CSCI 567 - Machine Learning - Spring 2021

Project: Checkpoint 2 (05/03/21)

Team StochasticResults:
- Abel Salinas
- Angel Nieto
- Misael Morales

Youtube link: https://youtu.be/p4We5DwuleQ

See figures.pdf for figures.

Regression Modeling Experiments

We start by clearly defining the training ("train") and testing ("test") data set. The training data set comes from our data preprocessing, where we selected the features with the highest correlation and therefore the most relevant to the regression modeling. We will use these main features to predict the (log10) SalePrice ("target") of the Housing dataset from the test set. While we used several metrics to understand the performance of our models, for the purpose of presenting a consistent metric, we will compute the cv_rmse on every model.

Comparing all regression models

Model	Test RMSE Loss	Detail
Linear	0.2326	Figure 1
Ridge	0.2324	Figure 2
Lasso	0.2194	Figure 3
ElasticNet	0.2194	Figure 4
Gradient Boosting	0.2371	Figure 5
Random Forest	0.2367	Figure 6
Linear SVM	0.2436	Figure 7
RBF SVM	0.2371	Figure 7
Neural Network	0.1734	Figure 8
Stacked Regression	**0.0684**	Next page

See Figure 9

Concluding Remarks

- Remarks
  - After our data preprocessing techniques, most regression algorithms will provide good predictions for the log10(SalePrice) target variable using the most high-correlated feature variables.
  - Fully-Connected Neural Network with nonlinear activations provides really good MSE (since we are optimizing with this loss function - but simpler regression methods like regularized regression provide a better distribution of the test predictions.
  - Further sensitivity analysis and hyperparameter tuning can provide improved testing accuracy compared to the standard techniques deployed.
- Conclusion
  - The SalePrice target variable from the Ames Housing data set can be easily predicted after detailed data preprocessing and advanced regression techniques.
  - Preprocessing and data wrangling is crucial for exploiting the full potential of all features.
  - Most regression algorithms will work to provide acceptable predictions.
  - Regularization helps in further discriminating the most important predictor features.

Best Model: Stacked Regression

Stacking (short for stacked generalization) is an ensemble method which uses trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble. The following figure shows such an ensemble performing a regression task on a new instance.

See Figure 10

Each of the bottom predictors predicts a different value and then the final predictor (called a blender or a meta learner) takes these predictions as inputs and makes the final prediction.

Next, the first layer's predictors are used to make predictions on the second set like in the previous figure. This ensures that the predictions are "clean" since the predictors never saw these instances during training. For each instance in the validation set, there are three predicted values. We can create a new training set using these predicted values as input features (which makes this new training set 3D), and keeping the target values. The blender is trained on this new training set, so it learns to predict the target value, given the first layer's predictions.

See Figure 11 & 12

It is actually possible to train several different blenders this way (e.g., one using Linear Regression, another using Random Forest Regression), to get a whole layer of blenders. The trick is to split the training set into three subsets: the first one is used to train the first layer, the second one is used to create the training set used to train the second layer (using predictions made by the predictors of the first layer), and the third one is used to create the training set to train the third layer (using predictions made by the predictors of the second layer). Once this is done, we can make a prediction for a new instance by going through each layer sequentially.

See Figure 13

To avoid the tedious work to find the best hyperparameters for every base model and for the meta-model manually we used Grid Search and Random Search to automatize fine tuning.

For the project purposes splitting our training set into 10 folds gave us the best result.

Concluding Remarks

- Remarks
  - Preprocessing choice is not as important to get higher accuracy, but it is an essential step to run our models and to get acceptable predictions.
  - Using fine-tune base models is more important than adding more base models.
  - After many experiments combining and using different "simple" regression methods as base models along LGBMRegressor or XGBoosting meta-models result in the best prediction.
  - Using PCA trained data from highly correlated features or simply discarding low correlated features produced a lower RMSE we assume the model is losing important information which low correlated features are providing to our best model.
  - Fix features skewness and remove outliers proved to be relevant to produce a higher accuracy.
  - Increasing the number of folds up to 10 showed to achieve higher accuracy.
  - Increasing the number of folds further showed no impact on accuracy.

- Conclusion
    - Fully understand the data set and different regression techniques are the capstone to get a highly accurate model.
    - Most regression algorithms provide acceptable predictions, but using the best quality of each method and stacked all predictions in a single model provided the best result for our project.

Feature Engineering Conclusion and Main Takeaways

Different Models Need Different Things
Through our testing and research on many different types of models, it was interesting to observe the differences between the different types of models. Some feature engineering strategies worked great for some models while reducing performance for others. Overtime, we learned the weaknesses and strengths of each model. Through parameter tuning and changes to our preprocessing, we could experiment on one model and later observe how the changes might affect the overall stack model or how the preprocessing would affect other individual models.
As we observe from the graph below, while some feature engineering steps proved to be helpful across the board, different models benefited from different strategies. Some perform better with dummy variables while others prefered an encoding strategy. SVM RBF seemed to work best with feature selection. Every model seems to have its own complexities and designing the preprocessor to easily change its functionality through keyword arguments allows us to iterate quickly as we test different strategies.

