



SAPIENZA  
UNIVERSITÀ DI ROMA

# Sapienza Università di Roma

## Machine Learning Course

**Prof: Paola Velardi**

**House Price Predicting: Advanced Regression Techniques**

**Students:**

Anxhelo Diko [1867167]

Sidorela Mema [1853933]

## Contents

Tabel of Figures.....	3
1.Introduction.....	4
1.1 Problem Description.....	4
1.2 Data set.....	4
1.3 Tools.....	5
1.4 Linear Regression (short definition).....	6
1.5 Approach and methodology.....	6
2. Preprocessing.....	6
2.1 Outliers.....	7
2.2 Target Variable (Sale Price).....	8
2.3 Missing values.....	9
2.4 Creating new variables.....	10
2.5 Binning values.....	12
2.6 Label encoding.....	12
2.7 Normalizarion.....	12
3. Models and hyperparameters.....	13
3.1 Introduction to regularization.....	13
3.2 Lasso regression.....	13
3.3 Elastic Net.....	14
3.4 Random Forrest.....	14
3.5 Gradient boosting.....	14
3.6 Stacked Regression.....	15
3.7 Hyperparameter tuning.....	16
4. Model evaluation and selection.....	17
4.1 Cross Validation.....	17
4.2 Learning Curves.....	18
4.3 RMSE.....	18
4.4 Our models Evaluation.....	18
5. Prediction and final result.....	23
6. Conclusion.....	24
7. References.....	24

## Tabel of Figures

Figure 1: Simple Linear Regresion.....	6
Figure 2: Hadling outliers for Garage Area variable.....	7
Figure 3: Garage Area plot with Sale Price.....	7
Figure 4: Garage Area Sale Price plot after removing outliers.....	8
Figure 5: Sale Price statistics.....	8
Figure 6: Sale Price before and after log-transformation using numpy.....	9
Figure 7:Missing values ratio for each variable.....	10
Figure 8:The relation of Year Built and Year Remod Add with Sale Pprice.....	11
Figure 9:Remod _Diff relation with Sale Price.....	11
Figure 10: Garage quality before and after encoding (manually).....	12
Figure 11: Stacking ensambling method.....	15
Figure 12:Grid search function implemented in python.....	16
Figure 13:Elastic net hyperparameter selected by the grid search.....	17
Figure 14:Gradient boosting metrics from Grid Search.....	19
Figure 15:Lasso metrics/results from Grid Search.....	19
Figure 16:Random forest metrics/results from Grid Search.....	19
Figure 17:Elastic Net metrics/result from Grid Search.....	20
Figure 18:Stacked Regresor RMSE score.....	20
Figure 19:Learning curve of Lasso.....	21
Figure 20:Elastic Net curve.....	21
Figure 21:Gadient Boosting leaning curve.....	22
Figure 22:Random Forest Learning Curve.....	22
Figure 23:Stacked Regressor Learning Curve.....	23
Figure 24:Predicting with our Stacking modeling.....	23
Figure 25:Score on kaggle.....	24

# 1.Introduction

## 1.1 Problem Description

House Price Prediction is a very common task in statistics and very popular in Data Science and Machine Learning world. We got this task from a kaggle competition(House price prediction: Advanced regression techniques). The problem is to build a model that will predict house prices with a high degree of predictive accuracy given the available data. With 79 explanatory variables we are challenged to build a regression model using advanced techniques.

Regression is a widely used predictive analysis which consists in a set of statistical processes for estimating the relationship among variables. This is one of most widely used statistical model and has a wide range of application in a lot of areas.

In this project we will try to apply regression predictive analysis. We will approach the problem by trying different algorithms based on regression and find which one performs better on this task. The main idea is to emphasize the difference between stacked ensambling method and other methods of applications plus the importance of hyperparameter tuning and the metric used based on the goal.

## 1.2 Data set

We are using a data set which describes the sale of individual residential property in Ames, Iowa from 2006 to 2010. The data set contains 3970 observations and a large number of explanatory variables (23 nominal, 23 ordinal, 14 discrete, and 20 continuous) involved in assessing home values. The variables focus on the quality and quantity of many physical attributes of the property. Most of the variables are exactly the type of information that a typical home buyer would want to know about a potential property.

The data set seems small but is presented with a lot of problems which needs a lot of feature engineering (preprocessing and cleaning) work to make it ready and to extract the best features for our goal.

Some of the variables we can find on the dataset:

- **SalePrice**- the property's sale price in dollars. This is the target variable that you're trying to predict.
- **MSSubClass**: The building class
- **MSZoning**: The general zoning classification
- **LotFrontage**: Linear feet of street connected to property
- **LotArea**: Lot size in square feet
- **Street**: Type of road access
- **Alley**: Type of alley access
- **LotShape**: General shape of property
- **LandContour**: Flatness of the property
- **Utilities**: Type of utilities available
- **LotConfig**: Lot configuration
- **LandSlope**: Slope of property
- **Neighborhood**: Physical locations within Ames city limits
- **Condition1**: Proximity to main road or railroad
- **Condition2**: Proximity to main road or railroad (if a second is present)

- **BldgType**: Type of dwelling

*We got this dataset from kaggle*

### 1.3 Tools

Package management tool: **Anaconda**, it is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.)

Language: Python 3.7 implemented on Anaconda, a high-level, general purpose OO language.

Analysis tool: **Regression analysis**, it is a form of predictive modeling technique which investigates the relationship between a dependent and independent variable. While there are many types of regression analysis, at their core they all examine the influence of one or more independent variables on a dependent variable. The process of performing a regression allows you to confidently determine which factors matter most, which factors can be ignored, and how these factors influence each other.

Main Libraries:

1) **Scikit-learn**: It is an open source machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machine, random forests, gradient boosting, k-mean and is designed to inter-operate with the Python numerical and scientific libraries **NumPy** and **SciPy**.

2) **Pandas**: It is a software library written for the Python programming language for data manipulation and analysis. It's a useful and easy tool for reading and writing data between data structures and different formats: CSV and text files, Microsoft Excel and SQL databases. We use pandas to interact with data.

3) **Numpy**: Numeric Python is a Python package that among others provides an alternative to the regular Python list, Numpy array. This allow us to analyze data much more efficiently. It performs calculation over the entire array it is easy and fast.

4) **Matplotlib**: It is a plotting library for the Python programming language. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy.

5) **Seaborn**: Statistic data visualization is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

6) **Scipy**: Scientific Python is an open source Python library used for scientific and technical computing. Scipy uses NumPy for more mathematical functions.

