House Prices: Advanced Regression Techniques

Lei Wang @Oct 2020

Email: ieeewangl@gmail.com

The task is to have the best prediction one can make with the House Prices: Advanced Regression Techniques on Kaggle.

Abstract

The goal of this project was to use EDA, data cleaning and preprocessing, as well as advanced model-ensemble methods to predict house prices given 80 features of houses (data originally available on Kaggle). Ten linear and nonlinear models were evaluated, e.g., Lasso was used to predict sale prices and to show which features are important to the house prices.

Model ensemble can further improve generalizability and robustness over a single model, i.e., to make a more robust model by combining the predictions of several base estimators. Compared with a simple model blending (easy to implement), in which coefficients are empirically chosen for each base model, model stacking (or stacking regressor) enables such coefficients computed at a secondary level model. It thus allows for better performance. However, using all available base models in a stacked model is not a good idea. (This will be demonstrated in the final evaluation section). On the other hand, it is computationally expensive to evaluate all possible combinations of stacked models. Here, we use a heuristic algorithm, sequential forward selection (SFS), aiming to find a (sub)optimal solution for the model stacking. See the figure below.

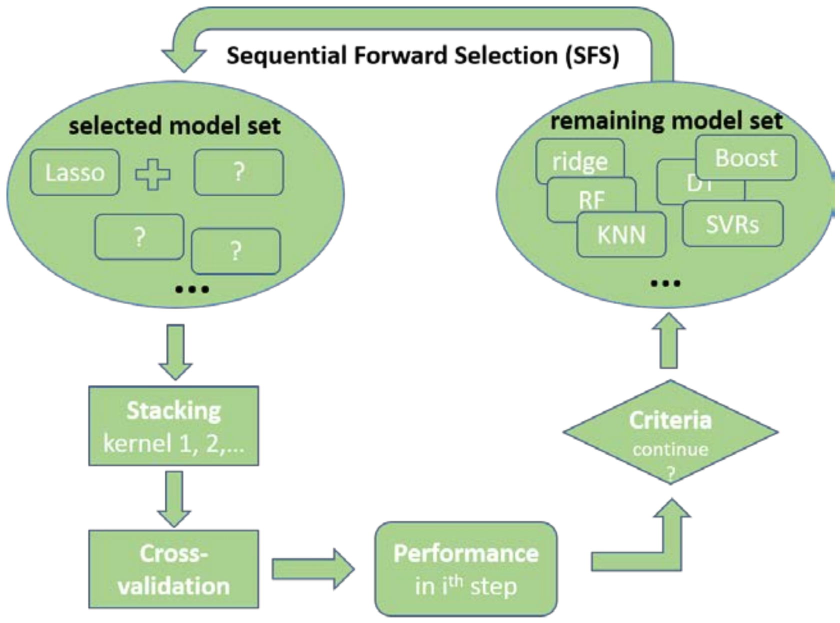


Figure. Illustration of constructing an SFS-model stacking. Here, SFS starts from a base model 'Lasso', then in each step, sequentially adds another base model (for stacking) that mostly improves the stacking performance (i.e., with the lowest RMSE) evaluated by CV, until that all models have been used or it meets the predefined performance criteria.

Contents of Jupyter notebook

For the demonstration purpose, three Jupyter notebooks (also available on my Github page) were used to cover the following three main sections: (1) EDA, (2) data preprocessing, and (3) model evaluation.

1. Exploratory Data Analysis
    - Load data
    - Numerical and categorical features
    - Features with missing values
    - Relation of features to target (sale price)
    - Save cleaned data for further analysis

2. Data preprocessing
    - Load cleaned data
    - Numerical features: reducing skewness
    - Categorical features: converting to numerical (option A and B)
    - Merging numerical and categorical data
    - Standardization train and test data
    - Save preprocessed data for further analysis

3. Model evaluation
    - Load preprocessed data
    - Modeling (linear and nonlinear models)
    - Comparison amongst models
    - Correlation of model predictions
    - Sequential Forward Selection (SFS) model stacking
    - Final evaluation

1. Exploratory Data Analysis (EDA)

A brief introduction

In this EDA part, I will briefly introduce the house price dataset. A more detailed description can be found in the attached Jupyter notebook 'House_Prices_LW_01_EDA.ipynb'.

'House Prices: Advanced Regression Techniques' includes both train (1460 samples) and test (1459 samples) datasets. In the train set, there are 81 columns including 80 features and the target, sale prices (see Fig.1), provided in the last column. In the test set, there are 80 column features without the target house prices given. There are numerical (37) and categorical (43) features, excluding the target sale price.

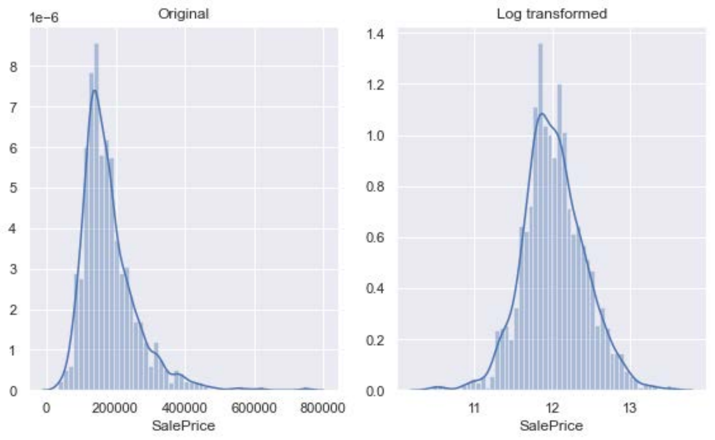


Fig. 1. Visualize the target 'SalePrice' before and after log transform.

Features with missing values

There are more than 10 features with missing values, shown in Fig. 2. However, from the dataset document, we learned that for some features, e.g., PoolQC, NaN is not missing data but means no pool, likewise for Fence, FireplaceQu, etc. Thus, we accordingly change the NaN to None.

