Documentation for Predictor.R with TidyModels

Apple Watch Data Predictor with Tidymodels

Programmed by: Arastoo Bozorgi & Glenn Tanjoh

1. Overview

This R script is designed to process Apple Watch data, perform feature engineering, train a predictive model using **Tidymodels**, and make predictions on activity levels. The goal is to improve model performance, speed, and efficiency by utilizing a structured workflow. The script also includes data preprocessing, model tuning, parallel processing, and model saving in an efficient format.

2. Script Structure

The script consists of several parts:

- **Loading required libraries**
- **Setting environment variables**
- **Defining utility functions**
- **Loading and preprocessing data**
- **Tidymodels recipe creation**
- **Model training and evaluation**
- **Saving model objects and predictions**

3. Setting Up the Environment

```
R
Copy code
rm(list = ls())
```

This clears the workspace, ensuring no previous variables interfere with the current session.

Required Libraries:
The script uses the
`pacman` package to load all necessary libraries, ensuring the environment is prepared for execution. The `pacman::p_load()` function checks if packages are installed, installing them if necessary, and then loads them.

4. Setting Time Zone and Arguments

```
R
Copy code
Sys.setenv(TZ = "America/St_Johns")
```

Sets the time zone for time-related calculations, ensuring consistency in date-time parsing.

Arguments Setup:

- The script takes five arguments specifying paths for data files, models, and directories. If the correct number of arguments is not provided, an error is raised to ensure proper execution.

```
R
Copy code
args <- c(
  "main_path", "model_path", "training_file", "file_name", "m
odel"
```

```
)
```

- These arguments help in organizing the file structure, making it easier to modify paths without altering the script.

5. Utility Function for Correlation Calculation

- This function calculates the Pearson correlation between two columns but handles cases where the standard deviation is zero to avoid errors.

```
R
Copy code
correlation <- function(x) { ... }
```

- This function is used later to calculate the correlation between heart rate and steps.

6. Loading and Preparing Data

The script reads the Apple Watch data from a CSV file using `fread()` and checks for missing columns, adding them if necessary to ensure consistent data structure.

Data Imputation:

- Missing values are filled using **linear interpolation**, applied separately to key variables like `Heart`, `Calories`, `Steps`, and `Distance`.

```
R
Copy code
applewatch_data <- applewatch_data %>%
  mutate(
    Heart = na_interpolation(Heart, option = "linear"), ...
  )
```

7. Feature Engineering

- The script generates new features based on existing data, including entropy, resting heart rate, normalized heart rate, intensity calculation, and correlation.
- Feature engineering helps enrich the dataset, enabling the model to identify complex patterns in the data.

```
R
Copy code
applewatch_data <- applewatch_data %>%
  mutate(
    EntropyApplewatchHeartPerDay_LE = ...,
    NormalizedApplewatchHeartrate_LE = ...,
    ApplewatchIntensity_LE = ...
  )
```

8. Using Tidymodels for Data Preprocessing

- The **Tidymodels recipe** defines the preprocessing steps in a modular way, covering imputation, normalization, and removal of zero-variance columns.

```
R
Copy code
applewatch_recipe <- recipe(activity_trimmed ~ ., data = appl
ewatch_data_sample) %>%
  step_rm(...) %>%
  step_impute_median(...) %>%
  step_normalize(...)
```

- This recipe is prepped and baked, creating a transformed dataset that's ready for modeling.

9. Model Training with Tidymodels

- A **Random Forest model** is defined using Tidymodels, setting parameters like the number of trees and the number of variables to consider at each split.

```
R
Copy code
rf_model <- rand_forest(mode = "classification", mtry = 3, tr
ees = 20) %>%
  set_engine("ranger")
```

- The model is added to a workflow that combines the recipe and model definition, ensuring a seamless training process.

10. Parallel Processing

- To speed up model training, the script uses **parallel processing** with the `doParallel` package.

```
R
Copy code
library(doParallel)
cl <- makeCluster(detectCores() - 1)  # Use all but one core
registerDoParallel(cl)
rf_fit <- fit(rf_workflow, data = train_data)
stopCluster(cl)
```

- This improves model training speed by distributing computation across multiple CPU cores.