See Figure 14 & Figure 15

Neural Networks are not Always the Best Solution
When researching machine learning, it is common to hear about neural network-based strategies for solving machine learning problems. As a result, our team spent a lot of time, especially in our initial experiments, trying to tune a neural networks solution to achieve results. It was only when we opted to explore other machine learning models that we found significant improvements in our score. While neural networks might be optimal in other tasks, there are many strategies for tackling machine learning problems that might have equal or even better results. By having an open mind and a willingness to experiment, we can achieve great results.

There is no Magic Model, Machine Learning is an iterative process
While experimenting for this project, we often found ourselves often hoping to ourselves that this change will be the magic step that will drastically improve our models performance, whether that change be using a new model, different feature engineering techniques, or some parameter tuning. We found that while our changes would affect the model's performance, the performance difference would be small. We learned overtime that machine learning projects are not about getting lucky and finding that perfect model for your dataset. Instead, it is an iterative process, slowly making changes and experimenting to find better results over time. Through an amalgamation of changes such as improving your feature engineering, trying different models, tuning parameters, we will eventually see the results we are looking for. This is a valuable lesson that we are excited to use as we work on more complex machine learning tasks.

References

Bescond, Pierre-Louis. “Cyclical Features Encoding, It’s about Time! - Towards Data Science.” Medium, 1 Feb. 2021, towardsdatascience.com/cyclical-features-encoding-its-about-time-ce23581845ca.

Brownlee, Jason. “How to Choose a Feature Selection Method For Machine Learning.” Machine Learning Mastery, 20 Aug. 2020, machinelearningmastery.com/feature-selection-with-real-and-categorical-data.

Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O’Reilly Media, Incorporated; 2019.

Mazzanti, Samuele. “Boruta Explained Exactly How You Wished Someone Explained to You.” Medium, 12 Feb. 2021, towardsdatascience.com/boruta-explained-the-way-i-wish-someone-explained-it-to-me-4489d70e154a.

Rençberoğlu, Emre. “Fundamental Techniques of Feature Engineering for Machine Learning.” Medium, 3 Apr. 2019, towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114.

PyData. “Daniel Chen: Cleaning and Tidying Data in Pandas | PyData DC 2018.” YouTube, uploaded by PyData DC 2018, 4 Jan. 2019, www.youtube.com/watch?v=iYie42M1ZyU&t=2168s.

CSCI 567 - Machine Learning - Spring 2021

Project: Appendix - Data Preprocessing (05/03/21)

Team <u>StochasticResults</u>:
- Abel Salinas
- Angel Nieto
- Misael Morales

See figures.pdf for figures.

Regression Modeling

Linear Regression
We build a simple linear regression to predict SalePrice as a function of a linear combination of the highly-correlated features in the dataset. Following that, we use this linear model to predict the test set, and compute the resulting MSE.

See Figure 1

We see that linear regression proves to be a sufficient algorithm to predict the SalePrice; however, further regularization can help with overfitting/underfitting issues. Moreover, cross-validation can also help in finding the best regularization parameters to predict the target.

Regularized Regression
Adding onto the linear regression, we use Ridge, Lasso, and ElasticNet regression to predict our SalePrice target, and compare the resulting MSE.

To all of these methods, we apply the automatic Cross-Validation algorithm from sklearn.

- Ridge regression, also known as Tikhonov regularization, applies and l2 penalty to the linear regression to reduce large deviations.
- Lasso regression, also known as shrinkage, sparsifies the model and reduces small values to 0 to try and improve the prediction.
- ElasticNet regression is the combination of Ridge and Lasso penalty terms into a single regularized linear regression model.
-

With Cross-Validated regularized regression, we see a good improvement from the simple linear regression. These techniques further exploit the most relevant features in predicting the target variables by shrinking or putting less weight on the less-correlated features. Here, we have used Leave-One-Out Cross-Validation (LOOCV) in order to find the best hyperparameters for the regularization terms in the regression models.

See Figure 2-4

Boosting
Gradient Boosting builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

See Figure 5

Random Forest
A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. This technique will help significantly with overfitting issues by using an ensemble of weak-learners to predict the target variable from the features.

See Figure 6

Support Vector Machines
We implement support vector machines as a regression algorithm in two ways: i) with linear kernels, ii) with radial-basis-function (gaussian) kernel. The first will help compare more closely to our previously implemented linear regression models, while the second will produce an improved support vector regressor for the given features and the target variable. We ensure to standard-scale the data so that the kernels are equivalent, and use slack variables and tolerances for the optimization algorithm to converge properly.

See Figure 7

Neural Network

Neural Networks provide enhanced regression power compared to other algorithms, as they can capture the nonlinearity in the data more accurately, and provide higher flexibility in terms of the weights used for the correlation of the hidden layers and variables. Here, we use a fully-connected neural network from the high-correlated training set to the target variable. We do not need to use dropout regularization since the network is relatively simple and still able to capture the relationships between the features and predictor. We also experimented vastly with hyperparameters such as learning rate (or step size selection in the optimization algorithm), the number of epochs, and other parameters. By using a validation split in the training we also control for overfitting, and by selecting the best number of epochs we avoid overfitting as well.

See Figure 8

Further Exploration

While the sections above provide a comparison of several machine learning models, in addition to an overview of our best model, we conducted much more comprehensive experiments throughout the semester. These experiments include, but are not limited to, feature engineering, feature selection, and alternative models and model compositions.

Feature Engineering

Our initial feature engineering implementation was enough to convert the input csv into a form that models can train on. This is achieved by converting categorical variables into numerical variables. While this implementation clearly worked well, there is room to experiment in hopes of achieving better results.
Our improved preprocessor provides more comprehensive preprocessing of the housing data. In addition to creating mappings for some categorical variables, we

- Engineer Custom Features
- Drop less useful data
- Alternate Strategies for dealing with missing data
- Use of dummy variables (Or Label Encoders) for remaining categorical variables

See Figure 16

This graph above compares the preprocessing method in section 6 with our new preprocessing method. The purple bar is the barebones preprocessor specified in section 3, which merely merely encodes all categorical variables. The red bar is the strategy from section 6 which uses the 8 highest correlated features. The green and red bars use our new preprocessing method. While the green bar's method uses dummy variables to convert categorical variables we felt could not be mapped to numerical values, the blue bar's method merely uses a label encoder. The dummy variable strategy prevents the model from assuming relationships between encodings. For example, if we used the following label encoding for Neighborhood:

CollgCr = 1; Veenker = 2; Crawfor = 3; NoRidge = 4;

The model might assume that CollegeCr is more similar to Veenker than NoRidge since their values are close together. The solution is to use dummy variables. This creates a new column for each category, assigning a 1 or a 0 if that sample is from said category. The downside of using dummy variables is that the number of features grow significantly. Our dummy variable strategy created a training set with 255 features while our label encoding strategy only has 90. As we can see, using either feature engineering strategy improves our performance on all models seen above. For most models, the dummy variable preprocessing strategy performs best, with the exception of svm_rbf (and rf, though the difference in rf performance is minimal). This tells us that, given the provided parameters for our model, most of our models were able to find useful patterns in the large number of features. SVM RBF seems to perform better with a smaller number of features, which we can test with slight modifications to our preprocessor. First, we check if dropping the columns used for feature engineering affects performance.

See Figure 17

Not dropping the redundant columns reduces performance for most models, though the performance drop is mostly minimal with the exception of svm_rbf. Interestingly, we see a slight increase in performance to the boost model, though the increase is so small we consider it negligible. Since our models seem to function well despite useless features, we see if including both dummy variables and label encoded versions of categorical variables affects performance.

See Figure 18

While we see no improvement, other than a negligibly small improvement to our boost model, the strategy of using both dummy variables and label encodings works as well as the best of the two strategies on their own, with the exception of SVM RBF. Based on our first test, this further shows that SVM RBF performs best with fewer features.

Feature Selection

We further experimented with feature selection to see if they would provide solutions to our models. Since we hypothesized above that SVM RBF performs better with fewer features, we will only show results of SVM RBF's performance with our selected features.

Boruta

As predicted, reducing the number of unhelpful features improved the performance of SVM RBF. In this case, our strategy for selecting features. This feature selection is handled through the boruta method. Boruta works by concatenating randomized columns of each variable to our training dataframe and using a random forest to filter out any training variables that perform worse than the best randomized column. We give a variable a "hit" in each iteration of the Boruta method where said variable is not filtered out. While it is impressive that only 12 features can achieve such an accuracy, the final accuracy is still worse than all other models tested.

See Figure 19

K-Best Feature Selection

This section uses the sklearn feature selection class to attempt to improve SVM RBF's performance. While the model performs best at only 20 features, the model performs worse with the feature selection at 20-220 features than it does at 255. In this experiment, Sklearn's SelectKBest algorithm does perform well on our data as it makes our SVM RBF model perform worse with many of its selections.

See Figure 20

Figure 1

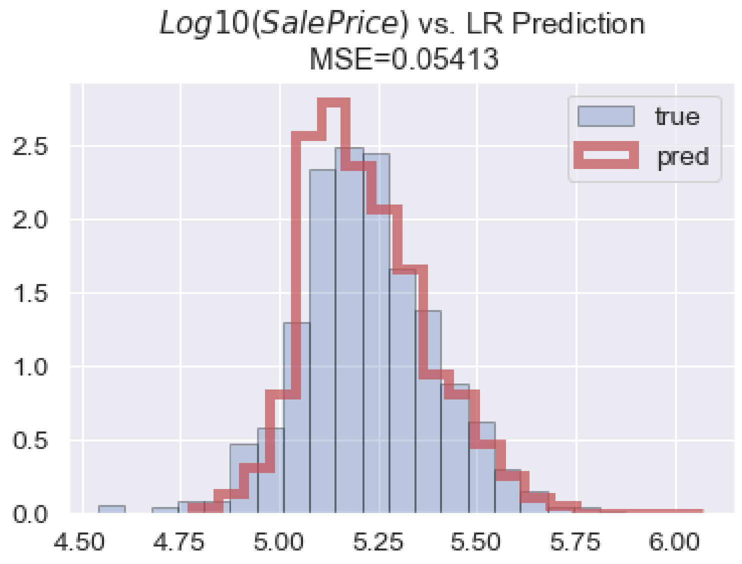


Figure 2

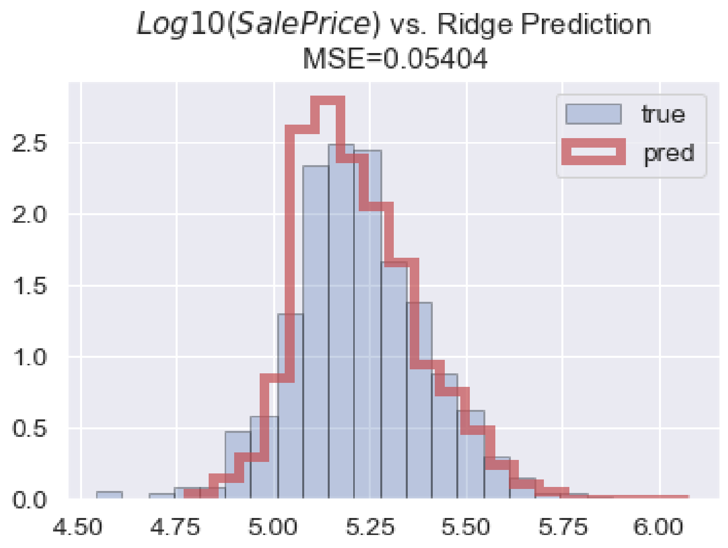


Figure 3

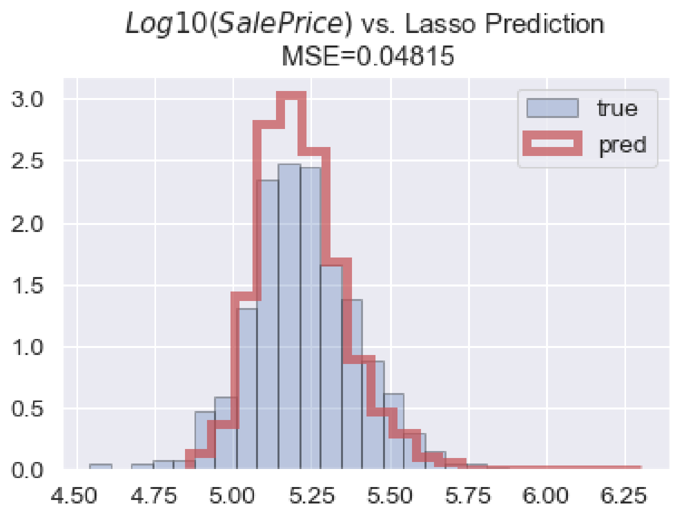


Figure 4

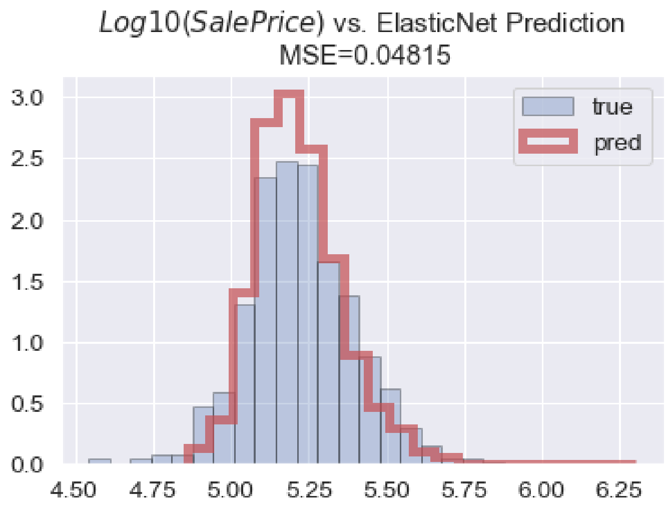


Figure 5

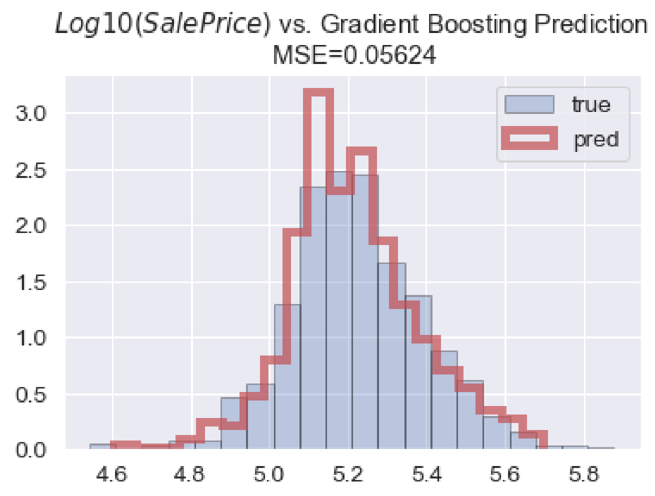


Figure 6

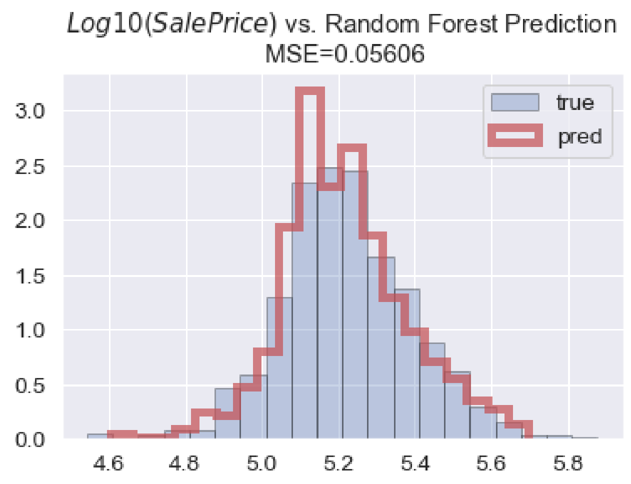


Figure 7

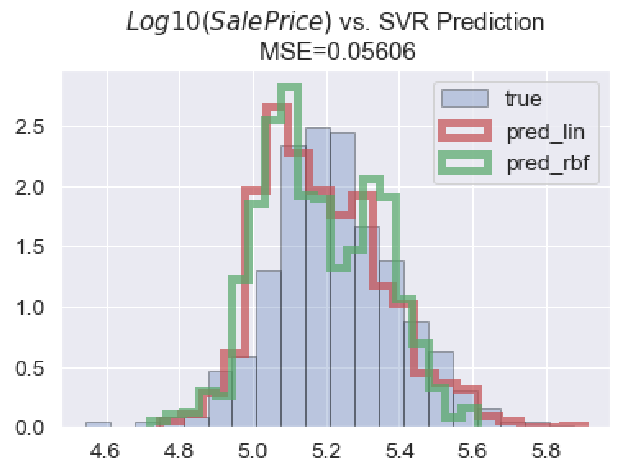


Figure 8

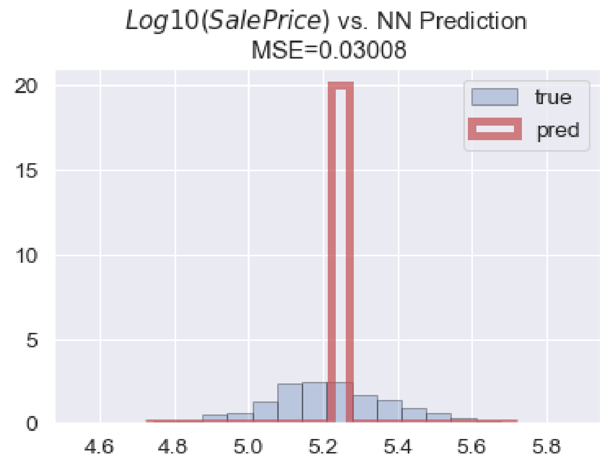


Figure 9

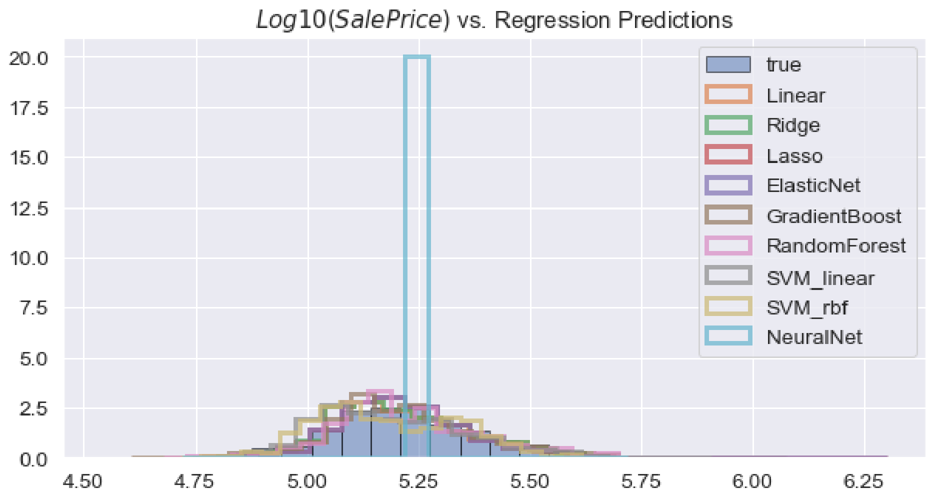