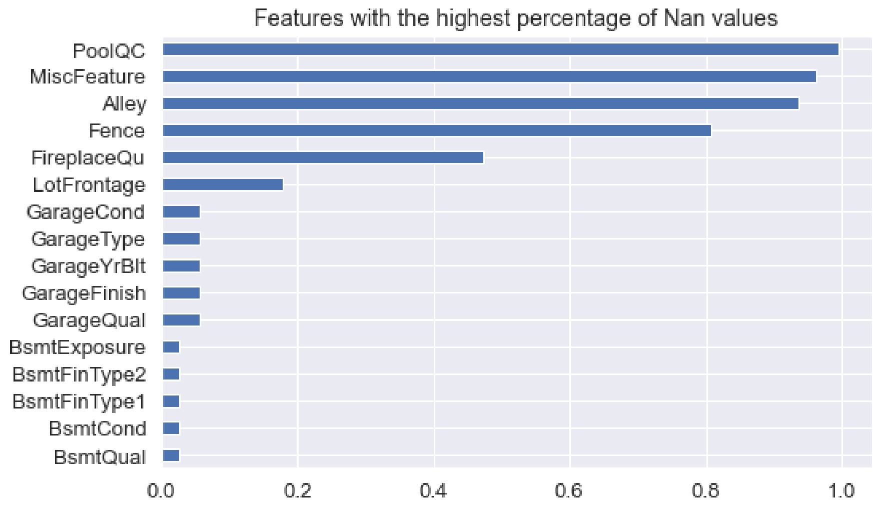


Fig. 2. Features with missing values ranking by the percentage.

Relation of features to target (sale price)

Spearman correlation (a.k.a. Spearman's rank correlation) is more suitable than the Pearson correlation to work within this case because it computes the correlation coefficient on rank values of the data, such that it picks up relationships between variables even when they are nonlinear. Therefore, a log transform of the target will not affect the values of the Spearman correlation. Moreover, Spearman correlation does not assume that data is normally distributed as Pearson correlation does.

Figure 3 shows all numerical features' correlation with the sale price. As expected, the features such as 'OverallQual' and 'GrLivArea' are the most correlated variable, and Fig. 4 shows their relationship in detail. Whereas the categorical features' correlation, in general, is lower than the numerical features.

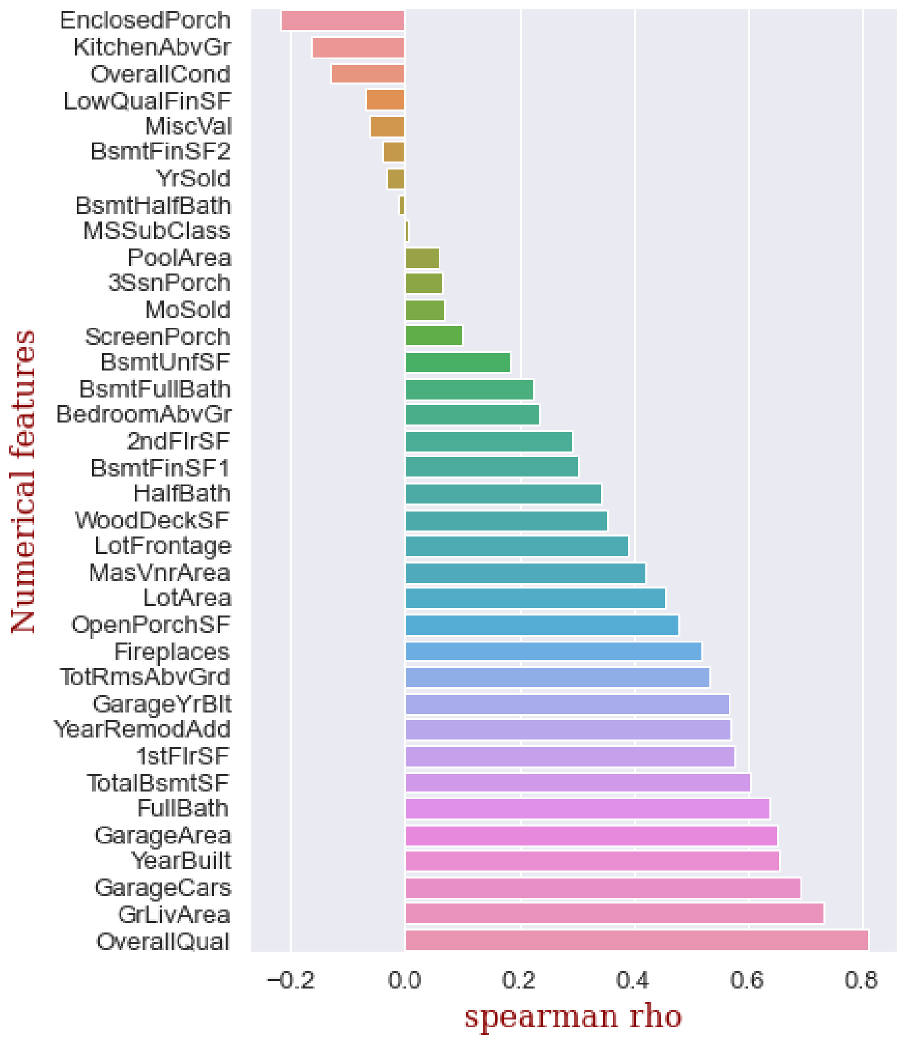


Fig. 3. Numerical features' Spearman correlation rho.