IDE/Environment: JupyterLab, it is the next-generation web-based user interface for Project Jupyter Notebook. Its documents are both human-readable documents containing the analysis description and the results (figures, tables, etc..) as well as executable documents which can be run to perform data analysis.

## 1.4 Linear Regression (short definition)

Linear regression is used to predict the value of a continuous variable  $Y$  based on one or more input predictor variables  $X$ . The aim is to establish a mathematical formula between the response variable ( $Y$ ) and the predictor variables ( $X$ s). You can use this formula to predict  $Y$ , when only  $X$  values are known.

**This mathematical equation can be generalised as follows:**

$$Y = \beta_1 + \beta_2 X + \epsilon \quad (\text{simple linear regression formula})$$

where,  $\beta_1$  is the intercept and  $\beta_2$  is the slope.

Collectively, they are called regression coefficients and  $\epsilon$  is the error term, the part of  $Y$  the regression model is unable to explain.

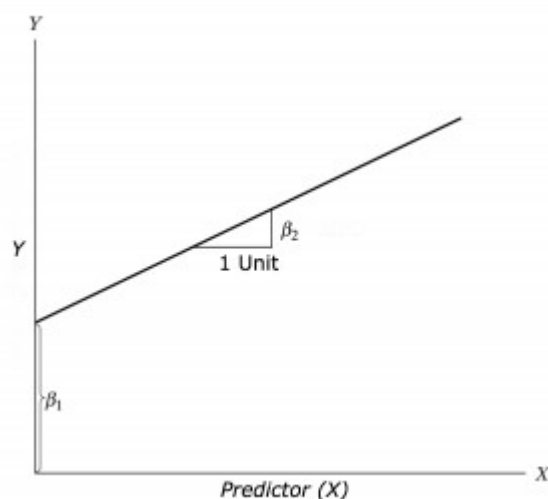


Figure 1: Simple Linear Regression

## 1.5 Approach and methodology

Before anything else we tried to understand the problem, this way we would have a better idea on how to work with data and make the right decision. We then tried to separate this work in three essential parts: Feature engineering, modeling and hyperparameter tuning. Both three parts plus the validation are core elements of every single machine learning project. In the end we chose the best model by evaluating their generalization ability using the kaggle score (based on RMSE).

## 2. Preprocessing

Everything starts when a problem is emphasized and every problem is associated with data. For us the problem is ready, house price prediction, even the dataset is ready, kaggle provided the iowa dataset but this is not enough to go straight to model building. Ready to use data would be the

dream of every machine learning and data science professional but that does not happen. Data have problems, have noise, have missing values, can be imbalanced, can have a non appropriate distribution, etc. This section is going to treat most of these aspects for our dataset.

## 2.1 Outliers

An outlier is an observation point that is distant from other observations. They can occur by chance in any distribution, but they often indicate either measurement error or that the population has a heavy-tailed distribution. Outliers are a very difficult thing to handle, you can drop those data because they affect your model or not to drop them because you lose information. Our models will all be based on linear regression, and linear regression uses statistics like mean, standard deviation and correlation. These statistics are very sensitive to outliers so we have a good reason to remove them from our data.

**Note:** *Removing outliers is not always the way to go. If our prediction data contains outliers too then our model will perform very badly on predicting these instances.*

To find the outliers we will follow the IOWA data set instructions about outliers. The first variable to take in consideration will be Garage Area. To do this, we first plot the variable in relation with our target variable, Sale Price, using matplotlib and seaborn libraries.

```
#Before joining the two dataset for feature engineering, we are going to remove some outliers from the trainset
# Something that can not be done with the testset
plt.subplots(figsize=(20,5))
plt.subplot(1,2,1)
plt.plot(dataset["GarageArea"], dataset["SalePrice"], "ro")
plt.ylabel('Sale Price')
plt.xlabel('GarageArea')
plt.savefig('./Garage_me_outlier')
mean = dataset["GarageArea"].mean()
stan_d = dataset["GarageArea"].std()
outlier_gr = dataset[dataset["GarageArea"] > mean + 3*stan_d]
outlier_gr = outlier_gr.append(dataset[dataset["GarageArea"] < mean - 3*stan_d])
dataset = dataset.drop(outlier_gr.index)
#after removing the outlier
plt.subplot(1,2,2)
plt.plot(dataset["GarageArea"], dataset["SalePrice"], "ro")
plt.ylabel('Sale Price')
plt.xlabel('GarageArea')
plt.savefig('./Garage_pa_outlier')
```

Figure 2: Handling outliers for Garage Area variable

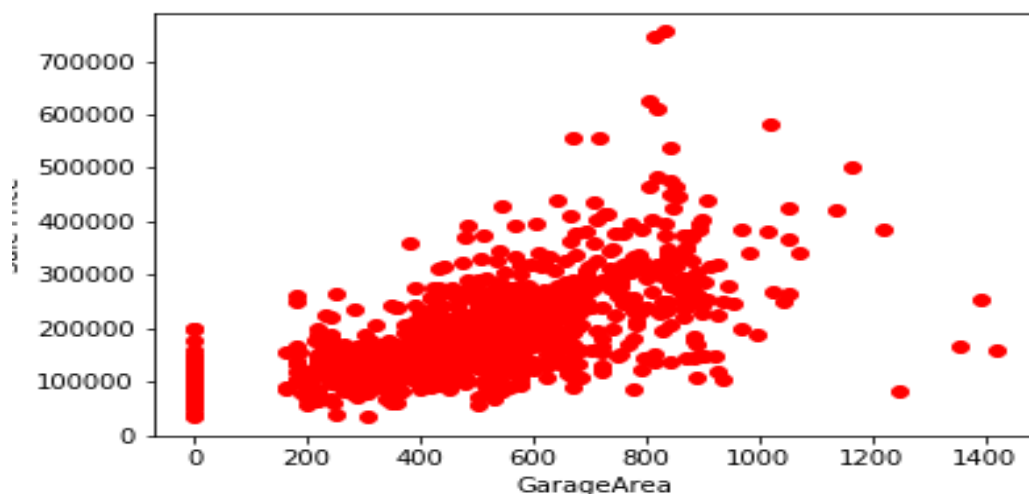


Figure 3: Garage Area plot with Sale Price

From the picture we can easily identify that Garage Area contains outliers. To remove them we have used a statistical approach, dropping values that are outside the interval  $[\text{mean} - 3 \cdot \text{std}; \text{mean} + 3 \cdot \text{std}]$ . So the number of data we drop will be very small and probably only outliers.

