Advanced Housing Regression: SalePrice Prediction

SCS_3253_009 Machine Learning

Submitted to University of Toronto
December 13, 2018

Boris Korotkov - korotko1
Robert Shaheen - qq267674
Vladimir Taubes - taubesvl

❏ Introduction

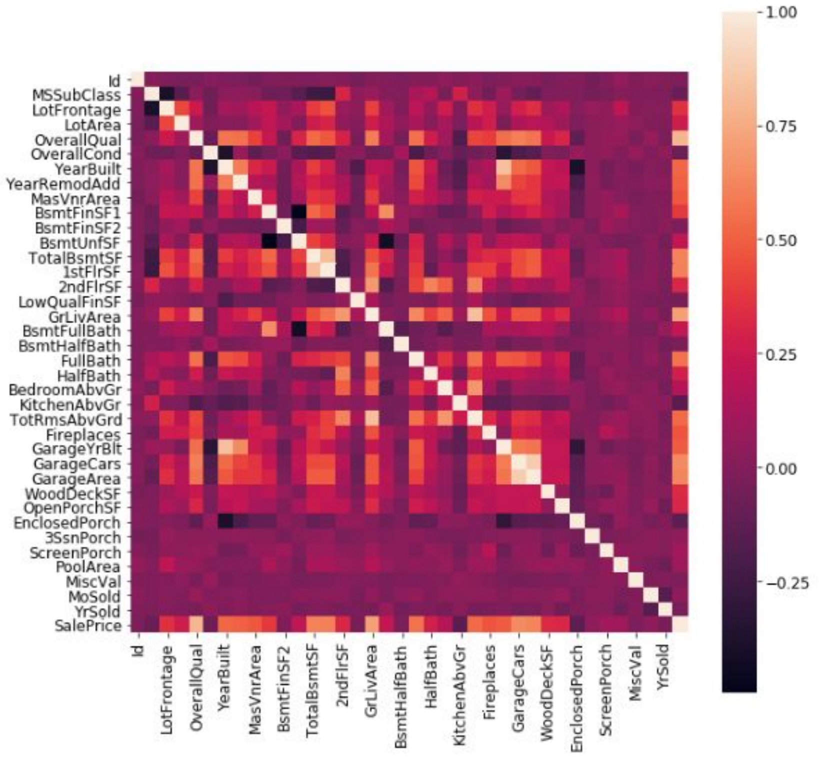
This paper investigates a dataset from Kaggle used to predict housing prices. The data contains 79 features regarding real-estate variables but no information about the market, interest rate, or housing policies that have a direct effect on the housing prices. Kaggle provides two datasets: a train and a test set with 1460 and 1459 elements, respectively. The train set comes with a labelled feature column 'SalePrice' whereas the test set is not labelled. This project splits the training set into two parts: 80% of the elements to (train_X, train_y) and 20% of the elements to (val_X, val_y). Datasets (train_X, train_y) and (val_X, val_y) are used to train machine learning models and tune hyperparameters. The test dataset is used to predict SalePrice for the unlabelled data, and these predictions are uploaded to Kaggle to check for model accuracy (measured in root mean squared error), demonstrated by a ranking on the Kaggle leaderboard. These Kaggle results give a good idea to the models' generalizability.

The scope of this course is on machine learning algorithms in scikit-learn and tensorFlow therefore the emphasis in this report is placed on investigating many different regression models. Time was allocated to preparing the data, dealing with outliers, and some rudimentary feature engineering, but the majority of the work was put into investigating different machine learning models. The performance of more than 10 models is calculated starting with vanilla linear regression advancing all the way to neural networks.

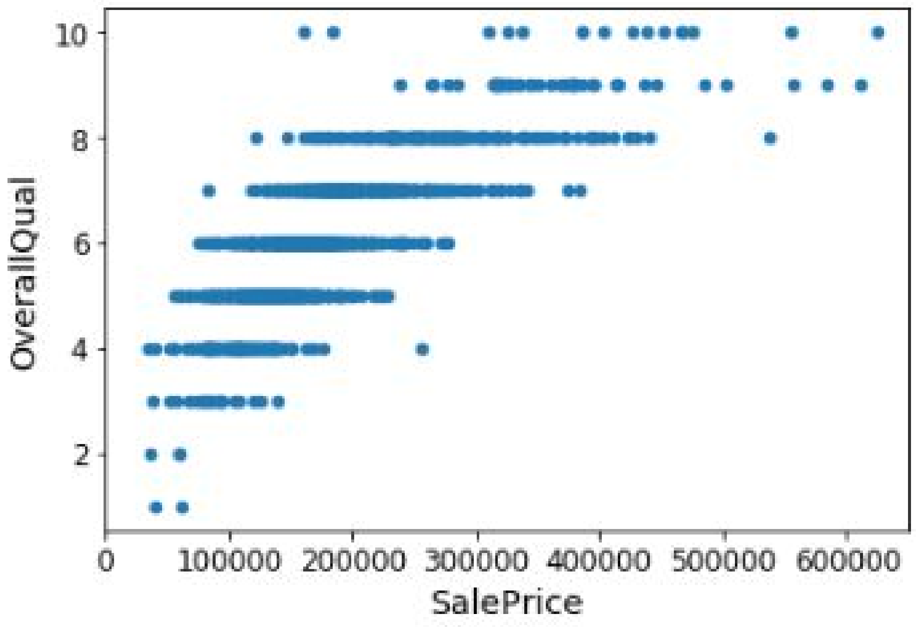
❏ Data exploration

The data exploration is started with retrieving the overall information about the dataset, number of features, and statistics over the numerical features. The dataset has 79 explanatory features that describe almost every aspect of residential homes in Ames, Iowa.

