Capstone Project
Machine Learning Engineer Nanodegree

Christoph Hilty

01. September 2018

1 Definition

1.1 Project Overview

The project is an ongoing kaggle competition[1] based on the 'Ames Housing Dataset'[2] of Dean De Cook. With a modernized and expanded data set, the task provides an interesting use case for various data science and machine learning methods. The prediction of house prices is clearly a much needed and important part in the real estate world and there is a lot of money involved. An accurate prediction is of high importance for the seller and the buyer as well. With a lot of features and a supposedly easy problem, the task offers a lot of room for data analysis, feature engineering and exploration of algorithms. This makes the project attractive and engaging and allows the author to apply, challenge and compare different techniques of the current machine learning space. In addition, the author is implementing software solutions for real estate companies since a decade and the insights from the project might be of interest for the author as well as the clients.

1.2 Problem Statement

How can the price of a house be predicted from a variety of numerical and categorical, correlated and uncorrelated features? This is the main problem we are trying to solve. However, we want to explore answers to other questions as well in the process: Are there other possibilities to solve the problem that a regression approach? How important is feature engineering and how do the different algorithms compare? The authors try to answer these questions with a thorough analysis with a workflow according to the next section.

1.3 Workflow

Here we quickly summarize the workflow that has been followed in the progress of the project. In section 2 and 3 of the document, we will focus on the results of this process.

Data exploration

The data is explored for underlying directions. The most important features are determined and an intuitive understanding of the data is achieved. Correlations are clarified with solid reasoning to prevent data dredging[3].

Data preparation

With the gained insights from the exploration step the data is now getting transformed to allow better exposition for the different algorithms.

Model exploration and experimentation

Different models are explored and methods for regularization are defined and evaluated.

Model selection, combination and optimization

Some promising, complementary models are then combined to ensembles and advanced methods like bagging, boosting and stacking are explored.

Final model

The final model then is selcted and some last tunings are applied.

1.4 Metrics

The evaluation metric for the competition[1] is the root mean squared error between the logarithm of the predicted and the observed price.

$$E = \sqrt{\frac{\sum_{t=1}^{T}(\log price_t^{(predicted)} - \log price_t^{(observed)})^2}{T}}$$

For this project, we decided to work with the same metric for the following reasons:

- The price is the only variable to predict and the error needs to take the predicted and the observed price into account.
- With the log function, we are able to even out prediction errors between cheap and expensive houses. Otherwise, the same relative prediction error would have a different impact.

- Keeping the prediction error the same as the competition allows a direct comparison with the leader board.

2 Analysis

2.1 Data Exploration

The provided data for the competition is consisting out of two different data sets. A training set with the observed price (SalePrice) of the houses included and a test set without the target variable. The training set has 1460 rows and 79 features of which 38 are of numerical nature.

A complete description of all the features can be found in the original documentation of the data[4].

	GrLivArea	OverallQual	YearBuilt	FullBath	Fireplaces	PoolArea	GarageCars	SalePrice
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	1515.463699	6.099315	1971.267808	1.565068	0.613014	2.758904	1.767123	180921.195890
std	525.480383	1.382997	30.202904	0.550916	0.644666	40.177307	0.747315	79442.502883
min	334.000000	1.000000	1872.000000	0.000000	0.000000	0.000000	0.000000	34900.000000
25%	1129.500000	5.000000	1954.000000	1.000000	0.000000	0.000000	1.000000	129975.000000
50%	1464.000000	6.000000	1973.000000	2.000000	1.000000	0.000000	2.000000	163000.000000
75%	1776.750000	7.000000	2000.000000	2.000000	1.000000	0.000000	2.000000	214000.000000
max	5642.000000	10.000000	2010.000000	3.000000	3.000000	738.000000	4.000000	755000.000000

Figure 1: Statistics of the target variable and a sample of numerical features

Due to the sheer number of features the process of feature engineering is important. There might be features that are highly correlated and need to be removed or combined to prevent possible detrimental co-linearity between the regressors.

	GrLivArea	OverallQual	YearBuilt	PoolArea	CentralAir	Neighborhood	CentralAir	SalePrice
0	1710	7	2003	0	NaN	NaN	NaN	208500
1	1262	6	1976	0	NaN	NaN	NaN	181500
2	1786	7	2001	0	NaN	NaN	NaN	223500
3	1717	7	1915	0	NaN	NaN	NaN	140000
4	2198	8	2000	0	NaN	NaN	NaN	250000
5	1362	5	1993	0	NaN	NaN	NaN	143000
6	1694	8	2004	0	NaN	NaN	NaN	307000
7	2090	7	1973	0	NaN	NaN	NaN	200000
8	1774	7	1931	0	NaN	NaN	NaN	129900
9	1077	5	1939	0	NaN	NaN	NaN	118000

Figure 2: Excerpt of the data with numerical and categorical features

Also, we recognize missing values for various features which need to be properly handled.