11. Model Evaluation and Predictions

- Predictions are made on the test set, and the accuracy is calculated to evaluate model performance.

```
R
Copy code
```

```
predictions <- predict(rf_fit, test_data, type = "class") %>%
  bind_cols(test_data)
accuracy <- mean(predictions$.pred_class == predictions$activ
ity_trimmed)
```

12. Measuring Execution Time and Memory Usage

- The script measures execution time and memory usage during prediction to ensure performance optimization.

```
R
Copy code
start_time <- Sys.time()
start_mem <- pryr::mem_used()
# Code for predictions
end_time <- Sys.time()
end_mem <- pryr::mem_used()
```

13. Saving the Model and Predictions

- The fitted model is saved as an RDS file using `saveRDS()`, making it easy to load and use later.

```
R
Copy code
saveRDS(fitted_rf_model, file = "Tidymodels_RFModel_AppleWatc
h.rds")
```

- Predictions are saved as a CSV file for further analysis or deployment.

14. Conclusion and Usage

This script provides a complete workflow for predicting Apple Watch activity levels using **Tidymodels**. It is designed to be modular, flexible, and efficient, making it suitable for research and deployment.

15. Potential Enhancements

- Implement model tuning with cross-validation (`tune_grid()`) to optimize hyperparameters.
- Test other classifiers (e.g., SVM, Decision Tree) within the Tidymodels framework for potential performance improvement.
- Deploy the model using **TidyPredict** to integrate predictions into SQL databases or production environments.

Detailed Information:

1. Example Usage

- To run this script, ensure that all required files are in place and use the following command:

```
R
Copy code
Rscript AppleWatchDataPredictor.R <main_path> <model_path> <training_file> <file_name> <model_type>
```

- For testing purposes, you can define the arguments directly in the script as shown.

2. Prerequisites

Before running this script, ensure the following:

Software Requirements:

- **R version:** 4.0.0 or higher
- **R packages:**
    - The script uses the `pacman` package for managing dependencies. If `pacman` is not installed, install it using:

      ```
      R
      Copy code
      install.packages("pacman")
      ```

    - The script requires the following R packages: `imputeTS`, `lubridate`, `data.table`, `dplyr`, `tidymodels`, `pryr`, `ranger`, `caret`, `doParallel`, `zoo`.
    - These packages will be installed and loaded automatically using:

      ```
      R
      Copy code
      pacman::p_load(imputeTS, lubridate, data.table, dplyr,
      tidymodels, pryr, ranger, caret, doParallel, zoo)
      ```

Required Files:

The following files are necessary for execution:

- **Training data file:**
    - `aggregated_fitbit_applewatch_jaeger.csv` (or a similar file).
    - This file should contain historical data for model training.
- **Test data file:**
    - `applewatch_data.csv` (or a similar file).
    - This file should have Apple Watch data that is used for prediction and must be located at the specified file path.

- **Model Directory:**
  - An empty directory for saving model objects, specified by `model_path`.

3. Column Requirements

The input files should have the following columns:

- **Mandatory Columns:**
  - `Heart`: Heart rate data from Apple Watch.
  - `Calories`: Calorie data from Apple Watch.
  - `Steps`: Step count from Apple Watch.
  - `Distance`: Distance covered, measured by Apple Watch.
  - `DateTime`: The date-time field for each record, used for extracting time-based features.
- **Additional Features Created:**
  - `Applewatch.Heart_LE`: Heart rate with linear interpolation applied.
  - `Applewatch.Steps_LE`: Step count with linear interpolation applied.
  - `Applewatch.Distance_LE`: Distance with linear interpolation applied.
  - `Applewatch.Calories_LE`: Calories with linear interpolation applied.
  - `EntropyApplewatchHeartPerDay_LE`: Entropy of heart rate per day.
  - `EntropyApplewatchStepsPerDay_LE`: Entropy of step count per day.
  - `RestingApplewatchHeartrate_LE`: Resting heart rate, calculated as the 5th percentile of heart rate data.
  - `NormalizedApplewatchHeartrate_LE`: Heart rate normalized by subtracting resting heart rate.
  - `ApplewatchIntensity_LE`: Intensity, calculated using the Karvonen formula.
  - `SDNormalizedApplewatchHR_LE`: Standard deviation of normalized heart rate.
  - `ApplewatchStepsXDistance_LE`: Product of steps and distance.

