection or statistical information result in misleading information.

Calculating the number of missing values per feature shows that some feature has many missing values and this point need to treat because can be a noisy in the data for the learner. The percentage of missing value vary between 1% until 99% where 15 numerical features exhibit approximately more than 30% of missing values. See the image below:



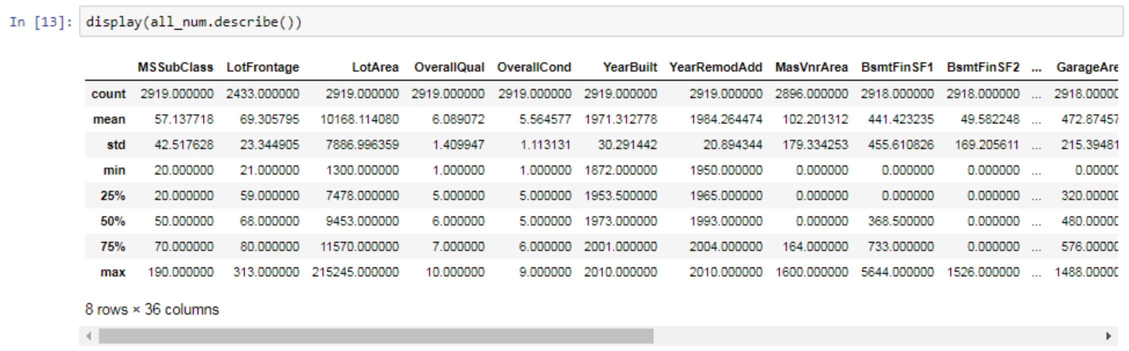
```
In [11]: missing = all_data.isnull().sum(axis=0)

         print "The features where the number of missing valueslarge than 5 are: "
         display(missing[missing > 5])

         The features where the number of missing valueslarge than 5 are:

         LotFrontage      486
         Alley           2721
         MasVnrType        24
         MasVnrArea        23
         BsmtQual          81
         BsmtCond          82
         BsmtExposure      82
         BsmtFinType1      79
         BsmtFinType2      80
         FireplaceQu     1420
         GarageType       157
         GarageYrBlt      159
         GarageFinish     159
         GarageQual       159
         GarageCond       159
         PoolQC          2909
         Fence           2348
         MiscFeature     2814
         dtype: int64
```

Figure 2- Number of missing value per feature

Analyzing the statistical properties of numerical features, the high number of missing values for some features causes the first quartiles to exhibit values equal zero. This fact is a big problem to the step of outlier detection and must be solve. See the output of Jupyter Notebook (*Data Exploration and Preprocessing.ipynb*) archive below, the percentage of zero value of each numerical feature are:

```
The percentage of ZEROS for MasVnrArea is 0.595409386776.
The percentage of ZEROS for BsmtFinSF1 is 0.318259677972.
The percentage of ZEROS for BsmtFinSF2 is 0.880781089414.
The percentage of ZEROS for BsmtUnfSF is 0.0825625214114.
The percentage of ZEROS for TotalBsmtSF is 0.0267214799589.
The percentage of ZEROS for 2ndFlrSF is 0.571428571429.
The percentage of ZEROS for LowQualFinSF is 0.986296676944.
The percentage of ZEROS for BsmtFullBath is 0.584104145255.
The percentage of ZEROS for BsmtHalfBath is 0.939362795478.
The percentage of ZEROS for FullBath is 0.00411099691675.
The percentage of ZEROS for HalfBath is 0.62829736211.
The percentage of ZEROS for BedroomAbvGr is 0.00274066461117.
The percentage of ZEROS for KitchenAbvGr is 0.00102774922919.
The percentage of ZEROS for Fireplaces is 0.486467968482.
The percentage of ZEROS for GarageCars is 0.0537855429942.
The percentage of ZEROS for GarageArea is 0.0537855429942.
The percentage of ZEROS for WoodDeckSF is 0.521754025351.
The percentage of ZEROS for OpenPorchSF is 0.444672833162.
The percentage of ZEROS for EnclosedPorch is 0.842754367934.
The percentage of ZEROS for 3SsnPorch is 0.987324426173.
The percentage of ZEROS for ScreenPorch is 0.912298732443.
The percentage of ZEROS for PoolArea is 0.995546420007.
The percentage of ZEROS for MiscVal is 0.964713943131.
```

