Documentation for Predictor.R with Tidymodels

Project: Apple Watch Data Predictor with Tidymodels

Programmed by: Arastoo Bozorgi & Glenn Tanjoh

Contact: glenntanjoh@gmail.com, ab1502@mun.ca

---

1. Overview

This R script is developed to process Apple Watch data, engineer features, train a predictive model using `Tidymodels`, and make predictions on activity levels. The goal is to optimize model performance, processing speed, and efficiency through a structured workflow that includes data preprocessing, feature engineering, model training, parallel processing, and saving model outputs in a reusable format.

---

2. Script Structure

The script is organized into the following components:

1. Loading required libraries
2. Setting up environment variables and arguments
3. Defining utility functions
4. Loading and preprocessing data
5. Creating a Tidymodels recipe for data transformation
6. Training and evaluating the model
7. Saving model objects and predictions
8. Cleaning up resources

---

3. Setting Up the Environment

The environment is initialized by clearing existing variables with `rm(list = ls())`, ensuring no previously stored data affects the current session.

Required Libraries

The script uses `pacman` to manage dependencies, installing and loading packages automatically:

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

This command installs any missing packages and loads all required libraries for data manipulation, model training, and parallel processing.

---

4. Time Zone and Argument Configuration

The time zone is set to `America/St_Johns` to ensure consistent time-based operations. The script accepts five arguments for specifying paths and model type, with error handling to ensure proper input.

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

If the expected number of arguments is not provided, an error is raised, ensuring paths are organized for flexible modification.

---

5. Utility Function for Correlation Calculation

The `correlation` function computes the Pearson correlation between two columns while handling cases where standard deviation is zero to avoid errors.

```
R
Copy code
correlation <- function(x) {
  if (sd(x[, 1]) == 0 || sd(x[, 2]) == 0) return(NA)
  else return(cor(x[, 1], x[, 2], method = "pearson"))
}
```

This function is later used to evaluate correlations between heart rate and step count.

---

6. Loading and Preparing Data

Data is loaded from a CSV file using `fread`, ensuring a consistent structure by checking for mandatory columns (`Heart`, `Calories`, `Steps`, `Distance`) and adding any missing columns as `NA`.

Data Imputation

Missing values in key columns are filled using linear interpolation with `imputeTS`, which provides continuity for time series data.

```
R
Copy code
applewatch_data[, Heart := imputeTS::na_interpolation(Heart,
option = "linear")]
```

This step ensures that all required features have complete data for model training.

---

7. Feature Engineering

The script creates various features to improve model accuracy:

- **Entropy Measures**: Calculates daily entropy of heart rate and step counts.

- **Resting Heart Rate**: Defined as the 5th percentile of heart rate data.
- **Normalized Heart Rate**: Heart rate adjusted relative to resting heart rate.
- **Intensity Calculation**: Based on normalized heart rate to quantify workout intensity.
- **Rolling Statistics**: Includes rolling means, lagged values, and rate of change for heart rate and steps.

```
R
Copy code
applewatch_data[, Heart_RollMean := zoo::rollmean(Heart, k =
5, fill = NA, align = "right")]
```

Feature engineering enriches the dataset, enabling the model to capture complex patterns.

---

8. Activity Level Labeling

The script labels activity levels based on step count thresholds and removes rows with missing activity levels:

```
R
Copy code
applewatch_data[, activity_trimmed := fifelse(Steps < 60, "Se
dentary",
                                              fifelse(Steps >
= 60 & Steps < 100, "Moderate",
                                                      fifelse
(Steps >= 100, "Vigorous", NA_character_)))]
```

This categorization provides meaningful target classes for training the classifier.

---

9. Data Preprocessing with Tidymodels Recipe

A Tidymodels `recipe` defines the preprocessing steps, making the workflow modular and organized.

- **Remove Zero-Variance Columns**: Excludes predictors with no variability.
- **Imputation and Normalization**: Ensures data integrity by removing rows with missing predictors and scaling numeric values.

```
R
Copy code
applewatch_recipe <- recipe(activity_trimmed ~ ., data = appl
ewatch_data) %>%
  step_zv(all_nominal_predictors()) %>%
  step_naomit(all_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

The prepared recipe is then baked to transform the data before modeling.

---

10. Model Training with Tidymodels

A `Random Forest` model is created using Tidymodels and trained on the data split into training and testing sets. Parallel processing is used to speed up model training by distributing computation across CPU cores.