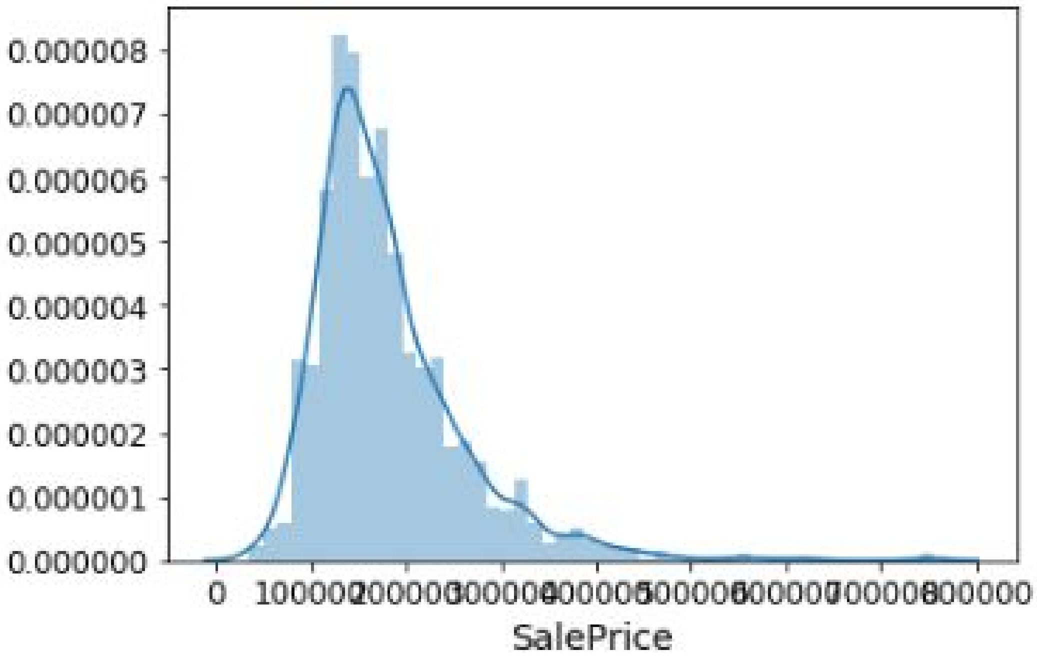
To understand the overall correlation, a numerical features heatmap plotted and analyzed.



There are some features that correlated better than others. The feature with the best correlation is OverallQuall:



The SalePrice distribution is close to normal, but it has positive skewness.



The Sale Price mean is $180,921.2 and standard deviation is $79,442.5

❑ Feature engineering

To improve the prediction accuracy four new features are added.

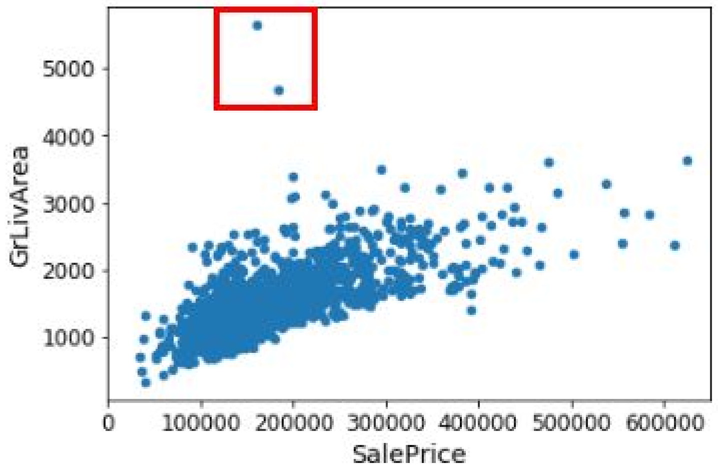
- Years after renovation: a number of years between last renovation and sale date
- Ratio of living area to lot size
- Ratio of living area to the number of bedrooms
- Ratio of overall quality to overall condition

Further assessment showed that new features improved RMSE by 0.4% only. Thus, the focus from hereon is tuned more into model selection and tuning rather than new feature engineering.

❑ Data preprocessing

❑ Outliers removal

The first step in data preprocessing is to remove the outliers. Two houses with living area greater than 4000 and price was less than $300,000 were removed because they negatively impacted the model prediction.



❏ Data cleaning and normalization

The source data had a lot on null values and numerical columns had different scales, so the following operations were performed with the data:

- Replaced all null values in numerical columns with median value
- Replaced all null values in text columns with the most frequent value
- Scaled all numerical data in the range of 0 to 1
- Converted categorical feature values into one-hot vector

All transformations listed above were completed in the most efficient manner, e.g. using SimpleImputer, MinMaxScaler, OneHotEncoder, ColumnTransformer and Pipeline.

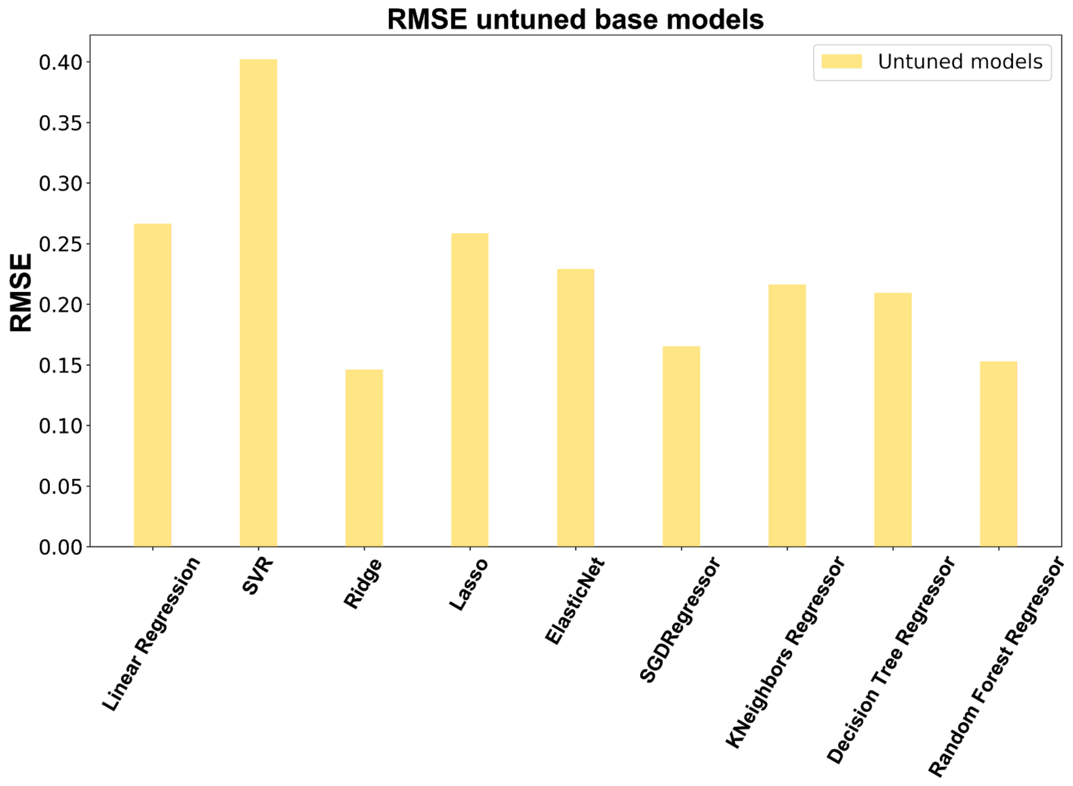
The resulting number of features increased to 291 features from 79.

❏ SalePrice predictions

❏ Base model selection

The SalePrice vector, a continuous feature, was earlier chosen as the advanced housing data label therefore a regression technique is sufficient to model the data. The following regression types are selected for initial assessment of the housing SalesPrice:

- Linear
- Support Vector
- Ridge
- Lasso
- Elastic Net
- Linear with stochastic gradient descent
- K-Neighbors
- Decision Tree
- Random Forest



The primary metric used to evaluate model performance is the root mean squared error (RMSE) of the log of the SalePrice and log of the predicted SalePrice to match the Kaggle Leaderboard metric. It is represented as:

$$RMSE = RMSE(log(SalesPrice),\ log(predicted\ SalesPrice))$$

Each learning model is fit to the preprocessed training data and then used to predict the SalePrice on the validation dataset. The performance of each model on the validation data is shown in the figure: RMSE for untuned base models.

It can be seen that, without any hyperparameter tuning, the best performing model is the linear ridge regression with an RMSE of about 0.14 and the worst performing model is the support vector regressor with an RMSE of about 0.40.

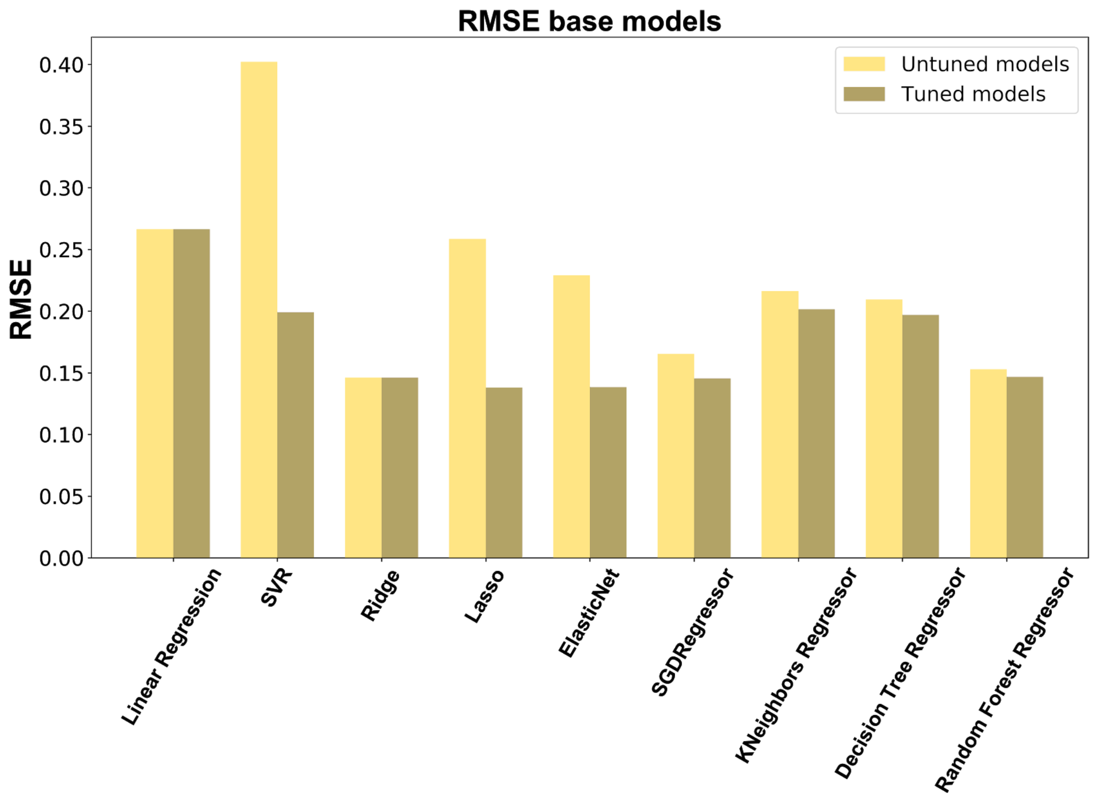
❑ Base model hyperparameter tuning

The hyperparameters of each base model are tuned using either a grid search or randomized search. The following table illustrates the tuned model hyperparameters:

Model name (sklearn)	**Tuned hyperparameters**

LinearRegression	{'fit_intercept': True}
SVR	{'max_depth': 16, 'max_features': 125, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 1000}
Ridge	{'alpha': 1.0, 'fit_intercept': True, 'solver': 'auto'}
Lasso	{'alpha': 71, 'max_iter': 100}
ElasticNet	{'alpha': 100, 'l1_ratio': 1.0}
SGDRegressor	{'eta0': 0.001, 'learning_rate': 'constant', 'loss': 'squared_epsilon_insensitive', 'max_iter': 50, 'penalty': 'none', 'power_t': 0.5}
K-Neighbours Regressor	{'algorithm': 'auto', 'n_neighbors': 7, 'p': 1, 'weights': 'distance'}
Decision Tree Regressor	{'max_depth': 9, 'max_features': 'auto', 'max_leaf_nodes': None, 'min_samples_leaf': 2, 'min_samples_split': 18}
Random Forest Regressor	{'max_depth': 16, 'max_features': 125, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 1000}

Some model's performance were drastically increased while other models did not change. The following graph illustrates the tuned versus untuned model RMSE on the validation dataset.

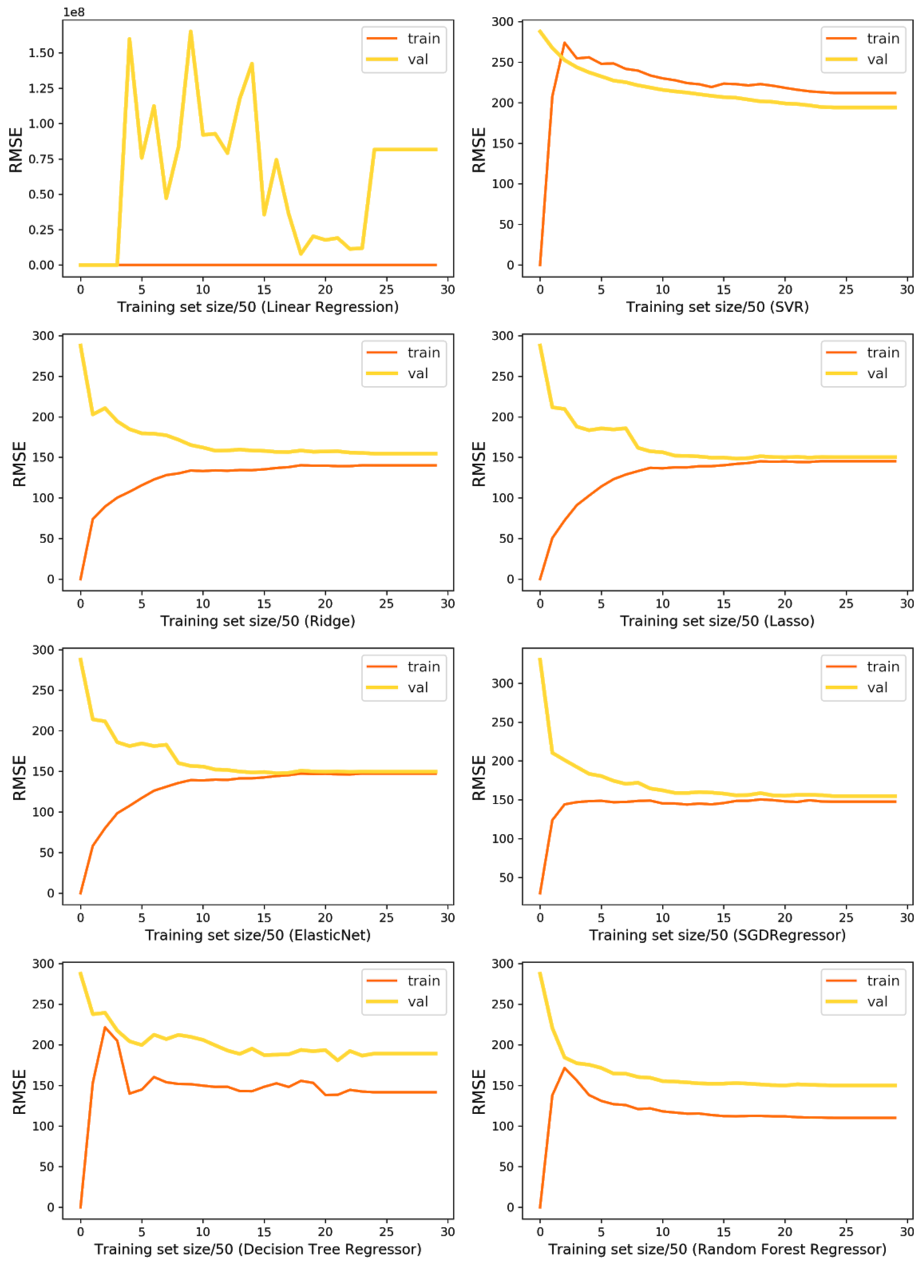


The model improvements are quantified in the following table.

Model name (sklearn)	**Rank**	**Untuned RMSE**	**Tuned RMSE**	**Performance Increase**	**Untuned** r^2	**Tuned** r^2
LinearRegression	9	0.26636	0.26639	0.00%	0.57903	0.87793
SVR	7	0.40208	0.19893	□ 50.53%	0.04099	0.74334
Ridge	4	0.14586	0.14586	0.00%	0.87380	0.89722
Lasso	**1**	**0.25845**	**0.13809**	□ **46.57%**	**0.60378**	**0.90787**
ElasticNet	**2**	**0.22906**	**0.13824**	□ **39.65%**	**0.68876**	**0.90916**
SGDRegressor	**3**	**0.16523**	**0.14529**	⇧ **12.07%**	**0.83804**	**0.89647**
KNeighboursRegressor	8	0.21635	0.20128	⇧ 6.96%	0.72234	0.78280
DecisionTreeRegressor	6	0.20935	0.19671	⇧ 6.04%	0.74002	0.76758
RandomForestRegressor	5	0.15267	0.14660	⇧ 3.98%	0.86175	0.90821

It can be seen from the graph and table above that SVR , Lasso, and Elastic net learning models show the highest improvement, while Linear Regression and Ridge models show no improvement with hyperparameter tuning. The highest performing model is Lasso regression.

