Research Article

Research on the Construction and Realization of Data Pipeline in Machine Learning Regression Prediction

Hua Zhang,1,2 Guoxun Zheng3,1,2 Jun Xu,2 and Xuekun Yao1,2

1School of Computer Technology and Engineering, Changchun Institute of Technology, Changchun, Jilin 130012, China
2Jilin Provincial Key Laboratory of Changbai Historical Culture and VR Reconstruction Technology, Changchun Institute of Technology, Changchun, Jilin 130012, China

Correspondence should be addressed to Guoxun Zheng; zhengguoxun@ccit.edu.cn

Received 11 January 2022; Revised 22 March 2022; Accepted 28 March 2022; Published 14 April 2022

Academic Editor: Naeem Jan

Copyright © 2022 Hua Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The dataset used by machine learning usually contains missing value and text type data, and sometimes, it is necessary to combine the attributes in the dataset. This requires cleaning and necessary conversion of the dataset before the model is built. This is usually a series of actions. If each action is completed step by step, the whole process will be cumbersome and messy. We will complete it by building a data pipeline [1, 2]. The data pipeline will form an orderly and clear process for all data processing before machine learning, which is streamlined and smooth, which can greatly improve the efficiency of data processing and reduce the complexity of operations. In addition, in traditional model training, the dataset is divided into a training set and a test set. The data of the test set are randomly selected. The prediction result is often related to the data selection or, to a certain extent, depends on luck. The model trained by machine learning needs an evaluation standard that can objectively evaluate the performance of the model. We combine the k-fold cross-validation with the data pipeline to complete the training and evaluation of the model.

1. Introduction

The data set used by machine learning usually contains missing value and text type data, and sometimes, it is necessary to combine the attributes in the data set. This requires cleaning and necessary conversion of the dataset before model is built. This is usually a series of actions. If each action is completed step by step, the whole process will be cumbersome and messy. We will complete it by building a data pipeline [1, 2]. The data pipeline will form an orderly and clear process for all data processing before machine learning, which is streamlined and smooth, which can greatly improve the efficiency of data processing and reduce the complexity of operations. In addition, in traditional model training, the dataset is divided into a training set and a test set. The data of the test set are randomly selected. The prediction result is often related to the data selection or, to a certain extent, depends on luck. The model trained by machine learning needs an evaluation standard that can objectively evaluate the performance of the model. We combine the k-fold cross-validation with the data pipeline to complete the training and evaluation of the model.

The following is a summary of the research: Section 2 explains how to handle data for regression prediction. Section 3 addressed how to create a data pipeline. Section 4 looked into cross-validation and grid search. Section 5 discusses the experimental design and analyses. Finally, the research job is completed in Section 6.

2. Data Processing in Regression Prediction

2.1. Missing Value Processing. The majority of machine learning algorithms are unable to cope with lacking features. There are usually 3 methods to deal with missing values:

(1) Delete instances with missing values
(2) Delete attribute columns with missing values
(3) Use 0, mean, or median to fill in missing values

If you use the average or median to fill in missing values, it can only be done on numeric attributes, or you can use SimpleImputer in Scikit-Learn to do it.

2.2. Processing of Text and Classification Attributes

2.2.1. Converting Text and Classification Attributes to Numeric Values. Most machine learning algorithms use numbers for processing, so certain attributes and categories
in the data set need to be converted from text to numbers. The Label Encoder class can be used for conversion before Scikit-Learn 0.20 version, and the Ordinal Encoder class can be used for conversion after version 0.20.

We should pay attention to the following 3 aspects when using the Label Encoder and Ordinal Encoder:

1. The input and output of the Label Encoder are one-dimensional arrays, while the input and output of Ordinal Encoder are two-dimensional arrays.
2. Label Encoder uses the classes_instance variable to get the category list, whereas Ordinal Encoder uses the categories_instance variable to get the category list.
3. Label Encoder is generally used to process the target column, and Ordinal Encoder is generally used to process the attribute column (feature column).

The Ordinal Encoder is equivalent to the Label Encoder and supports multicolour version, which has performance advantages when there are many columns. The Ordinal Encoder is preferred when processing features.

2.2.2. One-Hot Encoding. The feature column usually contains categorized text attributes. The machine learning algorithm converts these categorized text attributes into numbers such as 0, 1, and 2. The machine learning algorithm will think that two similar numbers are more similar than two numbers that are farther apart. However, this is not the case. For example, the similarity between category 0 and category 4 may be higher than the similarity between category 0 and category 1. In order to solve this problem, a common solution is to create a binary attribute for each category. If there are several categories in the data set, create several attributes. In each instance, only one attribute is 1 and the rest are 0. For example, when the category is "A," this attribute is 1 and the others are 0. When the category is "B," this attribute is 1 and the others are 0, and so on. Because only one attribute is 1 (hot) and the others are 0 (cold), this is called one-hot encoding. When there are thousands of category attributes, after the one-hot encoding is completed, a matrix of thousands of columns will be obtained and it is basically all 0s, and there is only one 1 in each row. It is very wasteful to take up a lot of memory to store 0s. The One-Hot Encoder class in Scikit-Learn can not only convert integer classification values into one-hot vectors, but the output result is a SciPy sparse matrix, which is very convenient to use.

2.2.3. Simplification of Processing. Text and categorical attributes are often processed in two steps: first, text categories are converted to integer categories, and then, integer categories are converted to one-hot vectors. When there are thousands of category attributes, the one-hot encoding will result in a matrix with thousands of columns. The following two conversions can be accomplished at the same time using the Label Binarizer class in Scikit-Learn, and a dense NumPy array is returned by default. To acquire the sparse matrix, set the sparse output parameter to True in the Label Binarizer class' function object () .

2.3. Custom Converter. Although Scikit-Learn provides many useful converters, sometimes it is necessary to customize data cleaning operations or combine specific attributes, etc., which can be done through custom converters. The custom converter is best adapted to Scikit-Learn’s own functions (such as pipeline). Since Scikit-Learn relies on the compilation of duck types rather than inheritance, we can create a class and then define fit () (return self) , transform () , and fit_transform () .

In addition, if useful attributes cannot be filtered out of all attributes, you can also add hyperparameters in the custom converter to determine whether an attribute helps improve the accuracy of classification or regression. By setting hyperparameters, data processing is automated and many combinations of attributes are automatically tested as possible, so as to find important combinations and save a lot of time.

2.4. Feature Scaling. Feature scaling [3, 4] is the most common converter applied to data. If the ratio of the numerical attributes of the input data is quite different, it often leads to poor performance of the machine learning algorithm. For example, in the case of California housing price prediction, the total number of rooms ranges from 6 to 39,320 and the median income ranges from 0 to 15. In this case, the input data need to be feature-scaled, but the target value usually does not need to be scaled. The scaler is only used to fit the training set, not the complete data set (including the test set). Two common methods of feature scaling are min-max scaling and normalization:

1. Scaling the value to a rate between 0 and 1 is called minimum-maximum scaling (normalization). By subtracting the minimum value from the current value and dividing it by the gap between the maximum and minimum values, this is accomplished. The Min Max Scaler converter in Scikit-Learn can achieve minimum-maximum scaling. If you want the range to be different from 0 to 1, you can adjust the hyperparameter feature range to change it.

2. Standardized scaling is used to first subtract the average value and then divide it by the variance, so that the distribution of the results has a unit variance. The advantage of the standardized method is that it is less affected by outliers. For example, in the case of California housing price prediction, if the average income of a certain area has the wrong data of 100, the min-max scaling will scale all other values from 0–15 to 0–0.15 and the standardized scaling will not be greatly affected. The mathematical calculation formula is as follows:

$$x_i^* = \frac{x_i - \mu}{\sigma},$$

(1)
Mathematical Problems in Engineering

where $\mu$ is the mean of the data and $\sigma$ is the standard deviation of the data.

The disadvantage of the standardized method is that it does not scale the value to a specific range, which may be a problem for some algorithms. For example, the expected input value range of the neural network is usually 0 to 1. Obviously, using standardized scaled data does not meet the requirements. The Standard Scaler converter in Scikit-Learn can achieve standardized scaling.

3. Building a Data Pipeline

Before machine learning, data processing generally has to go through the above-mentioned 4 steps. If each action is completed separately, the entire processing procedure will lack logic and consistency. We can build these related steps into a data pipeline to form a sequence operation to improve the efficiency of data processing. In machine learning, a data pipeline is a sequential data processing component, which usually runs asynchronously. Each component collects a vast quantity of information, processes it, and sends the results to another central repository. Components are independent, and the connection between components is only a data warehouse, which is robust.

Scikit-Learn provides the pipeline to build a data pipeline and complete various conversions of data. The pipeline constructor defines a sequence of steps through a series of name/estimator pairings. Except for the last one being the estimator, the front must be a converter, that is, there must be a fit transform () method. When the fit () method of the pipeline is called, fit transform () is called in sequence on all converters and the output of one calling method is passed as a parameter to the next calling method, until the final estimator calls the fit () method.

In the early Scikit-Learn, the Feature Union class was used to add transformations to the pipeline. After version 0.20, Column Transformer was introduced. It does not need to define a selector transformer, so it works well when used with pandas data frames.

The Column Transformer class constructs a column transformer by obtaining a list of numeric column names and a list of category column names. The constructor requires a list of tuples, where each tuple contains a name, a converter, and a list of column names (or indexes) to which the converter can be applied. In the case of California housing price prediction, we can specify the numeric column to use the previously defined num_pipeline for conversion and the category column to use the One-Hot Encoder for conversion. The pseudo-code is as follows:

```python
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]
# Processing value conversion
num_pipeline = Pipeline([
    ("imputer," SimpleImputer(strategy = "median")),
    ("attribs_adder," CombinedAttributesAdder()),
    ("std_scaler," StandardScaler())
])
# Complete pipeline
full_pipeline = ColumnTransformer([
    ("num," num_pipeline, num_attribs),
    ("cat," OneHotEncoder(), cat_attribs)
])

housing = strat_train_set.drop("median_house_value," axis = 1) # contains only the independent variable X, a total of 9 columns (8 columns of value + 1 column of text)
housing_prepared = full_pipeline.fit_transform(housing) # 8 numeric attributes + 3 combined numeric attributes + 5 columns of text one-hot encoding = 16 columns
print (housing_prepared) # print view
```

It can be seen from the above pseudo-code that all the steps of the previous complex data processing are concentrated in the data pipeline for processing, the code is streamlined, the logic is clear, and the degree of automation is high.

4. Cross-Validation and Grid Search

4.1. Cross-Validation Evaluation Model Performance. In traditional model training, the data set is divided into a training set and a test set. The test set is generally 1/4 to 1/3 of the ratio. The prediction results of classification problems can be observed with the confusion matrix, and the prediction results of regression problems can be observed using the root mean square error, but the data of the test set are randomly selected. The prediction result has a lot to do with luck. If you are lucky, the prediction result will be better. If you are not lucky, the variance of the test set will be randomly selected if it is large, and if the deviation is large, the prediction result will be relatively poor. If you use the traditional method, the final result will be related to luck and a very random value, which is obviously not the best criterion for evaluating the performance of the model. In machine learning, k-fold cross-validation, often known as cross-validation [4–7], is a better strategy. It randomly divides the training set into 10 different subsets, each of which is called a fold, and the predictor is then trained and evaluated ten times, each time selecting one fold for evaluation and nine folds for training; the outcome is an array comprising ten test scores.

In the regression problem, the typical performance measurement index is the root mean square error (RMSE), which measures the standard deviation of the prediction error during the prediction process. The "68-95-99.7" rule means that approximately 68% of the values fall within 1σ, 95% fall within 2σ, and 99.7% fall within 3σ. For example, in the case of California housing price prediction, if RMSE is equal to 30,000, approximately 68% of the system's predicted value falls within $50,000 and approximately 95% falls within $100,000. The mathematical calculation formula of RMSE is as follows:

$$
RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n}(y_i - \hat{y}_i)^2}
$$
\[
\text{RMSE}(X, h) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (h(X^{(i)}) - y^{(i)})^2},
\]

(2)

where \(m\) is the number of instances in the validation set used when measuring RMSE, \(X^{(i)}\) is the vector of all the feature values of the \(i^{th}\) instance (except for label features) in the data set, and \(y^{(i)}\) is the label, that is, the predicted output value of the instance.

SciKit-Learn’s cross-validation function tends to use a utility function (the larger the better) rather than the cost function (the smaller the better), so the function for calculating the score is actually a negative MSE function.

4.2. Grid Search to Determine Hyperparameters. In many models of machine learning, in addition to the parameters that the model can automatically fit, there are some parameters that cannot be fitted with data. These parameters are called "hyperparameters." A hyperparameter is a parameter that cannot be learned by data. For example, the number of decision trees set in a random forest is a hyperparameter, which is a parameter of a model that we need to choose by ourselves. SciKit-Learn’s GridSearch provides a grid search method. It only needs to tell it what the hyperparameters to be tested are and the values to be tried. It will use cross-validation to evaluate all possible combinations of hyperparameter values.

Each model has different hyperparameters. The random forest model has six main parameters: \(n\)-estimators (the number of subtrees, default 100), \(max\_depth\) (the maximum growth depth of the tree), \(min\_samples\_leaf\) (the minimum number of samples of leaves), \(min\_samples\_split\) (branch node, the minimum sample size of the classification decision tree), \(max\_feature\) (the maximum number of selected features), criterion (evaluation index, the regression decision tree defaults to the mean square error, and the other is the absolute value difference; in the classification decision tree, there are two parameters to choose from, entropy and gini; the default is gini). \(n\)-estimators is the most influential parameter. When tuning parameters, you can first set the parameters to a larger range, then set the step size, use the loop to get the model accuracy in this range, draw the learning curve, find the optimal value, finally reduce the range of the parameters near the optimal value, and recalculate the accuracy of the model to obtain the maximum score.

Some data preparation steps can also be treated as hyperparameter. Collaborative filtering, for instance, will quickly decide whether or not to include unclear features. Similarly, it could also be used to find the optimum route to manually deal with problems, such as handling outliers, missing features, and feature selection.

Evaluating the final model with the test set only needs to obtain predictors and labels from the test set and run full_pipeline to transform the data. It should be noted that transform () is called instead of fit_transform (), and finally, the final model is evaluated on the test set.

5. Experiment

In the above paragraphs, we described in detail the construction process and performance evaluation of the regression model in machine learning. We applied these steps to the California housing price prediction dataset, used linear models, decision tree models, and random forest models for model training, and calculated RMSE. The RMSE of the three models are shown in Table 1.

It can be found that the prediction effect of the random forest model is better than the simple linear model and the decision tree model. In the random forest model, we further tried to combine the typical hyperparameter bootstrap, \(max\_features\), and \(n\)-estimators and calculate the RMSE, respectively. The results are shown in Table 2, and the chart is used for visual representation, as shown in Figure 1.

As it can be seen from Figure 1, when the 9th hyperparameter combination is selected, the model will obtain a smaller RMSE, which is reduced compared to the random forest model that does not use hyperparameter.
6. Conclusion

We tried to develop an autonomous data pipeline in this article to make it easier and more flexible to handle missing values and text type data in the data set and to apply cross-validation to improve model performance evaluation criteria. The ideal value of the hyperparameter of the prediction model was determined via grid search. Experiments show that it can reduce the root mean square error of the regression prediction model and improve prediction accuracy. Experiments have proved that the root mean square error of the regression prediction is indeed reduced and a prediction model with better performance can be established.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was jointly supported by the Foundation of Jilin Provincial Science and Technology Department (202002029JC and 20200301045RQ).

References