# Report of Data Science Final

WANG Xiangyi, YANG Yixin, LIU Shubing, SUN Jinmeng

We selected three models: Random Forest, Lasso Model, and Neuralnetwork for prediction, and judged which model is the best according to R-squared.

The first step is to process the data, represent the first row (column names) of the source data with numbers, and change the column names of the data frame X to add the string "x" before each original column name for the next steps.

## Random forest

First, the random Forest and h2o packages are installed. "h2o" is a machine learning framework in R language, which provides tools and algorithms for building and deploying large-scale machine learning models. The "h2o" library uses big data processing frameworks such as Hadoop and Spark, and provides efficient distributed computing capabilities, allowing fast data mining and predictive modeling on large-scale data sets. Before starting the forecast, we did some processing on the data. First, we merged the cpi with a lag of one period and x data, and deleted the column identification of the first column. Since we need some data to verify our predictions, we extract the first 80% of the data in the time series as the training set, and the last 20% of the data as the prediction set, which is convenient for testing our results. localH2O = h2o.init(nthreads = -1) This code will initialize an instance of the H2O machine learning platform on the local machine. The nthreads parameter is used to specify the number of threads used for parallel processing, -1 means that H2O should use all available threads on the machine. By default, H2O only uses a single thread, so specifying -1 can improve performance significantly. train\_h2o = as.h2o(train) is to convert the data frame (data frame) train in R language to the format of H2O frame (H2O frame) train\_h2o. Test\_h2o is the same.

If the value of the variable inClass is FALSE, the following block of code is executed:

Fit a random forest model using the h2o.randomForest() function, where x = 1:151 specifies that the first 151 columns are used as input features, and y = 152 specifies that the last column is the target variable;

Pass the training set train\_h2o to the training\_frame parameter;

ntrees = 500 specifies the number of trees in the random forest to be 500, you can tune it to a higher number for better performance;

stopping\_rounds = 2 specifies to stop training when the error does not decrease further for two consecutive rounds;

model\_id = "rf\_MNIST" specifies the unique identifier of the model to facilitate saving and loading the model.

Then, use the h2o.saveModel() function to save the fitted model to the specified path.

If the value of the variable inClass is TRUE, use the h2o.loadModel() function to load the existing model and store it in the variable rf\_fit for subsequent operations, such as making predictions on the test set.

Also, we create a h2o.performance object perf by passing rf\_fit and test\_h2o as parameters. This object contains various metrics that can be used to evaluate the performance of the model.

The second line of code uses h2o.mae to calculate the mean absolute error (MAE) of rf\_fit on its training data.

The third line of code uses h2o.r2 to calculate the R-squared value of rf\_fit on its training data. The R-squared value is a measure of how well the model fits the training data. A value of 1 indicates a perfect fit, while a value of 0 indicates that the model does not fit the data at all.

We then performed a cross-validation Create a cross-validation model

The ntrees parameter specifies the number of trees to grow in the forest, in this case 500. The stopping\_rounds parameter specifies that the algorithm will stop growing the tree if the cross-validation error rate does not improve after 2 rounds. The nfolds parameter specifies the number of cross-validation folds to use.

The seed parameter sets a random seed for reproducibility, while the model\_id parameter sets a unique identifier for the trained model so that it can be referenced later.

best\_model\_index <- which.min(cv\_perf@metrics$mae): The model index with the smallest mean absolute error (MAE) in the cross-validation performance (ie cv\_perf) is found, and the index is stored in the variable best\_model\_index.

The which.min() function returns the index of the minimum value in the vector. Therefore, which.min(cv\_perf@metrics$mae) returns the index position where the mean absolute error minimum is located. This index can be used to select the best model from the list of cross-validation models.

h2o.saveModel(new\_model, path = "new1\_mnist"): Use the h2o.saveModel() function of "H2O" to save your trained model. This function requires two parameters: the first parameter is the model to be saved, namely new\_model, and the second parameter specifies the path and name to save the model to

## Lasso

Here the mutate() function is used to modify the columns of X. The across() function specifies the columns to be modified, and "everything()" as a parameter indicates that all columns should be included. Finally, the as.numeric() function is applied to each selected column, converting any non-numeric variables to numeric.