```
R
Copy code
rf_model <- rand_forest(mode = "classification", mtry = 3, tr
ees = 100) %>%
  set_engine("ranger", num.threads = num_cores)
```

The model is integrated into a Tidymodels `workflow`, which combines the recipe and model definition for seamless training.

---

11. Parallel Processing

The script uses `doParallel` for efficient parallel processing, reducing training time by utilizing multiple CPU cores:

```
R
Copy code
num_cores <- detectCores() - 1
cl <- makeCluster(num_cores)
registerDoParallel(cl)
```

The cluster is stopped after training to free up system resources.

---

12. Model Evaluation and Prediction

The model is evaluated on the test set by calculating accuracy, providing a performance metric for model effectiveness.

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

Resource usage (execution time and memory) is measured to identify performance bottlenecks.

---

13. Saving Model and Predictions

After evaluation, the model and predictions are saved:

- **Model**: Stored as an RDS file for future loading and use in other applications.
- **Predictions**: Exported as a CSV for further analysis or integration into other systems.

```
R
Copy code
saveRDS(rf_fit, file = model_file)
```

```
write.csv(predictions, output_file, row.names = FALSE)
```

14. Conclusion

This script provides a complete and efficient workflow for predicting Apple Watch activity levels, structured to facilitate both research and deployment. The modular approach allows flexibility, while parallel processing enhances speed.

15. Potential Enhancements

- **Model Tuning**: Implement cross-validation with `tune_grid()` for hyperparameter optimization.
- **Classifier Comparison**: Experiment with other classifiers (e.g., SVM, Decision Tree) within Tidymodels for potential performance gains.
- **Deployment**: Integrate predictions into production using TidyPredict or similar tools for real-time applications.

Detailed Information and Example Usage

1. **Example Usage**:
   Run the script with appropriate file paths:

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

2. **Prerequisites**:
   - **R Version**: 4.0.0 or higher
   - **Packages**: `imputeTS`, `lubridate`, `data.table`, `dplyr`, `tidymodels`, `pryr`, `ranger`, `caret`, `doParallel`, `zoo`

- **Data Requirements**: Input files must contain `Heart`, `Calories`, `Steps`, `Distance` as required columns.

3. **Output Files**:
   - **Preprocessed Data**: Cleaned and transformed data file.
   - **Predictions**: CSV containing activity level predictions.
   - **Model File**: RDS file of the trained Random Forest model for reuse.
4. **Troubleshooting**:
   - **Missing Columns**: Automatically handled by creating empty columns with NA values.
   - **Permission Errors**: Ensure write access to specified directories.
   - **Inconsistent Data Types**: Verify input data for consistency; adjust preprocessing if necessary.

Script Structure Recap and Component Breakdown

Here's a detailed breakdown of each section in the script, clarifying the purpose, approach, and sequence of operations:

1. Environment Initialization

- **Command**: `rm(list = ls())`
- **Purpose**: Clears the existing environment to ensure a fresh start without leftover variables that could interfere with the script's execution.

2. Library Installation and Loading

- **Command**: `pacman::p_load(imputeTS, lubridate, data.table, dplyr, tidymodels, pryr, ranger, caret, doParallel, zoo)`
- **Purpose**: Loads essential libraries, handling dependencies to ensure that all required packages are installed and ready. This setup ensures all modules for data processing, modeling, and performance measurement are accessible.

3. Parallel Processing Setup

- **Commands**:

```
R
Copy code
num_cores <- detectCores() - 1
cl <- makeCluster(num_cores)
registerDoParallel(cl)
```

- **Purpose**: Enhances processing speed by using parallel computation, leveraging available cores for multi-threaded operations. This setup leaves one core free to keep the system responsive.

4. Setting Time Zone and Arguments

- **Command**:

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

- **Purpose**: Sets a consistent time zone for all time-related calculations and data parsing.
- **Arguments**:
  - The script expects five arguments that specify paths for the main directory, model storage, training data file, prediction data file, and model type. Missing arguments raise an error, ensuring correct execution.

5. Utility Functions

- `correlation` **Function**:
  - **Command**:

```
R
Copy code
```

```
correlation <- function(x) { ... }
```

- **Purpose**: Computes Pearson correlation between two columns and returns `NA` if the standard deviation is zero in either column, preventing potential runtime errors.

6. Data Loading and Preparation

- **Commands**:

```
R
Copy code
applewatch_data <- fread(file_name)
```

- **Purpose**: Loads Apple Watch data from a CSV file using `fread` for efficient reading. This section checks for required columns (`Heart`, `Calories`, `Steps`, `Distance`), adding them as `NA` columns if absent.
- **Data Imputation**:
  - Uses linear interpolation to fill missing values in key columns, ensuring complete data for model training.

7. Feature Engineering

- **Purpose**: Enhances the dataset with new calculated features based on heart rate, steps, and other metrics.
- **New Features**:
  - **Entropy Calculation**: Measures variability for heart rate and steps.
  - **Resting Heart Rate and Normalization**: Adds features for normalized heart rate based on a resting value.
  - **Rolling Mean, Lag, and Rate of Change**: Captures rolling averages and differences to highlight time-based patterns in the data.

8. Activity Level Labeling

- **Command**:

```
R
Copy code
applewatch_data[, activity_trimmed := fifelse(Steps < 60,
"Sedentary", ...)]
```

- **Purpose**: Categorizes data into activity levels (Sedentary, Moderate, Vigorous) based on step count thresholds, establishing target classes for classification. Rows without activity labels are removed to ensure data quality.

9. Tidymodels Recipe for Preprocessing

- **Recipe Definition**:
  - **Command**:

```
R
Copy code
applewatch_recipe <- recipe(activity_trimmed ~ ., data
= applewatch_data) %>%
  step_zv(all_nominal_predictors()) %>%
  step_naomit(all_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

  - **Purpose**: Defines a structured preprocessing workflow with Tidymodels, removing zero-variance columns, handling missing values, and normalizing numeric predictors. This modular approach standardizes data preparation for model training.

10. Model Training with Random Forest

- **Model Configuration**:
  - **Command**:

```
R
Copy code
rf_model <- rand_forest(mode = "classification", mtry =
3, trees = 100) %>%
  set_engine("ranger", num.threads = num_cores)
```

- **Purpose**: Configures a Random Forest classifier with Tidymodels, setting parameters for the number of trees and variable selection at each split. This section leverages parallel processing for efficient training.

- **Workflow Creation**:
  - **Command**:

```
R
Copy code
rf_workflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(applewatch_recipe)
```

  - **Purpose**: Combines the preprocessing recipe and model into a workflow, ensuring smooth integration of data preparation and training steps.

11. Model Evaluation and Performance Monitoring

- **Accuracy Calculation**:
  - **Command**:

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

- **Purpose**: Measures model accuracy on the test set to assess predictive performance.

- **Execution Time and Memory Usage**:
  - **Commands**:

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

  - **Purpose**: Captures execution time and memory usage, providing insights into resource requirements and allowing for potential optimizations.

12. Saving the Model and Predictions

- **Model Storage**:
  - **Command**:

```
R
Copy code
saveRDS(rf_fit, file = model_file)
```

  - **Purpose**: Saves the trained model as an RDS file, allowing for future use without retraining.

- **Prediction Output**:
  - **Command**:

```
R
Copy code
```

```
write.csv(predictions, output_file, row.names = FALSE)
```

- **Purpose**: Exports predictions to a CSV file, facilitating analysis or integration with other systems.

13. Clean-Up and Finalization

- **Commands**:

```
R
Copy code
stopCluster(cl)
registerDoSEQ()
print("Model training and prediction process completed suc
cessfully!")
```

- **Purpose**: Stops the parallel processing cluster and resets to sequential processing. This section prints a success message to indicate that the script has completed.

---

17. Potential Enhancements and Future Directions

Model Tuning and Optimization

- Implement cross-validation with `tune_grid()` to optimize hyperparameters, which can help in improving model accuracy and robustness.
- Explore additional classifiers within Tidymodels, such as SVM and Decision Trees, to benchmark and possibly enhance prediction performance.

Production-Ready Deployment

- Integrate the model with a real-time system using TidyPredict or deploy to SQL databases for production-level predictions.
- Create an API wrapper around the model for web or mobile applications, enabling easy access to predictions.

Advanced Feature Engineering

- Introduce time-series analysis and more sophisticated feature extraction to further capture temporal patterns, such as periodic trends in heart rate or steps data.
- Develop an ensemble model that combines multiple algorithms to achieve higher accuracy and potentially reduce overfitting.

Example of Full Script Execution

1. **Run the Script**:
   Ensure all required files and directories are in place. Execute the script from the command line using:

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

2. **Expected Output Files**:
   - **Preprocessed Data**: Contains the transformed dataset.
   - **Predictions**: File with predicted activity levels.
   - **Model File**: Saved Random Forest model object.
3. **Error Handling and Troubleshooting**:
   - **Missing Columns**: The script automatically generates empty columns with `NA` values for any missing mandatory fields.
   - **File Permissions**: Ensure the specified directories have write permissions to avoid permission errors.
   - **Data Type Consistency**: Verify that input data fields are consistent (e.g., numeric data for heart rate, steps). Adjust preprocessing steps as necessary.