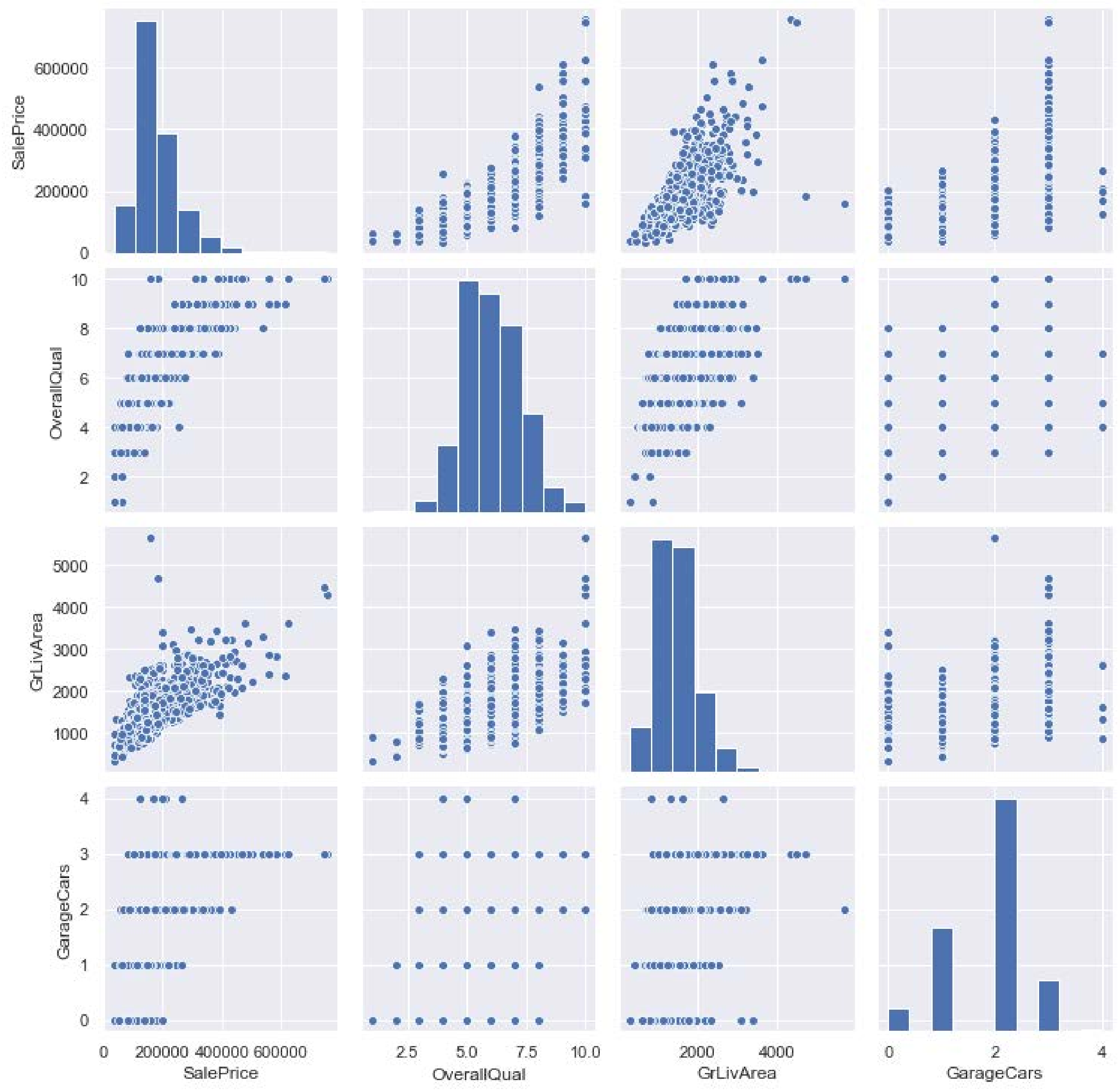


Fig. 4. A plot of relation to target for top-3 numerical features.

2. Data preprocessing

Numerical features

I dealt with numerical and categorical features separately. For numerical features, we reduced the skewness of features. Skewness is a measure of the symmetry in distribution. The symmetrical dataset will have a skewness equal to 0. So, a normal distribution will have a skewness of 0. Skewness essentially measures the relative size of the two tails. As a rule of thumb, skewness should be between -1 and 1. In this range, data are considered sufficiently symmetrical.

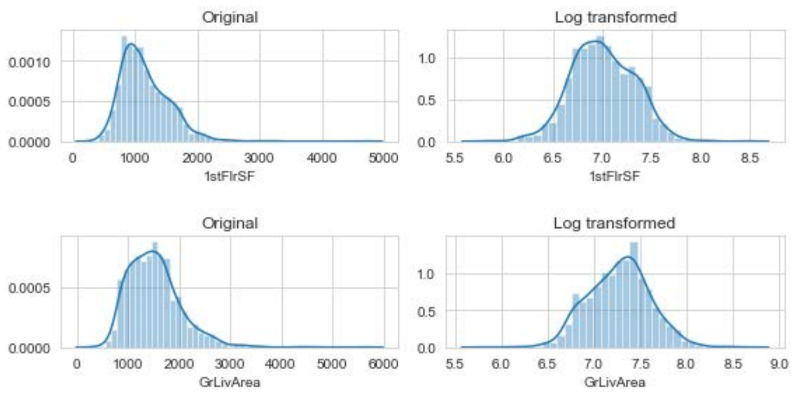


Fig. 5. Two example features before and after log(1+x) transform. The skewness of '1stFlrSF' reduced from 1.38 to 0.08, while 'GrLivArea' reduced from 1.37 to -0.01.

Categorical features

Categorical features need to be converted to numerical values, which is required by most machine learning algorithms. Here, I provided two options for converting categorical features to numerical ones.

- Option A: maintaining the column numbers.
  Each categorical variable will be replaced by its corresponding mean value of the target, i.e., log(saleprice) (see Fig. 6). The advantage of this approach is that it does not change the original column numbers so that it enables easier model interpretation.
- Option B: one-hot encoding category (expanding the column numbers).
  Alternatively, 'one-hot encode' can be used to convert categorical variables into numeric columns (which is required by most Machine Learning algorithms). This means that all (not ordinal) factor values are getting separated columns with 1s and 0s (1 means Yes/Present).

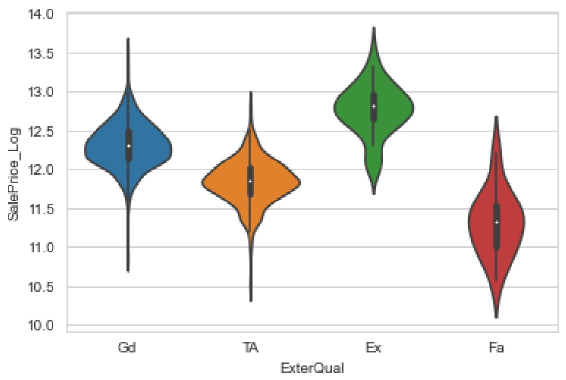


Fig. 6. A categorical variable 'ExterQual' and its subgroups' corresponding mean values of the target log(saleprice).

Standardization

Standardization ($X^{'} = \dfrac{X-\mu}{\sigma}$) is a common requirement for many machine learning estimators. It takes mean removal (to 0) and variance scaling to 1. Note that both the train and test sets were standardized at the same time, so that the same linear transformation (between the train and test sets) was performed for each feature.

The advantages of standardization include:

1) Having features on a similar scale can help the gradient descent converge (as in neural network or boosting algorithms) more quickly towards the minima.
2) Distance-based algorithms, such as KNN, K-means, and SVM, are most affected by the range of features. They will also converge faster after standardization.
3) Compared with normalization (e.g., between 0 and 1), standardization does not affect much by outliers.
4) It facilitates model interpretation (i.e., important features) based on coefficients such as Lasso.

3. Model evaluation

Performance criteria

(1) Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

(2) R-squared or coefficient of determination (R^2)

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)^2}$$

R-squared is the percentage of the dependent variable variation that a linear model explains. R-squared is always between 0 and 100%. The higher, the better.

Single model's performance

Linear and nonlinear models

We employed 10 models in total for this task. The linear models include standard linear regression (least square), Lasso, ridge regression, and SVM regressor (or more precisely, SVR) with a linear kernel (SVM_lin). The nonlinear models include SVM with the polynomial kernel (SVM_poly) and Gaussian kernel (SVM_rbf), decision tree regressor (DT), random forest regressor (RF), nearest neighbors regressor (KNN), and gradient boosting regressor (Boost).

Lasso as an example

$$\hat{\beta}^{Lasso} = \arg\min_{\beta}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{d} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{d}|\beta_j|\}$$

The L1-norm penalty $\lambda\sum_{j=1}^{d}|\beta_j|$ enables some (less important) coefficients β_j to be set to zero (Fig. 7), such that it facilitates model interpretation, meanwhile improves the generalizability (thanks to the reduced model complexity) [1]. In a Bayesian framework, the lasso is mathematically equivalent to a maximum a posterior (MAP) solution of coefficients β_j with a Gaussian prior [2].

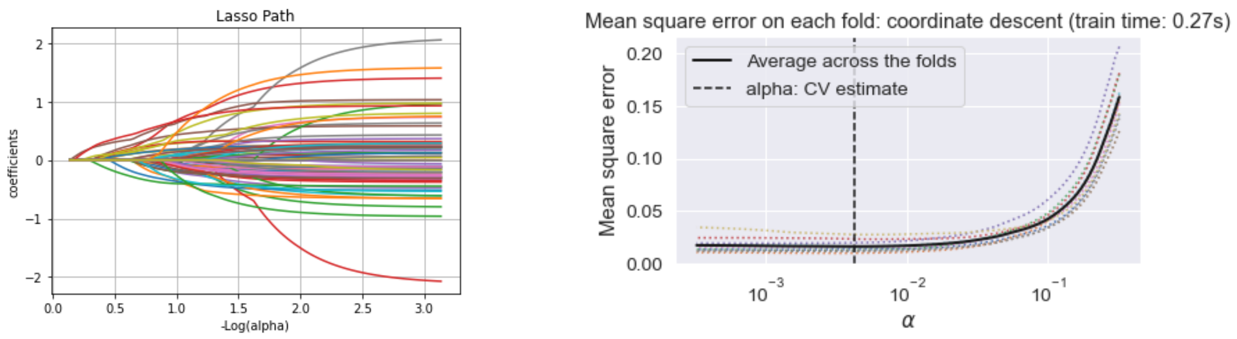


Fig. 7. Left panel: the relation (lasso path) between the penalty term α (i.e., λ in the formula) and coefficients β_j. Right panel: the optimal value α is determined by cross-validation (CV).

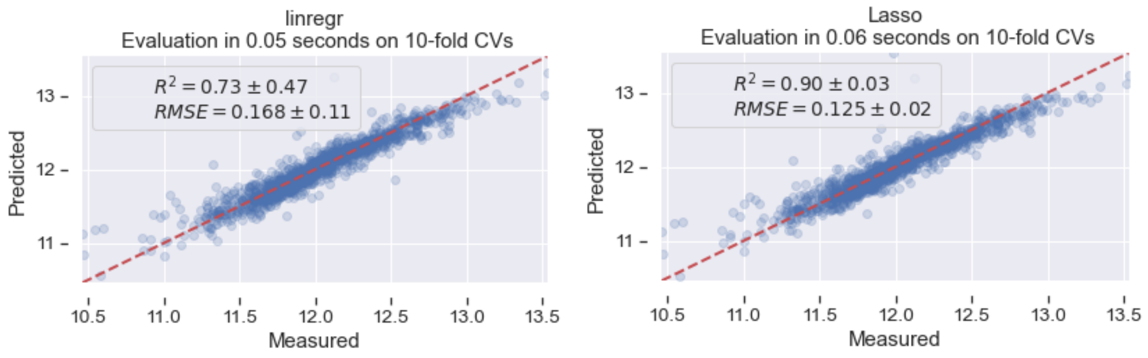


Fig. 8. Model prediction on 10-fold CVs. Left: linear regression; right: Lasso.

As expected, the feature importance ranked by the Lasso model's coefficients is in line with the features' Spearman correlation. Three of the top-5 features are the same as those in Fig. 3.

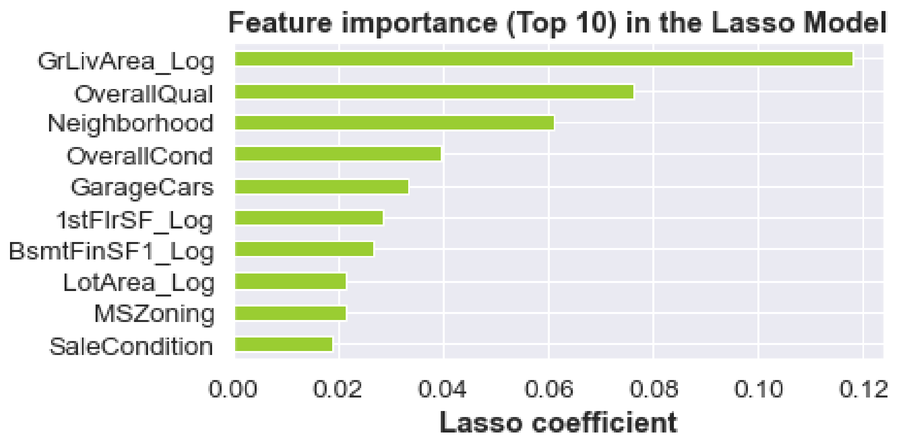


Fig. 9. Important features are identified by the Lasso model's coefficient ranking.

Comparison amongst models

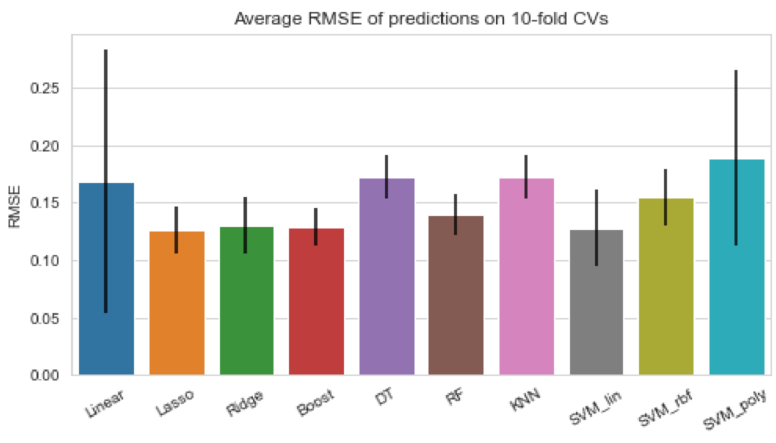


Fig. 10. Prediction performance on 10-fold CV sets of all signal models.

In general, the standard linear regression shows high prediction error and variance. It suggests the existence of 'noisy' features in the training set. (Recall that we used all given features in the preprocessing). Whereas more complex models tend to suffer little influence from 'noisy' features, thanks to regularization such as L1-norm in Lasso and L2-norm in Ridge regression, or model averaging of tree-based algorithms (e.g., random forests).

Another model, SVM with the polynomial kernel ('SVM_poly', degree=3, chosen from CVs), also shows high error (bias) and variance. It is because a 3-order polynomial model might be a biased one for this linear regression problem. Meanwhile, higher model complexity (degree =3) caused higher variance.

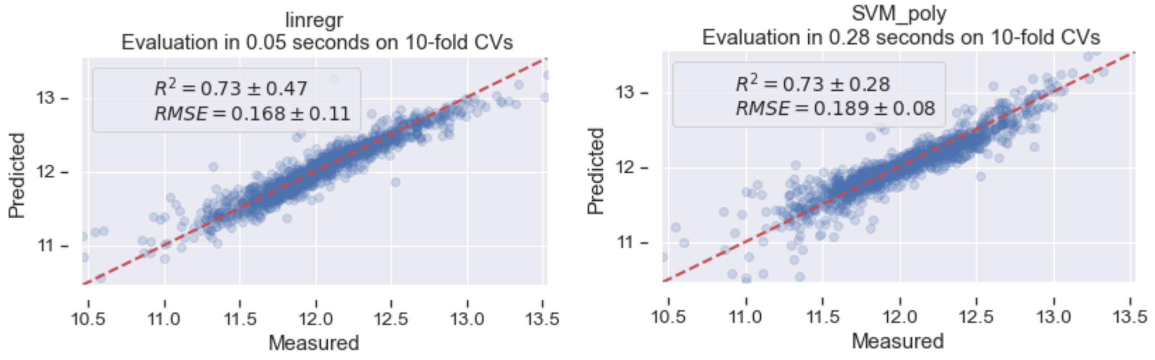
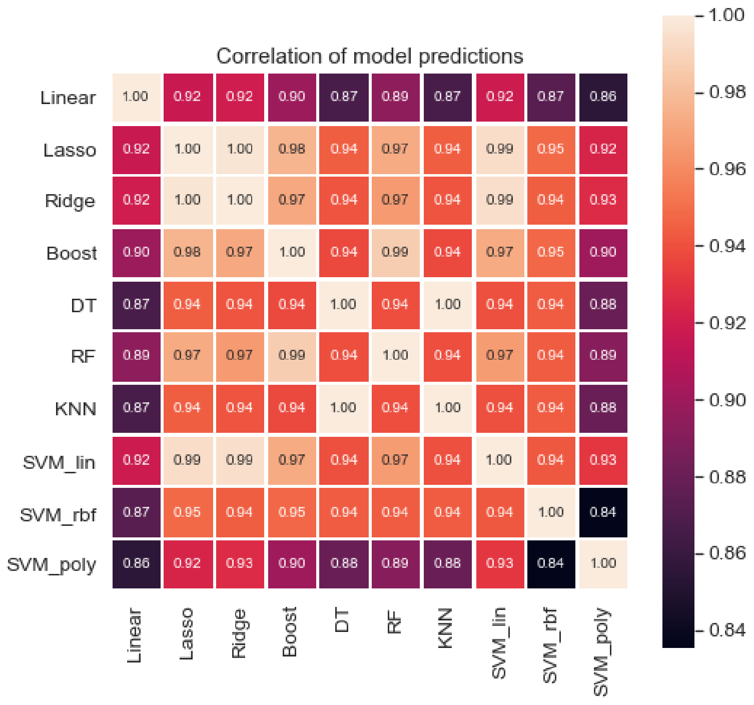


Fig. 11. Model prediction on 10-fold CVs. Left: standard linear regression; right: SVM with the polynomial kernel.

Correlation of model predictions

To help determine which models can be further used for a model ensemble, we checked the correlation of model predictions (see Fig. 12). In general, the correlation amongst linear models (i.e., Linear, Lasso, Ridge, SVM_lin) is higher than those amongst nonlinear models. The correlation between linear models and nonlinear models is lower than 1. Therefore, the model ensemble between linear and nonlinear models will potentially improve performance.



	Linear	Lasso	Ridge	Boost	DT	RF	KNN	SVM_lin	SVM_rbf	SVM_poly
Linear	1.00	0.92	0.92	0.90	0.87	0.89	0.87	0.92	0.87	0.86
Lasso	0.92	1.00	1.00	0.98	0.94	0.97	0.94	0.99	0.95	0.92
Ridge	0.92	1.00	1.00	0.97	0.94	0.97	0.94	0.99	0.94	0.93
Boost	0.90	0.98	0.97	1.00	0.94	0.99	0.94	0.97	0.95	0.90
DT	0.87	0.94	0.94	0.94	1.00	0.94	1.00	0.94	0.94	0.88
RF	0.89	0.97	0.97	0.99	0.94	1.00	0.94	0.97	0.94	0.89
KNN	0.87	0.94	0.94	0.94	1.00	0.94	1.00	0.94	0.94	0.88
SVM_lin	0.92	0.99	0.99	0.97	0.94	0.97	0.94	1.00	0.94	0.93
SVM_rbf	0.87	0.95	0.94	0.95	0.94	0.94	0.94	0.94	1.00	0.84
SVM_poly	0.86	0.92	0.93	0.90	0.88	0.89	0.88	0.93	0.84	1.00

Fig. 12. Correlation (Pearson) amongst model predictions. Note that the standard linear regression shows a correlation of 1 with Lasso and Ridge, thus will be excluded for the further model ensemble.

Sequential Forward Selection (SFS) model stacking

A brief review of the model ensemble

To improve generalizability and robustness over a single model, we employ model ensemble methods, i.e., to combine the predictions of several base estimators and make a more robust one. Compared with a simple model blending (easy to implement) method, in which coefficients are empirically chosen for each base model, model stacking (or stacking regressor) (see Fig. 13) enables such coefficients computed in a secondary level model [1]. It thus allows for better performance.

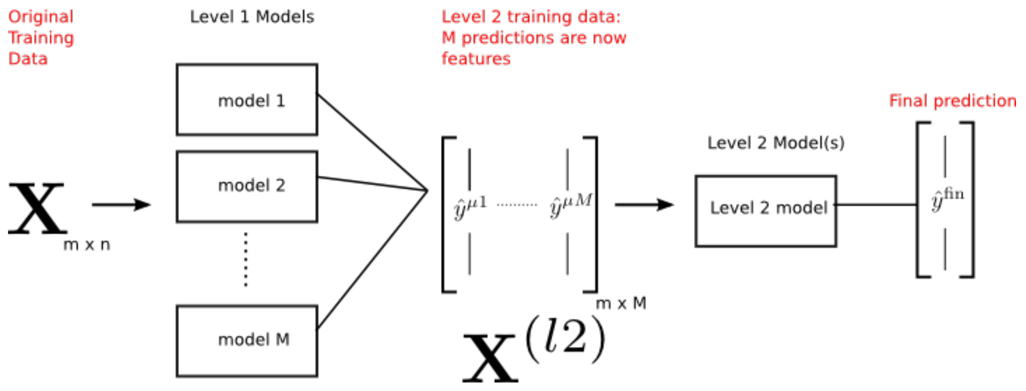


Fig. 13. Illustration of constructing a predictive model by model stacking, i.e., combining different base models' predictions. In level 1, base models are trained on X (using optimal parameters chosen by cross-

validation). In level 2, the final model (a.k.a., meta model) is trained on M predictions from the base models. The optimal parameters of the final model are determined by cross-validation.

SFS model stacking

Using all available base models in a stacked model (or a blending model) is not a good idea. (This will be demonstrated in the final evaluation section). It is because that there might be 'bad' models that we want to exclude, and it needs to be evaluated considering the interaction amongst models. However, it is computationally expensive to evaluate all possible combinations of stacked models. Here, we use a heuristic sequential forward selection (SFS) algorithm, aiming to find a (sub)optimal solution for the model stacking.

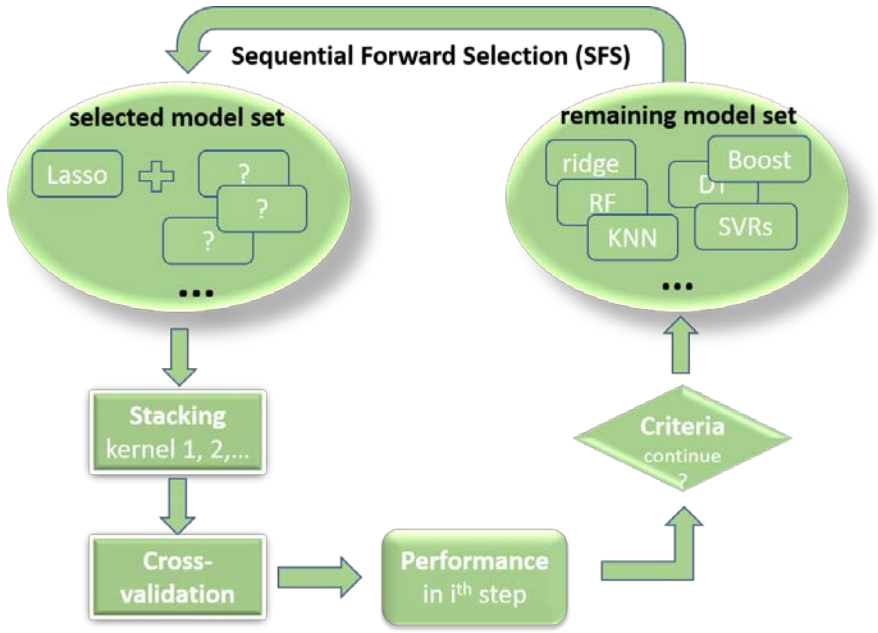


Fig. 14. Illustration of constructing an SFS-model stacking. Here, SFS starts from a base model 'Lasso', then in each step, sequentially adds another base model (for stacking) that mostly improves the stacking performance (i.e., with the lowest RMSE) evaluated by CV, until that all models have been used or it meets the predefined performance criteria.

SFS-model stacking results

Firstly, Lasso was selected as the starting base model for the SFS because it showed the best single-model performance (see Fig. 10). Then, we compared two choices of the final model of stacking: Ridge and Lasso. The detailed results of using Ridge as the final model of stacking is shown in Fig. 15. The comparison between the two final models of stacking is shown in Fig. 16. In which, we also evaluated that SFS starting from the worst base model SVMpoly. The code for the following figure is 'HP_modeling_stacking3.py'.

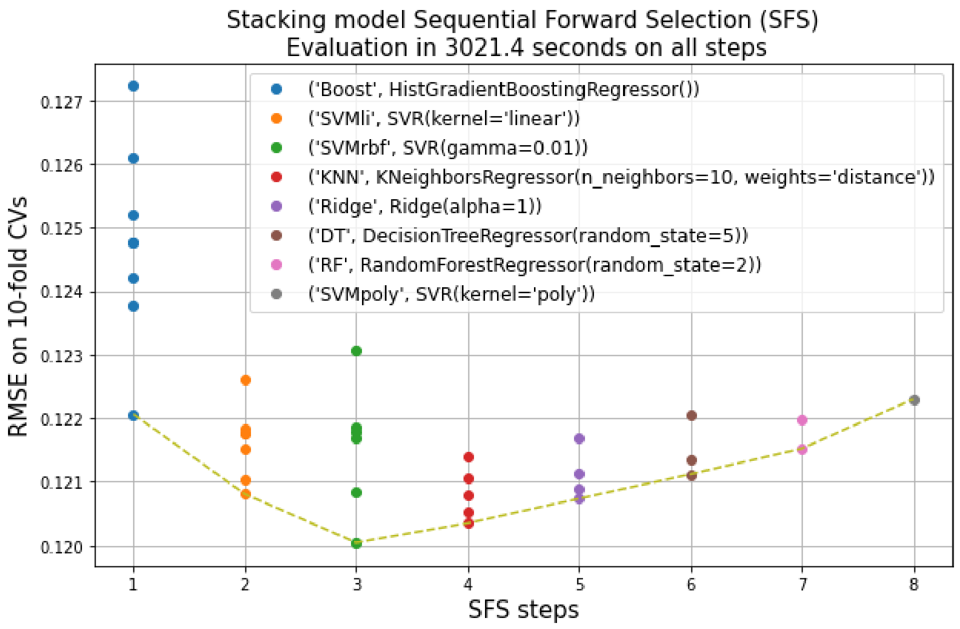


Fig. 15. Illustration of SFS for a series of stacking models (final model = 'RingeCV'). SFS starts from a base model 'Lasso', then on each step, sequentially adds another base model (the dot connected by the dashed line) that mostly improves the stacking performance (i.e., with the lowest RMSE) until that all models have been used. On step 1, the best model 'Boost' is added into the base models of stacking, and on step 2, 'SVMli', etc. The best performance was achieved in step 3, with four base model selected: 'Boost', 'SVMli', 'SVMrbf', and 'Lasso' (the default one).

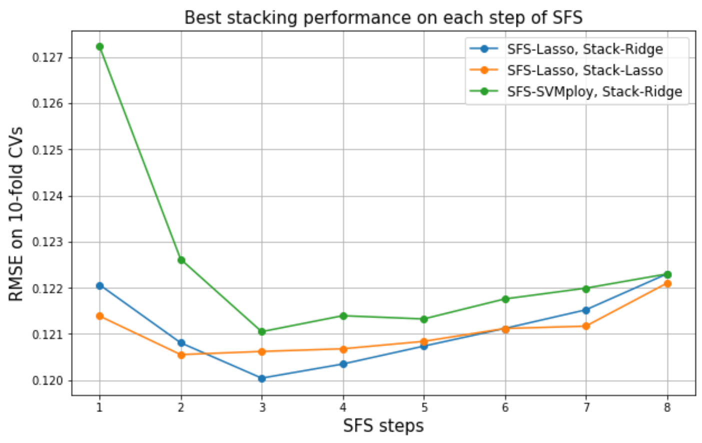


Fig. 16. Comparison of best stacking performance on each step of Sequential Forward Selection (SFS) under different conditions. The blue line (SFS starting from Lasso, and Ridge as the final model of stacking) corresponds to the dashed line in Fig. 15. For the other two models, see Fig. A1 and Fig. A2 in the Appendix for detailed results. @load_SFS_results.py

Figure 16 shows that the best performance was achieved in step 3 with Lasso as the preselected base model and Ridge as the final stacking model. Figure 15 shows that, in addition to 'Lasso', three selected base models are 'Boost', 'SVMli', and 'SVMrbf'.

Figure 17 shows the performance of each single model and the final stacking model.

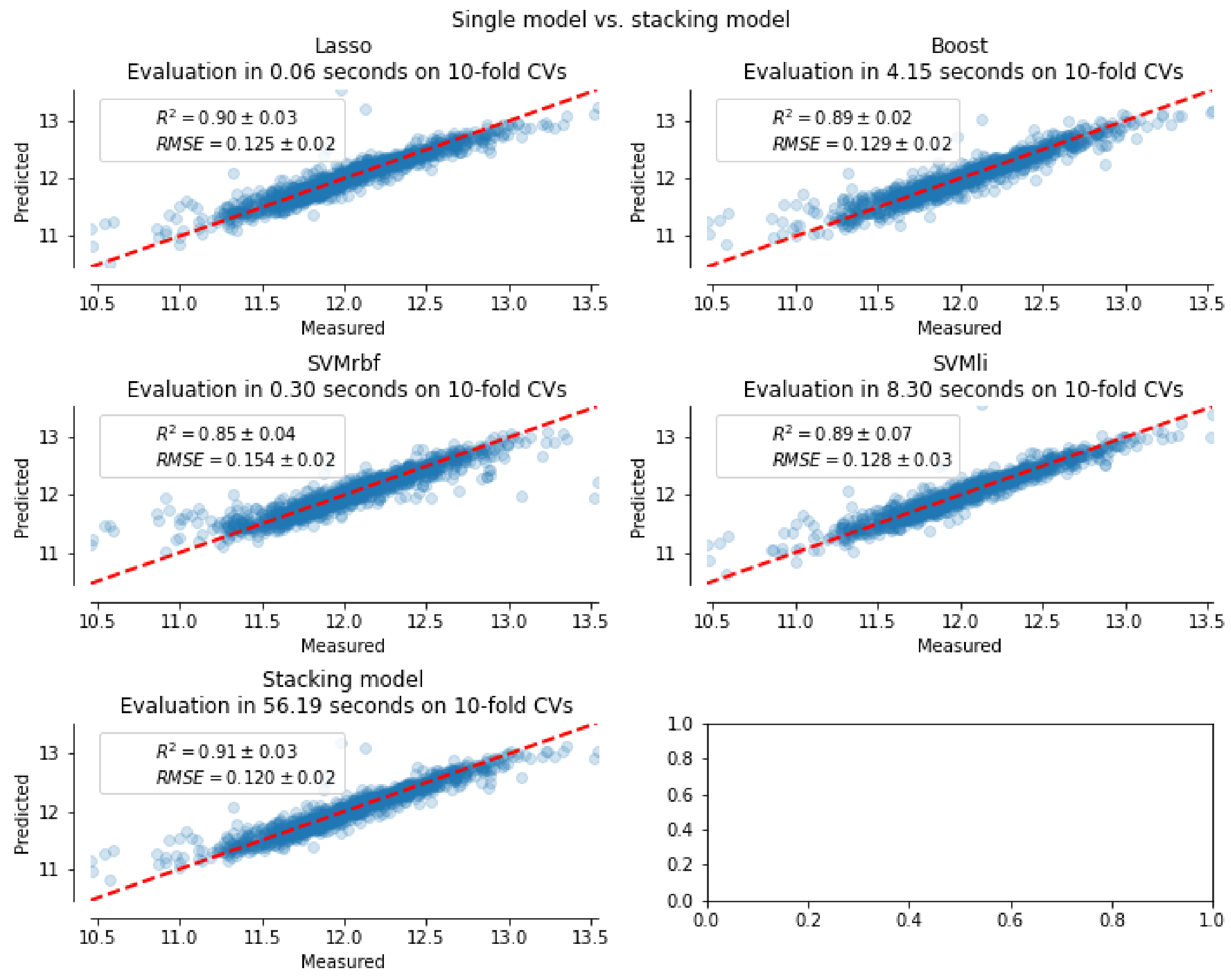


Fig. 17. The stacking model (last row) used the four above base models determined by the SFS-model stacking. These selected base models will 'best' combine the strengths of each other. As expected, the stacking model achieved the lowest RMSE of 0.120.