2.2 Exploratory Visualization

2.2.1 Distribution of target variable SalePrice

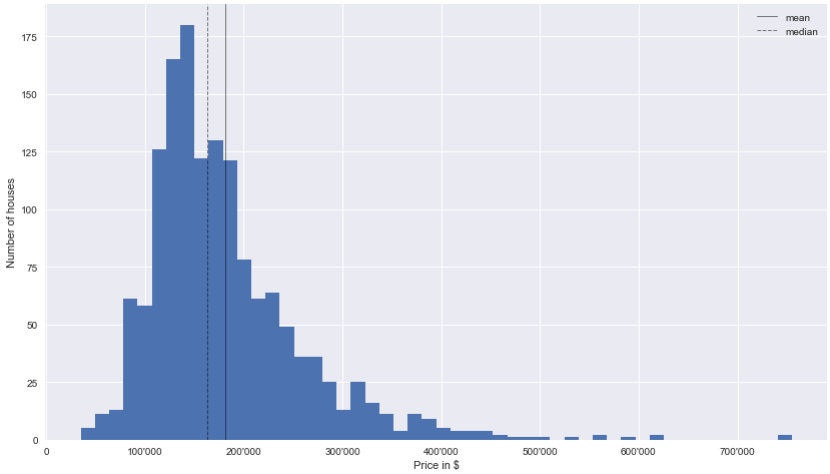

Figure 3: Price distribution in the training set

By visually examining the distribution of the house prices we can see, that the target variable is rightly skewed (skew of 1.88). This can also be evaluated by fact that the median of the data is lying left from the mean.

2.2.2 Correlation of different features

Perfect collinearity between regressors would violate the regression assumptions and we examine the correlation between the independent variables with a correlation matrix which we visualize with a heatmap.

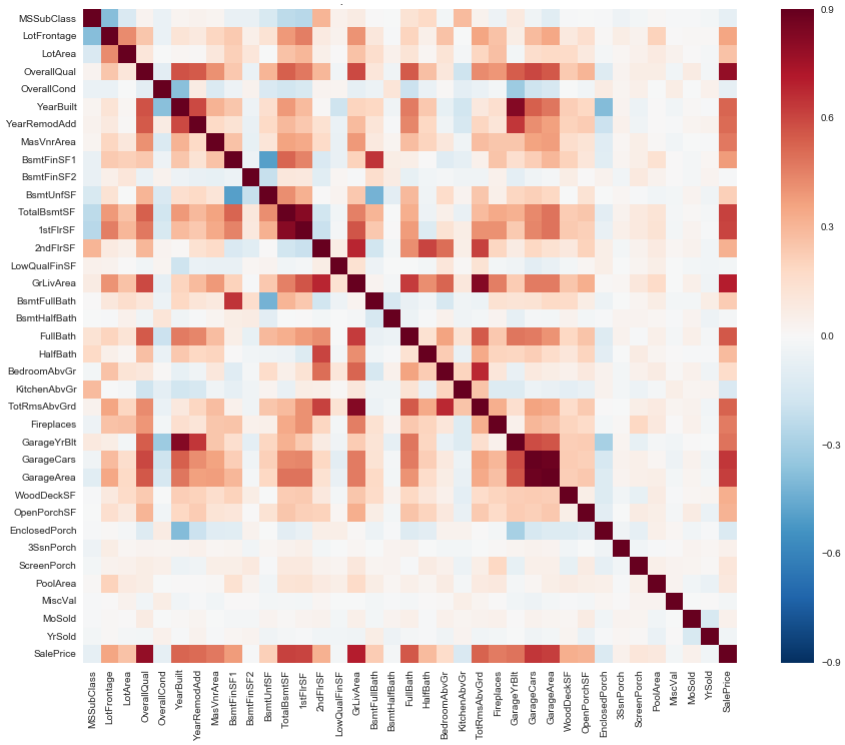

Figure 4: Visualized correlation between independent variables

We can see strong correlations within the features and examine the highest absolute correlations coefficients:

```
Top Absolute Correlations
GarageCars    GarageArea      0.882475
YearBuilt     GarageYrBlt     0.825667
GrLivArea     TotRmsAbvGrd    0.825489
TotalBsmtSF   1stFlrSF        0.819530
2ndFlrSF      GrLivArea       0.687501
BedroomAbvGr  TotRmsAbvGrd    0.676620
BsmtFinSF1    BsmtFullBath    0.649212
YearRemodAdd  GarageYrBlt     0.642277
GrLivArea     FullBath        0.630012
2ndFlrSF      TotRmsAbvGrd    0.616423
```

Figure 5: Top 10 correlation coefficients

There is no perfect correlation between features however, there are four feature pairs with an absolute coefficient above 0.8, suggesting that some features can be removed or optimized. Intuitively it is obvious that there has to be a high correlation between the listed features (e.g. GarageCars and GarageArea).

