RESEARCH PROJECT FINAL REPORT

Advanced Radiomics Analysis and Machine Learning in Multiple Sclerosis Patients

Student:
Zihui Weng
UNI:zw2985
zw2985@columbia.edu BS in CS
Exp. Grad. May 2025

Advisor:
Prof. Albert Boulanger
agb6@cumc.columbia.edu
Advisor:
Prof. Ansaf Salleb-Aouissi
ansafsalleb@columbia.edu
Mentor:
Prof. Korhan Buyukturkoglu
kb2997@cumc.columbia.edu

Department of Computer Science&
Center for Translational and Computational Neuroimmunology
January 5, 2024

Contents

1 Abstract **3**

2 Introduction **3**

3 Method **4**
3.1 Overview 4
3.1.1 Data Preprocessing - clean_up.py 4
3.1.2 Feature Selection and Machine Learning - ml_pipeline.py 5
3.2 Data set 8

4 Results **9**

5 Conclusion **10**

6 Appendices **11**

1 Abstract

This study introduces a novel statistical pipeline designed for analyzing radiomics datasets to predict SDMT scores using machine learning algorithms. The pipeline includes flexible feature selection and machine learning, emphasizing numerical radiomics features. It employs several feature selection methods, including Lasso Regression, Univariate Statistical Test, Tree-based Classifiers, Principal Component Analysis, and Mutual Information. Subsequently, machine learning algorithms - K-nearest Neighbors, Decision Tree, Random Forest, Extremely Random Trees, Adaptive Boosting, and Extreme Gradient Boosting - are applied, each tailored to the complex nature of radiomics data. The pipeline's effectiveness is evaluated using various performance metrics and is implemented on two distinct datasets: SNAPSHOT, with data from 126 MS patients, and GEMS, focusing on 154 individuals at varying MS (Multiple Sclerosis) risk levels. This approach showcases the potential of machine learning in enhancing medical predictive modeling.

2 Introduction

The analysis of radiomics data is a growing area of interest in medical research, offering insights into disease characteristics and patient outcomes. In this project, we explore a new statistical pipeline designed to analyze such data, focusing on its application in predicting cognitive scores based on SDMT (Symbol Digit Modalities Test) result among MS patients.

The Symbol Digit Modalities Test (SDMT) is a widely recognized cognitive test used in the evaluation of cognitive function in patients with Multiple Sclerosis (MS).[1] MS, a chronic autoimmune disease affecting the central nervous system, can lead to a range of neurological symptoms, including cognitive impairment.[2] The SDMT is particularly useful in this context because of its sensitivity to changes in cognitive function often observed in MS patients.Lower SDMT scores are often indicative of cognitive impairment, which is a common feature in MS, affecting approximately 40-65% of patients. [3] Changes in SDMT scores over time are indicative of MS progression and treatment efficacy. A decline in these scores may serve as an early predictor of disease advancement. [4]

In patients with Multiple Sclerosis (MS), one of the common findings on brain imaging is the presence of lesions, often indicative of the disease's ac-

tivity and progression.[5] Radiomics analysis involves extracting quantifiable data from medical images, such as pixel intensity, arrangement, color, and texture, converting these images features into a series of quantitative variables. This method reveals subtle image features that are often imperceptible to the human eye, enhancing diagnostic and prognostic capabilities in medicine.[6] Radiomics, with its advanced image analysis capabilities, might possess the potential to capture and quantify these lesions with high precision.

Building on this foundational understanding of SDMT's role in MS and the capabilities of radiomic analysis, this study introduces an innovative approach that combines these two domains. We developed a statistical pipeline that utilizes machine learning algorithms to analyze radiomic features derived from MRI scans of MS patients. The goal is to predict the SDMT scores, which are indicative of cognitive function, by identifying patterns and correlations within the radiomic data. This approach aims not only to enhance our understanding of the relationship between radiomic features and cognitive impairment in MS but also to explore the potential of machine learning in making predictive assessments based on complex medical imaging data.

3 Method

3.1 Overview

In this project, my primary goal was to develop a robust and flexible statistical pipeline tailored for the analysis of radiomic results. The central aim is to apply machine learning algorithms to numerical radiomic features for predicting SDMT scores. This pipeline consists of `clean_up.py` and `ml_pipeline.py`, intricately designed to process radiomic datasets with a particular emphasis on numerical radiomic features. The pipeline begins with data cleansing and organization, a task managed by `clean_up.py`. This is followed by a critical feature selection phase in `ml_pipeline.py`, after which machine learning algorithms are applied.

3.1.1 Data Preprocessing - clean_up.py

The preprocessing stage of the statistical pipeline, handled by `clean_up.py`, is designed to meticulously prepare the dataset for in-depth analysis. This

script is specifically programmed to exclude categorical data, focusing exclusively on numerical radiomic features. Additionally, it rigorously removes subjects that contain NaN entries and extraneous features, such as scanner versions and Python package versions. This ensures that the dataset retains only pertinent and quality data, setting a solid foundation for the subsequent stages of analysis.