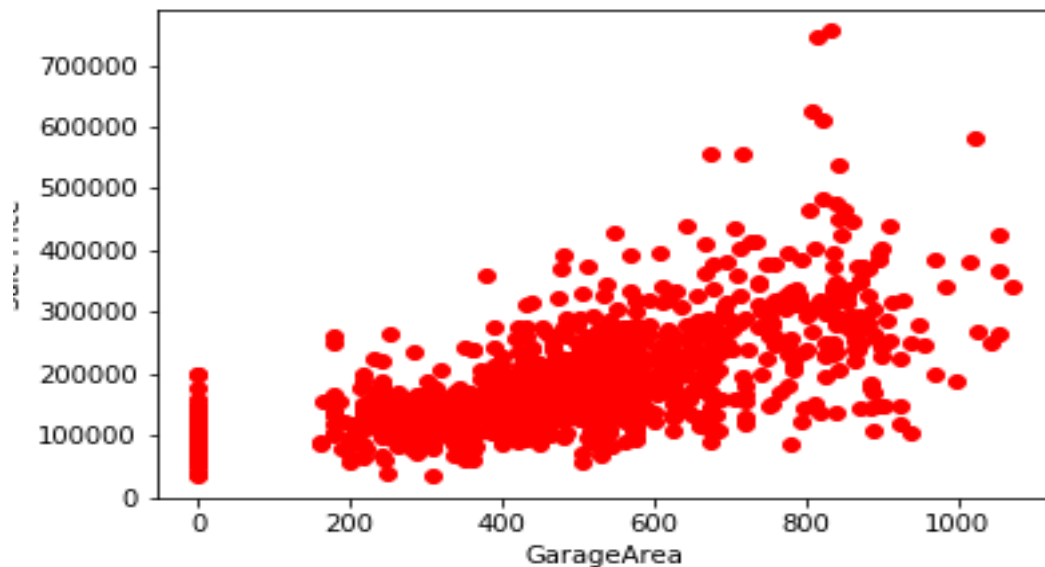


Figure 4: Garage Area Sale Price plot after removing outliers

We did the same procedure even with some other variables like GrLivArea, 1stFlrSF, OverallQual, etc.

## 2.2 Target Variable (Sale Price)

Sale Price is our target variable, our Y and analysing it is very important. This analysis will show its statistical metrics and distribution. Pandas library offers a function called describe which gives information about the selected variable, in our case it is Sale Price.

```
[10]: dataset.SalePrice.describe()  
#mean and std are very high, we are going to see the skewnes of the sale price dist.
```

	count	mean	std	min	25%	50%	75%	max
count	1417.000000							
mean		176845.394495						
std			70313.417561					
min				34900.000000				
25%					129500.000000			
50%						160200.000000		
75%							210000.000000	
max								582933.000000

Name: SalePrice, dtype: float64

Figure 5: Sale Price statistics



As we see from the data statistics, standard deviation is too high and if we plot the distribution we notice that the distribution is skewed, right skewed. This is not a mathematically convenient representation of the data so we need to take some action to make it more convenient because linear models work better with normally distributed data. We are going to use log transformation of data, this is the most used technique to transform skewed distributions into approximately normal distributions.. We will perform this technique to scale our feature so it will have approximately standard normal distribution properties,  $\mu=0$  and  $\sigma=1$ .

```
Skewness before applying log : 1.316316  
Skewness after applying log : 0.049842
```

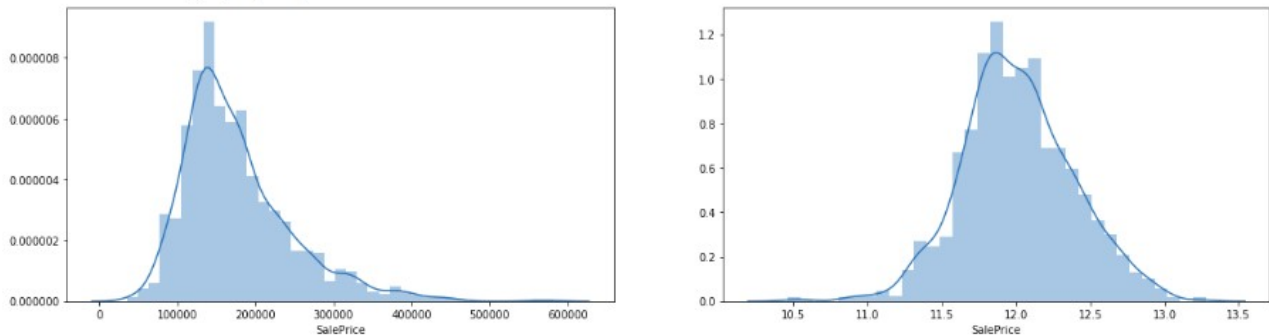


Figure 6: Sale Price before and after log-transformation using numpy

## 2.3 Missing values

A missing value is an entry in a column that has no assigned value. Handling them in the right way is crucial for the model performance and generalization. A missing value can mean multiple things:

- A missing value may be the result of an error during the production of the data set. This could be a human error, or machinery error depending on where the data comes from.
- A missing value in some cases, may just mean that a 'zero' should be present. In which case, it can be replaced by a 0. The data description provided helps to address situations like these.
- However, missing values represent no information. Therefore, does the fact that you don't know what value to assign an entry, mean that filling it with a 'zero' is always a good fit?

Some algorithms do not like missing values. Some are capable of handling them, but others are not. Therefore since we are using a variety of algorithms, it's best to treat them in an appropriate way.

If you have missing values, you have three options:

- Delete the entire row
- Fill the missing entry with an imputed value
- Or depending on the frequency of the missing value for some specific variable you can drop the column

Choosing between the three options may be a little bit tricky. Deleting an entire row may cause loss of information, especially if the data set is not so big, like our data set. Even dropping the column is dangerous, because the column might be very important for the model.

There is a lot of space for discussing how and what to choose when handling the missing values, there is no best practice for every situation. However we handled it using replacement and column drop the following way:

- 1) If one column had more than 80% missing values, we chose to drop it
- 2) If the missing value had a meaning 0 in the variable values we replaced it with 0 or No
- 3) In continuous variable where 0 had no meaning we replaced the missing value with the mean
- 4) In categorical variables where 0 had no meaning we replaced the value with the mode.