2.3 Algorithms and Techniques

This section gives a short overview and justification of the different techniques used. A detailed discussion however will follow below in section 3.2.

2.3.1 Regression

By looking at the different features, we did not find any non-linear relationship between independent and dependent variable. Therefore, a linear regression though its simplicity is a good fit. Because we have only around 1.5 thousand observations but a considerably amount of features a ridge regression will be evaluated [5]. It will allow us to use regularization to prevent overfitting which could be a problem because of the limited amount of data.

2.3.2 Grid search

For the optimization of the parameters we will apply a grid search where beneficial. With a grid search approach, we are not only able to optimize critical parameters for a single parameter, but also for a pipeline with different steps and for ensembles.

2.3.3 Neural Network

Neural networks have proved to be very efficient in supervised learning problems and even if the problem at hand is a very good fit for regression we want to explore and compare the performance of a multi-level perceptron. This will further allow us to build a stack with two different classes of algorithms.

2.3.4 Stacking

Because the method of stacking (also called meta ensembling) at the moment often outperforms the predictive power of a single model or an ensemble of equal models[6], we want to build a stacked model out of multiple, varying estimators. We will evaluate how the performance compares to a single algorithm as well as a stacked model out of different algorithms.

2.4 Benchmark

To have a baseline for the more elaborate models we decided to use a model without the need for feature engineering. These will allow us to also gain an understanding of how the step of data pre-processing (data imputation, feature engineering, dummy encoding etc.) will affect the performance of the models. We chose a linear model with the 10 most relevant numerical features and normalization.

3 Methodology

The project has been addressed by following the steps that have been outlined in section 1.3. In the next chapter however, we will discuss the resulting transformations, methods and algorithms that have been evaluated to build the final model.

3.1 Data Preprocessing

3.1.1 Data transformation

In the data exploration section, we have seen that our target variable is positively skewed. Because the regression assumptions do imply (when not using OLS method) that the target variable is also normally distributed[7] we apply a log transform to the target variable which brings us a much more even distribution:

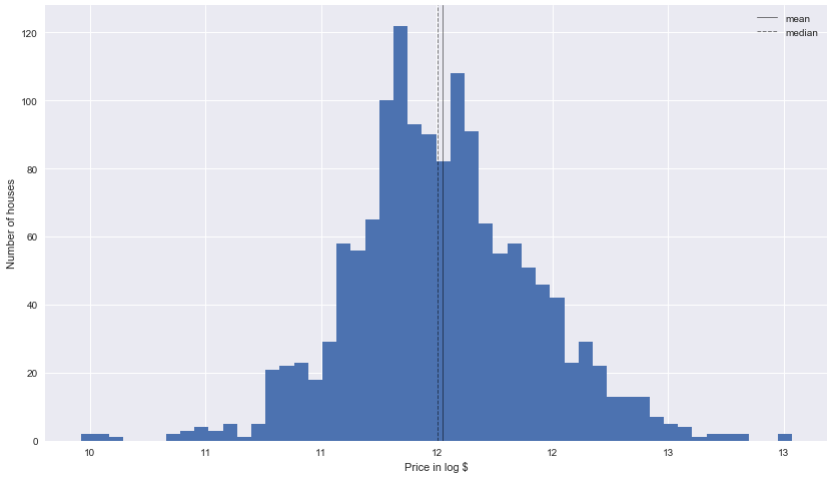

Figure 6: Price distribution after applying log transform

The log transformation $log(1 + y)$ of the sale price did have a substantial effect on the predictive power of a naive linear regression model with the five most important features:

Model	Skew	Score
Naive linear regression model	Positive (1.88094)	0.54923
	Minimal (0.12122)	0.18438

Table 1: Root mean squared error (RMSE) with different distributions

3.1.2 Outlier identification

Intuitively we understand that the size of a house is having a great impact on the sale price. This fact is validated by looking at the heatmap (see figure 4).