3.1.2 Feature Selection and Machine Learning - ml_pipeline.py

Usage Serving as the core of the statistical pipeline, `ml_pipeline.py` is responsible for the main statistical and machine learning processes. The script is executed via a command line interface, allowing for flexibility and ease of use. The command format is `python3 ml_pipeline.py <input_csv> <output_base_name> <target_col> <feature_selection_method> [<excluded_col>]`. The arguments required for the script are listed below:

- **input_csv:** A clean radiomic result csv file
- **output_base_name:** The base name for the output file
- **target_col:** The name of the target column, which must exist in the data frame.
- **feature_selection_method**: The method to use for feature selection with the possible options:
  - lasso: Lasso Regression
  - univariate: Univariate Statistical Test,
  - tree: Tree-based Classifier
  - pca: Principal Component Analysis
  - mutual: Mutual Information
- **excluded_col(optional):**A comma-separated list of column names to be excluded. This might include non-relevant columns or categorical data (e.g., subject names).

Process The pipeline starts with a crucial feature selection phase base on the choice of users. The approach then involves the application of six distinct machine learning algorithms, including K-nearest neighbors,Decision Tree, Random Forest, Extremely Random Tree, Adaptive Boosting and Extreme Gradient Boosting to further refine the model. Finally, the process culminates with the evaluation of model performance using various metrics including R^2 value, Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). The results, including the best model parameters and performance metrics, are systematically compiled into a CSV file for thorough analysis and comparison.

Rationale for Methodology Selection I chose six commonly used feature selection methods due to their effectiveness and widespread acceptance in the field. The first method, Lasso Regression (Lasso), stands for Least Absolute Shrinkage and Selection Operator. It is a regression analysis method that combines variable selection and regularization to enhance the prediction accuracy and interpretability of the statistical model it produces. The strength of Lasso lies in its ability to shrink the coefficients of less significant features to zero, effectively eliminating them from the model, thereby simplifying the model and reducing overfitting.[7]

Following Lasso, I utilized the Univariate Statistical Test method. This approach involves conducting statistical tests on individual variables to assess their relationship with the response variable. Features demonstrating strong relationships are then selected for inclusion in the model. The simplicity and ease of understanding of Univariate tests make them a popular choice for initial feature selection in many studies.[8]

Subsequently, I incorporated Tree-based Classifiers, including algorithms like Decision Trees, Random Forests, or Gradient Boosted Trees. These models are particularly advantageous as they provide feature importances, which are instrumental in selecting the most relevant features. Additionally, Tree-based classifiers are adept at handling nonlinear relationships and interactions between features, making them a robust choice for feature selection.[9]

Another method I employed is Principal Component Analysis (PCA). PCA is a dimensionality reduction technique that transforms the original variables into a new set of variables, known as principal components. These components are linear combinations of the original variables and are selected

to explain the maximum variance in the data. The use of PCA is particularly beneficial in reducing the feature space while retaining most of the critical information present in the data.[10]

I also utilized Mutual Information, a method that measures the amount of information one variable contains about another. This approach is used to gauge the dependency between variables, with a higher mutual information value indicating a stronger relationship. In feature selection, features that share more mutual information with the response variable are typically more relevant and thus are preferred for model development.[11]

Each of these methods contributes uniquely to the feature selection process. Their integration in my research not only simplified the models but also significantly enhanced their performance by reducing overfitting and ensuring that only the most relevant features were included. This comprehensive approach to feature selection is fundamental in achieving accurate and reliable results in predictive modeling.

After feature selection, I have meticulously chosen four methods that seem apt for handling the dataset: K-Nearest Neighbors, Decision Tree, Random Forest, and Adaptive Boosting. Given the intricate nature of radiomic features, which may not always exhibit a linear relationship, these four algorithms are particularly effective. They are known to perform robustly across both linear and more complex datasets.[12]

K-nearest neighbors is particularly effective in regression problems where the relationship between variables is not explicitly linear or complex. It predicts the value of a new data point based on the average values of the 'k' nearest neighbors in the feature space.[13] This proximity-based approach allows KNN to adapt to local data variations, making it suitable for datasets where patterns may vary across different regions of the feature space.

In regression, Decision Trees partition the feature space into distinct regions, making predictions based on the mean target value within each region. This method can effectively capture non-linear relationships between features and the target variable. The hierarchical structure of decision trees enables them to model complex interactions between variables, which is beneficial in many regression scenarios.[14]

As an ensemble of Decision Trees, Random Forest in regression works by averaging the predictions of numerous trees, each trained on a different subset of the data. This approach not only captures the intricate relationships between features and the target variable but also significantly reduces overfitting, a common problem in single Decision Trees. The averaging process

smoothes out predictions, leading to a more robust and generalizable model. Random Forest also handles varying feature scales well, which is a common challenge in regression tasks.[15]

Extremely Random Trees offer a unique approach to handling complex datasets by introducing more randomness into the decision-making process of tree construction. This method differs from traditional Random Forest by randomly selecting split points and cut-offs in trees, which substantially reduces the risk of overfitting. its randomness not only contributes to the model's robustness but also enhances its speed and efficiency, particularly beneficial when dealing with large datasets. Its capability to perform well even when the predictive power of individual features is not strong makes it an excellent choice for datasets with high variance or less pronounced relationships between features and the target variable.[16]

Adaptive Boosting, when applied to regression, sequentially applies multiple weak regression models (typically simple decision trees) to iteratively correct the errors of the previous models. The algorithm focuses more on instances that were previously modeled incorrectly, thereby refining the model's accuracy with each iteration. Adaptive Boosting is particularly useful for enhancing the performance of simple models and can lead to a more nuanced understanding of complex relationships in the data, making it a powerful tool for regression tasks.[17]