This was the path we chose but there are a lot of other techniques like for example using a function to predict the missing value based on the other ones, etc.

```
[15]: #Will deal with missing values before going on with other feature modification
Missing = pd.DataFrame({'Missing Values':data_union.isna().sum(), 'Missing Ratio':((data_union.isnull()).sum()
Missing.sort_values(['Missing Ratio'], ascending=False).head(20)
# i am going to drop data with missing ratio more than 80%
```

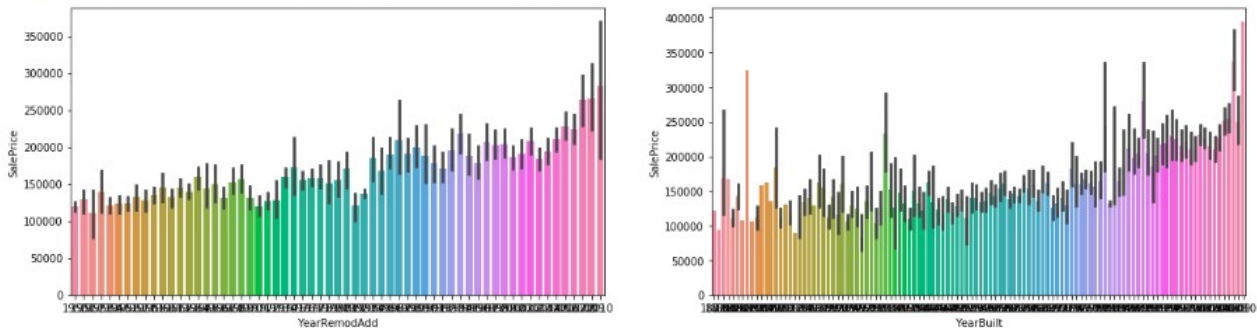
```
[15]:
```

	Missing Values	Missing Ratio
PoolQC	2869	99.756606
MiscFeature	2772	96.383866
Alley	2679	93.150209
Fence	2312	80.389430
FireplaceQu	1414	49.165508
LotFrontage	477	16.585535
GarageYrBlt	154	5.354659
GarageFinish	154	5.354659
GarageQual	154	5.354659
GarageCond	154	5.354659
GarageType	152	5.285118
BsmtExposure	81	2.816412
BsmtCond	81	2.816412
BsmtQual	80	2.781641
BsmtFinTpe2	78	2.712100

