POLITECNICO DI TORINO

in collaboration with

UNIVERSITÄT ZU KÖLN

Master's Degree Thesis
ICT for Smart Societies

Machine Learning Algorithms for Radiogenomics: Application to Prediction of the MGMT promoter methylation status in mpMRI scans

Data Science Lab at University Hospital Cologne

Candidate
Mostafa KARAMI

Supervisor at Polito
Prof. Monica VISINTIN

Supervisor at UC
Prof. Liliana CALDEIRA

October 2022

Abstract

Glioblastomas are the most aggressive and destructive forms of solid brain tumors of the central nervous system. Despite aggressive multimodal treatment approaches, the overall survival period is reported to be less than 15 months after diagnosis. MGMT gene silencing is a potentially proper predictive element in determining the mortality rate for glioblastoma patients undergoing chemotherapy. Analyzing the correlation between different medical image characteristics and MGMT promoter methylation status through machine learning tools could play an essential role in the automatic aided diagnosis approach.

By doing data preprocessing and transformation, this thesis aims to extract features from data provided by the Radiological Society of North America (RSNA) and investigate this relationship with various classification methods like Logistic Regression (LR), Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP). Using Pyradiomics, an open-source python package, 1153 features have been extracted from the images, their segmented forms, and Laplacian of Gaussian (LoG) and Wavelet filters to get features with more details. The Extreme Gradient Boosting (XGBoost) classifier is used since it is desirable to reduce the number of features to improve performance and identify the optimal number for the most relevant results.

The model used nested cross-validation (CV) to prevent information leakage and obtain better outcomes by finding the best set of hyperparameters for the chosen models. This approach is also exclusively applied to different MRI sequence types such as Fluid Attenuated Inversion Recovery (FLAIR), T1-weighted pre-contrast (T1), T1-weighted post-contrast (T1ce), and T2-weighted (T2) in order to point out the importance of each for final user support and future research purposes. Different statistical metrics such as accuracy, F1 score, and confusion matrix are considered for each classification algorithm.

This study demonstrated acceptable performance by the proposed feature extraction, feature selection methods, and machine learning classification algorithms. Although the deep learning approach would result in better performance metrics, considering computational load and time-space trade-off, using radiomics features and performing classification lead to valuable results in quicker time for the final user. Undoubtedly, better results can be obtained by accessing more extensive data, which points out the importance of data quantity. Analyzing different optimal features also would be a good starting point for future research to focus on the most critical aspect of brain tumor MRI images.

Acknowledgements

I would first like to thank my thesis advisor, Dr. Caldeira, at the University of Cologne. She consistently steered me in the right direction whenever she thought I needed it. I would also like to thank **Dr. Visintin**, my thesis advisor from the Polytechnic University of Turin, for accepting me as her student and letting me be part of this journey.

I must express my deepest gratitude to my parents and my two wonderful brothers, **Morteza** and **Alireza**, for providing unfailing support and continuous encouragement throughout my years of study and through researching and writing this thesis. This accomplishment would not have been possible without them.

Mostafa Karami

Table of Contents

List of Tables

List of Figures

Acronyms

GBM
Glioblastomas

MGMT
O^2-methylguanine DNA methyltransferase+

MRI
Magnetic Resonance Imaging

FLAIR
Magnetic Resonance Imaging

T1
T1-weighted pre-contrast

T1ce
T1-weighted post-contrast

T2
T2-weighted

DICOM
Digital Imaging and Communications in Medicine

NIFTI
Neuroimaging Informatics Technology Initiative

mpMRI
multi-parametric MRI

AI

Artificial Intelligence

ML

Machine Learning

IoT

Internet of Things

CV

cross-validation

SVM

Support Vector Machine

MLP

Multi-Layer Perceptron

LR

Logistic Regression

std

Standard Deviation

CNN

Convolutional Neural Network

LoG

Laplacian of Gaussian

XGBoost

Extreme Gradient Boosting

PR

Precision Recall

TR

True Positive

FP

False Positive

TN

True Negative

FN

False Negative

Chapter 1

Introduction

1.1 Background

Brain Glioblastomas (GBM) tumors represent about 50% of all primary brain tumors [1], and due to the fact that it is the most aggressive and exceptionally invasive type of brain tumor, the treatment of GBM has still been believed to be the most challenging job in clinical oncology. GBM suffers an unfavorable prognosis despite all the worldwide efforts such as surgical excision, chemotherapy, and radiotherapy [2].

O^2-methylguanine DNA methyltransferase (MGMT) is a DNA repair enzyme that removes the guanine-alkyl group induced by alkylating agents such astemozolomide (TMZ) [3]. The methylation of the MGMT promoter, which epigenetically silences the MGMT gene, has become a reliable prognostic and predictive biomarker of the World Health Organization (WHO) [4–6]. Accordingly, measurement of MGMT promoter methylation status would be a proper approach to consider for clinical evaluation of glioma patients. The typical implementation of MGMT methylation status in clinical practice is challenging even though substantial evidence emphasizing its role in prognosis. Some mentionable controversial scenarios specifying the statuses of MGMT are the optimal cut-off methods and definitions by pyrosequencing in glioblastoma. The main significant challenge is a variation in diagnostic methods, which are not the same between different laboratories.

One of the most effective tools for diagnosis and monitoring the treatment is Magnetic Resonance Imaging (MRI) which plays a vital role in glioma prognosis studies. In recent years, MRI-imaging features have been considered as predictive objective indications of the medical state of GBM. According to studies, imaging characteristics such as the extent of edema, preoperative tumor volume, degree of

contrast enhancement, and necrosis have particular GBM predictive value [7]. That is why to illuminate the meaningful link between imaging features and survival or GBM genomic alterations, the accurate and reproducible measurement of MR features is required [8].

Radiogenomic analysis connected the MR characteristics to gene expression profiles in GBM, and by extracting them, the molecular properties of tumors could be predicted [9]. However, since these characteristics are often considered qualitatively, interpretation becomes crucial. To clarify the relevant relationship between imaging characteristics and survival or GBM genomic mutations, precise and repeatable evaluation of MR features is essential [10].

As stated by [11], remote-controlled robots will become more widespread in various fields in the near future. Recent studies demonstrate the engagement of several researchers in wireless communication [12], [13], and [14] improve an existing system by addressing pertinent difficulties. Machine learning (ML) and other data analysis techniques are being developed to process medical data analysis. The integration of Machine Learning and other technologies, such as the Internet of Things (IoT), resulted in the smart hospital development project developed by [15] in order to efficiently manage hospitalized patients. ML is equivalent to the research and creation of models and algorithms for automated gaining knowledge from massive, complex data sets. Most biological studies that analyze enormous amounts of data consider using computational techniques like ML. These methods have several advantages, including the ability to exploit complicated correlations in high resolution and enable the quantification of tumor approaches that rely on imaging.

1.2 Motivation and Goals

To predict the MGMT value (indicator for the presence of MGMT promoter methylation), in this study, several features have been extracted using the Brain Tumor Segmentation (BraTS) dataset provided by the Radiological Society of North America (RSNA) [16]. All BraTS multiparametric MRI (mpMRI) scans are accessible in Digital Imaging and Communications in Medicine (DICOM) files (.dcm) for Task 2 (Prediction of the MGMT promoter methylation status) and NIfTI files (.nii.gz) for Task 1 (Brain Tumor Segmentation in mpMRI scans.) The dataset for each task provided is regarded separately, and that is why 26 patients from task 2 have not segmented form. Hence they are excluded from the dataset for this study since the segmented form is essential for the feature extraction stage. These mpMRI images were obtained using various clinical procedures and scanners

from several institutions which describe the following sequence types:

- **Fluid Attenuated Inversion Recovery (FLAIR)**, fluid suppression by setting an inversion time that nulls fluids.
- **T1-weighted pre-contrast (T1)**, measuring spin–lattice relaxation by using a short repetition time (TR) and echo time (TE).
- **T1-weighted post-contrast (T1ce)**, providing a novel approach to optimal myocardial nulling for late gadolinium enhancement imaging.
- **T2-weighted (T2)**, measuring spin–spin relaxation by using long TR and TE times.

Following the same annotation process, one to four raters manually segmented all imaging datasets, and expert board-certified neuroradiologists confirmed their annotations. Annotations comprise the GD-enhancing tumor (ET—label 4), the peritumoral edematous/invaded tissue (ED—label 2), and the necrotic tumor core (NCR—label 1). Visualizing the images and segmented area is a crucial aspect, that covered in this thesis. The best view of the tumor for each sequence typed has been found and presented to the final users.

By using the original images, the filtered versions (Laplacian of Gaussian and Wavelet), and their segmented form, raw data has been obtained using `pyradiomics`, an open-source python package for extracting radiomics features from medical imaging. The output data is formed by 11336 number of subjects and 1155 features, which are separated into different segmented areas for each MRI sequence type. Extreme Gradient Boosting (XGBoost) classifier is used for feature selection in different files that eliminate unnecessary and worthless features and keeps only those that are relevant. The top 20 features were separated as a means to associate MGMT methylation status with glioma progression. Various classification models such as LR, SVM, and MLP are used to investigate the statistical metrics such as accuracy, F1 score, confusion matrix and etc.

Another essential ideal of the thesis is to evaluate the result for each MRI sequence type as a different scenario and compare the outcome to provide a better interpretation of the dataset for final users.

1.3 Related Studies

A few surveys addressing the topic of Machine Learning-based processes for MGMT methylation status prediction have been proposed in recent years. This section

provides an overview of relevant research and explains the critical points of previous surveys. One of the noticeable research in this area is MRI-based machine learning for determining quantitative and qualitative characteristics affecting the survival of glioblastoma multiforme [17]. In this article, it has been shown that MGMT promoter methylation is related to better outcomes in glioblastoma (GBM) patients and may be a measure of chemotherapy sensitivity. It has been shown that MGMT promoter methylation is related to better outcomes in glioblastoma (GBM) patients and may be a measure of chemotherapy sensitivity. MRI scans from GBM patients were retrospectively analyzed. ML techniques were used to create multivariate prediction models, which were then evaluated using information from The Cancer Genome Atlas (TCGA) database. Accuracy of up to 73.6% was achieved in predicting the methylation status of the MGMT promoter. According to experimental investigation, the most critical factors affecting the level of MGMT promoter methylation in GBM were:

- The edema/necrosis volume ratio.
- Tumor/necrosis volume ratio.
- Edema volume.
- Tumor location and enhancement features.