Figure 10

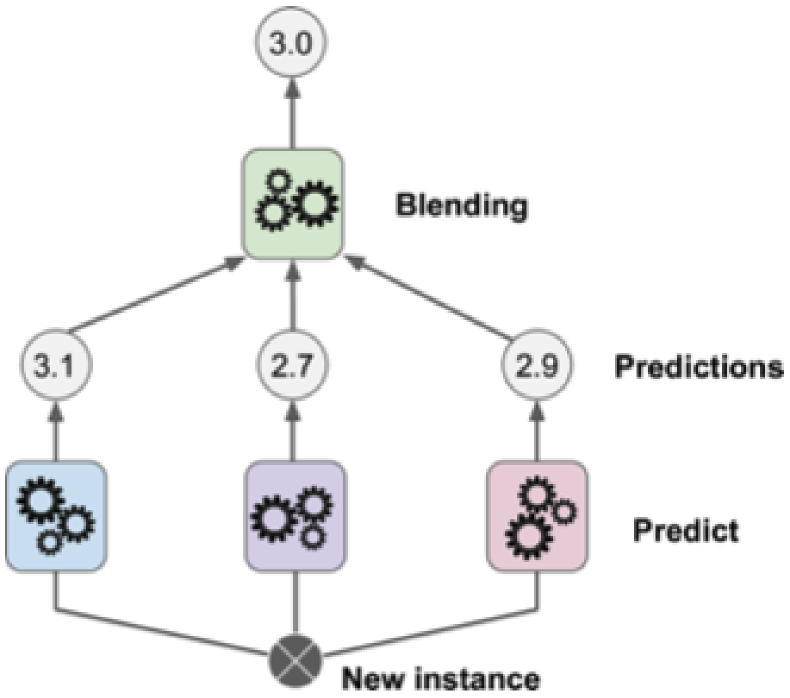


Figure 11

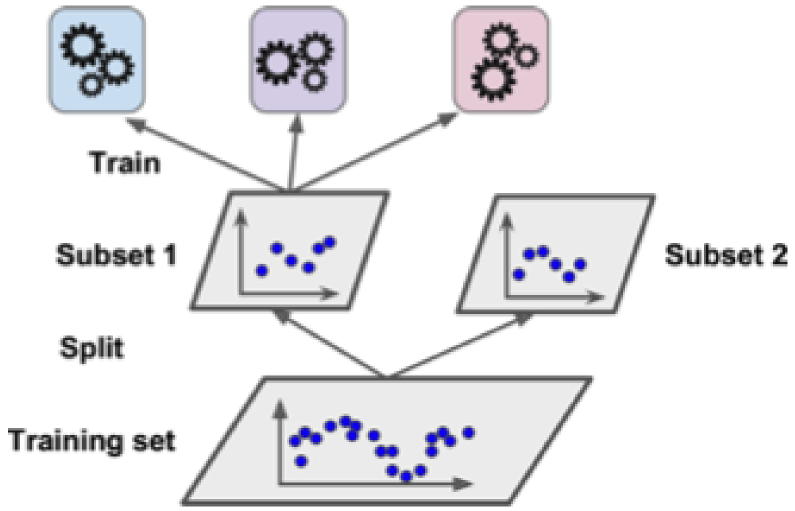


Figure 12

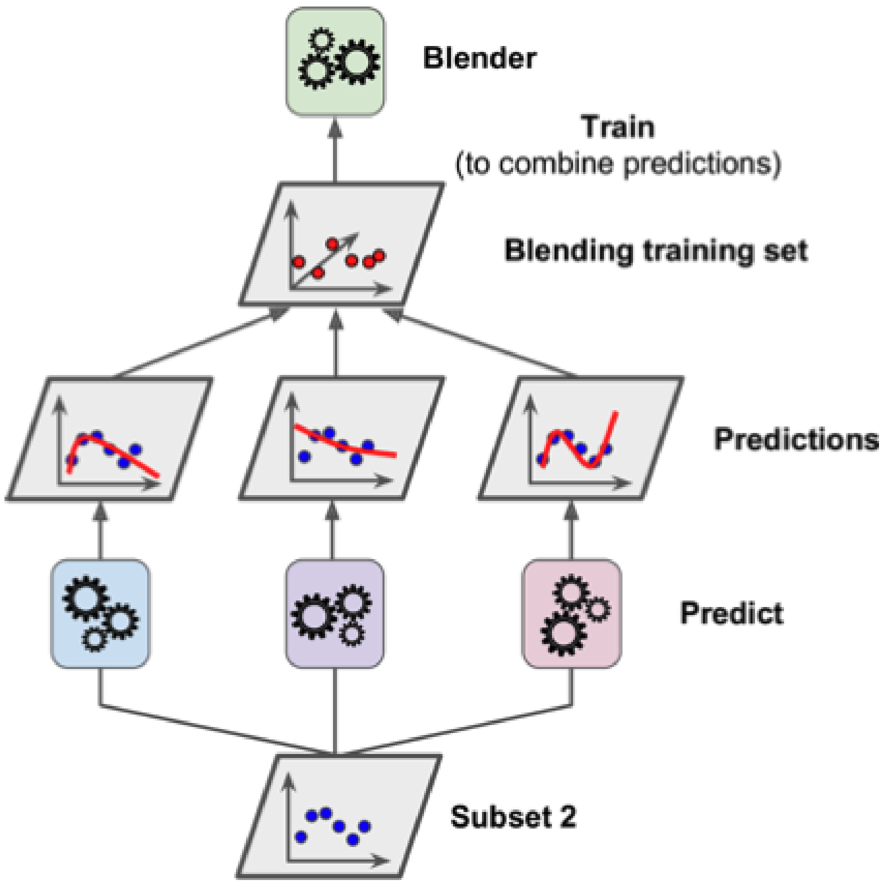

Blender
Train
(to combine predictions)
Blending training set
Predictions
Predict
Subset 2

Figure 13

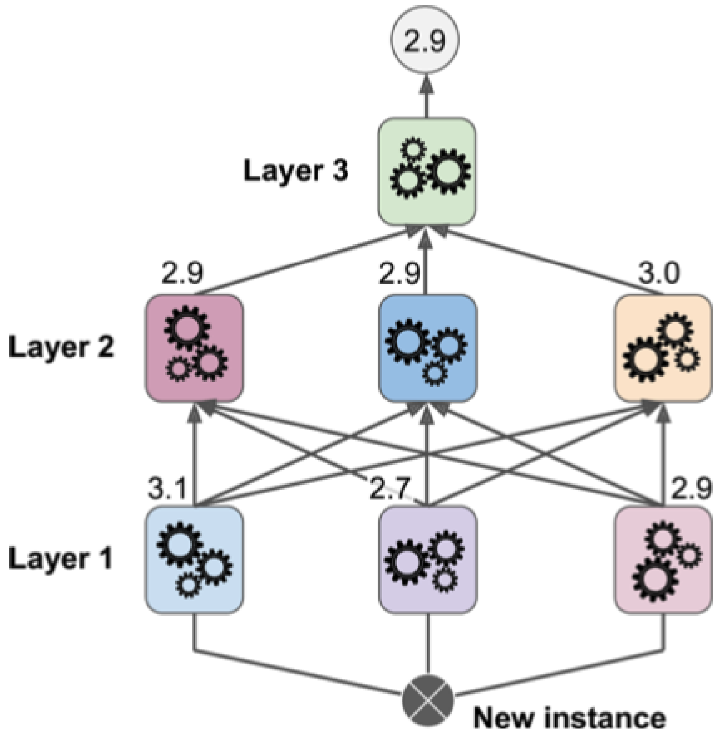


Figure 14

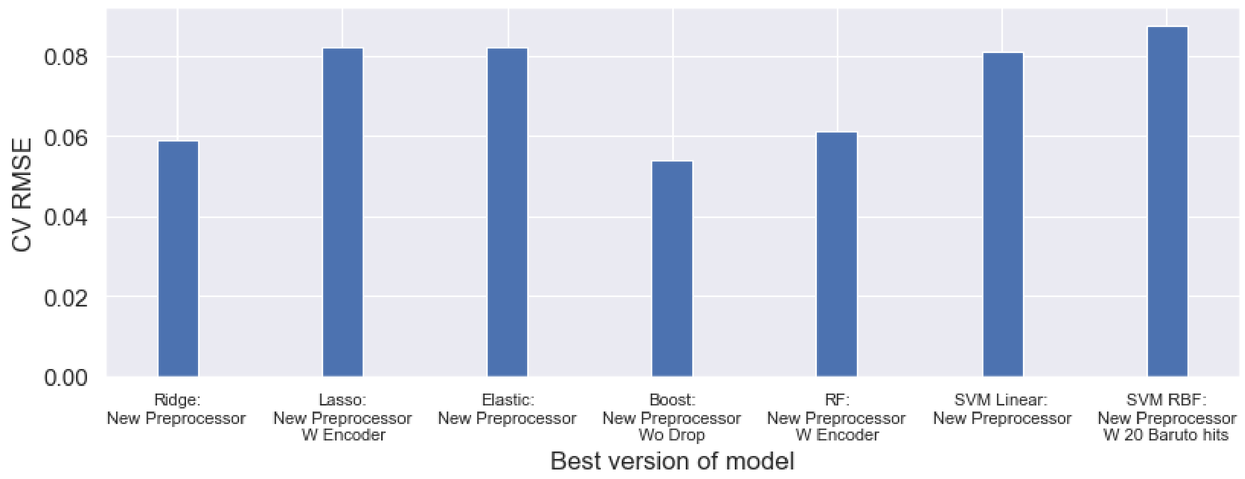


Figure 15

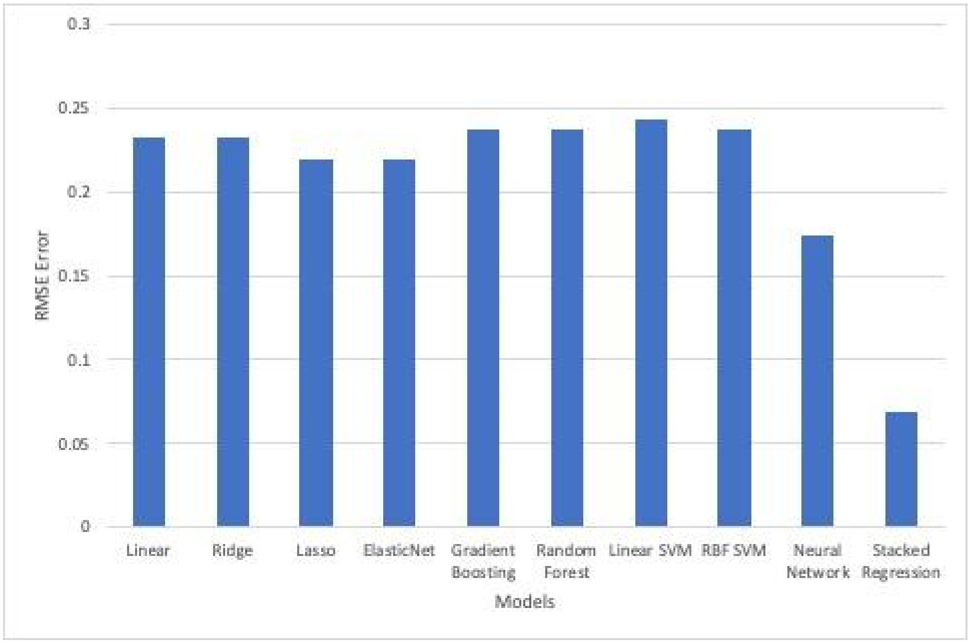

0.3
0.25
0.2
0.15
0.1
0.05
0
RMSE Error
Linear
Ridge
Lasso
ElasticNet
Gradient Boosting
Random Forest
Linear SVM
RBF SVM
Neural Network
Stacked Regression
Models

