Project: Predict Future Sales

-Vedant Rakesh Abrol

{Data Analytics with R}

Explanation of code

- **Installing Required Packages:**

```
if (!require("data.table")) install.packages("data.table", dependencies = TRUE)

if (!require("ggplot2")) install.packages("ggplot2", dependencies = TRUE)

if (!require("caret")) install.packages("caret", dependencies = TRUE)

if (!require("xgboost")) install.packages("xgboost", dependencies = TRUE)
```

This section checks if the required R packages (data.table, ggplot2, caret, xgboost) are installed. If not, it installs them using install.packages().

- **Loading Required Libraries:**

```
library(data.table)

library(ggplot2)

library(caret)

library(xgboost)
```

The necessary R libraries are loaded into the current R session using the library() function.

- **Loading Datasets:**

```
sales_train <-
fread('/Users/vedantabrol/Downloads/competitive-data-science-predict-future-sales/sales_train.csv')

test <- fread('/Users/vedantabrol/Downloads/competitive-data-science-predict-future-sales/test.csv')

shops <-
fread('/Users/vedantabrol/Downloads/competitive-data-science-predict-future-sales/shops.csv')

items <-
fread('/Users/vedantabrol/Downloads/competitive-data-science-predict-future-sales/items.csv')

item_categories <-
fread('/Users/vedantabrol/Downloads/competitive-data-science-predict-future-sales/item_categories.
csv')
```

This section loads the datasets (sales_train, test, shops, items, item_categories) from CSV files using the fread() function from the data.table package.

- **Checking Datasets**

```
print(head(sales_train))
print(head(test))
print(head(shops))
print(head(items))
print(head(item_categories))
```

The head() function is used to display the first few rows of each dataset to inspect its structure and contents

Formatting Date Column:

```
sales_train[, date := as.IDate(date, format = "%d.%m.%Y")]
```

The date column in the sales_train dataset is converted to the IDate format using the as.IDate() function with the specified date format.

Aggregating Sales Data:

```
sales <- sales_train[, .(item_cnt_month = sum(item_cnt_day)), by = .(shop_id, item_id,
date_block_num)]
```

The daily sales data is aggregated to monthly sales per shop and item using the sum() function within a data.table operation.

Preparing Data for Model:

```
sales <- merge(sales, items[, .(item_id, item_category_id)], by = "item_id")
```

The data is merged with the items dataset to include item category information.

Preparing Test Data:

```
test <- merge(test, items[, .(item_id, item_category_id)], by = "item_id")
```

Similar to the training data, the test data is also merged with the items dataset.

Splitting the Data:

```
set.seed(123)

train_rows <- createDataPartition(sales$item_cnt_month, p = 0.8, list = FALSE)

train_data <- sales[train_rows,]

test_data <- sales[-train_rows,]
```

The data is split into training and testing sets using the createDataPartition() function from the caret package. The set.seed() function ensures reproducibility of the random split.

Model Training using XGBoost:

```
model <- train(item_cnt_month ~ ., data = train_data, method = "xgbTree",

          trControl = trainControl(method = "cv", number = 10),

          tuneLength = 3)
```

- A predictive model is trained using the `train()` function from the `caret` package. XGBoost algorithm (`method = "xgbTree"`) is used. Cross-validation (`trControl`) with 10 folds is performed for tuning hyperparameters.

Prediction and Model Evaluation:

```
predictions <- predict(model, test_data)

results <- postResample(predictions, test_data$item_cnt_month)

print(results)
```

- The trained model is used to make predictions on the test data. The performance of the model is evaluated using the `postResample()` function, which calculates the RMSE (Root Mean Squared Error).

Predicting for submission

```
final_predictions <- predict(model, test[, .(shop_id, item_id, item_category_id)])

sample_submission <- fread('./sample_submission.csv')

sample_submission[, item_cnt_month := final_predictions]
```

- Final predictions are made on the test dataset for submission. The results are stored in a new dataframe called `final_predictions`.

```
print(summary(model))
```

- A summary of the trained model is printed, which includes information about variable importance, parameter tuning results, and model performance metrics.

Evaluating the Prediction Model

Objective: Assess the performance of the prediction model built using XGBoost algorithm on sales data.

Steps:

Installed and loaded necessary R packages (data.table, ggplot2, caret, xgboost).

Loaded datasets: sales_train, test, shops, items, and item_categories.

Checked the first few rows of each dataset to understand its structure and contents.

Ensured the date column in the sales_train dataset was correctly formatted.

Aggregated daily sales to monthly sales per shop and item.

Prepared the data for modeling by merging relevant datasets and splitting it into training and testing sets.

Trained the prediction model using XGBoost (xgbTree) through 10-fold cross-validation.

Made predictions on the test data and evaluated the model's performance using Root Mean Squared Error (RMSE).

Outcome: The model achieved a certain level of accuracy in predicting future sales, as indicated by the evaluation metrics.

Different Ways to Improve Your Model and Show the Improvements

Objective: Explore potential enhancements to the prediction model to further improve its performance.

Possible Improvements:

Feature Engineering: Create new features or modify existing ones to capture more relevant information from the data.

Hyperparameter Tuning: Experiment with different hyperparameter values to optimize the model's performance.

Ensemble Methods: Combine multiple models to leverage their strengths and mitigate weaknesses.

Cross-Validation Strategies: Explore alternative cross-validation techniques to ensure robustness and generalizability.

Implementation:

Iterate through each improvement strategy, implement changes, and evaluate the impact on model performance.

Visualize the results to demonstrate the effectiveness of each improvement.

Expected Outcome: By systematically exploring various enhancement strategies, we aim to achieve a more accurate and reliable prediction model, ultimately improving the forecast of future sales.

```
# Feature Engineering: Add additional features

# Add month and year features extracted from the date column

sales[, month := month(date)]

sales[, year := year(date)]


# Prepare test data

test <- merge(test, items[, .(item_id, item_category_id)], by = "item_id")


# Splitting the data

set.seed(123)

train_rows <- createDataPartition(sales$item_cnt_month, p = 0.8, list = FALSE)

train_data <- sales[train_rows,]

test_data <- sales[-train_rows,]
```

```r
# Model training using XGBoost with hyperparameter tuning

tune_grid <- expand.grid(

  nrounds = c(100, 200, 300),

  max_depth = c(3, 6, 9),

  eta = c(0.01, 0.03, 0.05)

)

model <- train(

  item_cnt_month ~ .,

  data = train_data,

  method = "xgbTree",

  trControl = trainControl(method = "cv", number = 10),

  tuneGrid = tune_grid

)
```

Model Training with Hyperparameter Tuning

Objective: Optimize the XGBoost model's performance through hyperparameter tuning.

Hyperparameters Explored:

nrounds: 100, 200, 300

max_depth: 3, 6, 9

eta: 0.01, 0.03, 0.05

Model Training Process:

Utilized the train() function with cross-validation (10-fold) to evaluate model performance.

Tuning grid specified the combinations of hyperparameters to explore.

Expected Outcome: Improved model performance by selecting the optimal combination of hyperparameters.

XGBoost Algorithm

The code is using the XGBoost algorithm for modeling. XGBoost (Extreme Gradient Boosting) is an implementation of gradient boosting with several enhancements that make it highly efficient and scalable. It's commonly used in machine learning competitions and real-world applications due to its excellent performance and flexibility. In the provided code, the `xgbTree` method is specified in the `train` function, indicating the use of XGBoost for regression (tree-based) modeling.