The acquired findings provide further proof that MGMT methylation status in GBM and common preoperative MRI variables are related. The following research is MRI-based machine learning for determining quantitative and qualitative characteristics affecting the survival of glioblastoma multiforme [18] that aims to examine the image biomarkers taken from MRI pictures to see how they affect the survival of individuals with glioblastoma multiforme (GBM). Finding its biomarker aids in improved illness management and therapy evaluation. Imaging characteristics have been shown to be potential biomarkers. This research aims to examine the correlation between MRI and clinical characteristics as a biomarker for GBM survival. Five clinical factors, 10 pre-operative MRI imaging features, and six quantitative features derived from the BraTumIA software were taken into consideration for 55 patients. To identify key variables, ANN, C5, Bayesian, and Cox models were run in two stages. The first stage has chosen the quality features that appear in at least three models and are quantitative in two models. The probability value of variables in each model was then determined in the second part, which revealed that the greatest sizes of the breadth and length, radiation, the volume of enhancement, the volume of nCET, enhancing margin, and age feature are vital characteristics.

In the subsequent study, Machine learning-based radiomic evaluation of treatment response prediction in glioblastoma [19], ML-based models that include clinical,

radiomic, and genomic data have been investigated to differentiate between genuine early progression (tPD) and pseudoprogression (psPD) in patients with glioblastoma. Following chemoradiotherapy for glioblastoma, 76 patients (46 tPD, 30 psPD) with the early enhancing illness underwent a retrospective study. Following patients for six months after chemoradiotherapy, the result was assessed. In early post-chemoradiotherapy contrast-enhanced T1-weighted imaging (T1), T2-weighted imaging (T2), and apparent diffusion coefficient (ADC) maps, enhancing disease and perilesional oedema masks were used to extract 307 quantitative imaging features. These models also included clinical characteristics and the status of the (MGMT) promoter's methylation. A random forest approach was used for feature selection inside bootstrapped cross-validated recursive feature removal. The resultant model was validated using a naive Bayes five-fold cross-validation procedure.

In summary, evaluating different sequence types in mentioned research remains a nontrivial task. Therefore, in this work, the dataset went through specific preparation to separate each sequence and its association with MGMT value by exploiting only the top selected features.

Chapter 2

Materials & Methods

2.1 Dataset

The 559 MRI images that were taken into consideration for this study create the dataset. A five-digit number uniquely identifies each separate case's specific folder, which is the patient's ID number. Within each of these case folders, four sub-folders correspond to the structural multi-parametric MRI (mpMRI) scans in DICOM format. The header and image data are both contained in a single file in the DICOM format. The amount of header information affects how big the header will be. The patient's ID, patient name, modality, and other information are included in the header. It also specifies the resolutions and the number of frames that are included. Many DICOM files will be generated for a single acquisition.

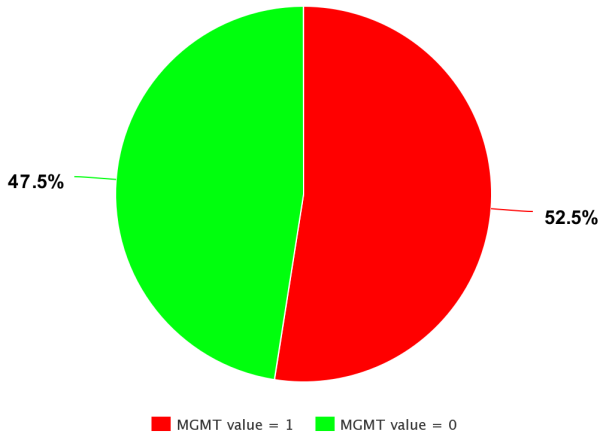

Figure 2.1: Data distribution according to the target variable, MGMT value

One of the other formats used to store brain imaging data obtained using MRI is Neuroimaging Informatics Technology Initiative (NIFTI), which originates in neuro-imaging but can be used in other fields as well. A major feature is that the format contains two affine coordinate definitions which relates each voxel index (i, j, k) to a spatial location (x, y, z).

The primary distinction between DICOM and NIfTI is that NIfTI saves the raw image data as a 3D picture, whereas DICOM uses 2D image slices. Because NIFTI is represented as a 3D picture, it may be more suitable for ML applications than with respect to the DICOM format. Aslo, it is simpler to manage a single NIFTI file than several hundred DICOM files. Unlike DICOM, which uses several files for each 3D picture, NIFTI only uses two.

In this thesis, by using `dicom2nifti` and `nibabel` python packages, all the dataset have been converted to NIFTI format for further process.

2.2 Visualization

Medical visualization research to date has focused primarily on supporting physicians in diagnosis and treatment and, to a lesser extent-medical students, particularly for anatomy education [21]. That is why representing medical data clearly and effectively is a primary step before any processing.

The converted dataset to NIFTI format, was used to project 3D images in 3 planes, which are axial in an X-Y, sagittal in a Y-Z, and coronal in X-Z. Fig. 2.2 shows an MRI image of the patient's ID 00009. One of the privileges of the used dataset is the presence of segmented images that are not only used for feature extraction but also have a positive representation point for the final user. To better understand the original images and the segmented forms, it is a good idea to find the best view in each projected plane that shows more tumor area. A searching algorithm identifies different voxel values as tumor labels, and the best-projected view is found. The figs. 2.3 to 2.6 display the best view of the original and the whole segmented tumor for the different patient's ID, and different sequence types.

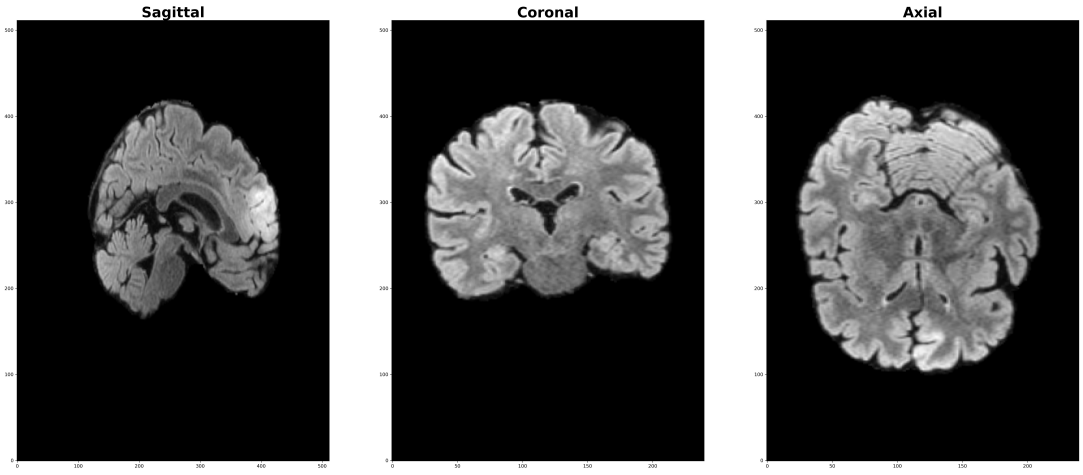

Figure 2.2: FLAIR MRI image in different planes

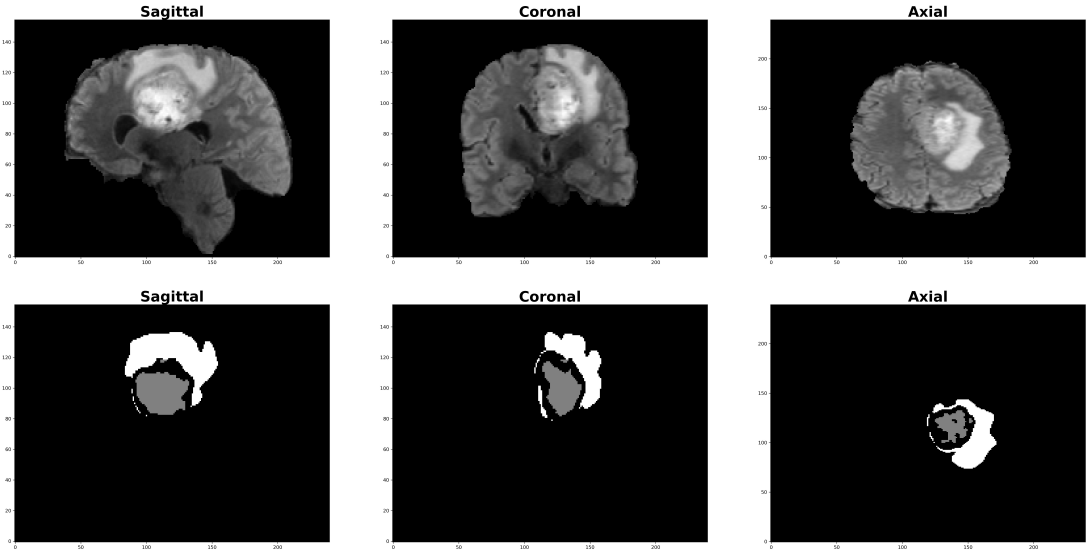

Figure 2.3: FLAIR original MRI image with the segmented form of the tumor in the best-projected views - Patient's ID: 00003