Figure 16

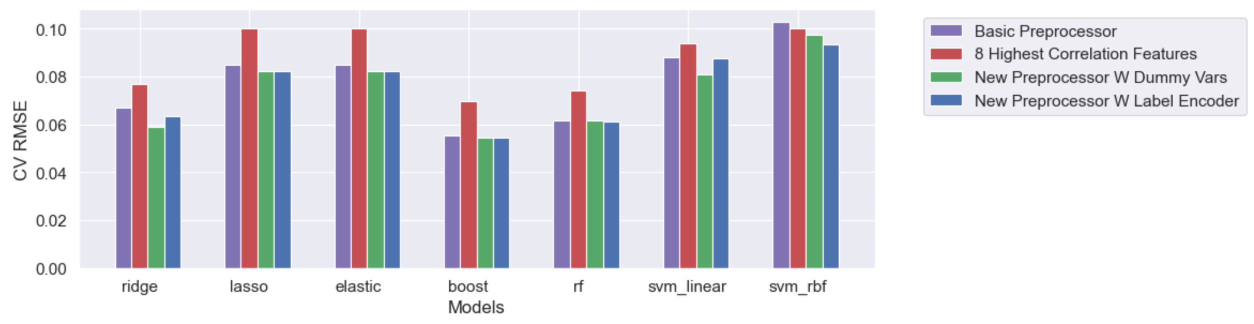


Figure 17

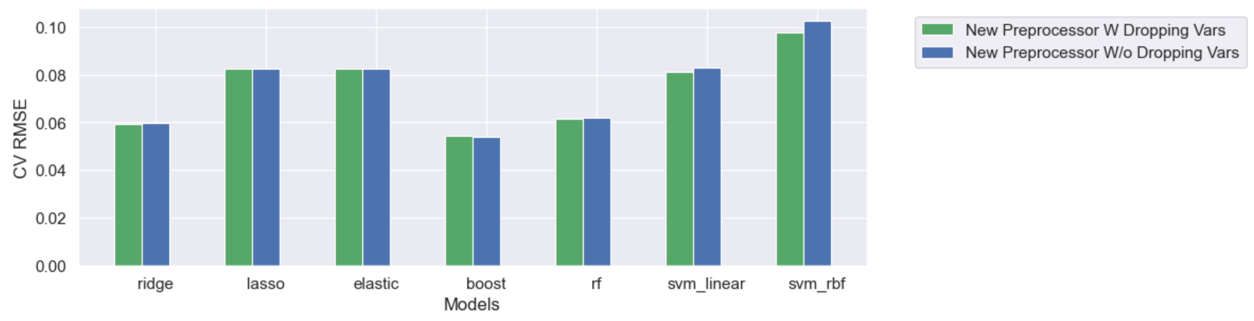


Figure 18

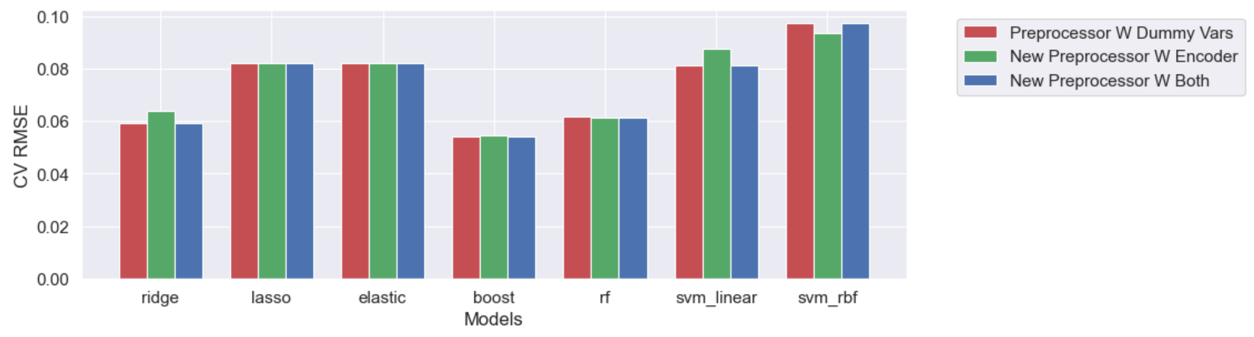


Figure 19

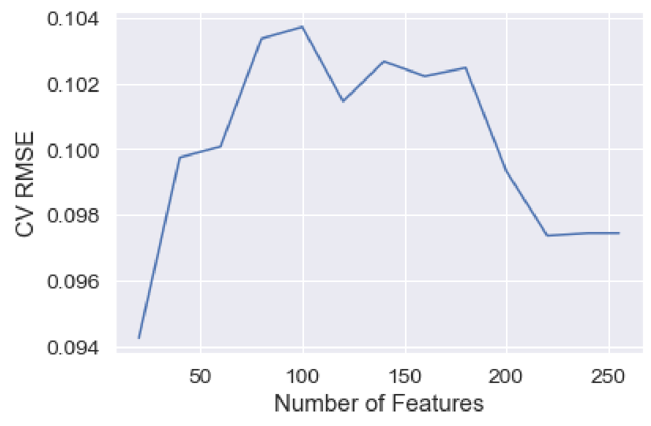
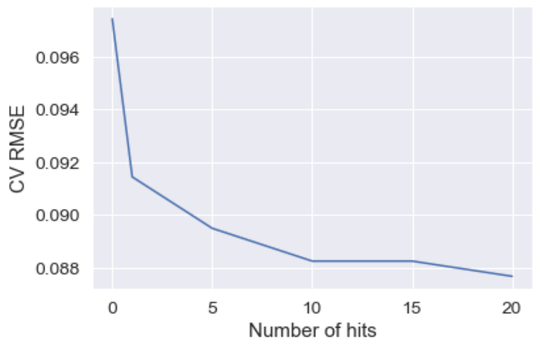


Figure 20


CV RMSE
Number of hits