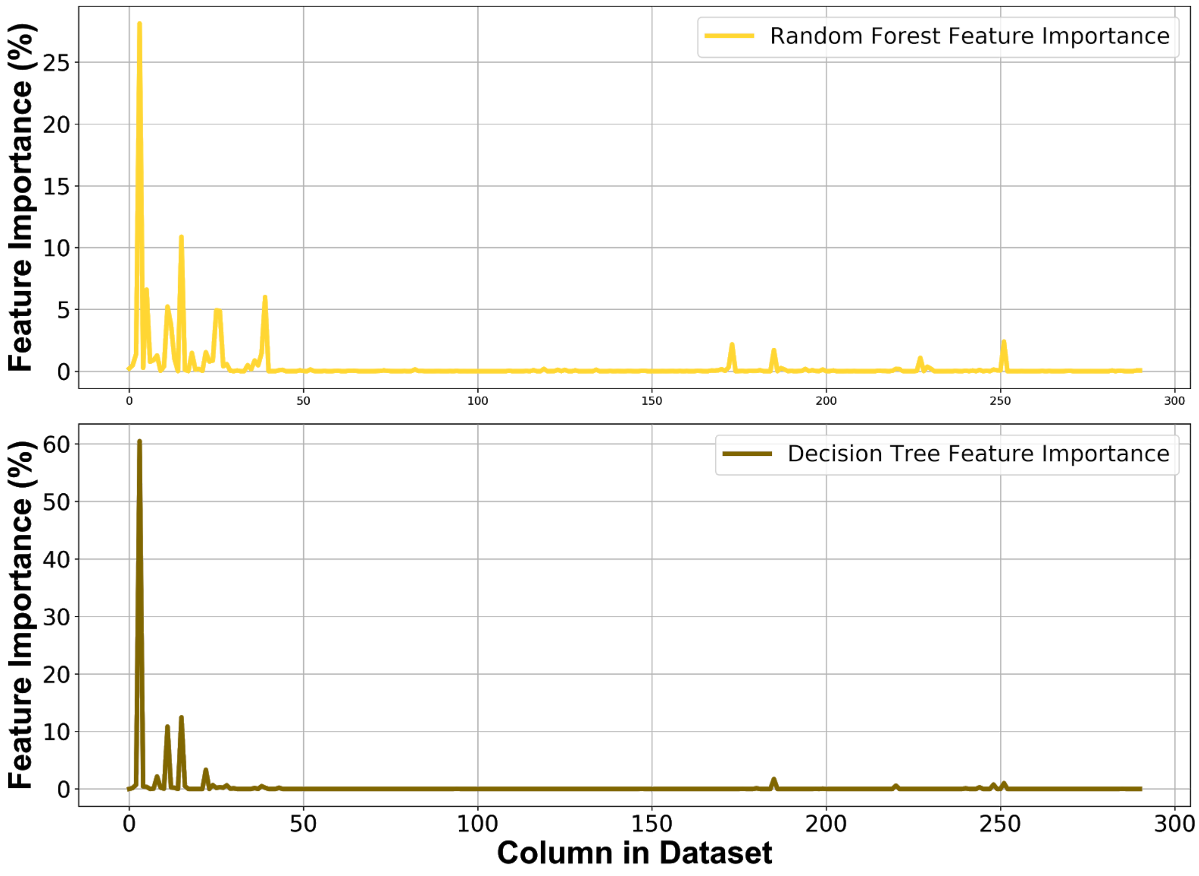
Next, the learning curves for the optimized models are examined to see how well the models generalize to the validation data. The learning curves for select models are are shown below.



First of all, it can be seen that the linear regression model performs poorly. Without any regularization the model grossly overfits the training data at each step and predicts a sporadic validation SalesPrice. It can also be seen from the above figure that each linear model, with regularization, exhibits bias in the learning curves. The best performing models according to these graphs are the various linear ElasticNet and Ridge regressors. The decision tree and random forest algorithms are shown to slightly overfit the training data from the gap between the plateau in training and validation curves.

❏ Feature importance

The most important features can be found from the random forest regressor and the decision tree regressor. The following graph illustrates the most important features of the processed dataset, where a column in the dataset represents one feature.



The most important features are closer to the beginning of the dataset columns. The most important feature is column 3, with a feature importance of about 28.2% for random forest and 60.4% for decision tree, which corresponds to LotArea. In reality LotArea is a good measure to determine housing cost, and both models are in good agreement with each other, therefore this analysis of feature importance can be accepted.

A lot was learned about the dataset through base model analysis, however, most models tend to either underfit or overfit the data even when tuned. The next session will focus on aggregation methods to create a strong learner from the optimized weak learners. Also, to note, it is understood by the authors that Random Forest Regressors is considered a strong learner and an aggregation method, however, it was used as a base model for the scope of this report.

❏ Aggregate learning methods

The optimized models from the previous section are combined to make strong regressors in an attempt to reduce the bias between predictions on the training and validation data. Three aggregate regression methods are explored: bagging, gradient boosting, and stacking. Each model is composed using the optimized weak learners from the previous sections and then tuned as a new model using a grid search. The final tuned parameters are explained in the following subsections

❏ Bagging

The bagging regressor uses a set of the same training algorithm for every predictor but train them on different random subsets of the training data, with bootstrapping. The results from each algorithm are then aggregated to form a final prediction. This is accomplished using the bagging algorithm in sklearn with a grid search between each optimized model from the previous section. The tuned parameters are illustrated in the table below.

Model name (sklearn)	**Tuned hyperparameters**
BaggingRegressor	{'base_estimator': SVR(C=113564, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.0007790692366582295, kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False), 'bootstrap': True, 'max_features': 1.0, ‘max_samples': 1.0, 'n_estimators': 7}

SVR used as the weak learner in the aggregate method perform the best on the validation data. The performance metrics of the tuned bagging regressor are displayed in the table below.

Model name (sklearn)	**Tuned RMSE**	**Tuned** r^2
BaggingRegressor	0.19855	0.74745

The results of the bagging regressor, when compared to the weak learners, are underwhelming. Compared to the RMSE of 0.13809 from the best performing weak learner, bagging regression actually shows a 43.8% decrease in performance on the validation set. This can be attributed to the lack of blending of the predicted results from the SVR on the base layer.