A scatter plot of the living area above ground and the sale price gives an overview of the correlation:

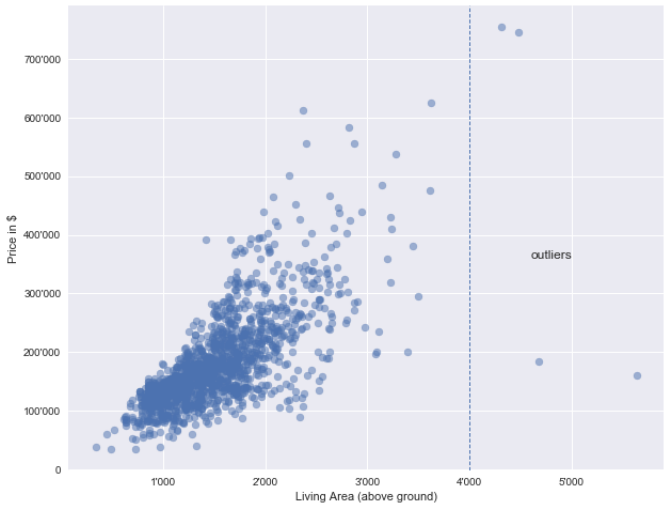

Figure 7: Scatter plot of living area and sale price

We can see here that there are two observations which are potentially outliers because the living area above ground is greater than four thousand square feed and the sale price is exceptionally low (below 200 thousand dollars). This observation is confirmed and expanded in the documentation[4] of the data set and we remove all observations above 4'000 square feet as outliers from the data set.

The removal of the outliers did also improve the results of our naive model:

Model	Outliers	Score
Naive linear regression model	Present	0.18438
	Removed	0.17668

Table 2: Root mean squared error (RMSE) in different stages

3.1.3 Data imputation

The data has been checked for missing values, which would badly affect the algorithms. Because we decided to use pipelines for the different steps we handled these missing inputs explicitly (and not through parameterization of the algorithms).

Numerical values

- There are 3 numerical features with missing data. With the documentation[4] and common sense we distinguished between missing input values due to data collection or feature specific meanings. In the first case we imputed the median (LotFrontage) and in the latter case we filled in an appropriate minimum (e.g. MasVnArea) or maximum value.

Categorical values

- There are a lot more categorical features with missing data. These are treated as separate categories and are addressed in the step 3.1.5. In order to align all the categorical features we impute some *None*'s into missing values of some features (e.g. GarageType).

3.1.4 Feature selection

We have evaluated the removal of the most highly correlated features with a linear regression model of all numerical features. The result was a slight decrease in performance and we decided to keep the correlating variables in our model:

Model	Correlating features	Score
Linear regression	Present	0.13575
model	Removed	0.13585

Table 3: Root mean squared error (RMSE) of models

3.1.5 Feature encoding

In order for the algorithms to work correctly, we need to encode the categorical features into binary features. For this process we used the function *pandas.get_dummies* from the pandas library[8]. In order to avoid multi-collinearity in the resulting features special care has been taken to prevent the dummy variable trap[9].

Several features could further be treated as ordinal features because they do have a clear ordering in their values. The feature *OverallQual* for example is documented as follows:

```
Overall Qual (Ordinal): Rates the overall material and finish of the house

       10      Very Excellent
       9       Excellent
       8       Very Good
       7       Good
       6       Above Average
       5       Average
       4       Below Average
       3       Fair
       2       Poor
       1       Very Poor
```

Figure 8: Excerpt from the documentation[4]

This makes the variable *OverallQual* unambiguously of ordinal nature and we have initially encoded these features with the label encoder from sklearn[10]. Surprisingly however, this did not lead to better results than the plain dummy encoding.

Overall, the consideration of categorical values did have a surprisingly subtle effect on the predictive power of the model:

Model	**Categorical features**	**Score**
Linear regression	Removed	0.13575
model	Encoded	0.13456

Table 4: Root mean squared error (RMSE) of models

3.2 Implementation

3.2.1 Benchmark Model

As stated in the analysis section we decided to use a simple linear regression with the ten most important numerical features as a baseline. The features have been selected with SelectKBest[11] and the algorithm has been parameterized with the appropriate scoring function *f_regression*. The three top scoring features were OverallQual (overall quality of the building), GrLivArea (living area above ground) and GarageCars (number of parkings). These results are aligned with our intuition for real estate prices. The algorithm additionally has been parameterized to normalize the data because there are various units present in the data (feet, year, square feet etc.). We evaluate this model with a 70/30 training and testing split which yields following results:

Model	**Score**	(Submission score)
Simple linear regression	0.16744	0.16473

Table 5: Root mean squared error (RMSE) of the model

3.2.2 Regularization

Due to the limited size of observation and the notable number of features, it is obvious that the model is prone to overfitting. To address this we chose a ridge regression (Ridge[12]) which has allowed us to add regularization to our model. Ridge regression uses a $l2$ regularization approach:

$$J(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{n=1}^{n} \theta^2$$

This approach has the following impact compared to a $l1$ (lasso regression):

- Computationally more efficient

- Provides a stable, single solution
- Not very robust to outliers (we have eliminated those in the data transformation step)

For solving the minimization problem we parameterize the algorithm with a *cholesky* solver (closed-form solution by André Louis Cholesky).