The basic statistical properties of numeric features are showed below:



```
In [13]: display(all_num.describe())
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	GarageAre
count	2919.000000	2433.000000	2919.000000	2919.000000	2919.000000	2919.000000	2919.000000	2896.000000	2918.000000	2918.000000	...	2918.0000
mean	57.137718	69.305795	10168.114080	6.089072	5.564577	1971.312778	1984.264474	102.201312	441.423235	49.582248	...	472.87457
std	42.517628	23.344905	7886.996359	1.409947	1.113131	30.291442	20.894344	179.334253	455.610826	169.205611	...	215.39481
min	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	...	0.0000
25%	20.000000	59.000000	7478.000000	5.000000	5.000000	1953.500000	1965.000000	0.000000	0.000000	0.000000	...	320.0000
50%	50.000000	68.000000	9453.000000	6.000000	5.000000	1973.000000	1993.000000	0.000000	368.500000	0.000000	...	480.0000
75%	70.000000	80.000000	11570.000000	7.000000	6.000000	2001.000000	2004.000000	164.000000	733.000000	0.000000	...	576.0000
max	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1526.000000	...	1488.0000

8 rows × 36 columns

Figure 3- Statistical description

For see the statistical properties of all (36) numerical features, look the archive Jupyter Notebook with the name *Data Exploration and Preprocessing.ipynb*

Exploratory Visualization

Exploring the distribution of the target (SalePrice), the skewness of these distribution need to be treat, because it's similar to a Log Normal, Gamma or Wedbul distribution and the objective is to become more normal (Gaussian). The image of this distribution:

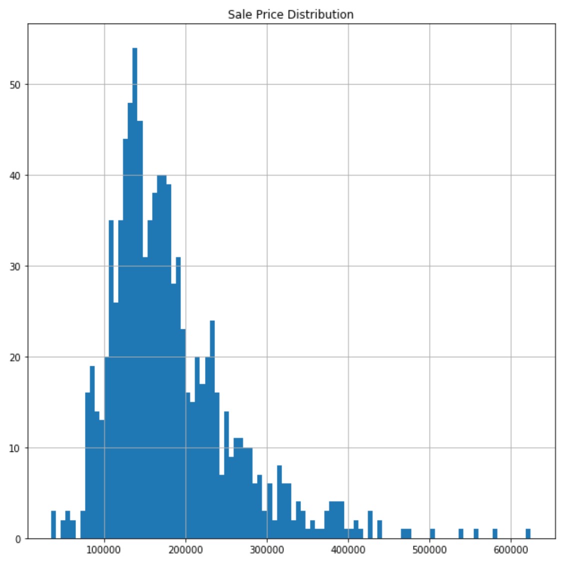


Figure 4- SalePrice (target variable) distribution

Analyzing the others numerical features, almost of their distributions are asymmetrical. Similar to target variable, these features need to be treat and even each one displaying a specific type of distribution the same transformation will be applied for all of them.

The image of all of these distributions might be found in the Jupyter Notebook with the name *Data Exploration and Preprocessing.ipynb.*

Algorithms and Techniques

Before the discussion about each Algorithm I used in this project, it's necessary to explain that as I have little experience with Machine Learning techniques. Thus, for find an acceptable solution I need to test many type of different models until I found good models, that is, I used "trial and error" to balance my lack of experience.

Models Used

Each one of algorithm (model) used is explained below:

1. Linear Regressor

Linear Regressor is a basic regression model that assume that the features can be combined linearly. Thus, this model tries to estimate a function of the form:

$$f(x) = C^T.x + C_0 ,$$

where x is the vector of inputs, $x \in R^i$; C is the coefficients, $C \in R^i$; C_0 is the y-intercept, $C_0 \in R$ and i = number of features.

For coefficients determination, is used Least Squared Error to find the C^T and C_0 that minimize this metric. So, basically minimizing the sum of the squared of the differences between the target value (real value) and the predicted values (output of model). In this model anyone regularization is used.

So, as the project is a Regression problem with many features, if a assume that prediction can be perform by linear combination, I can use this to solve this problem and if in the test step, the model exhibit a good metric evaluation (RMSE) we can identify that the assumption of linearity was correct.