*Figure 7:Missing values ratio for each variable*

## 2.4 Creating new variables

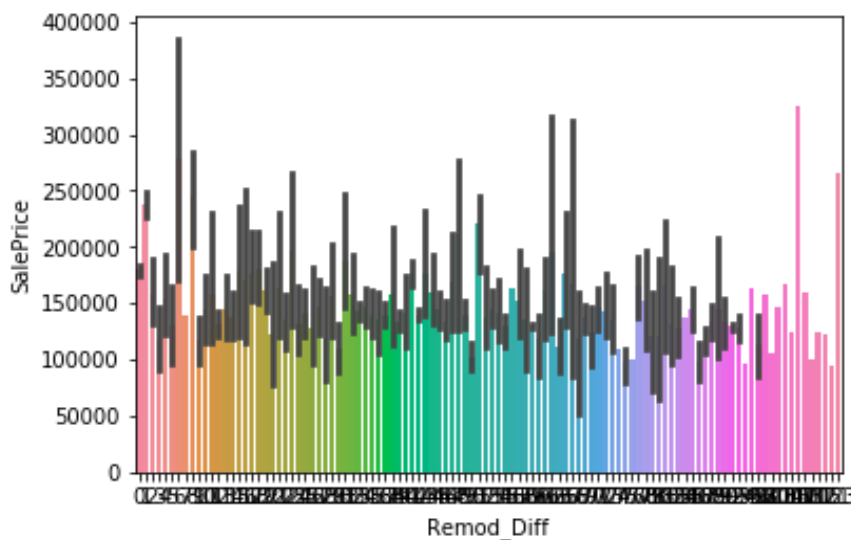
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd91d0070f0>
```



*Figure 9: The relation of Year Built and Year Remod Add with Sale Pprice*

In data science some processes are considered even art sometimes, one of these cases is creating new variables. To create new variables the data scientist or machine learning engineer must be creative and mix creativity with the analysis to try to create new variables which may be useful to the model. We tried to create some variables here on our model. Lets take two variables that our dataset has, the YearRemodAdd (the year when the house was remodeled) and YearBuilt (the year when the house was actually built). Studying these two variables we understand that they have a positive relation with the target variable. It is easily noticed from the graphics. and

Watching these relation we decided to create a new variable from these two, the difference between them and we named it Remod\_Diff.



*Figure 9: Remod\_Diff relation with Sale Price*

This is not the only variable we created, there are even others for example the total number of rooms, the total number of bathrooms, etc.

## 2.5 Binning values

Statistical data binning is a way to group a number of more or less continuous values into a smaller number of "bins". So basically is a way of grouping data. We used this technique unintentionally, we wanted to create a variable named season which shows the season when the house is sold, for example sold in summer, or sold in spring based on the month of sale. Then when created this variable we understood that it actually was a data binning, we binned the variable MoSold and named it season.

## 2.6 Label encoding

A lot of variables are represented as strings in our dataset (categorical and ordinal variables), but the machine learning algorithms learn only with numbers. So the input that we will feed to our model must be number. To transform the data from string to number we use LabelEncoder and One\_hot\_encoding from the scikit-learn. The LabelEncoder differs from One\_hot\_encoding in the way they replace the values with numbers. The LabelEncoder replace the values with integers meanwhile one\_hot\_encoding create n-1 variables (n is the number of possible values of the variable) and assign 1 to the actual value and 0 to the other ones, so it does a binarization of each category. Well, these two methods are not always so useful, in our dataset we have some ordinal variables, variables where the order has a meaning and doing the encoding with these methods (ready to use library methods) is not a good idea because we may lose important information so we did even manual label encoding. For example GarageQual (garage quality) is a variable which is ordinal and needed to be manually encoded. The original values were No (no garage) , Po (poor) , Fa (fair) , Ta (Typical/Average) , Gd (good) and Ex (excellent), doing the encoding with scikit-learn the values to the variable were assigned randomly but here the order matters so it should be No:0, Po:1, Fa:2, TA:3, Gd:4 and Ex:5.

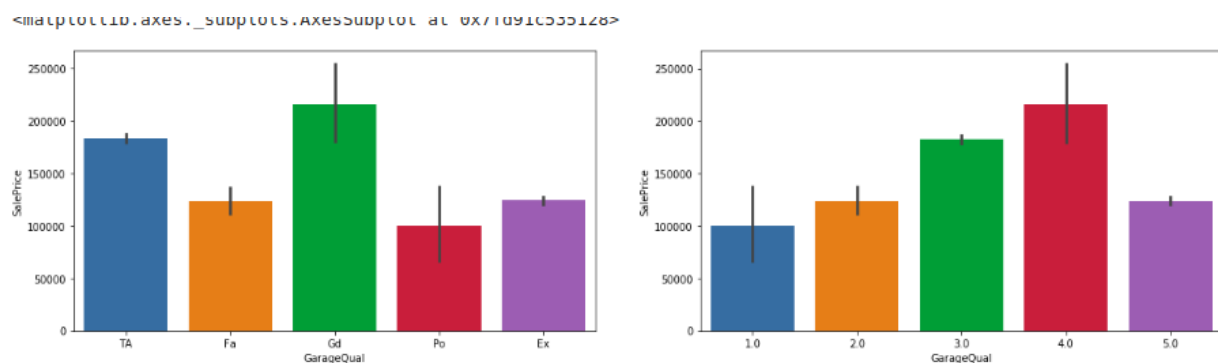


Figure 10: Garage quality before and after encoding (manually)

Note: Some variable represented as integers were actually categorical variables so we transformed them to strings and then one hot encoded them so this way we avoid that the model learns them as quantities.

## 2.7 Normalization

Because we are not using the OLS regression, where normalization is irrelevant, we can normalize our data. For the target variable we used  $\text{np.log1p}$  but this time is different, in our dataset there exist data with negative skew and  $\text{np.log1p}$  is not appropriate for negative skews. So to make the change of basis (log) we used  $\text{boxcox1p}$  from scientific python library (scipy). We applied this change only to variables which has skew more than 0.5 in absolute value.

### 3. Models and hyperparameters

The challenge is about regression techniques so our models will be regression based models. The ones that we choose are Lasso , ElasticNet, Random Forest, Gradient Boosting and a stacking ensemble methods self implemented made by the union of these scikit-learn ready to use regression algorithms plus two other gradient boosting implementation called XGBoost and LightGBM.

#### 3.1 Introduction to regularization

Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better, This can help in improving the accuracy of your regression models, it improves the model's performance on the unseen data as well. Regularization in Machine Learning is an important concept that it helps solving the overfitting problem.

This is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.

A simple relation for linear regression looks like this. Here Y represents the learned relation and  $\beta$  represents the coefficient estimates for different variables or predictors(X).

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

$$RSS = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 .$$

Now, this will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero. Regularization, significantly reduces the variance of the model, without substantial increase in its bias

#### 3.2 Lasso regression

In statistics and machine learning, **lasso (least absolute shrinkage and selection operator;** also Lasso or LASSO) 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. Its formula is:

$$L_{lasso}(\hat{\beta}) = \sum_{i=1}^n (y_i - x_i' \hat{\beta})^2 + \lambda \sum_{j=1}^m |\hat{\beta}_j| .$$

So from the formula we can easily say that it is RSS + a penalty of  $|\beta_j|$ . In statistics this technique is known as L1 norm. So it applies the L1 norm to regularization for further improvement of the prediction.

### 3.3 Elastic Net

**Elastic net regression** is a combination of Ridge regression and lasso. It is especially good at dealing with situation when there are correlation between parameters. This is because Lasso Regression tends to pick just one of the correlated terms and eliminates the others, whereas Ridge Regression tends to shrink all of the parameters for the correlated variables together. By combining this two Elastic net regression group and shrinks the parameters associated with the correlated variables and leaves them in equation or removes them all at once. The solution is to combine the penalties of ridge regression and lasso to get the best of both Elastic Net aims at minimizing the following loss function:

$$L_{enet}(\hat{\beta}) = \frac{\sum_{i=1}^n (y_i - x_i' \hat{\beta})^2}{2n} + \lambda \left( \frac{1-\alpha}{2} \sum_{j=1}^m \hat{\beta}_j^2 + \alpha \sum_{j=1}^m |\hat{\beta}_j| \right),$$

where  $\alpha$  is the mixing parameter between ridge ( $\alpha = 0$ ) and lasso ( $\alpha = 1$ ).

### 3.4 Random Forrest

**Random Forest** is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and Bootstrap Aggregation, or bagging. Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement. Random forest will handle missing values and maintain accuracy for missing values, won't overfit the model and handle large data sets.

The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base models. More formally we can write this class of models as:

$$a(x) = b_0(x) + b_1(x) + b_2(x) + \dots$$

where the final model  $a$  is the sum of simple base models  $b_i$ . Here, each base classifier is a simple decision tree. In random forests, all the base models are constructed independently using a different sub-sample of the data. The random forest model is very good at handling tabular data with numerical features, or categorical features with fewer than hundreds of categories. Unlike linear models, random forests are able to capture non-linear interaction between the features and the target.

### 3.5 Gradient boosting

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

Gradient boosting involves three elements:

A loss function to be optimized.

A weak learner to make predictions.

An additive model to add weak learners to minimize the loss function.

### 1. Loss Function

The loss function used depends on the type of problem being solved. It must be differentiable.

### 2. Weak Learner

Decision trees are used as the weak learner in gradient boosting.

### 3. Additive Model

Trees are added one at a time, and existing trees in the model are not changed.

## 3.6 Stacked Regression

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor (2-layerd stacking). The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features. The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous. The algorithm below summarizes stacking.

Algorithm	Stacking
1:	Input: training data $D = \{x_i, y_i\}_{i=1}^m$
2:	Output: ensemble classifier $H$
3:	<i>Step 1: learn base-level classifiers</i>
4:	<b>for</b> $t = 1$ to $T$ <b>do</b>
5:	learn $h_t$ based on $D$
6:	<b>end for</b>
7:	<i>Step 2: construct new data set of predictions</i>
8:	<b>for</b> $i = 1$ to $m$ <b>do</b>
9:	$D_h = \{x'_i, y_i\}$ , where $x'_i = \{h_1(x_i), \dots, h_T(x_i)\}$
10:	<b>end for</b>
11:	<i>Step 3: learn a meta-classifier</i>
12:	learn $H$ based on $D_h$
13:	return $H$

Figure 11: Stacking ensambling method

Our stacking model follows the same logic. The base level classifiers we choose are ElasticNet, Random Forrest, Gradient boosting, XGBoost (XGboost is a very fast, scalable implementation of gradient boosting) and LightGBM (Light GBM is a gradient boosting framework that uses tree based learning algorithm) and as a meta-classifier (the second layer of the stack) we choose Lasso (it was the one with the better performance as a meta-classifier).

## 3.7 Hyperparameter tuning.

In machine learning, **hyperparameter optimization** or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is



used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned. Our models has a lot of hyperparameters, lets take for example Elastic Net, some hyperparameters of this model are alpha (the constant that multiplies the penalty terms), l1\_ratio (the elastic net mixing parameter, mixes penalties applied from L1 and L2 norm), etc. This parameters and other needs to be fine tuned so we used the grid search method to tune them.

Grid search is a traditional way to perform hyperparameter optimization. It works by searching exhaustively through a specified subset of hyperparameters. Using sklearn's GridSearchCV, we first define our grid of parameters to search over and then run the grid search.