This model again has been trained on 70% of the data and evaluated on a test set of 30%.

Model	**Score**	(Submission score)
Ridge regression	0.12317	0.12114

Table 6: Root mean squared error (RMSE) of the model

3.2.3 Ensembling

In a first step a bagging approach for building an ensemble has been explored and the estimators have been trained in parallel a on random subsets of the data with replacement[13]. With this method each individual estimator had a higher bias but aggregation reduces both bias and variance[14]. The bagging estimator did score a little weaker on the local test set but did a slightly better job with generalization on the leaderboard:

Model	**Score**	(Submission score)
Ridge regression	0.12317	0.12114
Ridge regression with bagging	0.12399	0.12094

Table 7: Root mean squared error (RMSE) of the model

Better results however where achieved when multiple predictors have been trained sequentially on the data. This approach is called boosting and allows each new predictor to pay a bit more attention on training instances that the predecessor underfitted[15]. To implement boosting we chose the AdaBoostRegressor[16] of the scikit-learn package. We have initialized the ensemble with the ridge regression as base estimator and evaluated the correct loss function by performing grid search. A linear loss function for updating the weights after each boosting iteration yielded best results. Trained and tested on the same split as before (70%/30%) this ensemble method performed slightly better than the individual ridge regression on the kaggle site though the improvement was marginal:

Model	**Score**	(Submission score)
Ridge regressor with boosting	0.12417	0.12088

Table 8: Root mean squared error (RMSE) of the model

3.2.4 Diversification - Neural network

As stated above an approach with a multi-level perceptron has also been evaluated for the following reasons:

- Challenge an alternate approach for a classic regression problem.
- Have different models for the meta-ensembling steps so that potential weaknesses of an algorithm family can be minimized.

Overview

We used a keras wrapper for the tensorflow backend[17]. With all the features properly encoded we had an input vector of length 221. It is notable, that in this case label encoding of the ordinal features did in fact proved to yield better results than one hot encoding. We fed the input vector into the first dense layer of a sequential (linear stack of layers) model.

Topology

There are two hidden layers in the network, which are consisting of 100, respectively 50 neurons. Different topologies have been tried, but a gradually reduction of neurons per layer yielded the best results. More deep models as well as models with just one hidden layer were also less successful.

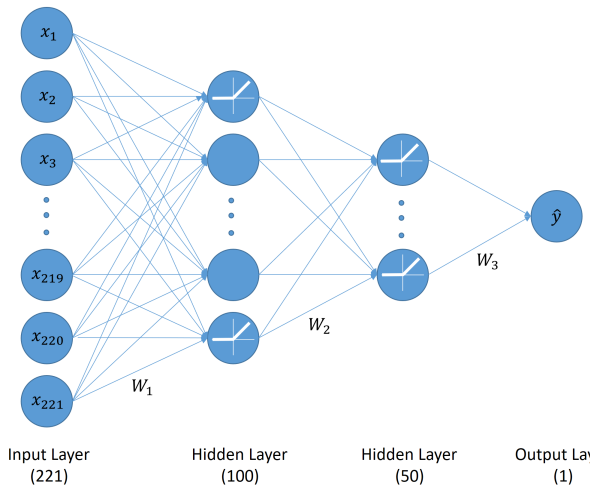

Figure 9: Neural network topology

Bias

The use of a bias neuron in the first three layers has been tested (*use_bias=True*) but did not decrease the resulting *RMSE*.

Activation

For all the layers of the network we used a $ReLU$ (rectifier linear unit) activation function. These have proved to enable better training of deeper networks[18] and ensured that only positive prices are resulting in the output layer.

Dropout

The amount of dropout in the different layers where crucial to prevent overfitting on the training set. Diminishing ratios of 0.3, 0.2 and 0.1 have been used.

Initialization

The tensors have been initialized with a random normal distribution.

Optimizer

To optimize the learning of the network an Adam optimization algorithm has been chosen to update the network weights instead of the classical stochastic gradient descent procedure. This has proven to allow for more efficient training.[19]

Result

The network did score better than the benchmark model but weaker than any requlatized implementation. The model did also show clear signs of overfitting when submitted to kaggle.