2. Kernel Ridge Regressor

Kernel Ridge Regressor can performs linear, non-linear and non-parametric regression and applies kernel trick combined with Ridge Regression. Therefore, this model uses squared error loss as loss functions and l2-norm as regularization looking solve a regression problem and avoid overfitting.

Ridge Regression is the Tikhonov regularization in the context of regression where the penalty is equivalent to square of the magnitude of coefficients and all penalties of coefficients are equals, except the y-intercept coefficient that is not penalizing.

Kernel Trick is the trick of manipulate the original dimensions (features) and combined than in a high dimension space where is possible to find a separating hyperplane. The image below represents this idea in 2D to 3D dimension:

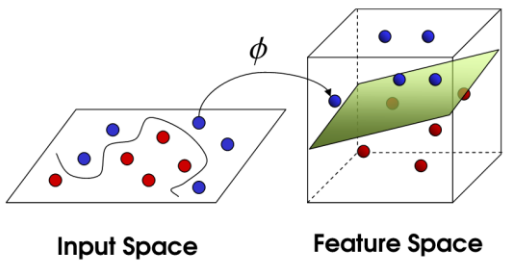


Figure 5- Kernel Trick illustration [9]

Therefore, this model can be used for this project problem because may exhibit a good result assuming linear or non-linear kernels. Thus, this model is versatile and is a good option when the dataset is not clearly linear or non-linear.

3. Lasso Regressor

Lasso regressor is a linear model that uses the shrinkage and provide a sparse solution, using L1 regularization. L1 regularization adds a penalty equal to the absolute value of the magnitude of coefficients. This regularization can result in a sparse model and some coefficients may become zero and are remove from model, thus, this model might execute feature selection indirectly. Therefore, Lasso works well with high number of features.

Although Lasso is similar to Ridge Regression, Ridge doesn't result in zero's coefficient, sparse models or probably works worse with high number of features based on computational cost.

So, the Lasso Regressor is a good model that might result satisfactory result in the regression problem of this project because getting a sparse solution may be a great computational advantage.

4. Gradient Boosting Regressor

“Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.” (Wikipedia)

The model Gradient Boosted Regression Tree (GBRT) uses a Decision Tree Regressor (DTR) multiple times where the second DTR is trained with the residual (difference between true values and prediction of first DTR), the third DTR is trained with the residual of second DTR and so on. This idea is shoed in the image below:

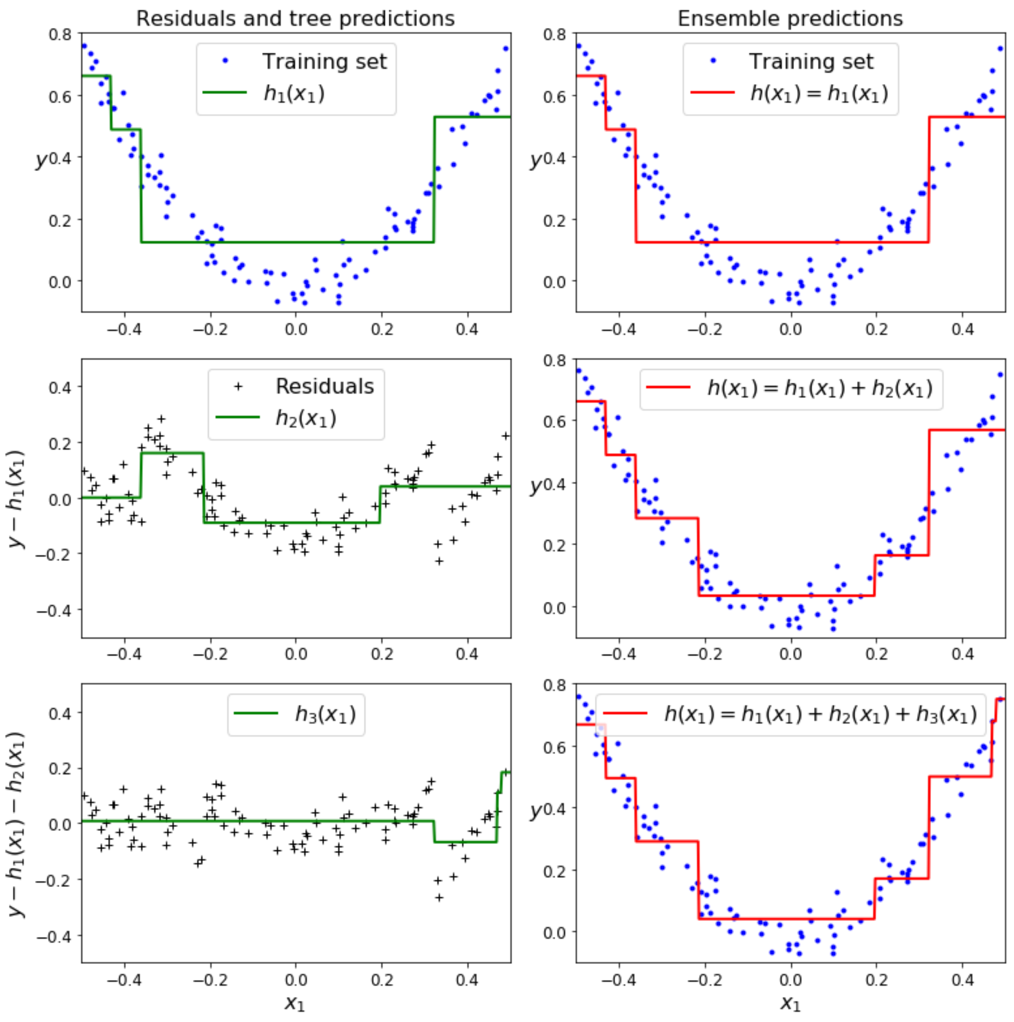


Figure 6- Gradient Boosting illustration [8]

As the image show, the iterative process converges to a better solution for regression problem. A great point of this model is that works with different loss functions. Thus, this model might be useful for the problem of this project and may work for linear and non-linear problems.

5. Random Forest Regressor

Random Forest Regressor (RFR) will be used as an Ensemble Learning, that is, this model works on top of many other models (all models above in this project). The base model of Random Forest Regressor, as GBRT, is Decision Tree and as the name suggest this model works by training many Decision Tree on random subsets of the features and them calculates the average of these predictions looking to improve result precision and control overfitting. The image below illustrate as this model works:

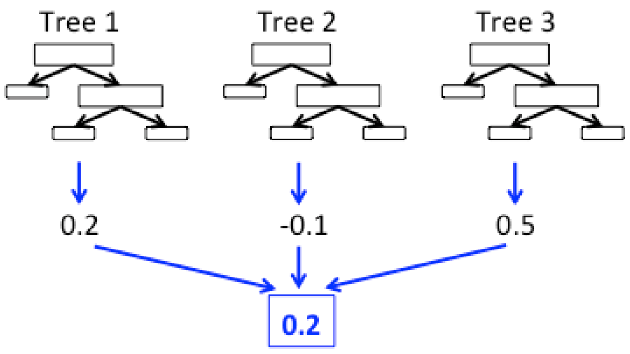


Figure 7- Random Forest Regressor illustration [10]

In this project, I will use this model as meta model (blender) to perform an Ensemble Method called Stacking that is explained forward.