Final evaluation

The final testing performance evaluated by Kaggle is shown below.

Single best model (Lasso): 0.12725

All model stacking (final = Lasso): 0.12349

All model stacking (final = Ridge): 0.12436

SFS-model stacking (final = Lasso): 0.12284

SFS-model stacking (final = Ridge): 0.12264 (best)

Conclusion

SFS-model stacking (using four selected base models) has achieved the best test performance. Using all available models as base models did not yield better performance.

Discussion

The SFS algorithm does not guarantee an optimal solution, but a suboptimal one. We started from the base model that show the best single-model performance, and sequentially added other base models. In practice, if the performance does not further improve, the searching can stop earlier. In this way, it enables a stacked model with low complexity (fewer base models) but reaching the desired performance (not necessarily the best). According to the Occam's razor principle, such a model could be the closest to a real one (with good interpretability).

In this report, results were based on the converted category features by using option A: maintaining the column numbers. The one-hot encoding category features (option B) were not reported here, because in general, they yield worse performance.

Feature engineering is the key to further improve performance, but it usually requires domain knowledge. For this task, the prediction performance (on CVs of train set) seems to be biased for those houses with the lowest and highest prices, this might be partly fixed if previously excluding outliers (i.e., sale prices much lower or higher than usual) in the training data. In addition, appropriate feature combination, e.g., X = f(x1, x2, …) with f() as a linear or a quadratic function, also potentially improve the performance.

Appendix

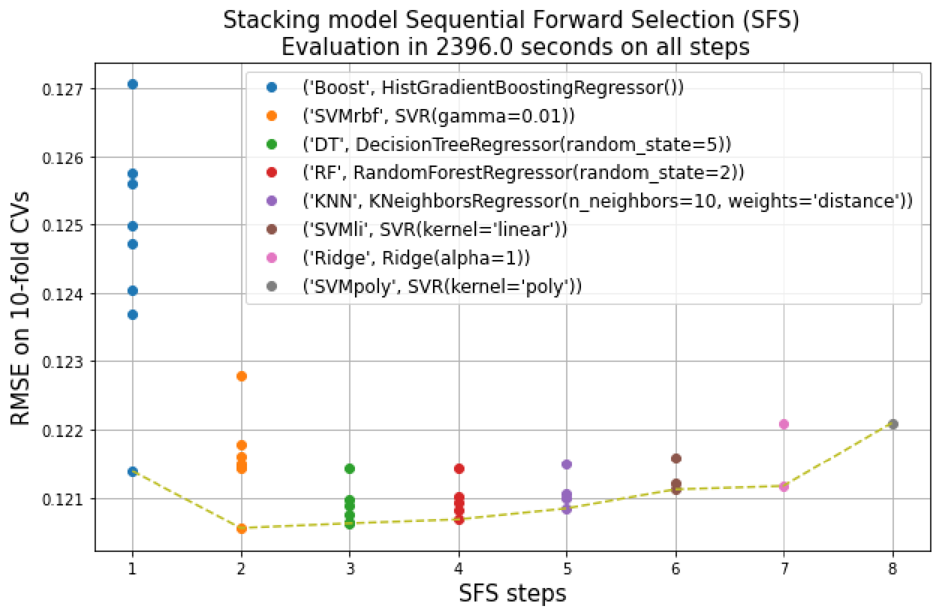


Fig. A1. Illustration of SFS for a series of stacking models (final model = 'LassoCV'). SFS starts from a base model 'Lasso', then on each step, sequentially adds another base model that mostly improves the stacking performance (i.e., with the lowest RMSE) until that all models have been used. On step 1, the best model 'Boost' is added into the base models of stacking, and on step 2, 'SVMrbf', etc.

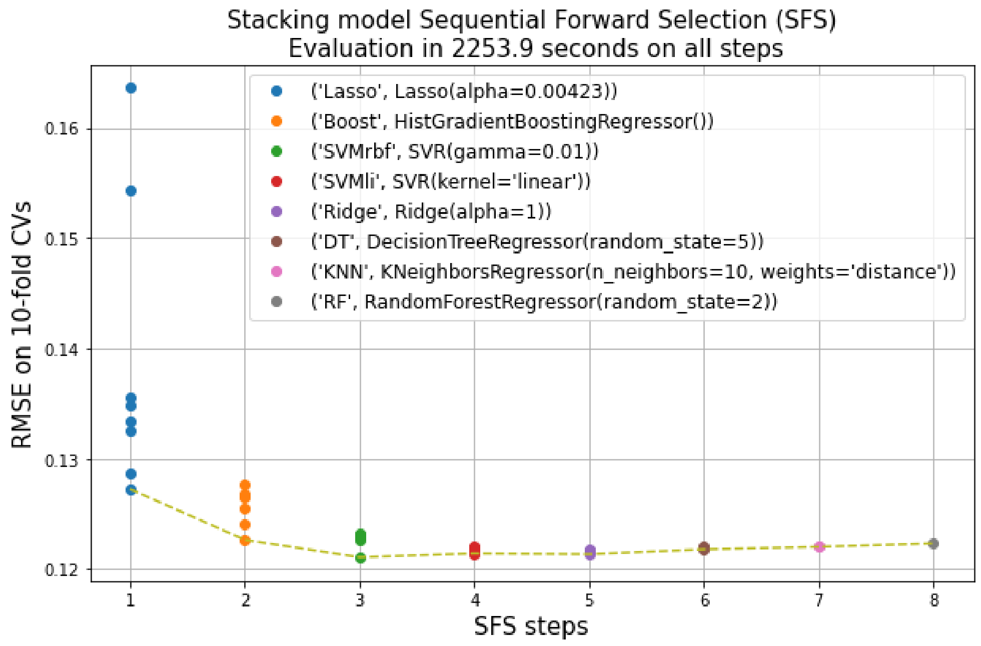


Fig. A2. Illustration of SFS for a series of stacking models (final model = 'RingeCV'). SFS starts from a base model 'SVMpoly', then on each step, sequentially adds another base model that mostly improves the stacking performance (i.e., with the lowest RMSE) until that all models have been used. On step 1, the best model 'Lasso' is added into the base models of stacking, and on step 2, 'Boost', etc.

References

[1] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning: data mining, inference, and prediction, Springer Science & Business Media, 2009.
[2] K.P. Murphy, Machine learning: a probabilistic perspective, MIT press, 2012.