Model	**Score**	(Submission score)
Neural Network (221x100x50x1)	0.14798	0.16425

Table 9: Root mean squared error (RMSE) of the model

3.2.5 Meta-Ensembling

To implement stacking[14] two boosting models and the neural network where combined into a first prediction layer. Then a ridge regression model (as blender) was trained on these predictions and the target variable to make the final estimation.

For the first layer the adaptive boosting algorithm from the previous section has been combined together with a gradient boosting library xgboost[20]. This library was introduced to further diversify the algorithms of our stacked model. These both algorithms differ on how they create the weak learners for the successive iterations. While the adaptive boosting algorithm is changing the sampling distribution by changing the weights, the weak learners from the gradient boosting algorithm are trained on the remaining errors. For the meta learner also called blender, a second ridge regression model has been used with the same parameters as in the first prediction layer but with a much smaller regularization parameter (0.001 instead of 10) because we did not expect to have the same overfitting situation as with much less and diverse features.

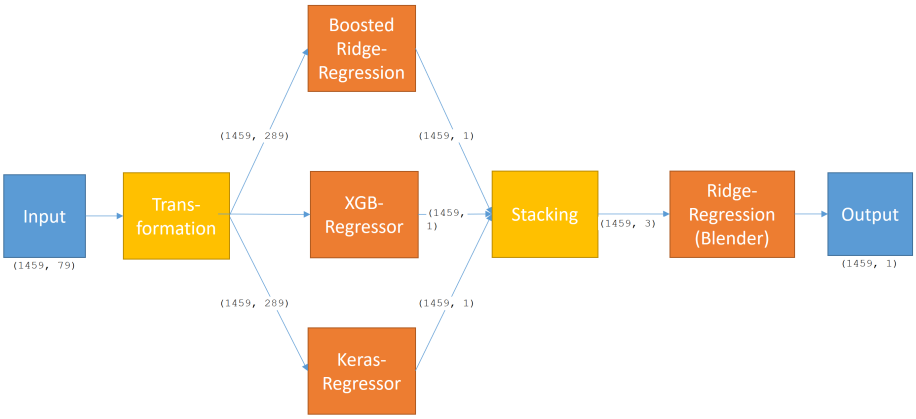

Figure 10: Stacked model overview

The result of this stacked model was better than the neural network alone but did not yield better results than the ensemble model from the previous section.

Model	Submission score
Stacked model	0.13128
Ridge regressor with boosting	0.12088

Table 10: Root mean squared error (RMSE) of the model

We can think of several possible reasons for the fact that the stacking did not perform better than the more simple boosting ensemble:

- Additional splitting is necessary when performing a stacking as in our example. This reduces the amount of training data for the optimization of the first layer. This might be critical when the amount of observations are limited.
- The data transformation might need to be optimized for the different algorithms

3.3 Final model

3.3.1 Model selection

In the following table the most relevant explored models and their score is listed:

Model	**Score**	(Submission score)
Simple linear regression	0.16744	0.16473
Ridge regression	0.12317	0.12114
AdaBoostRegressor	0.12400	0.12088
Neural Network (221x100x50x1)	0.14798	0.16425
Stacked model	0.14103	0.13128

Table 11: Root mean squared error (RMSE) of the different models

The ensemble consisting of multiple ridge regression models (AdaBoostRegressor) is the best performing of all evaluated models and has therefore been chosen as the final model.

Hyperparameter optimization for the ensemble as well as for the base estimator has been performed with a grid search with cross validation. For a discussion of the selected parameters for the algorithms see table 12.

Algorithm	**Name**	**Value**	**Justification**
AdaBoosting-Regressor (Ensemble)	learning_rate	0.0001	Small learning rate to allow high precision.
	loss	*square*	Using the square distance for minimization (l2 loss).
	n_estimators	500	500 estimators have been chosen to perform boosting.
	random_state	42	Random state for reproducible results.
Ridge Regressor (Base estimator)	alpha	10	A regularization strength of 10 has been selected to prevent overfitting.
	fit_intercept	*True*	The interecept needs to be fit as well, because we did not center all the features in advance.
	max_iter	*None*	The number of iterations have not been limited.
	normalize	*False*	The data has not been additionally normalized.
	solver	*cholesky*	The standard solver which enables a closed-form solution yielded best results.
	random_state	42	Random state for reproducible results.