❏ Gradient boosting

The gradient boosting algorithm employs multiple decision tree regressors, in series, to predict a SalePrice value. The process is executed as follows. The first decision tree is trained with the test data and predicts the labels. These predicted labels are compared to the true labels, and their difference is calculated. The next regressor is trained on these residual errors and is used to predict the labels, yet again. This process is repeated until specified. The overall gradient boosting prediction on the validation set is simply the sum of all the predictions from each individual decision tree. This method was implemented using scikit-learn and tuned with a grid search. The optimized model is described in the following table.

Model name (sklearn)	**Tuned hyperparameters**
GradientBoostingRegressor	{'learning_rate': 0.1, 'max_depth': 6, 'max_features': 'auto', 'max_leaf_nodes': 10, 'min_samples_leaf': 2, 'min_samples_split': 50, 'n_estimators': 300, 'subsample': 0.9, 'warm_start': True}

The performance metrics of the tuned gradient boosting regressor are displayed in the table below.

Model name (sklearn)	**Tuned RMSE**	**Tuned** r^2
GradientBoostingRegressor	0.11468	0.91971

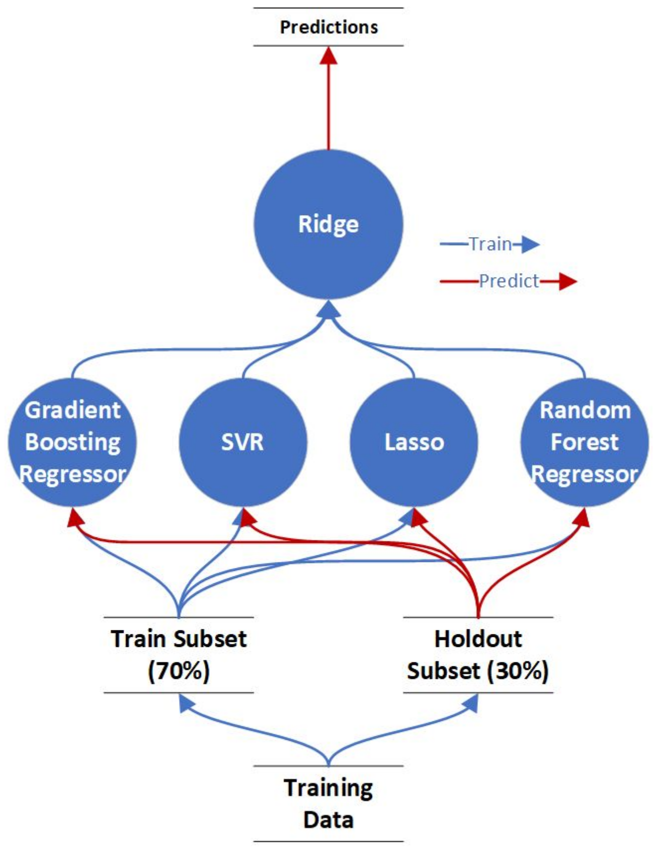
Performance increase on the validation set over the best weak learner is 17.0%. This performance is considerably better than each algorithm shown yet. If the validation set results were to be compared to the Kaggle leaderboard it would end in the top 4% of competitors.

The last aggregation method will combine all of the learning models presented until this point and then blend their predictions to hopefully increase the performance on the validation data.

❏ Stacking

Stacking is an aggregation method that acts similarly to bagging but it can use different predictors in the base layers and it also employs a meta learner to blend the predictions of all the results. Since there is no prepackaged model for stacking available in scikit-learn, one was coded from scratch in Python.

The custom stacking model simply employs a base layer followed by a meta learner to blend the results. A grid search of the tuned weak learners from the previous section is performed to decide the best combination of models to achieve the lowest RMSE on the validation dataset. The following figure illustrates the optimized configuration and method to train the stacking regressor.



The algorithm shown above provides yields the following performance metrics on the validation data.

Model name (sklearn)	**Tuned RMSE**	**Tuned** r^2
Stacking Regressor	0.11450	0.91778

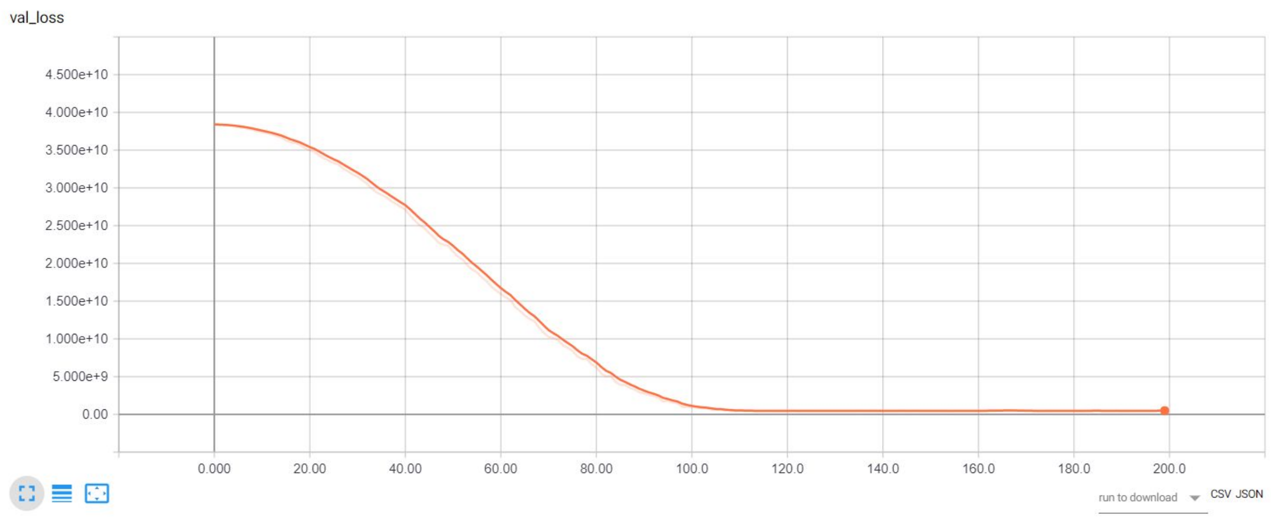
Performance improvement over the best weak learner on the validation set is 17.1%. This model provides the best performance on the validation data. Although its performance is only slightly better than the gradient boosting regressor, its combination of different learning models promises a more generalized model and increase of performance on the testing set.