Extreme Gradient Boosting (XGBoost), brings a high level of efficiency and effectiveness to model building, especially in scenarios involving sparse data. it is capable to handle missing values and zero-variance variables which makes it highly suitable for complex datasets. A key feature that sets XGBoost apart is its incorporation of regularization techniques, which helps in reducing overfitting and improving the model's generalization. This aspect is crucial for maintaining predictive performance across diverse data scenarios. Moreover, XGBoost's design focuses on computational efficiency, allowing for fast processing and scalability to larger datasets. The method also offers a high degree of flexibility with a wide range of tunable parameters, enabling precise model optimization.[18]

3.2 Data set

I have implemented this pipeline on two studies: SNAPSHOT and GEMS. The SNAPSHOT dataset encompasses data from 126 unique MS patients and features a comprehensive suite of 1,592 columns. These columns encapsulate

a broad spectrum of features and metadata, with the dataset containing a mix of textual elements (such as file paths and version details) and numerical data, primarily radiomic features.[19]

The GEMS dataset, comparable to SNAPSHOT in structure, consists of 1,589 columns. Similar to its counterpart, it encompasses textual elements such as file paths and version details, coupled with an extensive array of numerical radiomic features. A distinctive aspect of GEMS is its focus on 157 unique individuals, all healthy yet varying in MS risk levels due to family history and other factors. This diverse patient demographic offers a contrasting profile to that of the SNAPSHOT study, emphasizing potential future developments of MS in these patients.[20]

4 Results

In the GEMS study the results showcased diverse outcomes across different feature selection methods and machine learning algorithms. With the Lasso Regression feature selection, the top-performing models included K Nearest Neighbors (with a leaf size of 5 and 7 neighbors), Decision Tree (max depth 1 and min samples split of 2), Random Forest (max depth 2 and min samples split of 2), ExtraTrees, and AdaBoost. These models yielded Testing R2 scores ranging from -0.231 to -0.016, indicating some challenges in model performance.

In the Mutual Information feature selection, models such as K Nearest Neighbors (leaf size 5, 19 neighbors), Decision Tree (max depth 3, min samples split 4), Random Forest, ExtraTrees, and AdaBoost demonstrated Testing R2 scores between -0.188 and -0.030. In the Principal Component Analysis (PCA) feature selection approach, the models varied in parameters and exhibited Testing R2 scores from -0.204 to -0.099.

Using Tree-Based Classifiers, the study observed performances of K Nearest Neighbors, Decision Tree, Random Forest, ExtraTrees, and AdaBoost, with Testing R2 scores ranging from -0.175 to -0.099. The Univariate Statistical Test method highlighted models like K Nearest Neighbors, Decision Tree, Random Forest, ExtraTrees, and AdaBoost, with Testing R2 scores varying from -0.432 to -0.057.

Conversely, the analysis of the SNAPSHOT dataset showed a range of results with different feature selection methods as well. Under Lasso Regression, the best-performing models were K Nearest Neighbors (leaf size 5,

19 neighbors), Decision Tree (max depth 6, min samples split 5), Random Forest (max depth 10, min samples split 2), ExtraTrees, and AdaBoost, with Testing R2 scores from 0.019 to 0.377.

In the Mutual Information method, the top models included K Nearest Neighbors (leaf size 15, 7 neighbors), Decision Tree (max depth 7, min samples split 10), Random Forest (max depth 10, min samples split 4), ExtraTrees, and AdaBoost, showing Training R2 scores between 0.182 to 0.950.

The PCA feature selection highlighted K Nearest Neighbors, Decision Tree, Random Forest, ExtraTrees, and AdaBoost as standout models, with Testing R2 scores ranging from 0.065 to 0.696. Similarly, using Tree-Based Classifiers, models like K Nearest Neighbors, Decision Tree, Random Forest, ExtraTrees, and AdaBoost showed Testing R2 scores between 0.193 and 0.440.

Finally, the Univariate Statistical Test method in the SNAPSHOT study identified K Nearest Neighbors, Decision Tree, Random Forest, ExtraTrees, and AdaBoost as the best models, with Testing R2 scores ranging from 0.205 to 0.425.

5 Conclusion

In conclusion, the analyses of the GEMS and SNAPSHOT datasets using various machine learning algorithms and feature selection methods have provided valuable insights into the potential and limitations of predictive modeling in medical research, especially in the context of Multiple Sclerosis (MS).

The GEMS study, which focused on a cohort of healthy individuals at varying MS risk levels, showed generally poor model performance. This could be attributed to the nature of the dataset. Since the participants were all healthy, it's likely that normal brain features did not show a significant correlation with the SDMT scores. This lack of correlation likely led to the low predictive accuracy of the models, as reflected by the predominantly negative R2 scores. This suggests that in a healthy population, the radiomics features used may not be strong indicators for cognitive performance as measured by SDMT.

In contrast, the SNAPSHOT study, involving data from 126 MS patients, demonstrated better results. The presence of more pronounced radiomic variations in MS patients could correlate more with changes in SDMT scores. Importantly, the highest R2 score on the testing set was observed using the

Tree-Based Classifiers feature selection method, achieving a score of 0.440. This indicates a stronger predictive capability compared to the GEMS study.

These outcomes highlight the critical importance of considering the dataset's nature when applying machine learning models in medical research. While promising in disease-focused studies like SNAPSHOT, their effectiveness might be limited in populations without overt disease characteristics, as seen in the GEMS study. This underscores the need for context-specific approaches in predictive modeling based on the data being analyzed.

6 Appendices

Appendix A: Python Scripts

`clean_up.py`

This script is responsible for the preprocessing of the radiomic datasets. It includes functions for removing non-numerical data, handling missing values, and excluding irrelevant features to ensure the dataset is ready for analysis.[21]

```
# Author: Zihui (Aurora) Weng
# Preprocessing for the statistical pipeline

# This Python script processes and cleans the DataFrame for PyRadiomic results.
# The script is currently tailored with hard-coded values specific to a particular study.
# Note that adjustments may be required to adapt it for different studies due to varying data structures and requirements.

# "Usage: python3 clean_up.py <input_csv> <output_csv>"


import pandas as pd
import sys


def main():
    if len(sys.argv) != 3:
        print("Usage: python clean_up.py <input_csv> <output_csv>")
        sys.exit(1)

```

```
    input_csv = sys.argv[1]
    output_csv = sys.argv[2]
    df = pd.read_csv(input_csv)
    df3 = pd.read_csv('GEMS_SNAPSHOT_Aurora.csv') #This is just used to get SDMT score

    df = df.dropna()
    df.drop(df.iloc[:, 1:16], axis=1, inplace=True)
    columns_to_drop = ['diagnostics_Image-original_Minimum', 'diagnostics_Image-original_Maximum', 'diagnostics_Mask-original_Hash','diagnostics_Mask-original_Size','diagnostics_Mask-original_Spacing','diagnostics_Mask-original_BoundingBox','diagnostics_Mask-original_CenterOfMassIndex','diagnostics_Mask-original_CenterOfMass']
    df.drop(columns_to_drop, axis=1, inplace=True)
    # add SDMT score to the dataset
    # Convert 'Subject' in df3 and 'ID' in df to string
    df3['Subject'] = df3['Subject'].astype(str)
    df['ID'] = df['ID'].astype(str)

    # Merging df with df3
    df = pd.merge(df, df3[['Subject', 'SDMT']], left_on='ID', right_on='Subject', how='left')

    # Drop the 'Subject' column from the merged DataFrame as it's redundant
    df = df.drop(columns=['Subject'])

    # Reorder columns to make 'SDMT' the second column
    cols = df.columns.tolist() # Get a list of all columns
    cols.insert(1, cols.pop(cols.index('SDMT'))) # Move 'SDMT' to the second position
    df = df[cols]
    df = df.dropna()

    df.to_csv(output_csv, index=False)


if __name__ == "__main__":
    main()
```

Listing 1: clean_up.py

`ml_pipeline.py`

This script contains the machine learning pipeline used for feature selec-

tion and model training to predict SDMT scores. It applies various machine learning algorithms and evaluates their performance, with the flexibility to adjust parameters via the command-line interface.[22-26]

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: auroraweng
"""
# ml pipeline

# This Python script requires 4 to 5 arguments:
# 1. Input CSV File: A clean PyRadiomic result file.
# 2. Output File Base Name: The base name for the output file (method name will be appended).
# 3. Target Column Name: The name of the target column, which must exist in the DataFrame.
# 4. Feature Selection Method: The method to use for feature selection possible options
#    - lasso
#    - univariate
#    - tree
#    - rfe
#    - pca
#    - mutual
# 5. (Optional) Excluded Columns: A comma-separated list of column names to be excluded.
#    These might be columns not relevant to the analysis or categorical data (e.g., subject names).

# Usage: python3 ml_pipeline.py <input_csv> <output_base_name> <target_col> <feature_selection_method> [<excluded_col>]

# Example for GEMS dataset using Lasso for feature selection:
#   python3 ml_pipeline.py GEMS_output.csv GEMS_stat SDMT lasso ID

# Example for SNAPSHOT dataset using Univariate feature selection:
#   python3 ml_pipeline.py SNAPSHOT_output.csv SNAPSHOT_stat SDMT univariate ID

# Notice: This analysis is only suitable for numerical targets and numerical features.
```

```
31
32 import sys
33 import time
34 import pandas as pd
35 import numpy as np
36 from sklearn.preprocessing import StandardScaler
37 from sklearn.linear_model import LassoCV
38 from sklearn.ensemble import RandomForestRegressor
39 from sklearn.feature_selection import SelectKBest, f_classif,
       RFE, mutual_info_regression,VarianceThreshold
40 from sklearn.model_selection import train_test_split,
     GridSearchCV
41 from sklearn.metrics import mean_absolute_error,
     mean_squared_error, r2_score
42 from sklearn.linear_model import LinearRegression
43 from sklearn.neighbors import KNeighborsRegressor
44 from sklearn.tree import DecisionTreeRegressor
45 from sklearn.ensemble import RandomForestRegressor,
     AdaBoostRegressor, ExtraTreesRegressor
46 from xgboost import XGBRegressor
47 from sklearn.decomposition import PCA
48
49 #feature slection
50
51 def remove_constant_features(X):
52     """
53     Remove features with zero variance, i.e., constant
     features.
54
55     :param X: DataFrame containing the features.
56     :return: DataFrame with constant features removed.
57     """
58     selector = VarianceThreshold()
59     return pd.DataFrame(selector.fit_transform(X), columns=X.
     columns[selector.get_support()])
60
61 def corrSelect(X, y, num_features):
62     """
63     Select features based on correlation with the target
     variable.
64
65     :param X: DataFrame containing the features
66     :param y: Series or array containing the target variable
67     :param num_features: Number of top features to select
68     :return: DataFrame containing the selected features
```

```
    """
    # Calculate correlation with the target
    correlation_with_target = X.corrwith(y)

    # Sort by absolute value in descending order and select top 'num_features'
    top_features = correlation_with_target.abs().sort_values(ascending=False).head(num_features).index

    # Select the top features from the original DataFrame
    selected_X = X[top_features]

    return selected_X

def LassoSelect(X, y):
    """
    Perform feature selection using Lasso regression.

    :param X: DataFrame containing the features
    :param y: Series or array containing the target variable

    :return: DataFrame containing only the features selected by Lasso
    """
    # Standardize the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Apply Lasso Regression
    lasso = LassoCV(cv=5, max_iter=200000, tol=0.01)
    lasso.fit(X_scaled, y)

    # Identify the features that Lasso kept (non-zero coefficients)
    selected_features = X.columns[np.where(lasso.coef_ != 0)[0]]
    return X[selected_features]


def univariateSelect(X, y, num_features):
    """
    Feature selection using univariate statistical tests
    """
    # Apply SelectKBest class to extract top 'num_features' features
```

```
    best_features = SelectKBest(score_func=f_classif, k=num_features)
    fit = best_features.fit(X, y)
    df_scores = pd.DataFrame(fit.scores_)
    df_columns = pd.DataFrame(X.columns)

    # Concat two dataframes for better visualization
    feature_scores = pd.concat([df_columns, df_scores], axis=1)
    feature_scores.columns = ['Feature','Score']  # Naming the dataframe columns
    selected_features = feature_scores.nlargest(num_features, 'Score')['Feature']

    return X[selected_features]

def treeSelect(X, y, num_features):
    """
    Feature selection using tree-based feature importance
    """
    model = RandomForestRegressor()
    model.fit(X, y)
    importances = model.feature_importances_
    indices = np.argsort(importances)[::-1]

    selected_features = X.columns[indices[:num_features]]

    return X[selected_features]

def rfeSelect(X, y, num_features):
    """
    Feature selection using Recursive Feature Elimination (RFE)

    :param X: DataFrame containing the features
    :param y: Series or array containing the target variable
    :param num_features: Number of top features to select
    :return: Array of selected feature names
    """
    model = RandomForestRegressor()  # Suitable for regression tasks
    rfe = RFE(model, n_features_to_select=num_features)
    fit = rfe.fit(X, y)
    selected_features = X.columns[fit.support_]
    return X[selected_features]
```

```
147
148 def pcaSelect(X, num_components):
149     """
150     Feature selection using Principal Component Analysis (PCA
    )
151
152     :param X: DataFrame containing the features
153     :param num_components: Number of principal components to
    select
154     :return: DataFrame with the principal components
155     """
156     pca = PCA(n_components=num_components)
157     principal_components = pca.fit_transform(X)
158     columns = [f'PC{i+1}' for i in range(num_components)]
159     return pd.DataFrame(principal_components, columns=columns
    , index=X.index)
160
161 def mutualSelect(X, y, num_features):
162     """
163     Feature selection using Mutual Information
164     """
165     mi = mutual_info_regression(X, y)
166     mi /= np.max(mi)  # Normalize the MI scores
167     selected_features = X.columns[mi.argsort()[-num_features
    :][::-1]]
168     return X[selected_features]
169
170 # Machine learning
171 class Regressors():
172     def __init__(self, X,y):
173         '''
174         Convert the given pandas dataframe into training and
    testing data.
175         '''
176         X = X.to_numpy()
177         y = y.to_numpy()
178         self.training_data, self.testing_data, self.
    training_labels, self.testing_labels = train_test_split(X,
     y, test_size=0.2, random_state=42)
179         self.outputs = []
180
181
182     def test_regressor(self, reg, regressor_name=''):
183         # Fit the regressor and extract metrics
184         reg.fit(self.training_data, self.training_labels)
```

```
185
186         # Extract the best parameters if GridSearchCV is used
187         best_params = reg.best_params_ if isinstance(reg,
    GridSearchCV) else 'N/A'
188
189         # Predictions for training and testing sets
190         y_pred_train = reg.predict(self.training_data)
191         y_pred_test = reg.predict(self.testing_data)
192
193         # Calculate metrics for training data
194         mae_train = mean_absolute_error(self.training_labels,
     y_pred_train)
195         mse_train = mean_squared_error(self.training_labels,
    y_pred_train)
196         rmse_train = np.sqrt(mse_train)
197         r2_train = r2_score(self.training_labels,
    y_pred_train)
198
199         # Calculate metrics for testing data
200         mae_test = mean_absolute_error(self.testing_labels,
    y_pred_test)
201         mse_test = mean_squared_error(self.testing_labels,
    y_pred_test)
202         rmse_test = np.sqrt(mse_test)
203         r2_test = r2_score(self.testing_labels, y_pred_test)
204
205         # Create a dictionary for the results
206         results_dict = {
207             'Model': regressor_name,
208             'Best Parameters': best_params,
209             'Training MAE': mae_train,
210             'Training MSE': mse_train,
211             'Training RMSE': rmse_train,
212             'Training R2': r2_train,
213             'Testing MAE': mae_test,
214             'Testing MSE': mse_test,
215             'Testing RMSE': rmse_test,
216             'Testing R2': r2_test
217         }
218
219         # Append the results dictionary to the outputs list
220         self.outputs.append(results_dict)
221
222     def regressWithKNeighbors(self):
223         # Code to run a K Nearest Neighbors regressor
```

```
224         reg = GridSearchCV(KNeighborsRegressor(), {'
    n_neighbors': list(range(1, 20, 2)), 'leaf_size': list(
    range(5, 31, 5))}, cv=5)
225         self.test_regressor(reg, 'K Nearest Neighbors')
226 
227     def regressWithLinear(self):
228         # Linear Regression does not have hyperparameters for
     GridSearch in its basic form
229         reg = LinearRegression()
230         self.test_regressor(reg, 'Linear Regression')
231 
232     def regressWithDecisionTree(self):
233         # Decision Tree Regressor with GridSearch
234         reg = GridSearchCV(DecisionTreeRegressor(),
235                            {'max_depth': list(range(1, 51)),
    'min_samples_split': list(range(2, 11))},
236                            cv=5)
237         self.test_regressor(reg, 'Decision Tree')
238 
239     def regressWithRandomForest(self):
240         # Random Forest Regressor with GridSearch
241         reg = GridSearchCV(RandomForestRegressor(),
242                            {'max_depth': list(range(1, 11)),
    'min_samples_split': list(range(2, 11))},
243                            cv=5)
244         self.test_regressor(reg, 'Random Forest')
245 
246     def regressWithExtraTrees(self):
247         # ExtraTrees Regressor with GridSearch
248         param_grid = {
249             'n_estimators': list(range(10, 101, 10)),
250             'max_features': [None, 'sqrt', 'log2']
251         }
252         reg = GridSearchCV(ExtraTreesRegressor(), param_grid,
     cv=5)
253         self.test_regressor(reg, 'ExtraTrees')
254 
255     def regressWithAdaBoost(self):
256         # AdaBoost Regressor with GridSearch
257         reg = GridSearchCV(AdaBoostRegressor(),
258                            {'n_estimators': list(range(10,
    101, 10)), 'learning_rate': [0.01, 0.1, 1]},
259                            cv=5)
260         self.test_regressor(reg, 'AdaBoost')
261 
```

```
    def regressWithXGBoost(self):
        param_grid = {
            'n_estimators': [100, 200, 300],
            'learning_rate': [0.01, 0.1, 0.2],
            'max_depth': [3, 4, 5]
        }
        reg = GridSearchCV(XGBRegressor(objective='reg:squarederror'), param_grid, cv=5)
        self.test_regressor(reg, 'XGBoost')

def main():
    start_time = time.time()
    if len(sys.argv) not in [5, 6]:
        print("Usage: python statistical_pipeline.py <input_csv> <output_csv> <target_col> <feature_selection_method> <excluded_col(optional)>")
        sys.exit(1)

    # Assign command-line arguments to respective variables
    input_csv = sys.argv[1]
    output_csv = sys.argv[2]
    target_col = sys.argv[3]
    feature_selection_method = sys.argv[4].lower()  # Capture the feature selection method
    excluded_col = sys.argv[5] if len(sys.argv) == 6 and sys.argv[5].lower() != 'none' else None

    output_csv = f"{output_csv}_{feature_selection_method}.csv"
    # Read the input CSV file into a DataFrame
    df = pd.read_csv(input_csv)
    y = df[target_col].astype(int)
    excluded_col_list = excluded_col.split(',') if excluded_col else []
    if target_col not in excluded_col_list:
        excluded_col_list.append(target_col)
    X = df.drop(columns=excluded_col_list, errors='ignore', axis=1).astype(int)
    X = remove_constant_features(X)

    # Feature selection based on the method specified
    if feature_selection_method == 'lasso':
        X_selected = LassoSelect(X, y)
    elif feature_selection_method == 'univariate':
        X_selected = univariateSelect(X, y, num_features=10)
```

```
299     elif feature_selection_method == 'tree':
300         #X = corrSelect(X, y, num_features=100)
301         X_selected = treeSelect(X, y, num_features=10)
302     elif feature_selection_method == 'rfe':
303         #X = corrSelect(X, y, num_features=100)
304         X_selected = rfeSelect(X, y, num_features=10)
305     elif feature_selection_method == 'pca':
306         #X = corrSelect(X, y, num_features=100)
307         X_selected = pcaSelect(X, num_components=10)
308     elif feature_selection_method == 'mutual':
309         #X = corrSelect(X, y, num_features=100)
310         X_selected = mutualSelect(X, y, num_features=10)
311     else:
312         print("Invalid feature selection method.")
313         sys.exit(1)
314 
315     # Print the feature selection method and the names of the
    selected features
316     print(f"Selected Features using {feature_selection_method
    .capitalize()} method:")
317     for feature_name in X_selected.columns:
318         print(feature_name)
319     print()
320     models = Regressors(X_selected, y)
321 
322 
323     # Running all the regression methods
324     #print('Regressing with Linear Regression...')
325     #models.regressWithLinear()
326     print('Regressing with K Neighbors...')
327     models.regressWithKNeighbors()
328     print('Regressing with Decision Tree...')
329     models.regressWithDecisionTree()
330     print('Regressing with Random Forest...')
331     models.regressWithRandomForest()
332     print('Regressing with ExtraTrees...')
333     models.regressWithExtraTrees()
334     print('Regressing with AdaBoost...')
335     models.regressWithAdaBoost()
336     print('Regressing with XGBoost...')
337     models.regressWithXGBoost()
338     # End timing
339     end_time = time.time()
340     # Calculate total runtime
341     total_time = end_time - start_time
```