First, the split <- sample.split(X$x1, SplitRatio = 0.7) line of code uses the sample.split() function to divide the data frame X into two parts according to the given ratio (70% in this case). This function returns a logical vector named split containing for each row in the X data frame a TRUE or FALSE value indicating whether the row is in the training set.

The next two lines of code train <- subset(X, split == TRUE) and test <- subset(X, split == FALSE) respectively select the value of TRUE or FALSE in the logical vector split from the original data frame X line to divide the training set and test set.

First, the line fit <- Arima(train$x1, order=c(0,1,1)) uses the Arima() function to fit the ARIMA model. The first parameter of the function train$x1 is the time series data, and order=c(0,1,1) specifies the order of the ARIMA model. Where (0,1,1) means no AR part (that is, the AR order is zero), the order of the MA part is 1, and the difference order is 1. After fitting is complete, the model object is stored in the fit variable.

Next, the line train$y\_pred <- predict(fit)$pred uses the predict() function to generate predictions based on the training set data. The predict() function automatically selects the best prediction method and returns an object with various properties, including predicted values, standard errors, and confidence intervals, among others. Therefore, we need to extract only the predicted values using the $pred attribute and save them into a new column called y\_pred which is added to the train dataframe.

train$y\_arima <- residuals(fit) This line of code uses the residuals() function to extract the residuals of the ARIMA model on the training set data and saves them into a new column called y\_arima which is added to the train data frame middle. Residuals are the differences between observed values and the corresponding model predictions, and are often used to assess how well a model fits and to check for patterns or trends. After model building, we often want to check the quality of the model's fit to the training data in order to diagnose and improve the model.

Store the eigenvariables and exogenous variables in the training set in matrices and vectors, respectively.

After running the function, the cvfit variable will store an object containing the results of the cross-validation. This object contains information such as the error size of each cross-validation and the corresponding regularization parameter value, as well as the final selected model.

First, library(glmnet) loads the glmnet package, which provides a common regularized linear regression method that can automatically select the best regularization parameters.

Next, test$y\_arima <- predict(fit, n.ahead = nrow(test))$pred This line of code uses the ARIMA model that has been fitted to predict the test set and save the result to a new file named y\_arima column, which is added to the test data frame. The first parameter of the predict() function is the fitted model object fit, and the second parameter n.ahead indicates the number of steps to be predicted, which is set to the number of rows in the test set.

Next, x.test <- as.matrix(test[, c("CPI", "y\_arima")]) and y.test <- as.vector(test$CPI) will test the independent and response variables of the set Extract them separately and convert them into matrix and vector types.

Finally, the line y.pred <- predict(cvfit, newx = x.test) uses the best regularization parameters selected by cross-validation to make predictions on the test set and save the results to a file named y.pred variable. The first parameter of the predict() function is the object cvfit containing the cross-validation information, while the second parameter newx is the input test set argument matrix.

## Neuralnetwork

The first line of code scales the predictor variables in matrix X excluding the first column, which may contain the timestamp information. The scaled result is stored in x.nn.

The second line of code scales the target variable cpi.

The next two lines combine the scaled cpi variable with the unscaled timestamp from X and the scaled predictor variables in x.nn, and store them in a data frame called nndata. The column names for cpi and timestamp are set appropriately.

Similarly, the last four lines of code create another data frame nndata.ppi which combines the scaled ppi variable with the unscaled timestamp from X and the scaled predictor variables in x.nn. The column name for the timestamp is set appropriately.

This code converts the timestamp column in the nndata data frame to datetime format in R and sets its start time to "2020-01-01". The converted datetime information will be stored in nndata$timestamp.

Next, convert the nndata data frame to data in H2O format using the as.h2o() function. This is because H2O is a high-performance, distributed machine learning platform that can make data processing and modeling very efficient.

This code defines the hyperparameter set hyper\_params of the neural network model, which contains a number of different parameter combinations. Specifically:

The activation parameter specifies the activation function type of the neuron, and the candidate values are "Rectifier" (rectified linear unit), "Tanh" (hyperbolic tangent), and "RectifierWithDropout" (rectified linear unit with random dropout).