❏ Neural network

Another algorithm that was used to predict the sale price is neural network. As the data set is very small the neural network used is not very deep. We started with four dense layers and gradually went down to two. We used grid search to train more than one hundred and fifty neural networks using different combinations of neurons in every layer. It looks like there was not enough data to have more than two dense layers. The neural network could not be successfully trained.

Two dense layers with one layer having many neurons and the second layer having just one neuron was better. We again used grid search to find the right number of neurons for the first layer. This time the results were more interesting. The network could fit the data very well. It provided the best results on the validation dataset (it is also small); however, when we predicted the results using test dataset from Kaggle and uploaded the results they were not good. The network did not generalize well. It looks like just by going through a couple of thousand epochs the network could memorize all the data.

To deal with the overfitting with added dropout. The result was a way better generalization. On the test dataset the network didn't produce so impressive results; however, the results on validation set were way better. The number of epochs was limited to 200.



It looks like around 120 epochs is where the training stops and the neural network does not improve in performance. Adding more epochs adds lots of noise and increases memorization. The final architecture of the neural network is shown below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_181 (Dense)            (None, 128)               37376
_________________________________________________________________
batch_normalization_91 (Batc (None, 128)               512
_________________________________________________________________
dropout_91 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_182 (Dense)            (None, 1)                 129
=================================================================
Total params: 38,017
Trainable params: 37,761
Non-trainable params: 256
_________________________________________________________________
```

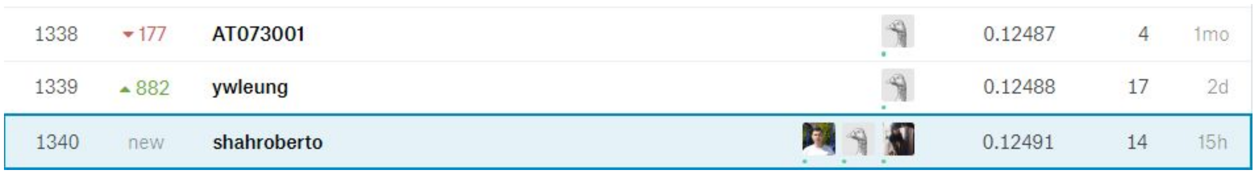
The resulting neural network showed by far the best performance on Kaggle test dataset. It generalized better than other algorithms that we used.

❑ Conclusions

❑ Kaggle leaderboard performance

The three best learning algorithms were used to predict the SalePrice of the test set and the results were submitted to Kaggle. The final leaderboard positions of each model follow:

1. Gradient boosting regressor (0.13247)
2. Stacking regressor (0.13819)
3. Neural network for regression (0.12491, placed in top 28.9%)



1338	▼177	AT073001	0.12487	4	1mo
1339	▲882	ywleung	0.12488	17	2d
1340	new	shahroberto	0.12491	14	15h

For a frame of reference, the top 10 scores are about 0.11 RMSE on the test data.

❑ Summary

This project focused on the exploration of multiple machine learning algorithms using a real-world dataset, with the goal of predicting housing prices from many features that accurately represent considerations when pricing a house. This analysis demonstrated that important features, such as lot size, provide the most weight when considering the price of a house.

Some challenges faced in this report that were not expected was the toughness in getting the model to generalize, especially with the weak learners such as linear regression and support vector regressors. It was noticed that models without regularization performed terribly, as they overfit the training set and predicted validation labels inconsistently. Models with regularization abilities tended to downplay the important features allowing for the models to generalize a bit better. The ensemble methods performed much better, especially those that linked regressors in series. The ability to link regressors in series allowed the overall strong learner to edit the errors of each individual model, while not overfitting the training data. Overall the best performance was demonstrated by the neural network. The networks ability to drop neurons prevented overfitting while allowing for great generalization on the data.

The model results on Kaggle leaderboard showed that good results can be achieved without significantly modifying the dataset. A total of 15 submissions were sent without grossly modifying th the dataset, and each submission ranked within the top 40%.