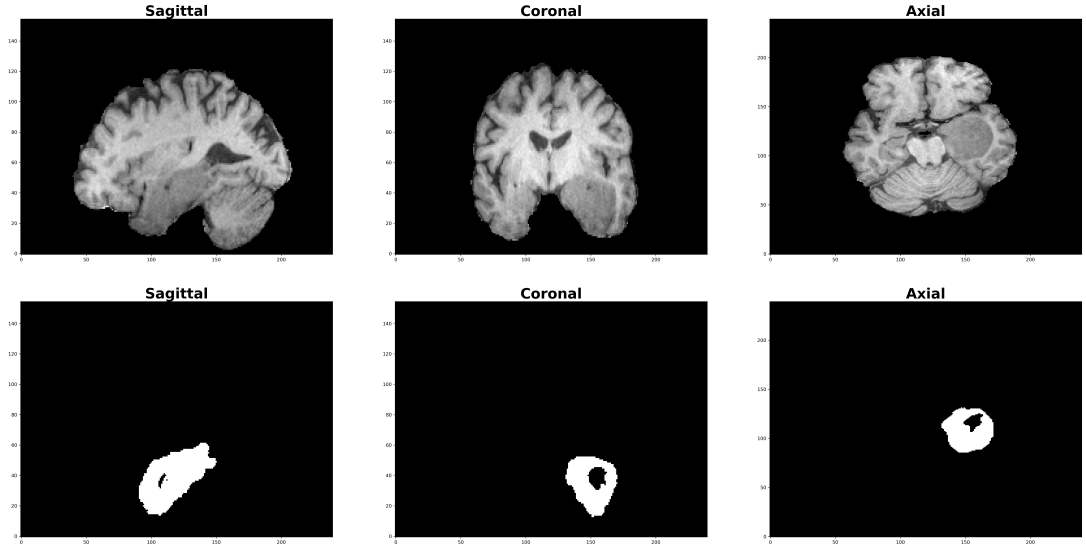

Figure 2.4: T1ce original MRI image with the segmented form of the tumor in the best-projected views - Patient's ID: 00014

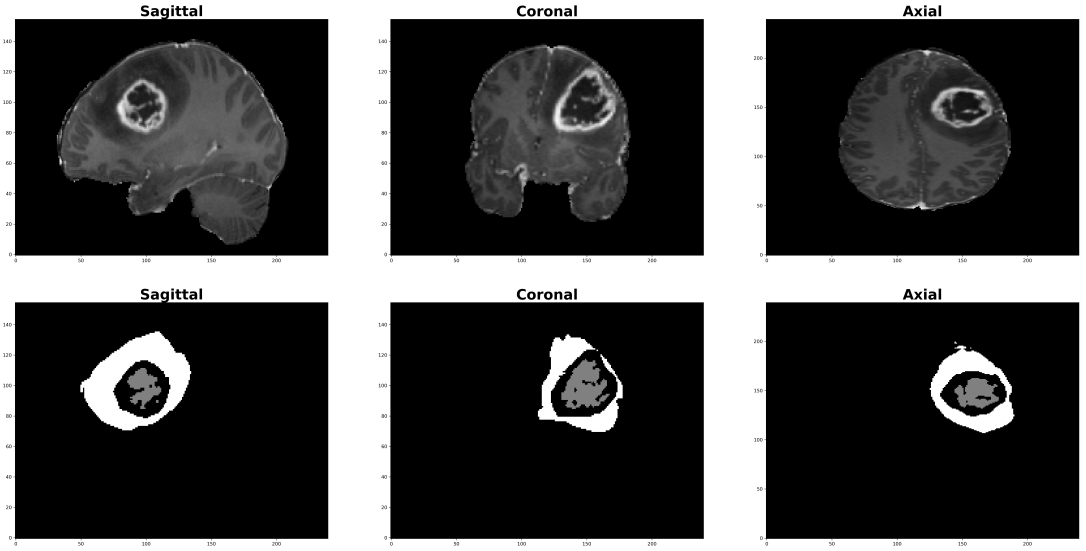

Figure 2.5: T1ce original MRI image with the segmented form of the tumor in the best-projected views - Patient's ID: 00070

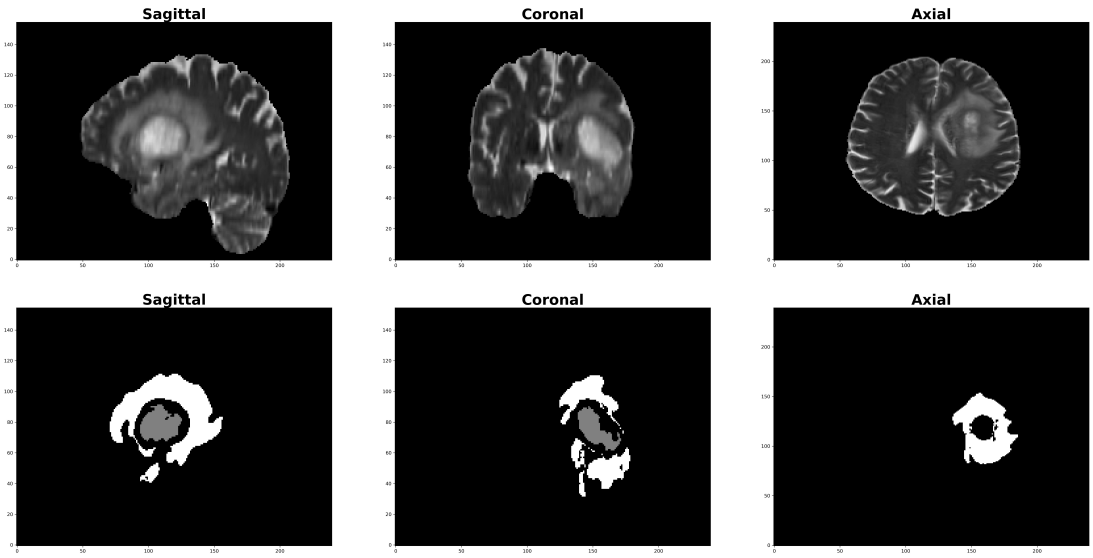

Figure 2.6: T2 original MRI image with the segmented form of the tumor in the best-projected views - Patient's ID: 00267

2.3 Feature Extraction

Increasing evidence suggests the strong impact of intra-tumor and inter-tumor heterogeneity on cancer treatment response. Radiomics aims to capture this heterogeneity by quantifying the tumor phenotype using high throughput mathematical algorithms applied to medical imaging data. These algorithms are either defined using engineered features requiring expert domain knowledge or deep learning methods that can automatically learn the feature representations from the data.

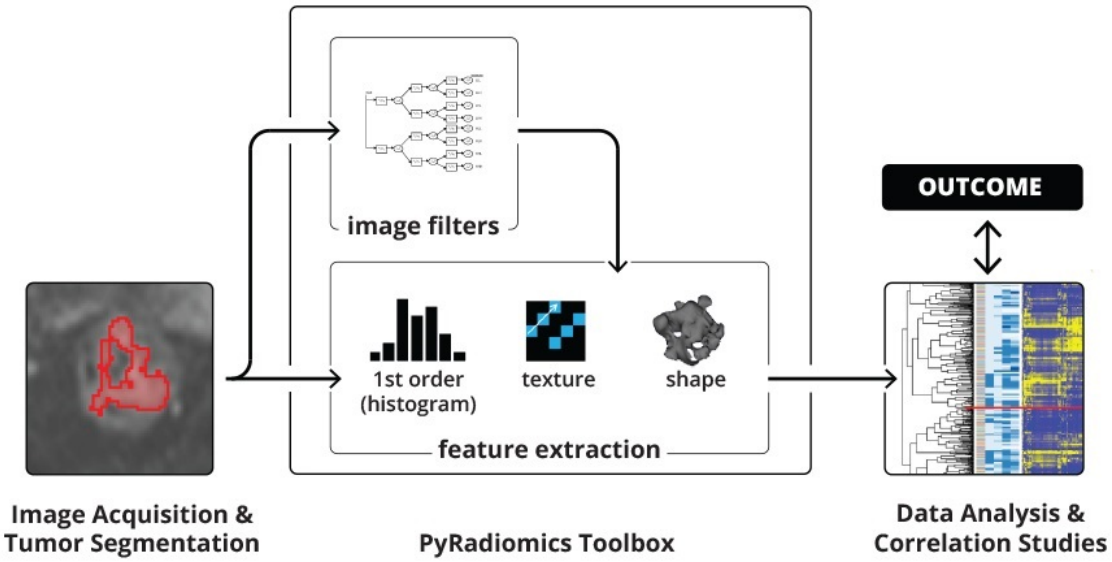

Figure 2.7: Overview of the pyradiomics process [21]

The `pyradiomics` package in python provides a pipeline to extract radiomic features from medical imaging data. It contains features for first-order [20] statistics as well as descriptors of the texture and shape of the region of interest. These features can either be extracted from the image directly or from images derived from the original image using a choice of available filters.

One of the filters used in this study is wavelet transformation which can provide comprehensive spatial and frequency distributions for characterizing intratumoral and peritumoral regions in terms of low and high-frequency signals. These properties may improve the performance of the radiomic model in term of compression, noise reduction and detection [23, 24]. Another filter is Laplacian of Gaussian (LoG) which is associated with image noise reduction and can improve the utility of the heterogeneity measures [25].

Once all features are extracted, results are combined and returned as a CSV file

which is used for prospective data processing.

2.4 Dimensionality Reduction

A dimensionality reduction technique is needed to transform the dataset from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original dataset. The variance threshold is a feature selector that removes all the low variance features from the dataset without significant modeling benefits. In the first step, the outcome of the feature extraction stage would be cleaned up by keeping only numerical features and applying a low variance threshold to reduce the number of features to 841. Since the number of features is still too high, the XGBoost classifier as a dimensionality reduction technique has been used, which will reduce the complexity of the model and remove some of the data noise. It also assists in reducing overfitting in this approach.

As part of its ensemble learning, the XGBoost classifier employs regularized learning and cache-aware block structure tree learning. L represents the loss function; f_t represents the tree and $t-th$ tree and $\Omega(f_t)$ is regularized term. The second-order Taylor series of L at the $t-th$ iteration is:

$$L^{(t)} \simeq \sum_{i=1}^{k} \left[l\left(y_i, y_i^{(t-1)}\right) + g_i f_t\left(x_i\right) + \frac{1}{2} h_i f_t^2(x) \right] + \Omega\left(f_t\right) \tag{2.1}$$

Gain is utilized to choose the best split node during XGBoost training where g_i and h_i stand for the first and second order gradients.

$$\text{gain} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda} \right] - \gamma \tag{2.2}$$

$$I = I_L \cup I_R \tag{2.3}$$

Where I_L and I_R, respectively, represent samples of the left and right nodes after segmentation and γ and I are a penalty parameters. Gain is the score assigned to each split of a tree, and the average gain is used to determine the final feature importance score. The average gain equals the sum of all tree gains divided by the sum of feature splits. The matching feature is considered to be more significant and impactful the higher the feature significance score of XGBoost is. In order to describe MGMT status, the top-ranked features are acquired based on feature relevance.

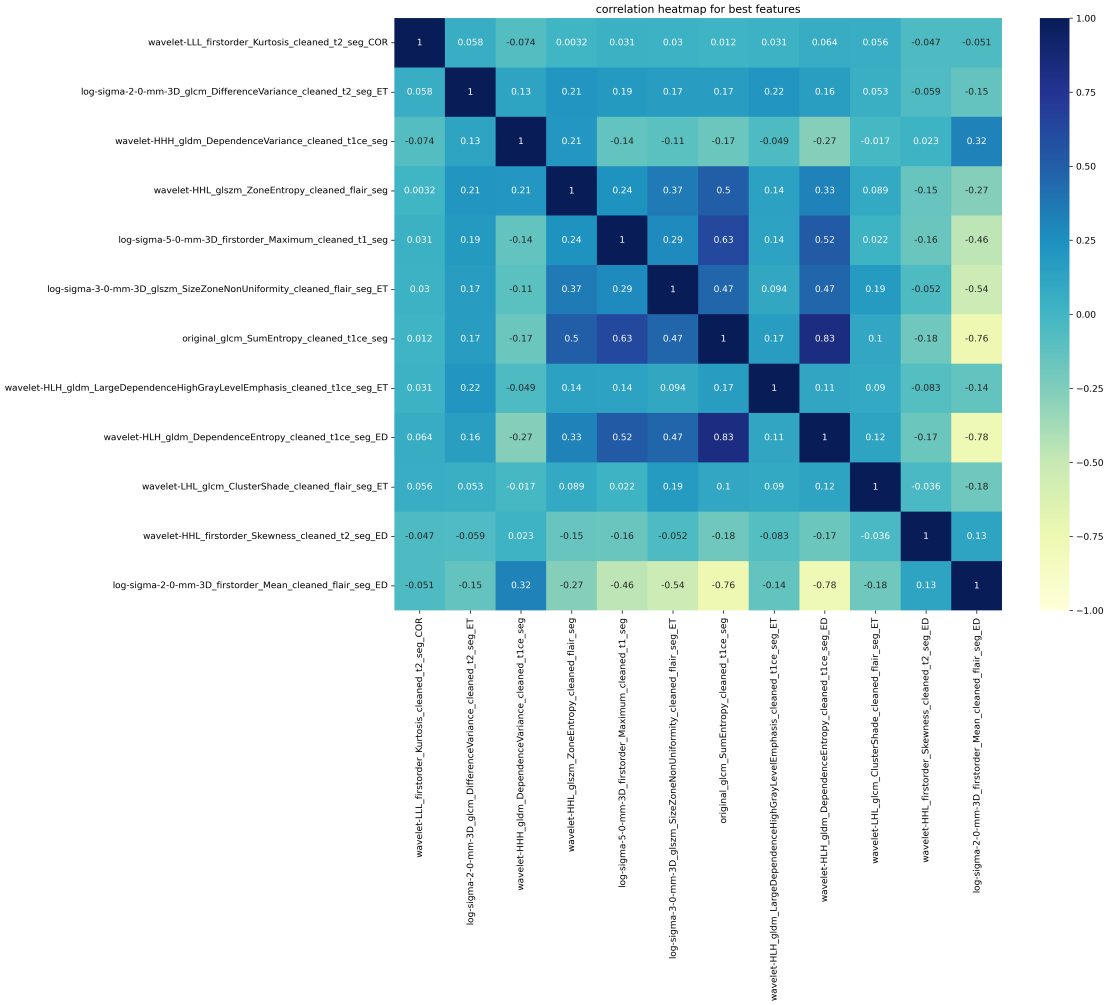

Figure 2.8: Correlation heatmap of top 12 ranked features of all mixed datasets used for the model

Fig. 2.8 shows correlation heatmap for the top 12 selected features in a mixed dataset containing all the MRI sequence types. These features will be used in binary classification to build the predictive model.

2.5 Classification

In this thesis, nested CV is considered as a splitting technique for evaluating the performance of ML algorithms. Nested CV is often used to train a model that needs its hyperparameters to be tuned. The generalization error of the underlying model and its hyperparameters search is estimated via nested CV. By selecting parameters that maximize non-nested CV, the model is biased toward the dataset and produces an excessively positive result.

The same data are used to adjust model parameters and assess model performance in model selection without a nested CV. As a result, the model may overfit the data, and information may leak into it. The size of the dataset and the model's stability has the most effects on the amount of this impact.

Nesting CV successfully uses a succession of train/validation/test set splits to get around this issue. The score is roughly maximized in the inner loop by fitting a model to each training set before being directly maximized by choosing hyperparameters across the validation set. The test set scores from various dataset splits are averaged in the outer loop to assess generalization error.
As it is shown in fig. 2.9, the whole dataset has been split in a way that the training part has 90% and the test 10%. The training part will go through a nested CV, in which the outer loop has 10 folds, and for the inner loop, 5 folds have been considered. The outer loop train the model with the best hyperparameters obtained with the inner loop and test on the held-back-test dataset. All average statistical measurements are reported separately for each classification approach through the outer loop iteration. In the following, a summary of applied classification methods has been reviewed.

2.5.1 Logistic Regression

Logistic Regression is a statistical learning technique in the category of supervised Machine Learning (ML) for classification problems. It is frequently used to calculate the likelihood that a certain instance belongs to a certain class. If the estimated probability exceeds 50%, the model predicts that the instance belongs to the positive class labeled "1". Otherwise, it predicts it belongs to the negative class labeled "0". This makes it a binary classifier. The Logistic Regression model computes a

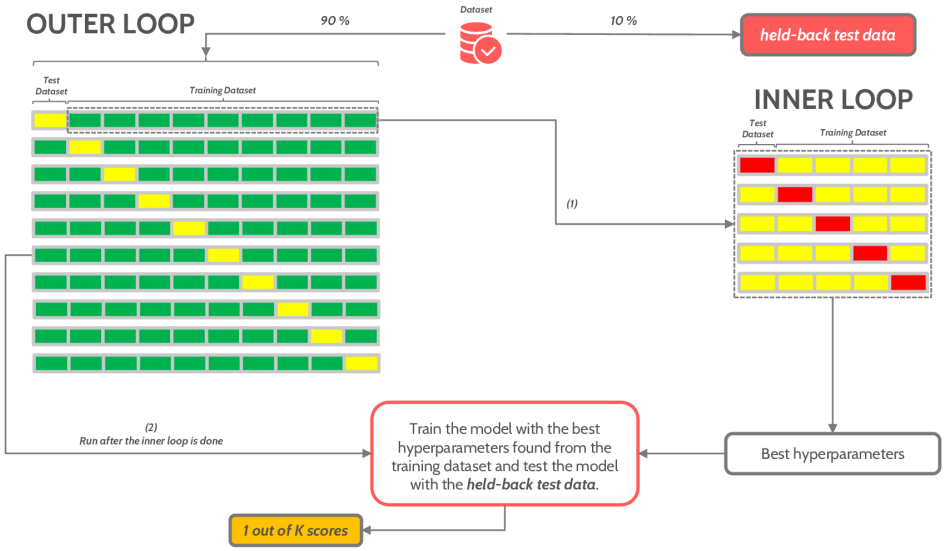

Figure 2.9: Process of nested CV

weighted sum of the input features (plus a bias term), and the output is equation 2.4:

$$\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\mathbf{x}^\top \boldsymbol{\theta}\right) \tag{2.4}$$

The logistic is a sigmoid function which is like an S-shaped that outputs a number between 0 and 1. It is defined as shown in equation 2.5 and fig. 2.10.

$$\sigma(t) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{2.5}$$

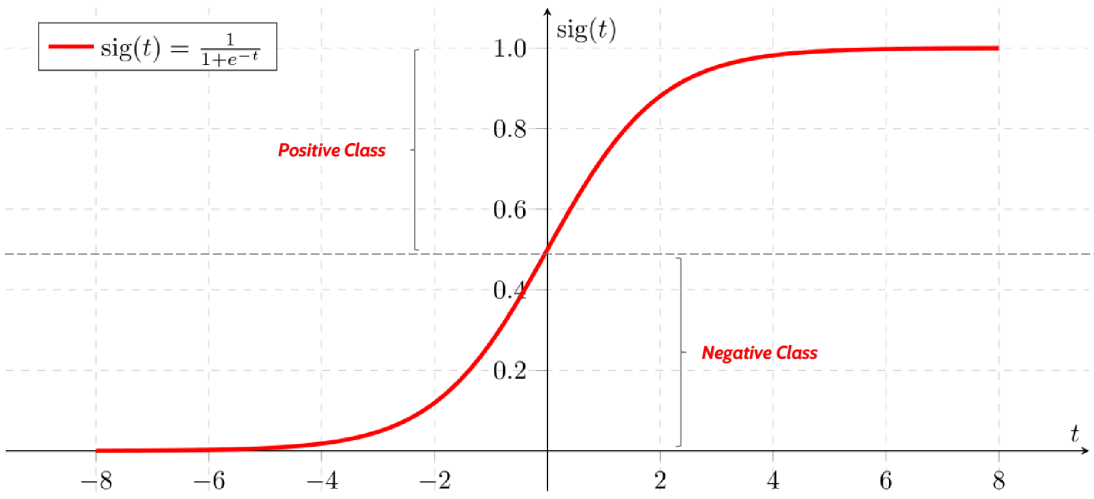

Figure 2.10: Logistic Sigmoid activation function

The Logistic Regression model could easily predict once it has calculated the probability $\hat{p} = h_\theta(x)$, which, an instance x, belongs to the positive class.

$$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5 \\ 1 \text{ if } \hat{p} \geq 0.5 \end{cases} \tag{2.6}$$

Equation 2.6 that $\sigma(t) < 0.5$ when $t < 0$, and $\sigma(t) \geq 0.5$ when $t \geq 0$, so a Logistic Regression model predicts 1 if $x^T\theta$ is positive and 0 if it is negative.
The `sklearn.linear_model.LogisticRegression` from sikit-learn package in Python has been used for creating LR model. The logistic model is penalized for having too many variables. As a regularization, this brings the coefficients of the less significant variables closer to zero. In this study, for specifying the norm of

penalty, four parameters have been considered:

```
"penalty":  ["none", "l1", "l2", "elasticnet"]
```

- "none": no penalty is added.
- "l1": penalizes the sum of absolute values of the weights.
- "l2": regularization penalizes the sum of squares of the weights.
- "elasticnet": penalized linear regression model that includes both the L1 and L2 penalties during training.

The second parameter is "C," which stands for the inverse regularization parameter. This control variable is positioned in the opposite direction from the Lambda regulator and maintains the regularization's strength modification.

$$C = \frac{1}{\lambda} \tag{2.7}$$

The relationship would be that lowering C would strengthen the Lambda regulator. The values considered for it are:

```
"C":  [0.001, 0.009, 0.01, 0.09, 1, 5, 10, 25, 50, 75, 100]
```

The third parameter is "solver," which allows for different gradient descent algorithms to set the β_i [26] in equation 2.5. Three solvers method has been taken into account:

- "newton-cg": Newton's technique that essentially employs an improved quadratic function minimization.
- "lbfgs": Limited-memory BFGS approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) using a limited amount of computer memory.
- "liblinear": It is Library for Large Linear Classification, which uses a coordinate descent algorithm that is based on minimizing a multivariate function by solving univariate optimization problems in a loop.

2.5.2 Support Vector Machine

A separating hyperplane is the formal definition of a Support Vector Machine (SVM), as a discriminative classifier. In other words, the method produces an optimum hyperplane that classifies fresh samples when given labeled training data (supervised learning). This hyperplane divides a plane into two halves in two-dimensional space, with one class lying on either side of the line.

Equation 2.8 is of a hyperplane where w is a vector normal to the hyperplane and b is an offset.

$$w.x + b = 0 \tag{2.8}$$

Defining a decision rule to classify a point as negative or positive is an essential step.

$$y = \begin{cases} +1 \text{ if } \vec{X} \cdot \vec{w} + \mathrm{b} \geq 0 \\ -1 \quad \text{if } \vec{X} \cdot \vec{w} + \mathrm{b} < 0 \end{cases} \tag{2.9}$$

To calculate 'd', as fig. 2.11 shows, the equation of L1 and L2 is needed. The assumption is that the equation of L1 is $w.x + b = 1$, and for L2, it is $w.x + b = -1$.

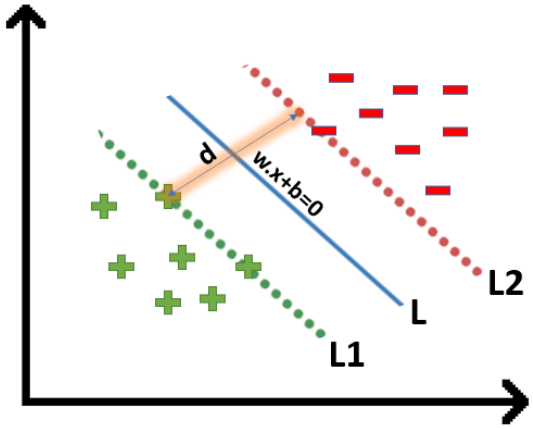

Figure 2.11: Classification of data by (SVM)

The `sklearn.svm.SVC` package from sikit-learn in Python is used for SVM classification. Parameters considered for the classification are "C," the same as the LR parameters, "gamma," and "kernel." "gamma" defines how far the influence of a

single training example reaches, and the values for it are:

```
"gamma":  [1, 0.1, 0.01, 0.001, 0.0001]
```

The "kernel" parameters are used as a tuning parameter to improve the classification accuracy which are:

```
"kernel":  ["rbf", "poly", "sigmoid"]
```

- "rbf": Radial Basis Function (RBF), has been utilized with great success in SVMs to differentiate across classes.

- "poly": To enable the learning of non-linear models, it denotes the similarity of vectors (training samples) in a feature space over polynomials of the original variables.

- "sigmoid": This function may be used to activate artificial neurons and is equal to a two-layer perceptron neural network model.

2.5.3 Multi-layer Perceptron

The feed forward neural network is supplemented by the multi-layer perceptron (MLP). As seen in fig. 2.12, it has three different kinds of layers: an input layer, an output layer, and a hidden layer. The input layer is where the input signal for processing is received. The output layer completes the necessary task, such as classification and prediction. The real computational engine of the MLP consists of an arbitrary number of hidden layers that are placed between the input and output layers. Data travels from the input to the output layer of an MLP in the forward direction, much like a feed forward network. With the help of the back propagation learning method, the MLP's neurons are trained. MLP may resolve issues that are not linearly separable since they are made to approximate any continuous function.

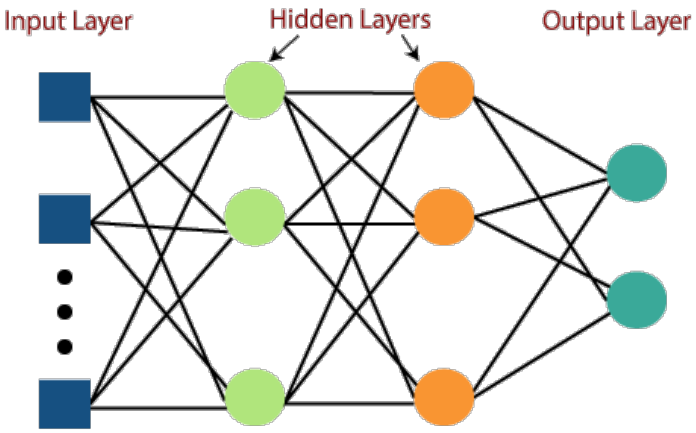

Figure 2.12: Multi-layer Perceptron (MLP) [27]

The parameters for MLP classifier are "batch_size", "momentum," "learning_rate_init,"¶ "solver", same as LR and SVM, "alpha," and "learnin_rate."

- "batch_size": The number of samples processed before the model is updated is 256.
- "momentum": The variant of the stochastic gradient descent. It replaces the gradient with a momentum which is an aggregate of gradients. 0.9 and 0.99 are considered for its values.
- "learning_rate_init": As initial learning rate, it controls the step size in updating the weights, which 0.001, 0.01, and 0.1 are considered for it.
- "alpha":
- "learnin_rate": It controls how quickly the model is adapted to the problem. `["constant", "adaptive"]` are the hyper-parameters. 'constant' is a constant learning rate given by 'learning_rate_init'. 'adaptive' keeps the learning rate constant to 'learning_rate_init' as long as training loss keeps decreasing [28].

Chapter 3

Results

3.1 Performance Evaluation

Reporting the final results for each classification method needs statistical tools to demonstrate the details in each scenario. For this thesis, in contrast to accuracy, the confusion matrix, precision, recall, and F1 score provide a deeper understanding of the results of the prediction. By getting the confusion matrix, other evaluation metrics could be obtained.

		Predicted	
	Total population	Positive (P)	Negative (N)
True	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Table 3.1: A 2x2 confusion matrix structure

Table 3.1 shows a 2x2 confusion matrix structure that consists of:

- **True Positive (TP)**: model correctly predicts the positive class
- **True Negative (TN)**: model correctly predicts the negative class
- **False Positive (FP)**: model gives the wrong prediction of the negative class
- **False Negative (FN)**: model wrongly predicts the positive class

Precision and recall are both essential for information retrieval, with positive class having a greater impact than negative. The goal of precision is to determine what proportion of all anticipated positives is actually positive. The precision value ranges from 0 to 1.

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

The goal of recall is to determine what proportion of overall positives are predicted positives.

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

The harmonic mean of recall and precision is the F1 score. It accounts for both false positives and false negatives. As a result, it works well with an unbalanced dataset. F1 score gives the same weigh to recall and precision.

$$F1score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{(Precision + Recall)} \tag{3.3}$$

The final results for each classification methods are shown in tables 3.2, 3.3 and 3.4. Each table's first column shows the ideal combination of features that results in a higher F1 score with the average F1 score in each outer loop of the nested CV. The second column in the tables shows the average Standard Deviation (std) of the F1 score in each outer loop of nested CV.

	num. of features / F1-score for LR	std for LR
All mixed dataset	12 / 0.57	0.018
FLAIR	10 / 0.55	0.01
T1	19 / 0.59	0.2
T1ce	3 / 0.57	0.009
T2	6 / 0.55	0.046

Table 3.2: Final results for LR classification

As table 3.2 demonstrates, the highest F1 score belongs to the T1 sequence type with 0.59 for the LR method. The optimal number of features to reach the result is 19. Although the T1ce type has the F1 score of 0.57, the optimal number of features is the least among all the other types with only 3 features, which also is the more stable result with an std of 0.009.

	num. of features / F1-score for SVM	std for SVM
All mixed dataset	10 / 0.6	0.021
FLAIR	16 / 0.58	0.048
T1	16 / 0.68	0.07
T1ce	5 / 0.6	0.067
T2	5 / 0.59	0.063

Table 3.3: Final results for SVM classification

The T1ce type has the best F1 score with 0.68 in SVM outcome according to table 3.3. The lowest number of the optimal selected features is for both T1ce and T2 types, which is 5. The most stable result, considering the std, is for the FLAIR type, which is equal to 0.58 F1 score.

	num. of features / F1-score for MLP	std for MLP
All mixed dataset	12 / 0.59	0.035
FLAIR	14 / 0.6	0.055
T1	16 / 0.59	0.09
T1ce	4 / 0.55	0.011
T2	6 / 0.62	0.066

Table 3.4: Final results for MLP classification

The outcome for the MLP approach shows the best result considering the F1 score for the T1 type, but the std for it is the highest with respect to others, which is equal to 0.09. The lowest std and number of optimal features needed to reach the better result belong to the T1ce type, which are 4 features and 0.011.
The following will report the confusion matrix for each MRI sequence type.

Table 3.5 shows the list of top-ranked features used for the SVM classifier and has the best outcome for T1ce. This is a sample for presenting the results to the radiologists as final users.

Rank	Feature Class	Feature Name	Image Type
1	Gray Level Dependence Matrix (GLDM)	Dependence Variance (DV)	Wavelet - Tumor Edematous
2	First Order Features	Mean	Wavelet - Enhancing Tumor
3	First Order Features	Entropy	LoG - Tumor Edematous
4	First Order Features	90th percentile	LoG - Whole Tumor's Core
5	Gray Level Run Length Matrix (GLRLM)	Long Run Emphasis (LRE)	Wavelet - Enhancing Tumor
6	Gray Level Dependence Matrix (GLDM)	Dependence Non-Uniformity (DN)	LoG - Enhancing Tumor
7	First Order Features	Median	LoG - The Whole Image
8	First Order Features	Range	LoG - Tumor Edematous
9	First Order Features	Mean	Wavelet - Tumor Edematous
10	First Order Features	Mean	LoG - Tumor Edematous
11	First Order Features	Median	Wavelet - Whole Tumor's Core
12	Shape Features (3D)	Maximum 3D diameter	Original - Enhancing Tumor
13	Gray Level Dependence Matrix (GLDM)	Large Dependence High Gray Level Emphasis (LDHGLE)	Wavelet - Tumor Edematous
14	Gray Level Co-occurrence Matrix (GLCM) Features	Contrast	Wavelet - The Whole Image
15	First Order Features	Kurtosis	Wavelet - Tumor Edematous
16	First Order Features	Mean	LoG - Tumor Edematous

Table 3.5: List of top-ranked features for the best outcome

3.1.1 FLAIR

The confusion matrix related to each binary classification approach's result are shown in the figs. 3.1 to 3.3.

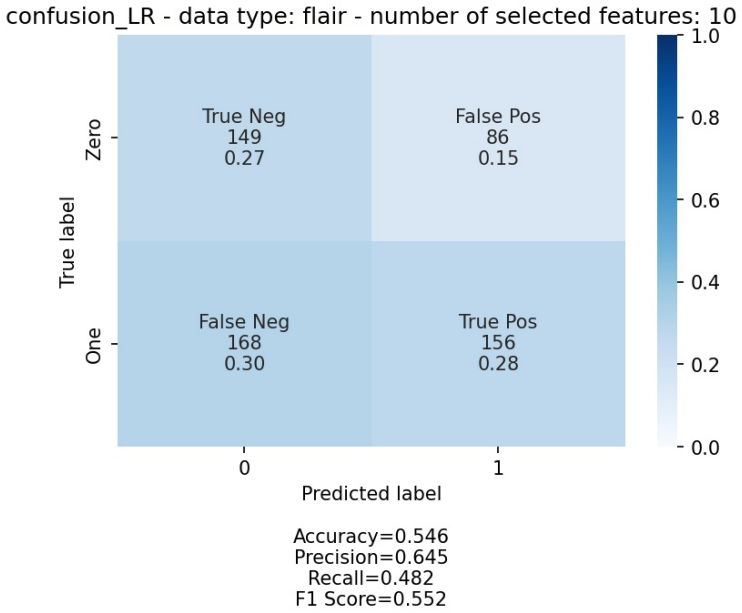

Figure 3.1: FLAIR - Confusion Matrix for LR with 10 selected features

The first number for each cell of the confusion matrix is the number of patients classified as TP, TN, FP, or FN.
The best selected hyperparameters for LR are:
`['C': 0.09, 'penalty': 'l2', 'solver': 'liblinear']`

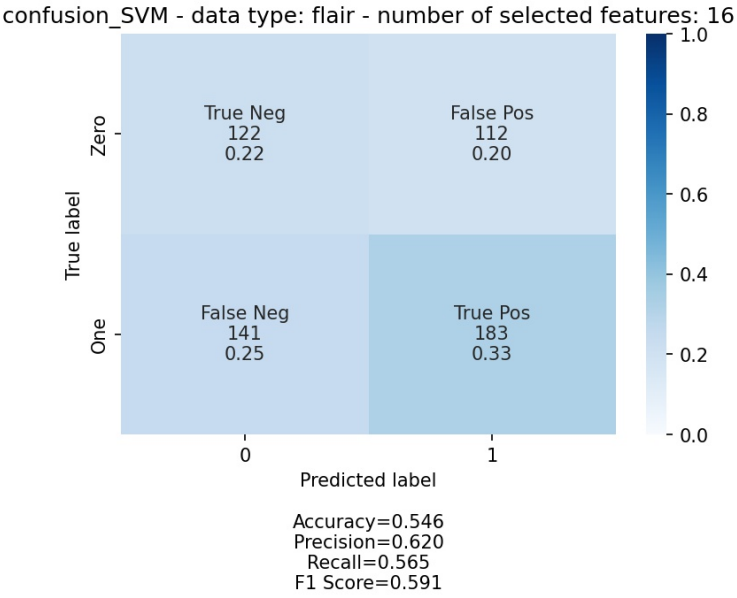

Figure 3.2: FLAIR - Confusion Matrix for SVM with 16 selected features

The best selected hyperparameters for SVM are:
`['C': 10, 'gamma': 0.1, 'kernel': 'rbf']`

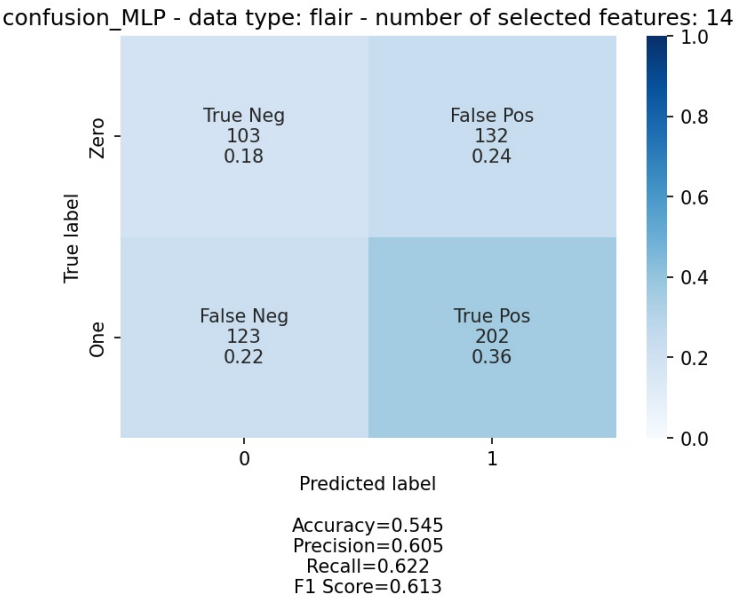

Figure 3.3: FLAIR - Confusion Matrix for MLP with 14 selected features

The best selected hyperparameters for MLP are:
`['alpha': 0.0001, 'batch_size': 256, 'learning_rate': 'constant',| 'learning_rate_init': 0.1, 'momentum': 0.9, 'solver': 'adam']`

3.1.2 T1

The confusion matrix related to each binary classification approach's result are shown in the figs. 3.4 to 3.6.

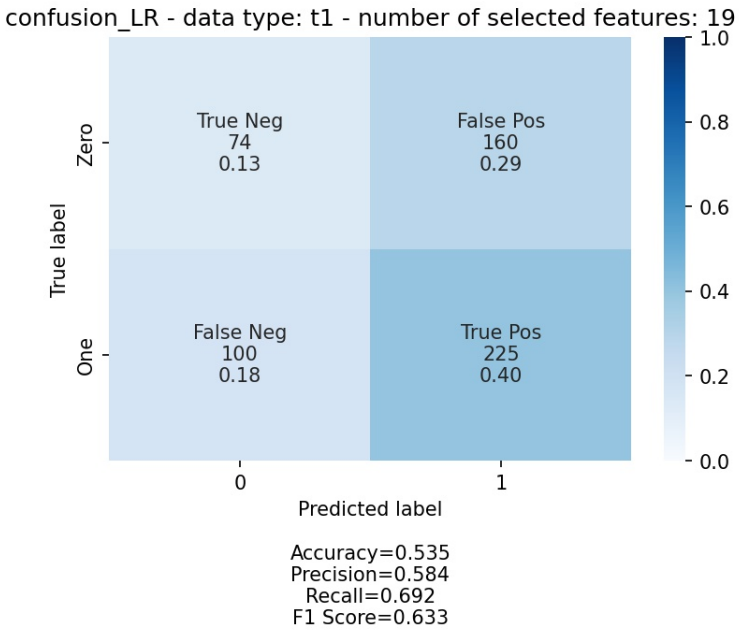

Figure 3.4: T1 - Confusion Matrix for LR with 19 selected features

The best selected hyperparameters for LR are:
`['C': 1, 'penalty': 'l1', 'solver': 'liblinear']`

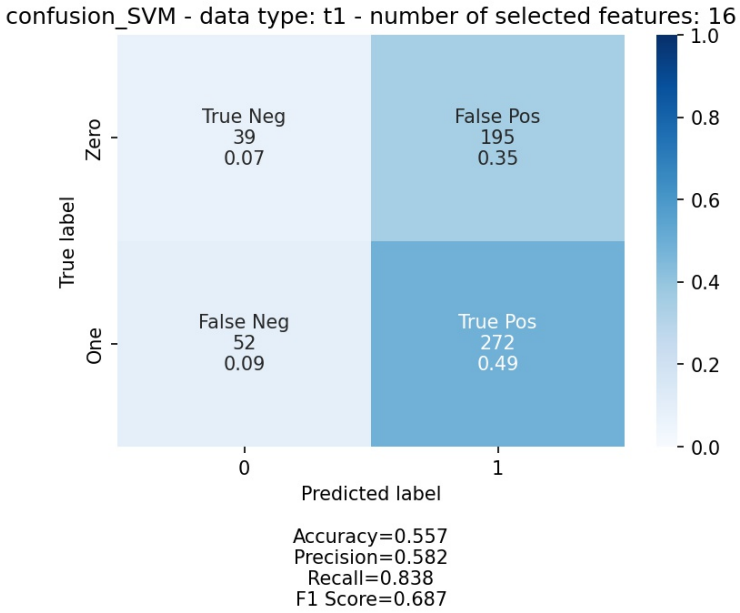

Figure 3.5: T1 - Confusion Matrix for SVM with 16 selected features

The best selected hyperparameters for SVM are:
`['C': 10, 'gamma': 1, 'kernel': 'poly']`

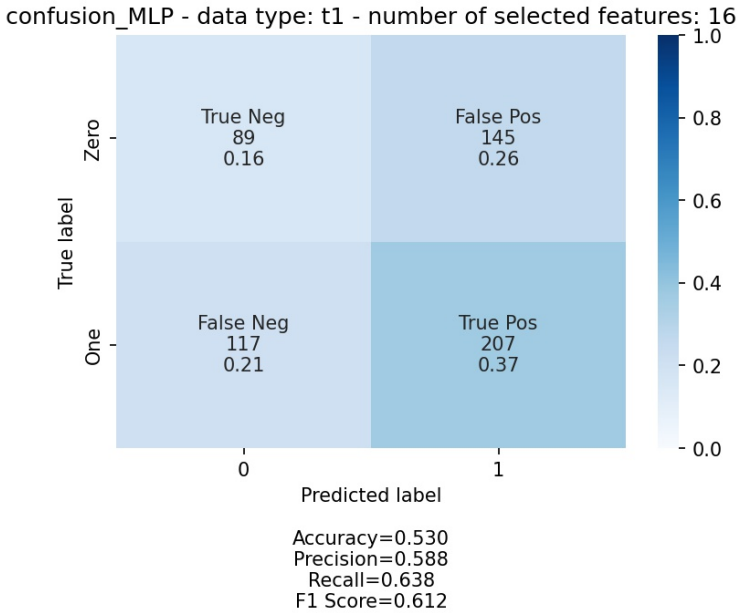

Figure 3.6: T1 - Confusion Matrix for MLP with 16 selected features

The best selected hyperparameters for MLP are:
['alpha': 0.0001, 'batch_size': 256, 'learning_rate': 'constant',
'learning_rate_init': 0.01, 'momentum': 0.9, 'solver': 'adam']

3.1.3 T1ce

The confusion matrix related to each binary classification approach's result are shown in fig. 3.7, fig. 3.8, and fig. 3.9.

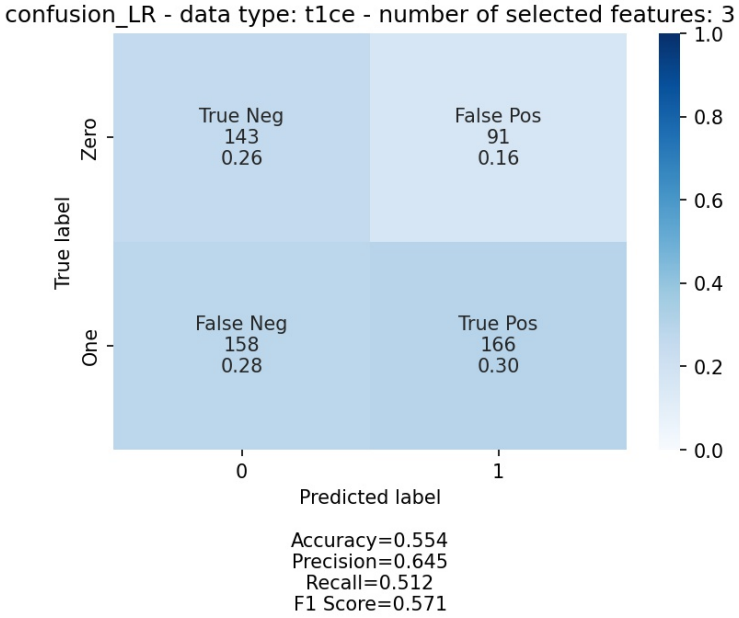

Figure 3.7: T1ce - Confusion Matrix for LR with 3 selected features

The best selected hyperparameters for LR are:
['C': 1, 'penalty': 'l2', 'solver': 'newton-cg']

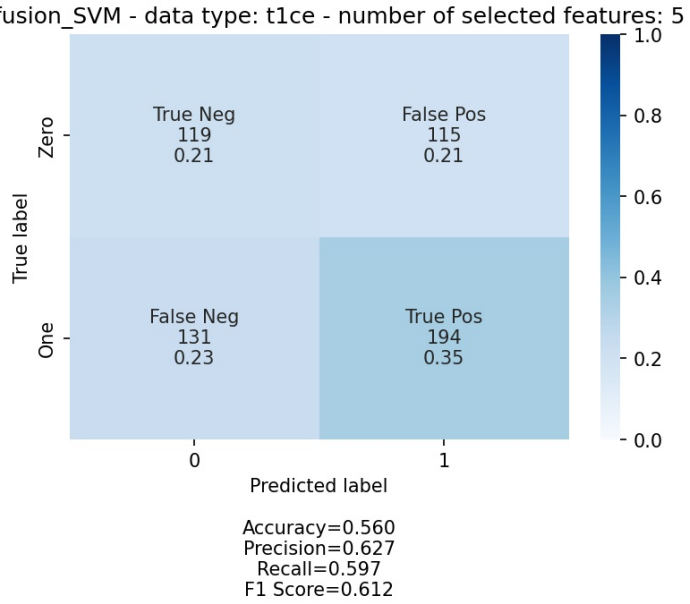

Figure 3.8: T1ce - Confusion Matrix for SVM with 5 selected features

The best selected hyperparameters for SVM are:
`['C': 10, 'gamma': 1, 'solver': 'rbf']`

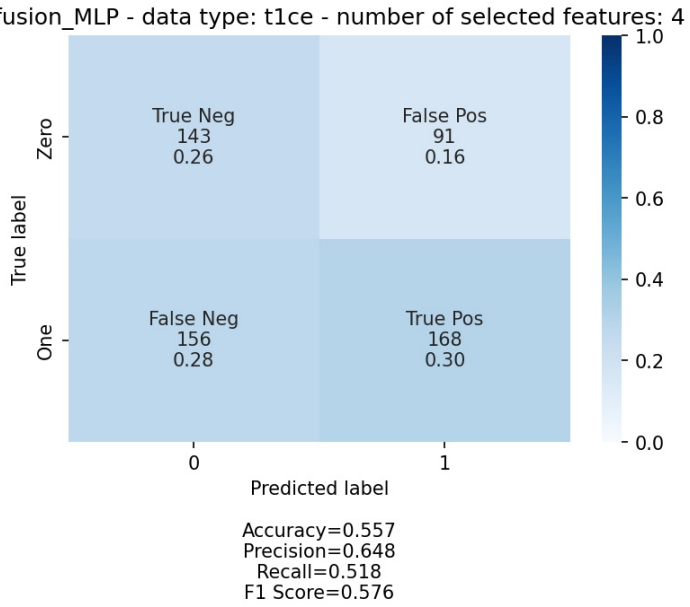

Figure 3.9: T1ce - Confusion Matrix for MLP with 4 selected features

The best selected hyperparameters for MLP are:
`['alpha': 0.05, 'batch_size': 256, 'learning_rate': 'constant', 'learning_rate_init': 0.01, 'momentum': 0.9, 'solver': 'adam']`

3.1.4 T2

The confusion matrix related to each binary classification approach's result are shown in fig. 3.10, fig. 3.11, and fig. 3.12.

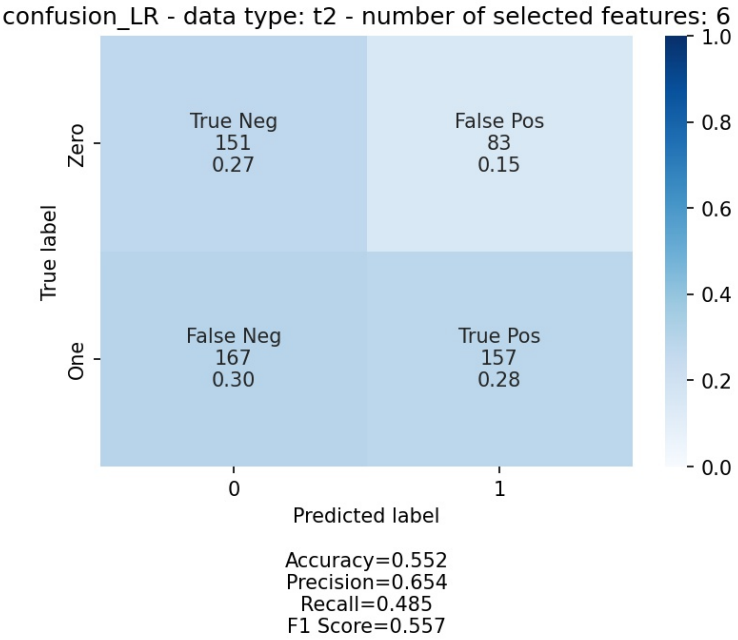

Figure 3.10: T2 - Confusion Matrix for LR with 6 selected features

The best selected hyperparameters for LR are:
`['C': 1, 'penalty': 'l2', 'solver': 'liblinear']`