Ensemble Method: Stacking

The idea for stacking is to train an adequate model perform the aggregation of prediction of different predictors. The model that perform this aggregation is called *Blender* or *Meta Learner* and all model used by the blender is called base models. Firstly, all base model that will predict the output need to be trained, see the images below:

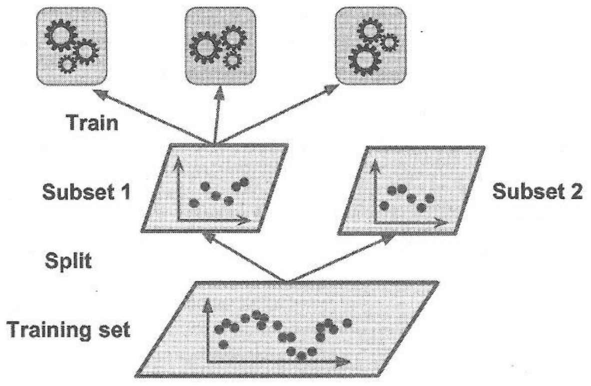


Figure 8- Division of dataset for Stacking Model [8]

Then, these base models perform prediction for new instances and these outputs are used as training set for blender model:

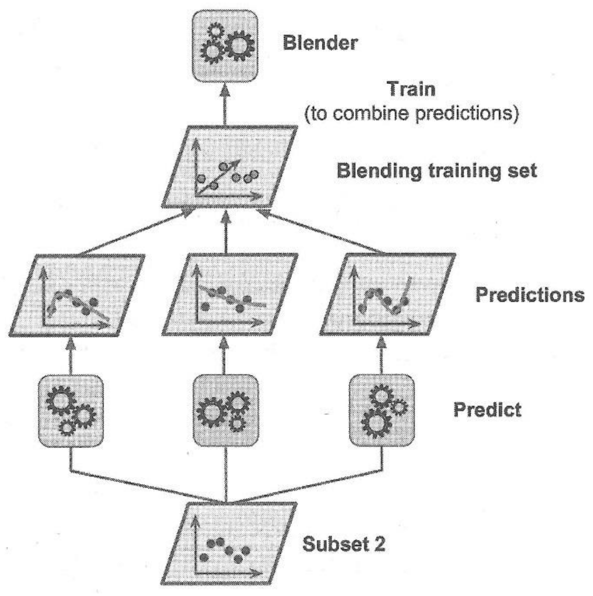


Figure 9- Stacking Method illustration [8]

After that, all models are trained, and the Stacking Method might be applied as the image below showed:

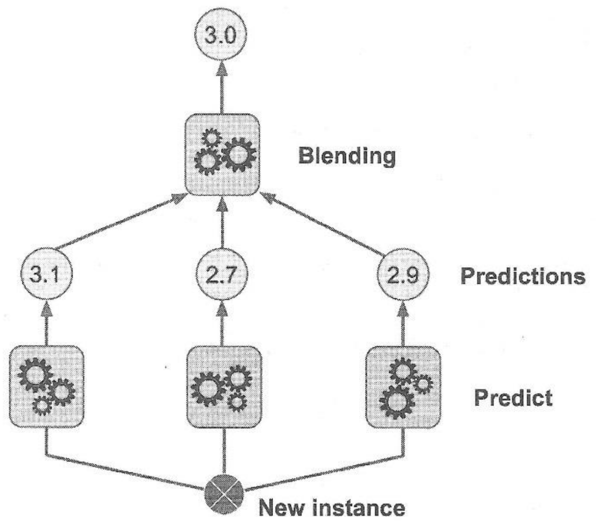


Figure 10- Stacking Model simulation [8]

As the image exhibited, the Blender works with the predictions of base models and that good because different models fail in different situation. So, use Stacking may be useful to prevent distinct types of fail and the Random Forest Regressor Model will be applied as blender on this project.

Benchmark

As this project is based on ***House Prices: Advanced Regression Techniques*** Kaggle competition, there are many submissions with different evaluation metric (Root Mean Squared Error) score. The Public Leaderboard exhibit a list of all submission scores and the first place (Dark Yoshi) in the competition achieved a score metric equal ZERO. For my Benchmark, I will try to reach the score approximately equal 0.14 or less because that’s the median score of this competition and as first competition it’s a challenging goal.

III. METHODOLOGY

Data Preprocessing

Load the Dataset

Firstly, the training and testing dataset were loaded and concatenated in only one, where the target variable of training set and the feature of identification (ID) of training and testing set were saved in separated variables.

Feature Transformation

Based on section Data Exploration, firstly, the type of ‘*MSSubClass*’ was change from numeric (int64) to categorical (object).

After that, the problem of skewness from numeric features was solve. The target variable of training set was transformed using natural logarithm of one plus the input, see:

$$f(SalePrice) = \log_e(1 + Saleprice) \ ,$$

where *Saleprice* is an array of one column.

The other numerical features of concatenated dataset were transformed if the value of skew from library Scipy exhibited a value high than 0.65, where a normal distribution exhibit a value 0.0. The transformation applied was the same, natural logarithm of one plus each numeric feature that need to be treated.

Feature Selection

In this step the objective is to identify which feature that have missing values need to be deleted or filled. If a feature displays a high number of missing values, must be deleted and if the feature has just few points of missing value, might be filled without influence a wrong tendency in the dataset. It’s necessary because many models exhibit problem in dealing with missing values. The minimum number of missing value that was deleted instead be filled was 23. That is a low value based on a total of 2919 samples, but an increase in this threshold (limit) result into a worse final result, in other words, false trends are created.

Therefore, the features removed and filled are:

```
Number of featues deleted:  18
['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'B
smtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Fireplac
eQu', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageQual', 'G
arageCond', 'PoolQC', 'Fence', 'MiscFeature']


Number of featues filled:  16
```

```
['MSZoning', 'Utilities', 'Exterior1st', 'Exterior2nd', 'BsmtFinSF1
', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Electrical', 'BsmtFul
lBath', 'BsmtHalfBath', 'KitchenQual', 'Functional', 'GarageCars',
'GarageArea', 'SaleType']
```

The process of fill the features with small number of missing values if different depending the type of features. Each Categorical feature was completed with their mode and each Numerical feature was completed with their mean. That is a simple and effective way to solve this problem.

Outlier Detection

This step was applied only on training set looking to identify outliers that may disturb the process of models training. The option to don't apply this step on testing set is consistent because is necessary to analyzing the performance of models with noise data points. Thus, in this point the features dataset was split into training and testing set, like originals, but now treated based on previous steps.

The method applied in this step of outlier detection was Tukey Method, where a data point is considered outlier if the numeric value is located below or above 1.5 times the interquartile range (IQR), thus, just numerical features of training set is analyzed in this step. The definition of IQR and Tukey Method illustration are showed below:

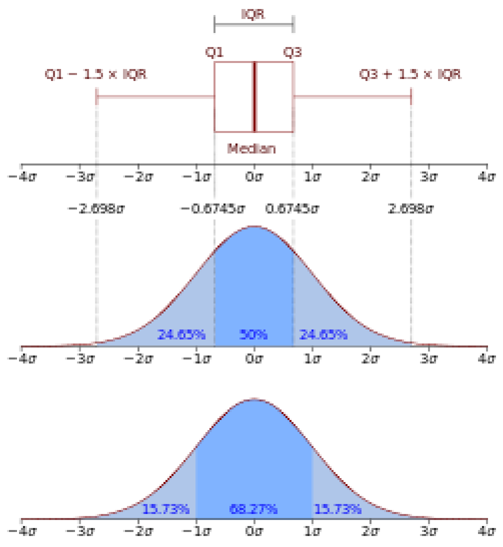


Figure 11- IQR illustration [11]

Wikipedia

As discussed on Data Exploration section, many numerical features exhibit a high percentage of zero values and this fact result in a high tendency of numerical features displayed first quartile (Q1) equal zero. Thereby, all numerical features that have this problem need to identify outliers without consider the zero values.

Each numerical feature applies the Turkey Method for all samples and each one identifies classify samples as outlier. Analyzing the result, anyone sample show up as outlier for all numerical features. Few samples were classified as outlier for more than 5 features and that represents just 15% of all numerical features.

Therefore, ZERO sample was removed because removing a sample based on the few features results in a loss of information instead better training set.

Individual Models Implementation

The initial step before implementation was load and split the preprocessed training data to create a sub test set looking to create a model proof of Overfitting and the remain to be used in true train, that is, part of training samples going to be used to verify if the model generalizes well. If a model using the same samples to train and evaluate, probably the performance metric will be good on the train, but new samples probably will exhibit a bad result, like a model that recorded the prediction for each sample but not learned the pattern of the problem.

Individual Models Test

During the project implementation, one of the hardest parts was decide which model I should use. There are many possibilities to perform this kind of regression and understand the differences between them is so difficult for an Data Scientist with few experience. Thereby, initially I search in the web which is the most used models for similar problems and I fond a large list of individual possibilities and decide to test the performance of them. The models tested were:

- Decision Tree Regressor
- K Neighbors Regressor
- Linear Regressor
- Linear Support Vector Regressor
- Epsilon Support Vector Regressor
- Kernel Ridge Regressor
- Lasso
- Elastic Net Regressor
- Gaussian Process Regressor
- Gradient Boosting Regressor
- Random Forest Regressor

Firstly, each model cited above was create with default parameters and fitted based on training samples (without see the part of training set destined to evaluation). After that, a cross validation method developed with a function called *rmse_CV* was applied for each method using the sub test set to evaluate the performance where the output is 3 scores of RMSE metric, the mean them and the stander deviation of these 3 scores.

Now, each model has a performance metric score and the bests of them were selected to be optimized. The models selected were:

- Kernel Ridge Regressor (Mean RMSE = 0.134)
- Gradient Boosting Regressor (Mean RMSE = 0.138)
- Linear Regressor (Mean RMSE = 0.163)
- Random Forest Regressor (Mean RMSE = 0.174)
- Elastic Net Regressor (Mean RMSE = 0.276)
- Lasso (Mean RMSE = 0.279)

The Lasso and Elastic Net Regressor were not chosen by your performance metric, but because the l2 regularization might improve the power of generalization when these promising models were combined. This hypothesis going to be verify further up. The Random Forest was selected because it will be applied as blender model. The other 4 models were chosen based on their performance metrics.

Promising Models Refinement

The 6 models selected above were fine-tuned looking to find the best hyperparameters combination, that is, that better fit the training points. The manual execution of this process of optimization is slow and longstanding, thus, for this process the method called Grid Search was used individually for each promising model.

Grid Search method might be found on the Scikit-Learn library as the name *GridSearchCV*. Basically, this method automates the search process and use the cross-validation as auxiliary function and using the RMSE as performance metric.

See the Wikipedia definition:

“The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.” (Wikipedia)

Before hyperparameters optimization, these promising models were:

```
KernelRidge(alpha=1, coef0=1, degree=3, gamma=None, kernel='li
near'      ,kernel_params=None)

GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse',
init=None, learning_rate=0.1, loss='ls', max_depth=3, max_feat
ures=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min
_impurity_split=None, min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto'
, random_state=None, subsample=1.0, verbose=0, warm_start=Fals
e)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1,
normalize=False)

RandomForestRegressor(bootstrap=True, criterion='mse', max_de
pth=None, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1
, min_samples_split=2,min_weight_fraction_leaf=0.0, n_estimat
ors=10, n_jobs=-1,oob_score=False, random_state=None, verbose
=0, warm_start=False)


ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_rat
io=0.5,max_iter=1000, normalize=False, positive=False, precom
pute=False, random_state=None, selection='cyclic', tol=0.0001
, warm_start=False)
```

```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=10
00,normalize=False, positive=False, precompute=False, random_
state=None,selection='cyclic', tol=0.0001, warm_start=False)
```

Applying Grid Search, the hyperparameters optimized for each model were:

```
KernelRidge:{'alpha': 1.0, 'degree': 2, 'gamma': 0.001, 'kern
el': 'linear'}

LinearRegression: there in not hyperparameters to optimizatio
n

Lasso: {'alpha': 0.0001, 'fit_intercept': True, 'normalize':
False, 'selection': 'random'}

ElasticNet: {'alpha': 0.0001, 'fit_intercept': True, 'l1_rati
o': 1.0, 'normalize': False, 'selection': 'random'}

GradientBoostingRegressor: {'alpha': 0.0001, 'criterion': 'ms
e', 'loss': 'ls', 'max_depth': 4, 'max_features': 'sqrt'}

RandomForestRegressor: {'criterion': 'mse', 'max_depth': 10,
'max_features': None, 'n_jobs': -1}
```

Based on these results, two important points need to be discuss.

- The optimization of the model Random Forest Regressor can’t be used because it’s a meta learner and the optimization was realized looking to improve the performance as base model.
- The Elastic Net model optimized exhibited the parameter *l1_ratio* equal 1.0, so, just *l1 regularizer* is used and that means that this model will work as the Lasso Regressor. Thereby, Elastic Net won’t be used.

Ensemble Method - Stacking

Apply Stacking is the last step of implementation. The model used as blender (meta learner) was Random Forest Regressor and the hyperparameters optimization was realized manually and just the number of trees in forest was changed to 500.

Blender uses Kernel Ridge Regressor, Gradient Booting Regressor, Linear Regressor and Lasso as base models, that is, as model that will be feed the blender model.

Firstly, as explained on the section about the Ensemble Learning Stacking, is necessary load and split the preprocessed training dataset. The testing dataset loaded just will be used in the end, for final predictions to submission as new sample of market. The training dataset was split, 20% of all samples were saved to be used as a test subset for blender validation. The remaining samples of the training dataset was split again, 35% was randomly selected as sub test set for individual model’s validation and the new remaining was utilized for individual model’s training.

It’s important observe that the training samples for blender will be the prediction of the individual models on your testing sub dataset and that each individual model utilized here was optimized.

After training, the base models were validated using your sub testing dataset. The performance metric of each individual base model was:

```
Kernel Ridge Regressor - RMSE = 0.12021920057347629
Gradient Boosting Regressor - RMSE = 0.12717293023638282
Linear Regressor - RMSE = 0.1258998978204327
Lasso - RMSE = 0.12021439241096132
```

As the performance metric showed, the process of optimization using Grid Search improve significantly the performance metric, all base model improves their metric.

Based on the predictions for this subset, a Data Frame was created with these results of base models and was used as training sample for blender. Looking to validate the blender model the blender was tested. Each base model predicted the target value for the blender test subset and the blender model predicted base on these outputs. So, a validation performance metric was fond:

The performance metric for each individual model in this blender test subset were:

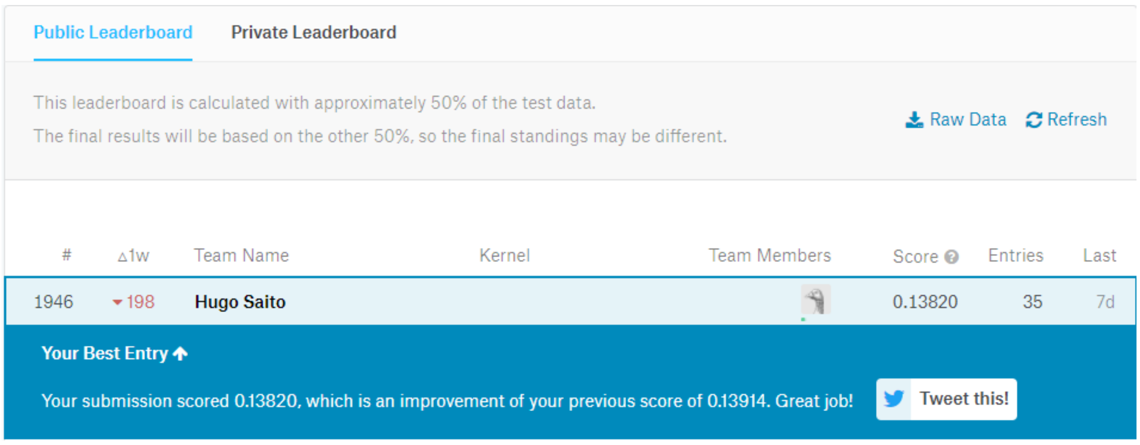
```
Kernel Ridge Regressor - RMSE = 0.12677709821050961
Gradient Boosting Regressor - RMSE = 0.1437172855552762
Linear Regressor - RMSE = 0.1337052011759785
Lasso - RMSE = 0.1262026135176259
```

The blender performance metric was:

```
Blender (Random Forest Regressor) - RMSE = 0.13737408597763429
```

Now, the final learner is ready for analyzing the submission test dataset. All models were trained and validated. Thereby, the testing dataset was applied, and the prediction were funds by the learner. This result was saved and submitted at Kaggle's Competition to be evaluated.

After submission, the score obtained from Kaggle competition was:



This leaderboard is calculated with approximately 50% of the test data.
The final results will be based on the other 50%, so the final standings may be different.

#	△1w	Team Name	Kernel	Team Members	Score	Entries	Last
1946	▾198	Hugo Saito			0.13820	35	7d

Your Best Entry ↑

Your submission scored 0.13820, which is an improvement of your previous score of 0.13914. Great job!

Figure 12- Submission Score, Kaggle

IV. RESULTS

Model Evaluation and Validation

The process to process to reach the better final model was so difficult. Firstly, as described previously, many individual models were tested, and the bests were optimized. Initially, the list of promising models was large, and many different combinations were tried, because is almost impossible to a new data scientist analyses how the models works and determine who will be the best for this problem project without test.

As example for some combinations realized, the model Gradient Booting was used as blender and Random Forest as base model; the Random Forest was used as base model and blender; different type of regressors (Decision Tree Regressor; K Neighbors Regressor; and so forth) were tested even these models aren't the most promising models;

After many different combinations, the best learner was found. The base models Linear Regressor, Kernel Ridge Regressor (where *kernel='linear'*) and Lasso are purely linear models but works without or with different regulators. The other base model (Gradient Booting Regressor) is an ensemble model that applies Decision Tree Regressor, that is, it learns local linear regressions approximating the sine curve. Therefore, as all models used perform linear regression, that indicates a linear correlation between features. The justification for each model used was discuss in the section II Analysis, Algorithms and Techniques.

The blender model chosen was Random Forest Regressor because, as explained previously, this model is excellent to tune the prediction of all base models and control overfitting. Although the performance metric of this blender was not the best score compared with the individual models, the blender is good to avoid errors, because even one of the base models fail, the other three (correct output) hold the final output. So, the final model become much more generalizer than any individual model.

As the *GridSearch* method worked exhaustively searching the best parameters combination (computationally expensive), the parameters combination for each individual model is the combination that result into better result, that is, the best performance metric score. Thereby, all configured parameters are properly defined.

A key point to be emphasize is the validation process of base models and blender. The robustness of each model need to be tested to verify the performance of models in specifies situations, that is, to verify manly two aspects of models:

- How is the training performance when different training dataset are used?
- How is the performance for unseen data points, in other words, the capacity of generalize?

Looking to answer the first question, during the project development, the original training dataset was split into some different train and test dataset. Thereby, the models learned from scratch using samples with various characteristics. The method *train_test_split* from Scikit-Learn library was used, the parameter *random_state* determine which pseudo random split will be executed, that is, which integer number will be used as a seed by random number generator. So, different integer number were used, and many random splits were tested.

As result, the models exhibited similar performance even learned from different data points, thus, the project final model exhibited a great behavior even when different environments were used as base of learning.

The test sub dataset, after original dataset split, were created to be used as unseen data points. The base model and the blender were tested using these sub datasets as previously described. That is a simple and effective way to evaluate the behavior of model after to be trained. As the model never seen the data points, the model sees these sample as new data points, that is, works as a simulation of real world of housing market. Thereby, the models are validated before working with real housing market. As these test points are part of training dataset, there are the true value for *SalePrice* variable (label data). So, it's possible to calculate the score of performance metric and evaluate the behavior of model on a simulating of real world.