Table 12: Explanation of the parameters for the boosting ensemble

3.3.2 Refinement

Below is a graph of the learning curve of the base estimator (ridge regression) of the final model:

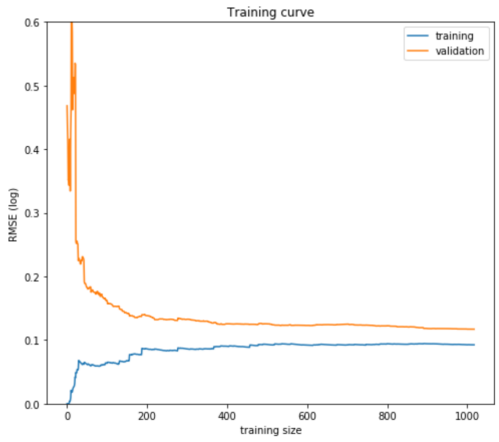

Figure 11: Learning curve of ridge regression model

Following observations and deductions were made from this graph:

- Both, training and validation are approaching to a decent small error. Therefore, we can deduce that the model does not suffer high bias.
- The curves are approaching each other and there is not a deciding difference in error between the sets. This demonstrates that the model does not suffer a high variance and is not overfitting the training data.

One specific optimization however has been applied after selecting the final model because there are various features present that are highly skewed:

```
Highly skewed features (> .8)

MiscVal                      24.4516
PoolArea                     14.8131
LotArea                      12.1951
3SsnPorch                    10.2938
LowQualFinSF                 9.00208
KitchenAbvGr                 4.48378
BsmtFinSF2                   4.25089
ScreenPorch                  4.11798
BsmtHalfBath                 4.09919
EnclosedPorch                 3.0867
MasVnrArea       2.6663261001607443
OpenPorchSF                  2.36191
LotFrontage      2.1608659947055435
BsmtFinSF1                   1.68377
WoodDeckSF                   1.53979
TotalBsmtSF                  1.52269
MSSubClass                   1.40621
1stFlrSF                     1.37534
GrLivArea                    1.36516
BsmtUnfSF                   0.919323
2ndFlrSF                    0.812194
```

Figure 12: Highly skewed features

The predictive power of the model could therefore be elevated by applying a one-parameter box-cox transformation[21] on features with

$$abs(skew(x)) > 0.8.$$

The box-cox transformation in its one parameter form (of which our log transformation of the target variable is a special case) looks like this:

$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln y_i & \text{if } \lambda = 0 \end{cases}$$

The resulting model did score slightly better on the local training set as well as on the leaderboard:

Model	**abs(skewness)**	**Score**	(Submission score)
AdaBoostingRegressor	< 25	0.12400	0.12088
	$\leq .8$	0.11943	0.11844

Table 13: Root mean squared error (RMSE) of the different models

The model has been trained on a 70%/30% split of the training set provided by kaggle to establish a local reference. Then it has been retrained on the complete training set provided by *kaggle.com* without a split. With the resulting model a prediction has been performed on the test set provided and the results have been submitted. The second training on all the data was certainly necessary to train the algorithm with as much data observations as possible to make the final prediction.

4 Results

4.1 Evaluation and Interpretation

It was interesting to observe the predictive power of the models evolving with the complexity of the models and then reaching a plateau where no additional benefits have been gained through complementary algorithms (neural network) or sophistication (stacking).

The improvment in the *RMSE* (root mean squared error) from the benchmark model (0.16473) to the final model (0.11844) means a decrease of 28.1% and cannot be explained by chance alone.

Translated back into the target variable of the domain, the error suggests that the price for a house can be roughly estimated with a precision of +/-12% by using the following approximation:

$$ln(x_1) - ln(x_2) \approx \frac{x_1 - x_2}{x_1} = \frac{\Delta x}{x}$$

For the mean of the sale price in our data set (around $180'000) this would imply a prediction error of +/- $25'000. However, to use this model in a real world scenario there are different aspects that need to be considered beforehand:

- House prices are heavily dependent on environmental aspects and cannot easily be transferred to different areas. The average price as well as the distribution might heavily differ as well as the impact of features in different topologies (e.g. country side versus town).
- Specific features like *Neighborhood* make the model additionally stationary and would not even allow for a direct transition into other locations.
- General economic changes like recession or inflation over time may also affect the model and would need to be addressed.

4.1.1 Ranking

The final model has been ranked $689th$ of 4,404 teams (top 16%) on the leaderboard.

5 Conclusion

5.1 Reflection

In the end we have achieved a model with a decent precision. The predictive power of the model might be comparable to a subject matter expert though this would have been examined further. Moreover, the main components of the model can be easily explained and the prediction process can be reconstructed intuitively by examining the coefficients of the regression. This is an advantage of our boosted regression model over more advanced algorithms (see section 3.2.4) when an estimation needs to be justified in front of a client.