```
    print("The code ran for", total_time, "seconds")
    # Output the results to a file
    with open(output_csv, "w") as f:
        # Print the header
        print('Model,Best Parameters,Training MAE,Training MSE,Training RMSE,Training R2,Testing MAE,Testing MSE,Testing RMSE,Testing R2', file=f)

        # Print the rows for each model's results
        for result_dict in models.outputs:
            # Create a list to store the formatted results
            result_list = [
                result_dict.get('Model', 'N/A'),
                '"' + str(result_dict.get('Best Parameters', 'N/A')) + '"',  # Ensure best parameters are quoted if they're a string representation of a dictionary
                result_dict.get('Training MAE', 'N/A'),
                result_dict.get('Training MSE', 'N/A'),
                result_dict.get('Training RMSE', 'N/A'),
                result_dict.get('Training R2', 'N/A'),
                result_dict.get('Testing MAE', 'N/A'),
                result_dict.get('Testing MSE', 'N/A'),
                result_dict.get('Testing RMSE', 'N/A'),
                result_dict.get('Testing R2', 'N/A')
            ]

            # Join the list items into a comma-separated string and write to file
            print(','.join(str(item) for item in result_list), file=f)

if __name__ == "__main__":
    main()
```

Listing 2: ml_pipeline.py

Appendix B: Bash Scripts
run_all_methods_GEMS.sh

```
#!/bin/bash

# Record the start time
start_time=$(date +%s)

# List of feature selection methods
feature_selection_methods=("lasso" "univariate" "tree" "rfe" "pca" "mutual")
```

```

# Loop through feature selection methods and run the commands
for method in "${feature_selection_methods[@]}"; do
    echo "Running ML pipeline for method: $method"
    time python3 ml_pipeline.py "GEMS_output.csv" "GEMS_stat" "SDMT" "$method" "ID"
    echo    # Empty line
done

# Record the end time
end_time=$(date +%s)

# Calculate and display the total execution time
execution_time=$((end_time - start_time))
echo "Total execution time: $execution_time seconds"

echo "All methods for GEMS completed."
```

Listing 3: run_all_methods_GEMS.sh

run_all_methods_SNAPSHOT.sh

```
#!/bin/bash

# Record the start time
start_time=$(date +%s)

# List of feature selection methods
feature_selection_methods=("lasso" "univariate" "tree" "rfe" "pca" "mutual")

# Loop through feature selection methods and run the commands
for method in "${feature_selection_methods[@]}"; do
    echo "Running ML pipeline for method: $method"
    time python3 ml_pipeline.py "SNAPSHOT_output.csv" "SNAPSHOT_stat" "SDMT" "$method" "ID"
    echo    # Empty line
done

# Record the end time
end_time=$(date +%s)

# Calculate and display the total execution time
execution_time=$((end_time - start_time))
echo "Total execution time: $execution_time seconds"

```

```
23 echo "All methods for SNAPSHOT completed."
```

Listing 4: run_all_methods_GEMS.sh

All these models were trained on a MacBook Pro that is 2023 model with an Apple M2 Max chip and 64 GB memory. The operating system on the MacBook Pro is macOS Sonoma 14.1. The bash script were done in around 5 minutes on GEMS and SNAPSHOT datasets.

Appendix C: CSV Files

GEMS_result.csv & SNAPSHOT_result.csv

These files contain the raw radiomic results from the GEMS and SNAPSHOT studies, respectively. They include the initial data as extracted from medical imaging prior to any preprocessing.

GEMS_output.csv & SNAPSHOT_output.csv

Processed datasets derived from the corresponding raw result files. These files represent the data post-cleanup, where non-numerical data and missing values have been addressed.

GEMS_stat_lasso.csv, GEMS_stat_mutual.csv, GEMS_stat_pca.csv, GEMS_stat_tree.csv, GEMS_stat_univariate.csv, SNAPSHOT_stat_lasso.csv, SNAPSHOT_stat_mutual.csv, SNAPSHOT_stat_pca.csv, SNAPSHOT_stat_tree.csv & SNAPSHOT_stat_univariate.csv

These files document the final results of the machine learning model performance for the GEMS and SNAPSHOT studies using the corresponding feature selection. The result include metrics such as MAE, MSE, RMSE, and R^2 values.

References

[1]Strober L, DeLuca J, Benedict RH, Jacobs A, Cohen JA, Chiaravalloti N, Hudson LD, Rudick RA, LaRocca NG; Multiple Sclerosis Outcome Assessments Consortium (MSOAC). Symbol Digit Modalities Test: A valid clinical trial endpoint for measuring cognition in multiple sclerosis. Mult Scler. 2019 Nov;25(13):1781-1790. doi: 10.1177/1352458518808204. Epub 2018 Oct 18. PMID: 30334474; PMCID: PMC6826875.

[2]Ghasemi N, Razavi S, Nikzad E. Multiple Sclerosis: Pathogenesis, Symptoms, Diagnoses and Cell-Based Therapy. Cell J. 2017 Apr-Jun;19(1):1-10. doi: 10.22074/cellj.2016.4867. Epub 2016 Dec 21. PMID: 28367411; PMCID: PMC5241505.

[3]Migliore S, Ghazaryan A, Simonelli I, Pasqualetti P, Squitieri F, Curcio G, Landi D, Palmieri MG, Moffa F, Filippi MM, Vernieri F. Cognitive Impairment in Relapsing-Remitting Multiple Sclerosis Patients with Very Mild Clinical Disability. Behav Neurol. 2017;2017:7404289. doi: 10.1155/2017/7404289. Epub 2017 Aug 15. PMID: 28912625; PMCID: PMC5574272.

[4]Gromisch ES, Dhari Z. Identifying Early Neuropsychological Indicators of Cognitive Involvement in Multiple Sclerosis. Neuropsychiatr Dis Treat. 2021 Feb 5;17:323-337. doi: 10.2147/NDT.S256689. PMID: 33574669; PMCID: PMC7872925.

[5]Lassmann H. Multiple Sclerosis Pathology. Cold Spring Harb Perspect Med. 2018 Mar 1;8(3):a028936. doi: 10.1101/cshperspect.a028936. PMID: 29358320; PMCID: PMC5830904.

[6]Elmahdy M, Sebro R. Radiomics analysis in medical imaging research. J Med Radiat Sci. 2023 Mar;70(1):3-7. doi: 10.1002/jmrs.662. Epub 2023 Feb 10. PMID: 36762402; PMCID: PMC9977659.

[7]R. Muthukrishnan and R. Rohini, "LASSO: A feature selection technique in predictive modeling for machine learning," 2016 IEEE International Conference on Advances in Computer Applications (ICACA), Coimbatore, India, 2016, pp. 18-20, doi: 10.1109/ICACA.2016.7887916.

[8]Abellana, D. P. M., & Lao, D. M. (2023). A new univariate feature selection algorithm based on the best–worst multi-attribute decision-making method. Decision Analytics Journal, 7, 100240. Elsevier BV. https://doi.org/10.1016/j.dajour.2023

[9]Chen, RC., Dewi, C., Huang, SW. et al. Selecting critical features for data classification based on machine learning methods. J Big Data 7, 52 (2020). https://doi.org/10.1186/s40537-020-00327-4

[10]Jolliffe Ian T. and Cadima Jorge 2016Principal component analysis: a review and recent developmentsPhil. Trans. R. Soc. A.3742015020220150202 http://doi.org/10.1098/rsta.2015.0202

[11]M. A. Sulaiman and J. Labadin, "Feature selection based on mutual information," 2015 9th International Conference on IT in Asia (CITA), Sarawak, Malaysia, 2015, pp. 1-6, doi: 10.1109/CITA.2015.7349827.

[12]Brnabic, A., Hess, L.M. Systematic literature review of machine learning methods used in the analysis of real-world data for patient-provider decision making. BMC Med Inform Decis Mak 21, 54 (2021). https://doi.org /10.1186/s12911-021-01403-2

[13]K. Taunk, S. De, S. Verma and A. Swetapadma, "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification," 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019, pp. 1255-1260, doi: 10.1109/ICCS45141.2019.9065747.

[14]Song YY, Lu Y. Decision tree methods: applications for classification and prediction. Shanghai Arch Psychiatry. 2015 Apr 25;27(2):130-5. doi: 10.11919/j.issn.1002-0829.215044. PMID: 26120265; PMCID: PMC4466856.

[15]Khaled Fawagreh, Mohamed Medhat Gaber & Eyad Elyan (2014) Random forests: from early developments to recent advancements, Systems Science & Control Engineering, 2:1, 602-609, DOI: 10.1080 /21642583.2014.956265

[16]Geurts, Pierre & Ernst, Damien & Wehenkel, Louis. (2006). Extremely Randomized Trees. Machine Learning. 63. 3-42. 10.1007/s10994-006-6226-1.

[17]Wang, R. (2012). AdaBoost for Feature Selection, Classification and Its Relation with SVM, A Review. Physics Procedia, 25, 800-807. https://doi.org /10.1016/j.phpro.2012.03.160

[18]Hasan, M. K., Alam, M. A., Roy, S., Dutta, A., Jawad, M. T., Das, S. (2021). Missing value imputation affects the performance of machine learning: A review and analysis of the literature (2010–2021). Informatics in Medicine Unlocked, 27, 100799. https://doi.org/10.1016/j.imu.2021.100799

[19]Z. Xia, S. U. Steele, A. Bakshi, S. R. Clarkson, C. C. White, M. K. Schindler, G. Nair, B. E. Dewey, L. R. Price, J. Ohayon, L. B. Chibnik, I. C. Cortese, P. L. D. Jager, and D. S. Reich, "Assessment of early evidence of multiple sclerosis in a prospective study of asymptomatic high-risk family members," JAMA Neurology, vol. 74, 2017.

[20]K. Buyukturkoglu, J. D. Dworkin, V. Leiva, F. A. Provenzano, P. Guevara, P. L. De Jager, V. M. Leavitt, and C. S. Riley, "Brain volumetric correlates of remotely versus in-person administered symbol digit modalities test

in multiple sclerosis," Multiple Sclerosis and Related Disorders, vol. 68, p. 104247, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2211034822007519

[21]McKinney, W., & others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

[22]Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2

[23]J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[24]Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021.

[25]Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[26]Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939785