Appling the two process described above, the predictions from final model become reliable. It's important emphasize that *reliable* does not means that all prediction will be *correct*, even because as the scores performance metrics never were zero in this project. *Reliable* means that the prediction from the final model have a high probability to be a value where the error is similar to the error of the blender when the test sub dataset from training data was applied. For this project, this score of performance metric (RMSE) of blender was 0.1374.

Justification

The benchmark established earlier is reach a median position in the Public Leaderboard. The final solution reached a score of performance metric equal 0.13820 and this score represents a position 1946 where the total number of teams was 4515. Thereby, the benchmark was reached a position close to the 2257 and this position represents the top 43% of all competitors and teams. As a first competition, that was a good result, but the leaderboards reached a performance metric score equal zero.

The final solution is significant enough to solve the problem of this project, the final performance metric of 0.138 is small because the scale of values of houses is much larger than the scale of the error. The number of this score doesn't represent the difference between the prediction and true value but based on the definition showed previously, it's obvious that this value is very small.

V. CONCLUSION

Free-Form Visualization

The most important quality about the project implementation is the ensemble method applied. The idea to use many different base models and a blender become the learner much a prove of errors than a single model. Below, see one more image that represent this method:

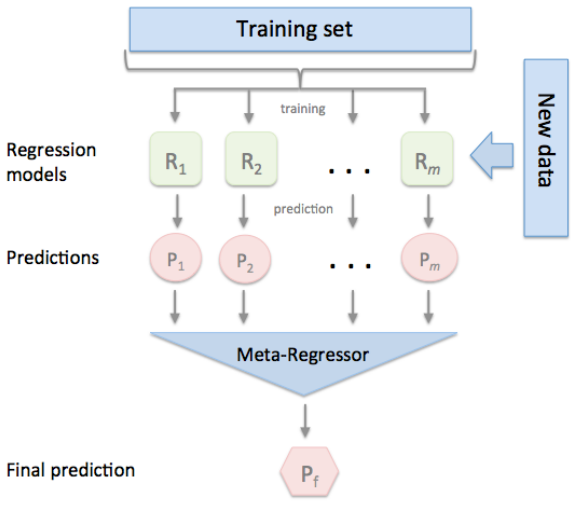


Figure 13- Stacking Regressor [12]

The base models used on final solution of this project are similar, but fail in different situation, different samples. When a base model (R_i) fails, the remaining base models probably won't fail and thereby the blender model receive more great predictions than worse. Consequently, the Meta-Regressor (Random Forest Regressor) will have more bias to output a great final prediction.

Therefore, the biggest challenge for Stacking Regression is determine what is the best combination of base model and which is the best Meta-Regressor for a specific situation and problem.

The idea of a Stacking model discussed above has the capacity of handling with several diverse types of problem. A unique model can solve a problem, but there will almost be always weaknesses, the goal is to find other model that works fine with these weaknesses trying to match them.

Improvement

Some parts of the project development might be changed, and better predictions can be found (or not).

In the step of filling the missing values, instead use mean and mode, is possible apply the models K Neighbors Regressor and Classifier to fill the numeric and categorical missing values, respectively. Using these models, each sample that needs filling any feature will be filled base on the K (integer number) similar samples, therefore, each sample will be analyzed individually. In this way, the filled ones are more likely to be the correct value.

About base model's selection, might be an improvement create a function to identify the biggest predictions errors. Identify the most problematics samples help to understand when the models are failing, consequently, it's might be useful in the process of model selection.

Looking to perform a more thorough refinement of base predictions, a multilayer stacking ensemble might be applied. The idea is basically applying stacking more than once, it is possible train several different blenders and create a layer of blender. Now, there is not a group called base models and a meta-learner, each group of models is called as layer's. The image below illustrates the process:

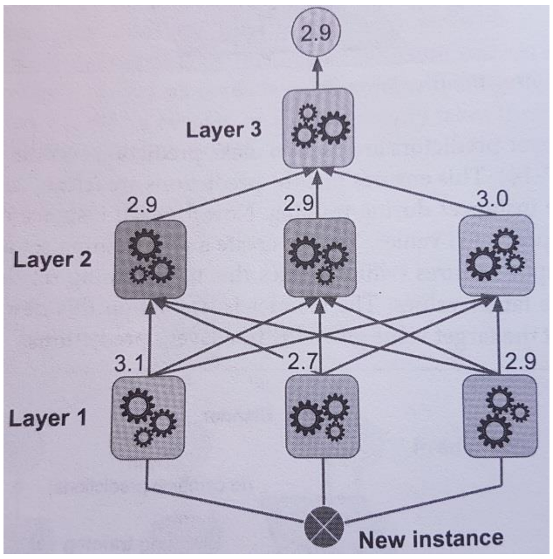


Figure 14- Multilayer Stacking Ensemble [8]

As the image showed, the new instances feed the first layer and the outputs feeds the next layer sequentially. In this way, the final output might be better than just one layer of stacking. It’s important observe that, after first layer, each model of a layer receive the predictions of all models of previous layer.

References

[1] Kaggle Competition - House Prices: Advanced Regression Techniques

[2] Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project

[3] Ames Housing Dataset Description

[4] Kaggle Metric Evaluation

[5] Root Mean Square Error - Wikipedia

[6] Ensemble Learning - Wikipedia

[7] Ensemble Methods – Scikit Learn

[8] Hands-On Machine Learning with Scikit-Learn and TensorFlow

[9] Kernel Trick – Stack Exchane

[10] Random Forest - databricks

[11] IQR – Wiipedia

[12] Stacking Regressor - mlxtend