Project Report

Big Data Methods for Economists

Wenjie Tu

April 2021

Introduction

In this project, I explored how machine learning algorithms can be used to solve a real-world problem, rents prediction, in particular. This report will walk you though all the steps I have taken for this project.

Data Preparation

Data Summary

The training dataset contains 72,000 observations and 83 columns (82 features and 1 target) and the given test dataset contains 18,001 observations and 82 columns (82 features). Both the training dataset and test dataset contain many missing values. In this project, the missing values are handled separately on the training dataset and test dataset to avoid data leakage in the training procedure.

Columns Dropping

In real world, different datasets may contain missing values for different reasons and thus require different imputing methods. However, in the given dataset, there is no further information on this. I therefore set a threshold (20%) for the proportion of missing values along a specific column and dropped those columns above the threshold. Most of those dropped columns contain more than 90% missing values so it makes more sense to drop them rather than impute them. Meanwhile, I also looked at those columns just above the threshold - *descr* (28%) and *dist_to_main_stat* (29%). *descr* is a text feature and requires text analysis, which makes imputation more complicated. It is also "risky'' to impute missing values on these columns since roughly 30% is still considered quite high according to the rules of thumb. In addition, I removed the column (*year*) with zero variance and the columns (*id*, *GDNER*) that have no impact on the prediction. I also removed the column *quarter_specific* to avoid collinearity since *quarter_specific* and *quarter_general* are perfectly collinear.

Data Visualization

I used latitude and longitude data to visualize the geographical distribution of the rents and found some interesting patterns. The location of rental houses/apartments is concentrated in Zurich, Bern, Lausanne, and Geneva. By adding *Anteil_auslaend* and *rent_full* to the plot, I see that proportion of foreigners and rents are quite high in those areas.

One-Hot Encoding

Some of the machine learning algorithms can work with categorical data directly while most of the machine learning algorithms cannot. There is only one categorical feature *home_type* and this feature is not ordinal, so I used one-hot encoding method to convert it into numerical values.

Missing Values Imputation

For simplicity, I assumed values in the given dataset are missing at random. Under this assumption, I used *IterativeImputer*, which models each feature with missing values as a function of other features. In particular, I used *BayesianRidge*, a regularized linear regression, as the estimator for imputation. Some previous studies show that using linear regression and tree-based models as estimators for imputation yield decent results and in most cases tree-based models even perform slightly better for imputation. However, I did not end up using tree-based models for imputation since they are much more computationally expensive. It should be noted that both target data and one-hot encoded categorical data are included for imputation since values on these columns (*rent_full* and *home_type*) are complete and can improve the accuracy on imputation.

Data Centering and Scaling

For this particular regression task, the objective function is the mean squared error, which is based on distance computation. Therefore, it is necessary to center and scale the numerical data to eliminate the impact from magnitude scales and to speed up the training procedure. I used *QuantileTransformer* on the dataset since this method is robust to outliers and alos applies non-linear transformation to the data (more Gaussian-like). Before I transformed the data, I noticed that there was a binary variable within the numerical data and I excluded that column from data transformation.

Model Selection

The training dataset is split into 80% for training and 20% for validation. My strategy is to first feed training dataset to different machine learning models with their default settings and then evaluate the performance on both training dataset and validation dataset. My prior guess is that random forest and neural network would yield better results since some similar tasks show that these two machine learning algorithms have better performance on regression tasks. The models I tried include linear regression, ridge regression, lasso regression, elastic net, Huber regressor, Bayesian ridge regression, stochastic gradient descent regressor, linear support vector regressor, support vector regressor (radial kernel), decision tree regressor, random forest regressor, k neighbors regressor, and multi-layer perceptron. As expected, random forest regressor and multi-layer perceptron outperformed other models on both training data and validation data.

Model Evaluation

I further looked at random forest and multi-layer perceptron. My strategy is to first make models overfitting and then solve the overfitting with fine-tuning methods since overfitting is easier to solve than underfitting. Overfitting can be easily detected by simply looking at the MSEs on training and vlidation dataset. If a model yields a lower MSE on training dataset but a higher MSE on validation dataset, then it is more likely to overfit the dataset. Scikit-learn library provides two approaches for optimal hyperparameter search - *GridSearch* and *RandomizedSearch*. *GridSearch* is an exhaustive search for optimal hyperparameters, which is inefficient and computationally expensive. *RandomizedSearch* is a random search for optimal hyperparameters, which yields a less accurate combination. Since there are a lot of empirically-derived rules of thumb on hyperparameter tuning, I decided to search for optimal hyperparameters manually and then use 5-fold cross validation on the full training dataset for evaluation.

Random Forest

Random forest regressor uses decision trees as building blocks to construct prediction models and outputs the mean of the classes as the prediction of all the trees. As is shown in the model selection procedure, the random forest regressor with defaulting settings overfits the training dataset, so I manually fine-tuned the following hyperparameters:

n_estimators: the number of trees in the forest.The lower the number of trees is, the more likely the model is to overfit training data, so this hyperparameter should be increased to improve generalization.

max_features: the number of features to consider when looking at split. This determines how many features each tree is randomly assigned. The smaller this number is, the less likely to overfit. A rule of thumb is to try 30-50% of the number of features.

max_depth: the maximum depth of the tree. Deeper trees have more splits and capture more information about the data. The larger this number is, the more likely to overfit. This hyperparameter should be decreased.

min_samples_leaf: the minimum number of samples required to be at a leaf node. It means that the branch will stop splitting once the leaves have that number of samples each. This hyperparameter should be increased to solve the overfitting.

Multi-Layer Perceptron

Multi-layer perceptron consists of three types of layers - the input layer, output layer and hidden layer, which is the true computational engine of the MLP. It is important to configure the number of neurons in hidden layers. The past common practice is to form a pyramid (the number of neurons per hidden layer is decreasing from top layer to bottom layer). The new trend is to use the same number of neurons in all hidden layers. I tried both and it turned out the structure of hidden layers from the new trend works better in this setting. Then I used the trial and error approach to pin down the optimal number of neurons for each hidden layer. I eventually chose three hidden layers with 60 neurons in each layer.

Model Prediction

I merged training dataset (without target) and test dataset into one dataset to better impute missing values and then detached test dataset from the entire dataset. The imputation and data transformation procedures are quite similar as before. I stored the predicted rents with corresponding *id* in a dataframe and exported the dataframe as a csv file.

Conclusion

The results of this project confirm the findings from previous studies that random forest and multi-layer perceptron perform better on general regression tasks.