The authors had the opportunity to participate in a *kaggle.com* challenge and now have an understanding of the submission process and the levels of the competitors. Moreover, we were able to apply and deepen different techniques and methods in a practical context:

- Explore various kinds of feature engineering and their effect.
- Implement and optimize various regression algorithms.
- Implement different kind of ensembles (e.g. bagging and boosting).
- Explore a stacking approach over different estimators.
- Build a machine learning pipeline (for transformation as well as prediction).
- Apply several parameter optimization's with grid search.
- Execute statistical exploration methods (e.g. correlation analysis).
- Exercise implementation of a neural network.
- Directly compare results of different algorithms.

These aspects made the project very profitable and rewarding. There where however several difficulties as well - some of them were evident from the beginning while others came up through the process:

- The large number of features made the feature engineering process very interesting but also very time consuming.
- The use of pipelines did make the individual predictions easier while also complicating the code. Some methods (eg. n-fold cross validation) can also become harder to implement when using pipeline.
- Being an active *kaggle.com* competition the process of training and evaluating was getting more complicated.
- The limited set of observations combined with the number of features did oppose some constraints with advanced methods because of the additional splits involved.

In retrospect, we learned a lot of technical as well as methodical things from this project. Subsequent are some of these points discussed:

- A sophisticated regression approach beats a neural network built with our current understanding when predicting a continuous target variable.
- Ensemble methods are a good way to improve the predictive power of model. However they might bring additional complexity.
- Meta-ensembling methods like stacking offer additional space for model optimization. The additional complexity in implementation as well as prediction interpretation need to be taken into consideration.
- The number of observations is crucial when evaluating more advanced methods like ensembling or meta-ensembling because additional splits might be necessary when cross validation might not always be easily possible.
- Pipelines are an interesting way to reuse and structure code. However there is some additional complexity involved.
- Jupyter notebooks are an easy way for data exploration. For more extensive project in the future however, we would try to avoid long notebooks for practical reasons (e.g. subtle bugs because of dynamic typing and global variable scope, annoying history exploration in source control).
- The feature engineering process with lots of features can become very time consuming and automation may not always be beneficial.
- When doing principal component analysis it is crucial to be aware of the underlying scales and units.

5.2 Improvement

We explored various methods of a typical machine learning project. Several additional aspects however could further be explored and have been postponed due to time constraints:

- Inflation and recession over the observed time span could be addressed.
- The time investment in feature engineering could be elevated.
- The prediction errors could be compared over different features groups for underlying patterns.
- The stacking example could be extended by performing uneven pre-processing steps for the different algorithms.

Smaller errors might be achieved with the listed actions but the generalization would have to be examined carefully.

References

[1] kaggle.com. House prices: Advanced regression techniques. `https://www.kaggle.com/c/house-prices-advanced-regression-techniques`.

[2] Dean De Cook. Alternative to the boston housing data as an end of semester regression project. `https://ww2.amstat.org/publications/jse/v19n3/decock.pdf`.

[3] wikipedia.org. Data dredging. `https://en.wikipedia.org/wiki/Data_dredging`.

[4] Dean De Cook. Ames iowa: Alternative to the boston housing data set. `http://ww2.amstat.org/publications/jse/v19n3/Decock/DataDocumentation.txt`.

[5] scikit learn.org. Choosing the right estimator. `http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html`.

[6] scikit learn.org. Api reference. `http://scikit-learn.org/stable/modules/classes.html`.

[7] Selva Prabhakaran. Assumptions of linear regression. `http://r-statistics.co/Assumptions-of-Linear-Regression.html`.

[8] pandas.pydata.org. pandas.get_dummies. `https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html`.

[9] wikipedia.org. Dummy variable. `https://en.wikipedia.org/wiki/Dummy_variable_(statistics)`.

[10] scikit learn.org. sklearn.tree.labelencoder. http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[11] scikit learn.org. sklearn.feature_selection.selectkbest. http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html.

[12] scikit learn.org. sklearn.linear_model.ridge. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.

[13] wikipedia.org. Bootstrap aggregating. https://en.wikipedia.org/wiki/Bootstrap_aggregating.

[14] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and Tensorflow*. O'Reilly, 2017.

[15] wikipedia.org. Boosting. https://en.wikipedia.org/wiki/Boosting_(machine_learning).

[16] scikit learn.org. sklearn.ensemble.adaboostregressor¶. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html.

[17] keras.io. Wrappers for the scikit-learn api. https://keras.io/scikit-learn-api/#wrappers-for-the-scikit-learn-api.

[18] Antoine Bordes Xavier Glorot and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

[19] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations, San Diego, 2015*, 2014.

[20] The XGBoost Contributors. xgboost. https://en.wikipedia.org/wiki/Xgboost.

[21] Princeton University Germán Rodríguez. Transforming the data. http://data.princeton.edu/wws509/notes/c2s10.html.