    - `CorrelationApplewatchHeartrateSteps_LE`: Rolling correlation between heart rate and steps over a window of 10 observations.
- **Data Checks:**
    - If any mandatory columns are missing, the script will create empty columns with NA values and issue a warning.

4. Script Structure

The code is divided into several components:

1. Load Required Libraries

- The `pacman::p_load()` function loads all necessary libraries. If any package is missing, it installs it automatically.

2. Set Time Zone and Arguments

- The script sets the time zone to `"America/St_Johns"` to ensure consistent time-based calculations.
- It accepts five arguments, which specify:
    1. **main_path:** Main directory for output files.
    2. **model_path:** Directory to save trained model objects.
    3. **training_file:** Path to the CSV file containing training data.
    4. **file_name:** Path to the CSV file containing data to predict.
    5. **model:** Model type, e.g., `"randomForest"`.

3. Utility Function for Correlation

- A utility function is defined to compute the Pearson correlation, handling cases where the standard deviation is zero to prevent errors.

4. Load and Preprocess Data

- The script reads data from `file_name`, checks for missing mandatory columns, and fills them if absent.

- **Data Imputation:** Missing values in numeric columns (`Heart`, `Calories`, `Steps`, `Distance`) are filled using linear interpolation to ensure a complete dataset.
- **Feature Engineering:** Several new features are generated to enrich the data and enhance model accuracy.

5. Tidymodels Integration

- The code utilizes **Tidymodels** for creating a structured workflow, making it easier to preprocess data, train models, and make predictions.

Data Preparation with Tidymodels:

- **Recipe Creation:**
  - The recipe includes steps to remove zero-variance predictors, impute missing values, remove remaining NAs, and normalize predictors.

```
R
Copy code
applewatch_recipe <- recipe(activity_trimmed ~ ., data = a
pplewatch_data_sample) %>%
  step_rm(...) %>%
  step_impute_median(...) %>%
  step_normalize(...)
```

- **Baking the Recipe:**
  - The prepared recipe is baked to apply transformations to the data before training.
  - Checks for missing values ensure data integrity after transformations.

6. Model Training

- The script uses a **Random Forest** model within the Tidymodels framework for classification.
- It splits the preprocessed data into training and testing sets for validation.

- **Parallel Processing:**
  - Parallel processing is used to speed up model training, utilizing available CPU cores.

Example Code for Parallel Processing:

```
R
Copy code
cl <- makeCluster(detectCores() - 1)
registerDoParallel(cl)
rf_fit <- fit(rf_workflow, data = train_data)
stopCluster(cl)
```

7. Model Evaluation

- The trained model is tested on the test set, and accuracy is calculated to evaluate model performance.
- The script also measures execution time and memory usage to identify performance bottlenecks.

```
R
Copy code
accuracy <- mean(predictions$.pred_class == predictions$activ
ity_trimmed)
```

8. Saving the Model and Predictions

- The model is saved as an RDS file using `saveRDS()`, ensuring that only the model object is saved without any data.
- The predictions are saved as a CSV file in the specified output directory.

Example of Model Saving:

```
R
Copy code
saveRDS(fitted_rf_model, file = "Tidymodels_RFModel_AppleWatc
h.rds")
```

9. Output Files

The script generates the following output files:

- **Preprocessed Data:** `preprocessed_data.csv` – Contains the cleaned and engineered data used for model training.
- **Predictions:** `applewatch_data_predicted_TinyModels.csv` – Contains the predicted activity levels with relevant features.
- **Model Object:** `Tidymodels_RFModel_AppleWatch.rds` – Contains the trained Random Forest model, saved separately for deployment.

12. Troubleshooting

- **Missing Columns:**
  - If any required columns are missing, the script will create empty columns and issue a warning.
- **Permission Denied Errors:**
  - If the script cannot save files, check the permissions for the specified directories.
- **Inconsistent Data Types:**
  - Ensure that input columns have consistent data types (e.g., numeric for heart rate, steps, etc.). Adjust preprocessing steps if necessary.