The hidden parameter specifies the hidden layer structure of the neural network. A list can be used to represent different hidden layer structures. Each list item contains an integer vector representing the number of neurons in the hidden layer. For example, c(30,30) means that there are two hidden layers with 30 neurons in each hidden layer.

The epochs parameter specifies the number of epochs to train the neural network with candidate values of 100, 150, 200, 250 and 300.

The adaptive\_rate parameter is a boolean specifying whether to use an adaptive learning rate. If set to TRUE, an adaptive learning rate is used, otherwise the default learning rate is used.

The input\_dropout\_ratio parameter specifies the dropout ratio of the input layer, and the candidate values are 0.05, 0.1 and 0.2.

By searching for the optimal parameter combination within the combined range of these hyperparameters, the prediction accuracy of the neural network model can be further improved.

This code uses H2O's h2o.deeplearning() function to create a deep neural network model nn\_model. The parameters of the function are set as follows:

The x argument specifies the data frame column names that are predictors. Here names(nndata)[-1] is used to mean all column names except the first column (i.e. timestamp).

The y parameter specifies the target variable column name, here "cpi".

The training\_frame parameter specifies the data frame used for training, here is data\_h2o.

The epochs parameter specifies the number of training rounds, here it is 100.

The hidden parameter specifies the hidden layer structure, in this case two hidden layers with 10 neurons in each hidden layer.

The activation parameter specifies the activation function type, here is "Rectifier", which is a rectified linear unit.

The input\_dropout\_ratio parameter specifies the random dropout ratio of the input layer, here is 0.2.

By adjusting these parameters, the prediction accuracy of the neural network model can be further improved, and appropriate parameter values can be selected according to actual application requirements.

This code uses H2O's h2o.grid() function to create a grid search object grid for parameter optimization within the hyperparameter space. The parameters of the function are set as follows:

The "deeplearning" parameter specifies the type of algorithm to optimize, in this case a deep neural network.

The grid\_id = "nn\_grid" parameter specifies the name of the grid search task as "nn\_grid".

The hyper\_params parameter specifies the set of hyperparameters, here is the previously defined hyper\_params.

The x parameter specifies the column names of the data frame used as predictors, where names(nndata)[-1] is used to indicate all column names except the first column (ie timestamp).

The y parameter specifies the target variable column name, here "cpi".

The training\_frame parameter specifies the data frame used for training, here is data\_h2o.

The seed parameter specifies the seed of the random number generator, which is used to control the reproducibility of the results.

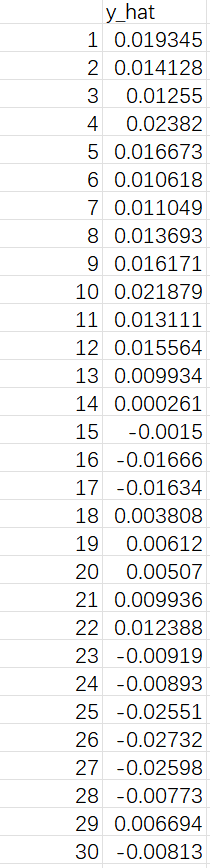
Grid search searches the hyperparameter space for the optimal combination of parameters and returns a list of all models and their corresponding scores. By comparing the performance of each model, the optimal combination of hyperparameters can be found, and the corresponding model can be selected for prediction.

This code selects the model with the highest score from the previous grid search results and assigns it to the variable best\_model. The model details are then obtained using the h2o.getModel() function and stored in the nnmodel variable. Finally, use the h2o.r2() function to calculate the R^2 score of the model, and store the result in the r2.nn variable; at the same time, use the h2o.varimp() function to check the influence of each variable in the model on the target variable, Note that this function only works for models with input variables (i.e. features).

Through these codes, we can obtain the details of the optimal deep neural network model nnmodel after grid search optimization, and understand its performance on the training data set. At the same time, we can also view the importance of each feature in the model to help us understand how the model predicts.

Combining the results of the above three models, we found that the neural network model can best explain and predict the data, so we finally chose the neural network model as our final prediction model.

Our forecast of cpi as follows:



Our forecast of ppi as follows:

