CSE 291-A: Group 8 Final Report

Hao-en Sung [A53204772] wrangle1005@gmail.com
Jui-Yu Chang [A53214192] b00901024@ntu.edu.tw
Shun-Jen Lee [A53209671] shunjen013@gmail.com

Abstract

This is a report for 291-A final project from group 8 in 2017 fall. Our goal is to be more familiar with machine learning pipeline from end to end by working on a house price prediction task on Kaggle. During the project, we try different models and tune parameters for each model with validation sets, respectively. Apart from that, we put a lot of effort in understanding models with support vectors, which have the best performance on hidden dataset.

1. Introduction

In the project, we are working on a house price prediction problem. Since we are new to the machine learning optimization task, we incline to choose a dataset that is

- Relatively small
- Relatively easy to parse and extract features
- A general regression problem

At the end, we find a dataset in Kaggle [1] meets all the above requirements, which can help us have more time in

- Data visualization
- Trials with different models

2. Introduction to Dataset

In this problem, we are given some explanatory variables describing different aspects of residential homes in Ames, lowa. With those information, we are asked to predict the final price for each home. There are overall 79 features, including both numerical and categorical ones. Target scores are real values, which describe the final house price. The evaluation metric used in this task is Rooted-mean-squared-error (RMSE) on the difference between log-scaled real price and log-scaled predicted price.

3. Methodology

3.1. Data Preprocessing

In the first glance, we find that there are many NAs in our data. It will cause some problems if we don't do anything about it. For simplicity, we filter out features (columns) that contain NA. We create a boolean mask for NAs and use it to filter out those infeasible columns in the dataset. Through this filtering, the number of features is greatly reduced from 79 to 45.

There are numerical and categorical features in our data, which both need proper adjustment. For numerical ones, we want the mean and variance of each feature to be the same. Hence, we normalize our data by columns. We utilize *StandardScalers*, a library in *SKlearn*, to subtract the mean from each column and then divide each value by column variance. There is also an alternative choice, *MinMaxScalers*, which rescales each column to $[0, 1]$. We use *MinMaxScalers* to process data for visualization instead of *StandardScalers*, since it is easier for reader to have intuition in those figures. The number of numerical features is 24 (out of 45).

For categorical features, we need to transform them into numerical values in order to apply general machine learning models on them. We use *get_dummies* function in *Pandas* to one-hot-encode categorical features, which results in columns filled with either 0 or 1. An example for one-hot-encoding is shown as Figure 1. After this process, the number of categorical features is greatly extended from 21 (out of 45) to 140.

After the preprocessing to numerical and categorical features separately, there are overall $24 + 140 = 164$ features available to us. At last, we need a validation set to tune parameters in our machine learning model. We split the training set into sub-training and sub-testing sets equally. To further stabilize the model performance, one can use k-fold cross-validation. Usually, one can set k to 3, 5, or 10.

3.2. Data Visualization

In this section, we perform data visualization to gain a deeper understanding of our data. The motivation is that some features in our data seem to be noisy, so some proper adjustments might need to be applied on them. In addi-

ID	Shape
1	square
2	circle
3	triangle

ID	Square	Circle	Triangle
1	1	0	0
2	0	1	0
3	0	0	1

Figure 1: one-hot-encoding

tion, the ratio between number of features and number of instances is too small, which might lead to the risk of overfitting to training data. Hence, we want to make use of data visualization to select a subset of meaningful features and neglect other noisy ones.

In order to find out those noisy features, we analyze the correlation between feature values and the target scores we are concerning. Our conjecture is that the higher the correlation is, the more effective the feature is.

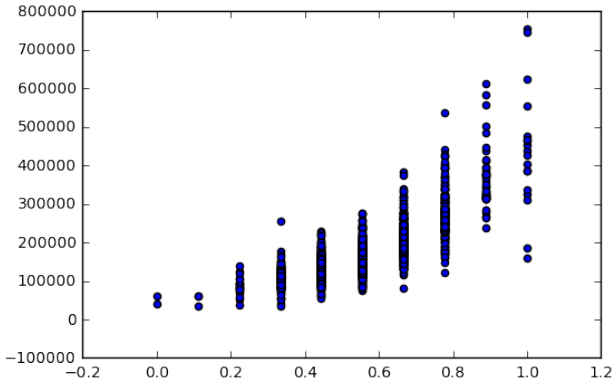
Some features with high correlation are shown in Figure 2a and Figure 2b. These features are overall quality and the area of the first floor. It is reasonable that these features should have great impact on the target house price.

On the other hand, two examples of feature with low correlation are shown in Figure 3a and Figure 3b. These two features are the area of open porch and the area of the second floor. We conclude that they will be less helpful and even become the noise to our model. However, there is an interesting fact regarding the area of the second floor. By ignoring the data with zero area, the correlation is high in the plot. Through further analysis, we know the reason there are so many data with zero area is that some of the houses simply do not have the second floor. In this case, a better strategy to deal with these features is to train two models for house with and without second floor separately. When testing, we check whether the instance contains the second floor or not and put it to the corresponding model.

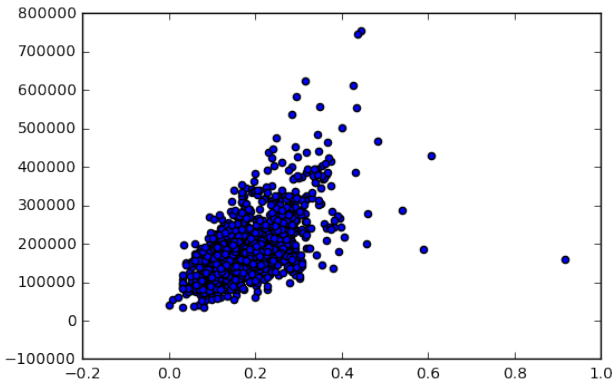
There are also categorical features that we want to visualize. Unlike numerical one, it is hard to see their correlations through an x-y diagrams. However, we can use box plot with some statistical indicators. The example of box plot we create is shown as Figure 4. *IR1*, *IR2* and *IR3* are three kinds of irregular shapes and *Reg* is the regular shape, which is rectangle. One can tell from the figure that the third quartile of the house price in *Reg* shape is almost the same as the first quartile of house price in *IR2* shape, which indicates that there is a huge clear trend of the house price between these two shapes. In other words, this feature might be very useful to our model.

3.3. Introduction to models with support vectors

Support Vector Machine (SVM) [2] is useful in classification. It finds the best hyperplane with largest margin to



(a) OverallQual



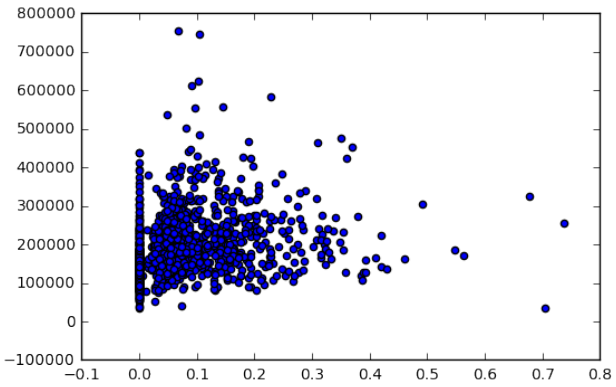
(b) 1stFlrSF

Figure 2: Example for two high correlation features

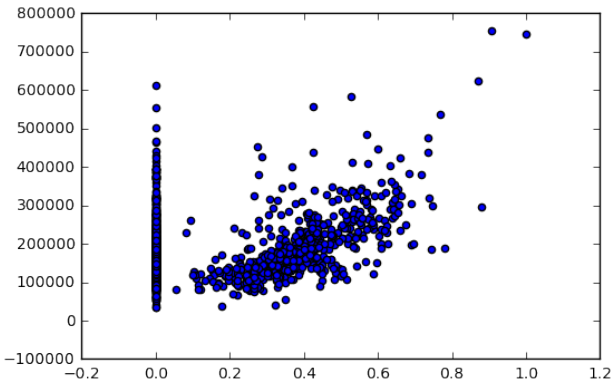
classify the data. In order to predict the real-value house price, we need to use Support Vector Regression (SVR) [3, 4, 5] instead. Similar to SVM, it would find the best ω, and it can be used to calculate the predicted value. To address with non-linear problem, we use kernel function. It is very powerful because it is equvialent to mapping original feature vector into higher dimensional vector.

3.4. Introduction to other models

Linear regression is just linear transform between input features and output value. Random forest constructs multiple decision trees and uses the mean to predict the final result. Gradient boosting is a powerful technique to make prediction by an ensemble of weak prediction models, typically decision trees. Deep learning is very popular right now, which builds neural network with multiple hidden layers to make prediction.



(a) OpenPorchSF



(b) 2ndFlrSF

Figure 3: Example for two low correlation features

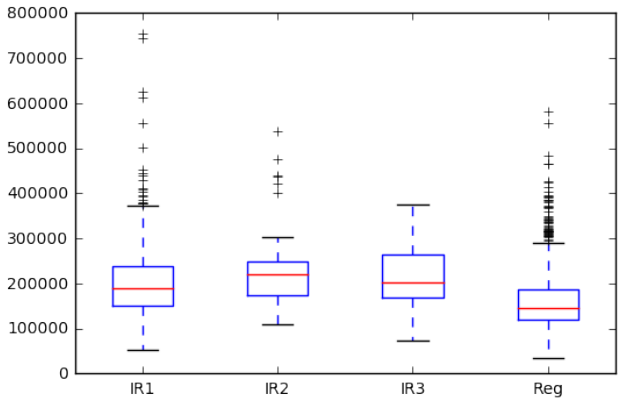


Figure 4: Example for one categorical feature: LotShape

4. Experiment Results

4.1. Linear Regressor

Linear Regressor is the simplest model, which simply calculates the weighted sum among all feature values linearly. No parameter needs to be tuned for this model. Results are recorded in Table 1

	Train	Test
Validation	0.09754	0.65637
Original Task	0.11598	0.46278

Table 1: Model Performance: Linear Regressor

4.2. Random Forest Regressor

Random Forest Regressor is a well-known tree-based model, which can deal with the non-linearity of features. Two parameters can be tuned for this model: *max_depth* and *n_estimators*. Usually, the deeper of the tree, the more overfitting the model is; while more forests, more stable the model is. Results are recorded in Table 2

	Train	Test
Validation	0.05046	0.17887
Original Task	0.05488	0.14964

Table 2: Model Performance: Random Forest Regressor
Best parameters: max_depth=30**, n_estimators=**1000

4.3. Gradient Boosting Regressor

Gradient Boosting Regressor is another widely used tree-based model. In practice, it usually has slightly better performance than *Random Forest*. Two parameters can be tuned for this model: *max_depth* and *n_estimators*. Similar to *Random Forest*, the deeper of the tree, the more overfitting the model is; however, the tree depth here indicates the level of feature interaction instead. On the other hand, the more trees, the more robust the model is. Results are recorded in Table 3.

	Train	Test
Validation	0.09359	0.16582
Original Task	0.10964	0.13324

Table 3: Model Performance: Gradient Boosting Regressor
Best parameters: max_depth=1**, n_estimators=**1000

4.4. Deep Learning Model

Deep learning model in general can be very powerful when there is sufficient data. In this task, the performance, however, is limited to the number of instances. There are many parameters can be tuned for this model. At the end, we only play with the size of hidden layers, number of hidden neurons, alpha value, activation functions, and type of solvers. At the end, we find out that number of hidden layers and hidden neurons have great impact on testing perfor-

mance, and decide to use one hidden layer with one hidden neuron for our task. Results are recorded in Table 4.

	Train	Test
Validation	0.10825	0.16060
Original Task	0.12525	0.13993

Table 4: Model Performance: Deep Learning Model
Best parameters: hidden_layer_sizes=$(1,)$**, alpha=**0.5**, activation='logistic', solver='lbfgs'**

4.5. Support Vector Regressor

Support Vector Regressor as we described in previous sections has very strong fitting power to training data. There are multiple choices of kernels available to us, and we at the end find out the one with *rbf* kernel has the best performance. Other parameters include C, eps, and gamma. Results are recorded in Table 5.

	Train	Test
Validation	0.09039	0.16673
Original Task	0.10337	0.13184

Table 5: Model Performance for Support Vector Regressor
Best parameters: kernel='rbf', C=10**, eps=**0.1**, gamma=**0.01

4.6. Summary

With five models, *Random Forest Regressor* has the best performance in terms of training data, *Deep Learning Model* has the best performance in terms of testing data in validation set, and *Support Vector Regressor* has the best results in terms of *Kaggle* submission score.

5. Conclusion

Through data preprocessing, not only our numerical features are rescaled and have lower variance, but our categorical features are also one-hot encoded into numerical values with either 0 and 1. With data visualization, we have better understanding of which feature is more important and which is more noisy. By doing so, we are able to manipulate features and train better models. In the end, we try several models with features we select and fine tune their parameters with proper validation set. We achieve the best performance using SVM with RBF kernel, which results in a submission error of 0.13184.

5.1. Future Work

There are ways to make us do better if we have more time. The first thing is to build different models for different set of data. Since we know the distribution of data through visualization, we could separate our data into meaningful groups. One example is to separate our data into houses with and without second floor. In this way, we could make better use of noisy data.

Another worth trying method is to do model ensemble. We could average the results from different models to achieve a better prediction. An advanced ensemble technique is to linearly combine different models and use yet another validation set to train the combining coefficient. Ensembles in general can bring some improvements to each single model.

6. Reference

References

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

[2] https://www.cs.umd.edu/~samir/498/SVM.pdf.

[3] http://www.cmlab.csie.ntu.edu.tw/~cyy/learning/tutorials/SVR.pdf.

[4] https://cs.adelaide.edu.au/~chhshen/teaching/ML_SVR.pdf.

[5] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.