```
def train_model(model, param_grid=[], X=[], y=[],
                splits=10, repeats=10):
    rkfold = RepeatedKFold(n_splits=splits, n_repeats=repeats)

    if len(param_grid)>0:
        gsearch = GridSearchCV(model, param_grid, cv=rkfold,
                               scoring=rmse_scorer,
                               verbose=1, return_train_score=True)

        gsearch.fit(new_dataset.values, SalePrice.values)
        model = gsearch.best_estimator_
        best_idx = gsearch.best_index_
        grid_results = pd.DataFrame(gsearch.cv_results_)
        cv_mean = abs(grid_results.loc[best_idx, 'mean_test_score'])
        cv_std = grid_results.loc[best_idx, 'std_test_score']

    # no grid search, just cross-val score for given model
    else:
        grid_results = []
        cv_results = cross_val_score(model, new_dataset.values, SalePrice.values, scoring=rmse_scorer, cv=rkfold)
        cv_mean = abs(np.mean(cv_results))
        cv_std = np.std(cv_results)

    # combine mean and std cv-score in to a pandas series
    cv_score = pd.Series({'mean':cv_mean, 'std':cv_std})

    # predict y using the fitted model
    y_pred = model.predict(new_dataset.values)
```

Figure 12: Grid search function implemented in python

The benefit of grid search is that it is guaranteed to find the optimal combination of parameters supplied. The drawback is that it can be very time consuming and computationally expensive. We applied this function to all of our models and used the hyper parameters for the final model used for predicting.

```
model = 'ElasticNet'
opt_models[model] = ElasticNet()

param_grid = {'alpha': np.arange(1e-4, 1e-3, 1e-4),
              'l1_ratio': np.arange(0.1, 1.0, 0.1),
              'max_iter': [1000]}

opt_models[model], cv_score, grid_results = train_model(opt_models[model], param_grid=param_grid,
                                                         splits=splits, repeats=2)

cv_score.name = model
score_models = score_models.append(cv_score)
```

Fitting 10 folds for each of 81 candidates, totalling 810 fits

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

```
ElasticNet(alpha=0.0009000000000000001, copy_X=True, fit_intercept=True,
           l1_ratio=0.5, max_iter=1000, normalize=False, positive=False,
           precompute=False, random_state=None, selection='cyclic', tol=0.0001,
           warm_start=False)
```

score= 0.9319255839805168

rmse= 0.09828193595693133

cross\_val: mean= 0.1113056277990968 , std= 0.010693494258527506

[Parallel(n\_jobs=1)]: Done 810 out of 810 | elapsed: 1.3min finished



*Figure 13:Elastic net hyperparameter selected by the grid search*

## 4. Model evaluation and selection.

Normally for evaluating a model in machine learning is needed a dataset for evaluation. In our case we do not have a specific dataset for evaluation, it is kept secret for the challenge and evaluates the model when you submit the prediction on kaggle. However there are other ways of evaluating the model and we gonna stick with cross validation, comparing the evaluation metric/s on training data and on cross validation evaluation. The base evaluation metric we are going to use is the RMSE as stated from the competition and is the metric that evaluates our model prediction on kaggle platform and we are going to show even the R-square metric when we build the learning curves.

### 4.1 Cross Validation

**Cross-validation** is a statistical method used to estimate the skill of machine learning models. It is commonly used to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and result in estimates that generally have a lower bias than other methods.

The cross-validation technique that we are going to use is the K-fold cross-validation. Removing a part of data for validation poses a problem of underfitting. By reducing the training data, we risk losing important trends in data set, which in turn increases error, bias. K-Fold cross validation is a method that provides sample data for training the model and also leaves sample data for validation. In K-Fold cross validation, the data is divided into k subsets. The method is repeated k times, such that each time, one of the k subsets is used as the test set/ validation set and the other k-1 subsets are put together to form a training set. The error estimation is averaged over all k trials to get total effectiveness of our model.

### 4.2 Learning Curves

**Learning curves** are plots that show changes in learning performance over time in terms of experience. Learning curves of model performance on the train and validation datasets can be used to diagnose an underfit, overfit, or well-fit model.

#### Types of learning curves

- Bad Learning Curve: High Bias
  - When training and testing errors converge and are high
    - No matter how much data we feed the model, the model cannot represent the underlying relationship and has high systematic errors
    - Poor fit
    - Poor generalization
- Bad Learning Curve: High Variance
  - When there is a large gap between the errors
    - Require data to improve
    - Can simplify the model with fewer or less complex features

- Ideal Learning Curve
  - Model that generalizes to new data
  - Testing and training learning curves converge at similar values
  - Smaller the gap, the better our model generalizes

## 4.3 RMSE

The **root-mean-square deviation (RMSD)** or **root-mean-square error (RMSE)** is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed.

## 4.4 Our models Evaluation

The evaluation of the models will be based on the scores they will output on their gridsearch with their best hyperparameters, and then a new function made by us which uses r2 metric to choose the best model on cross-validation and the learning curves we have plotted for them.

### 4.4.1 Numeric results (RMSE)

#### 1) Gradient Boosting

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='ls', max_depth=2, max_features=None,
                           max_leaf_nodes=None, min_impurity_decrease=0.0,
                           min_impurity_split=None, min_samples_leaf=1,
                           min_samples_split=7, min_weight_fraction_leaf=0.0,
                           n_estimators=250, n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0, warm_start=False)
cross_val: mean= 0.11601868147715579 , std= 0.00917456844995485
```

*Figure 14: Gradient boosting metrics from Grid Search*

#### 2) Lasso

```
Lasso(alpha=0.0005000000000000001, copy_X=True, fit_intercept=True,
       max_iter=1000, normalize=False, positive=False, precompute=False,
       random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
score= 0.9343987553626258
rmse= 0.09654197652682298
cross_val: mean= 0.11004634045651557 , std= 0.010080442161281239
```

*Figure 15: Lasso metrics/results from Grid Search*

### 3) RandomForest

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features=50, 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=200, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
score= 0.9840432715471471
rmse= 0.0476137156031438
cross_val: mean= 0.13049157715908324 , std= 0.01660197154528219
```

*Figure 16:Random forest metrics/results from Grid Search*

### 4) Elastic Net

```
ElasticNet(alpha=0.0005, copy_X=True, fit_intercept=True, l1_ratio=0.9,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
score= 0.9350504308593016
rmse= 0.09606126105379607
cross_val: mean= 0.10867650937959296 , std= 0.010184688425734471
```

*Figure 17:Elastic Net metrics/result from Grid Search*

### 5) Stacked Regressor

```
[60]: from sklearn.kernel_ridge import KernelRidge
stacked_averaged_models = StackingAveragedModels(base_models = (
    model['Lasso'], model['ElasticNet'],
    model['RandomForest'], model['GradientBoosting'],
    model_lgb, model_xgb),
    meta_model = model['Lasso'])

score = rmsle_cv(stacked_averaged_models)
print("Stacking Averaged models score: {:.4f} ({:.4f})".format(score.mean(), score.std()))

Stacking Averaged models score: 0.1069 (0.0061)
```

*Figure 18:Stacked Regresor RMSE score*

From the above pictures we can notice that the last line for each of them is the cross-validation evaluation of RMSE. The best model on RMSE metric is the Stacked Regressor with a score of 0.1069 (actually is not far from the real performance). Based on our main goal, which is making accurate predictions (that is the reason of using RMSE score), Stacked Regressor is the model we selected for final prediction.

#### 4.4.2 Learning curves on R-square (default metric on regression) metric

R-square is called differently the determination variable, it is the proportion of the variance in the dependent variable that is predictable from the independent variable(s). The  $SS_{res}$  and  $SS_{tot}$  in the formula showed in the picture are the residual Sum of Squares and the total Sum of Squares.

$$R^2 \equiv 1 - \frac{SS_{res}}{SS_{tot}}$$

We chose the R-square as metric for the learning curves for one purpose, to show that the best model is selected based on the goal and depending on the goal we have to choose the metric we want for evaluation. Our goal is prediction, so our best model would be the one that performs better on RMSE but if our goal was something else than the metric could have been R-square and our best model would have been a different one.

##### 1) Lasso Learning Curve

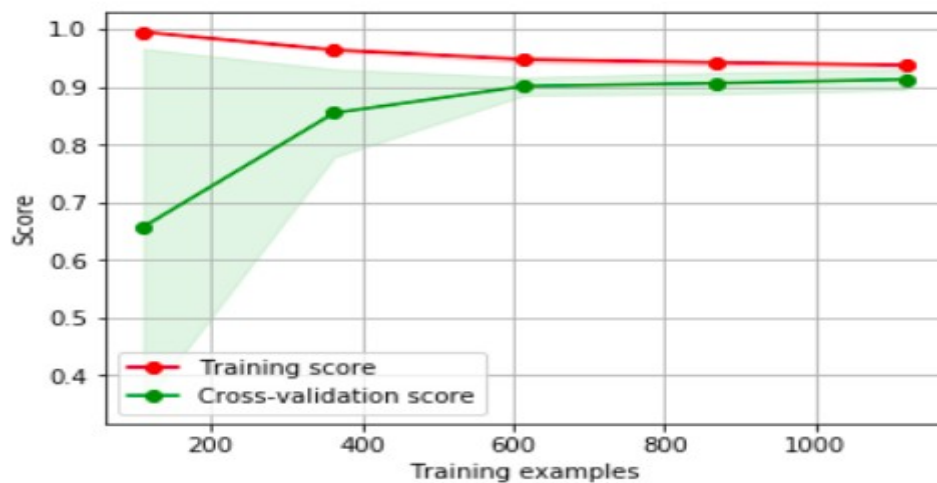


Figure 19: Learning curve of Lasso

From the learning curve we see that the models converge almost at the same point on R-Square metric with the value of ~92%. There are two possibilities, the first is that we are close the ideal learning curve and this convergence almost of the same point is because the model can generalize very well. The second is that we have a bad learning curve with high bias where the two evaluations converge in high values but the generalization is poor. Unfortunately we can not properly evaluate this because we do not have the test dataset.

##### 2) Elastic Net Learning Curve

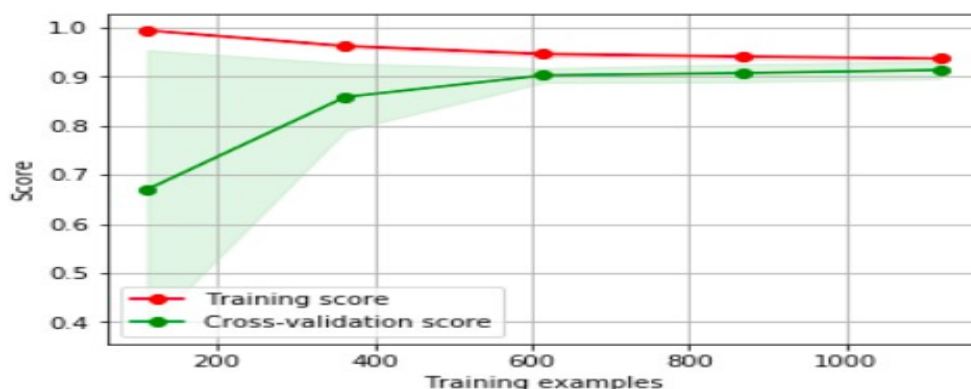


Figure 20:Elastic Net curve

The Elastic Net learning curve is similar to the one of lasso and the same comments can be made.

### 3) Gradient Boosting Learning Curve

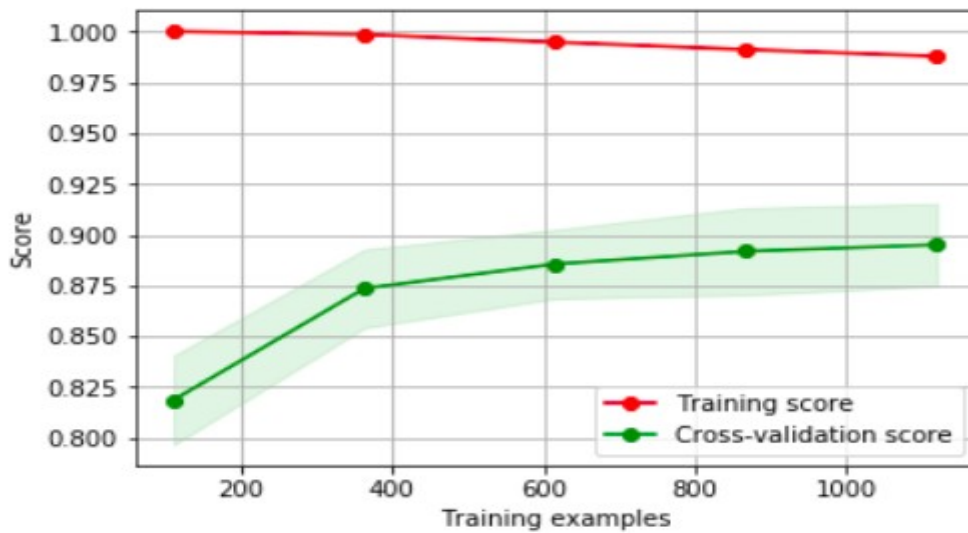


Figure 21:Gradient Boosting leaning curve

In this case the two evaluations, on training and cross-validation have a relatively big gap with each other which means we are in the bad learning curve with high variance case. The model does not generalize to well.

### 4) Random Forrest Learning Curve

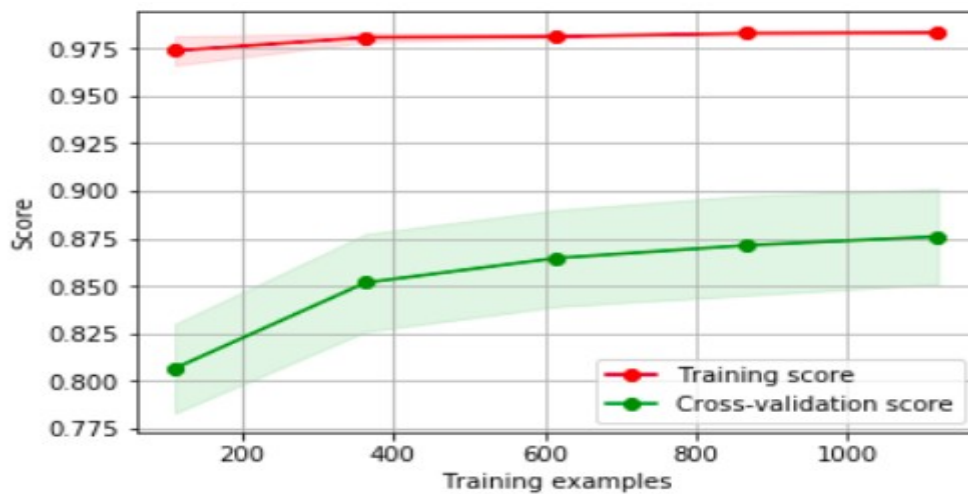


Figure 22:Random Forest Learning Curve

This learning rate is very similar with Gradient Boosting so the same comments take place.

## 5) Stacked Regression Learning Curve

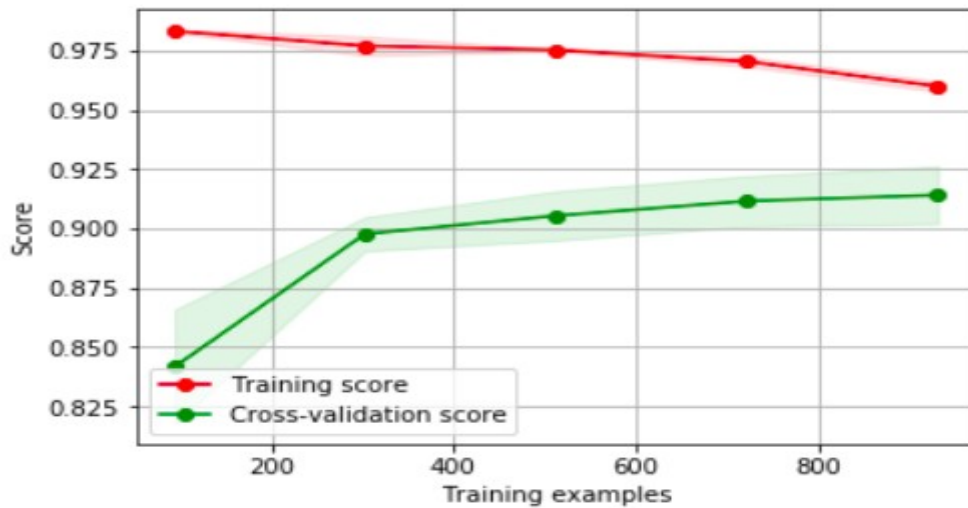


Figure 23: Stacked Regressor Learning Curve

The same thing happens even here, bad learning curve but the interesting thing here is that we are talking about stacking regressor, our best model in RMSE. This fact proves our comment that we have to select our metric based on our goal and if one model is the best with one metric does not mean that it is the best in all the metrics. However our goal is predicting so we gonna stick with Stacked regressor, make the prediction and submit it on kaggle.

## 5. Prediction and final result

After choosing the best model the prediction is the next and last step. For prediction we have the test set from kaggle only with the independent variables and we have to predict the dependent one, the sale price. Is important to say that all the changes done to the training set are done even to the test set with the only difference that in the test set we did not remove any data record. Following is showed the code for prediction and the result in kaggle.



```
#the best ensambling method, predicting with stacked_averaged_models
stacked = pd.DataFrame(prediction)
stacked.columns = ['SalePrice']
stacked.index = TestId
stacked.to_csv('submission_stacking_new.csv', header=True, index_label='Id')
```

```
#makind the training and prediction of the stacked regressor
stacked_averaged_models.fit(new_dataset.values, SalePrice.values)
stacked_train_pred = stacked_averaged_models.predict(new_dataset.values)
prediction = stacked_averaged_models.predict(new_testset.values)
print(np.sqrt(mean_squared_error(SalePrice, stacked_train_pred)))
```

Figure 24: Predicting with our Stacking modeling



332	Anxhelo Diko & Sidorela Mema		0.11495	1	now
Your Best Entry 					
Your submission scored 0.11495, which is not an improvement of your best score. Keep trying!					

Figure 25: Score on kaggle

## 6. Conclusion

This task seems easy but is actually very tough, the preprocessing and the selection of the best model is very complicated. In a machine learning task all the subtasks are very important. They are very connected with each other and to have a final good performing model all of this subtasks should be handled carefully and in the correct way. For example if you have the best model but the preprocessing is not done properly your model will perform poorly and vice versa. Another thing that should be noted is that ensambling methods are very powerful and in cases like our with a very limited dataset can help achive good results. The next point is that the hyperparameters can make the magic for the model, so a good optimization is very important. And the last thing is that we should pay attention to the metric we select for evaluation because it has to be appropriate for our goal.

## 7. References

1. <https://www.kaggle.com/agodwinp/stacking-house-prices-walkthrough-to-top-5>
2. [https://en.wikipedia.org/wiki/Linear\\_regression](https://en.wikipedia.org/wiki/Linear_regression)
3. <https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net>
4. <https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a>
5. <https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998>
6. <https://scikit-learn.org>

7. [https://en.wikipedia.org/wiki/Data\\_binning](https://en.wikipedia.org/wiki/Data_binning)
8. <https://blog.statsbot.co/ensemble-learning-d1dcd548e936>