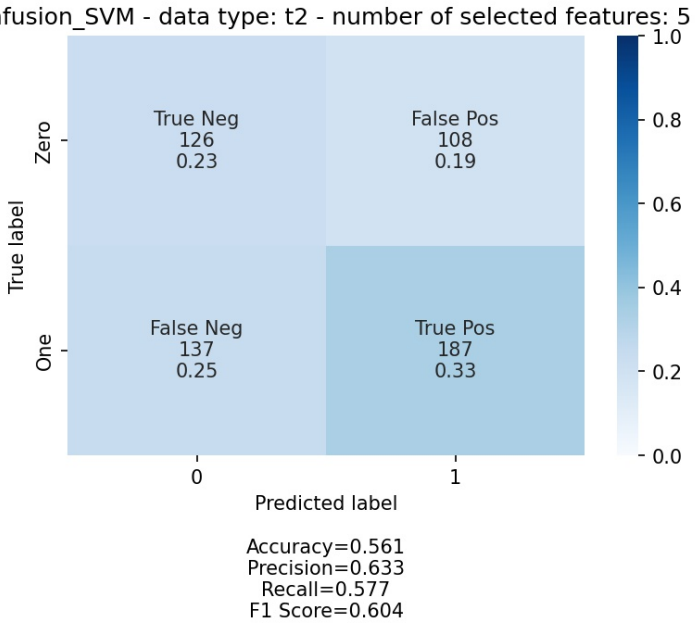

Figure 3.11: T2 - Confusion Matrix for SVM with 5 selected features

The best selected hyperparameters for SVM are:
`['C': 100, 'gamma': 1, 'solver': 'rbf']`

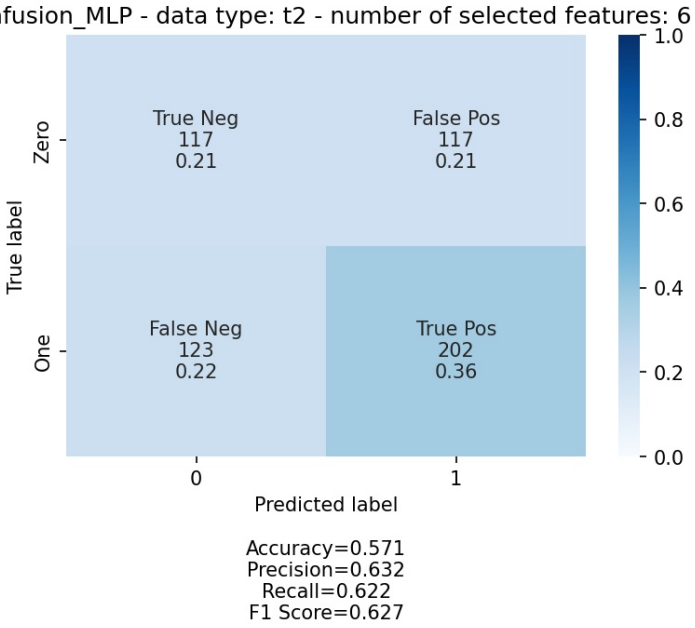

Figure 3.12: T2 - Confusion Matrix for MLP with 6 selected features

The best selected hyperparameters for MLP are:
`['alpha': 0.0001, 'batch_size': 256, 'learning_rate': 'constant',| 'learning_rate_init': 0.001, 'momentum': 0.9, 'solver': 'adam']`

3.2 Comparison

There are some points to b considered as a comparison between the different type's results:

- Considering the average statistical measurement for each MRI sequence type, T1 has shown to be more informative for predicting MGMT value.
- The T1ce and T2 types need fewer number of features to get their best results compare to FLAIR and T1 types.
- Although T1ce does not have the best result, it has the most stable outcomes by having the lowest std among all types.
- In average for all the types, the SVM classifier's results are superior with respect to other methods.

Chapter 4

Conclusion and Future Works

4.1 Summary and Conclusion

This study demonstrates the application of radiomics-based machine learning models to predict MGMT promoter methylation status in mpMRI scans. By extracting features through `pyradiomics` package, using Wavelet and LoG filters, the whole output metadata was analyzed and prepared for applying ML binary classification method. The data preparation stage focused on feature selection techniques in order to reduce the dimension by the XGBoost classifier. Three classification approaches were considered for the prediction: Logistic Regression, Support Vector Machine, and Multi-layer Perceptron.

Different evaluation metrics have been taken into account for analyzing the results, mainly focused on the confusion matrix, precision, recall, and F1 score, which deliver additional insights into the prediction than accuracy performance metrics. The proposed solution could reach 68.7% of the F1 score with the Support Vector Machine method for the T1ce MRI sequence type, which also has more acceptable outcomes in Deep Learning classification methods among all the MRI sequence types [29].

Using a 2-D Convolutional Neural Network (CNN) for the same dataset provided by the Radiological Society of North America (RSNA) results in better outcomes (74.8%) comparing to the ML-based radiomics approach [31]. The ML classification model, however, is a promising technique for identifying brain cancers when taking into account the data at hand. In the future, results might be more reliable by using a larger dataset with well-balanced and high-quality data to improve performance.

4.2 Future Works

To compare the outcomes of this work and enhance the performance of the model, it could be useful to take into account larger datasets. The effectiveness of a machine-learning model is closely correlated with the quality, and quantity of the data, making it a crucial component of any AI or ML application. This effect is more noticeable in radiology research related to a computer-aided approach. As a study shows [32], 1016 diffuse glioma patients were collected retrospectively from Beijing Tiantan Hospital in China. Deep convolutional neural networks (DCNN)-based predictive models and radiomics-based predictive models were created, respectively, and their effectiveness was evaluated. In conclusion, both the radiomics and DCNN models could preoperatively predict the molecular subtypes of diffuse glioma. This emphasizes the data quality since different medical protocols produce MRI images, and various scanners are used to obtain them.

Dealing with high dimensional data, which refers to the number of attributes or features a data collection may contain, is one of the major challenges in machine learning. Besides the XGBosst classifier, this study could process various feature selection algorithms, such as Recursive Feature Elimination, Cross-Validated (RFECV), and compare the results with the current ones.

Moreover, the segmented tumor images used in this study could be obtained directly from the task 2 dataset to guarantee the consistency and quality of the images, which leads to a more accurate outcome in the feature extraction stage.

Appendix A

Technicalities

Pre-processing class is used for different purposes:

- **Converesion** which is used to change the DCIOM datatype to NIFTI.
- **Best View** which is used to get the most informative view of the pictures.
- **Visualization** which is used to represent the data in JPG or PNG.
- **Image Info** which is used to extract the volume and size of images.

```
"""
libraries : data pre-processing
"""
import os
import dicom2nifti
import pandas as pd
import matplotlib.pyplot as plt
import nibabel as nib
import numpy as np

""" train_df : patients lables
    patientID : list of patiend's ID in input
  folder
    train_path : DICOM format path
    train_path_nifti : NIFTI format path -->
  output for conversion function
```

```
15      visualization_path : JPG format path -->
    output for visualization function
16      sequence_types : sequence type of mMRI
    pictures as a list of string
17  """
18
19
20  class PreProcess:
21      def __init__(
22          self,
23          train_df,
24          patientID,
25          train_path,
26          train_path_nifti,
27          visualization_path,
28          sequence_types,
29      ):
30
31          self.train_df = pd.read_csv(
32              "...[path to patient labels]"
33          )
34          self.patientID = os.listdir(
35              "... [list of patients's ID]"
36          )
37          # self.train_path = (
38          #     "...[path to raw dataset DICOM
    format]"
39          # )
40          # self.visualization_path = "... [path
    to save the visualized images]"
41          # self.train_path_nifti = "... [path to
     convertd dataset to NIFTI format]"
42          self.sequence_types = ["FLAIR", "T1w",
    "T1wCE", "T2w"]
```

```
43
44      """<<conversion>> is iterating over patient
        's ID and the sequence type (inner iteration
        )
45      to convert each picture into NIFTI format
        and save it in "converted_path"
        """
46
47
48      def conversion(self, original_path,
        converted_path):
49
50          for patient in self.patientID:
51              try:
52                  for types in self.
                    sequence_types:
53                      os.makedirs(
54                          converted_path +
                            patient + "/" + types + "/", exist_ok=True
55                      )
56                      dicom2nifti.
                        dicom_series_to_nifti(
57                          original_path + patient
                             + "/" + types,
58                          os.path.join(
59                              converted_path,
60                              patient + "/" +
                                types + "/" + patient + types,
61                          ),
62                      )
63              except:
64                  continue
65
```

```
66      """<<visualization>> is iterating over
    patient's ID and the sequence type (inner
    iteration)
67      to convert each picture into JPG format and
     save it in "jpg_path" for visualization
68      containing : MGMT_value status for each one
69      """
70
71      def visualization(self, nifti_path,
    jpg_path):
72
73          for patient in self.patientID:
74              try:
75                  for types in self.
    sequence_types:
76                      img = nib.load(
77                          os.path.join(
78                              nifti_path,
79                              patient + "/" +
    types + "/" + patient + types + ".nii",
80                          )
81                      )
82                      img_data = img.get_fdata()
83
84                      # extracting different
    slice of the brain for three dircetion views
85                      slice_0 = img_data[img.
    shape[0] // 2, :, :]
86                      slice_1 = img_data[:, img.
    shape[1] // 2, :]
87                      slice_2 = img_data[:, :,
    img.shape[2] // 2]
88
89                      # plotting 3 views
```

```
90                      fig, axes = plt.subplots(1,
    3)
91                      fig.set_figheight(17)
92                      fig.set_figwidth(40)
93                      axes[0].imshow(
94                          slice_0.T, cmap="gray",
    origin="lower", aspect="auto"
95                      )
96                      axes[0].set_title("Sagittal
    ", fontsize=30, fontweight="bold")
97                      axes[1].imshow(
98                          slice_1.T, cmap="gray",
    origin="lower", aspect="auto"
99                      )
100                     axes[1].set_title("Coronal"
    , fontsize=30, fontweight="bold")
101                     axes[2].imshow(
102                         slice_2.T, cmap="gray",
    origin="lower", aspect="auto"
103                     )
104                     axes[2].set_title("Axial",
    fontsize=30, fontweight="bold")
105                     plt.suptitle(
106                         f"Axial, Sagittal and
    Coronal view of the patient's brain \n
    Patient ID : {patient}    MGMT_value : {int(
    self.train_df.loc[self.train_df['BraTS21ID']
     == int(patient)]['MGMT_value'])}    Type: {
    types} ",
107                         fontsize=38,
108                         fontweight="bold",
109                     )
110                     os.makedirs(jpg_path +
    patient + "/" + types + "/", exist_ok=True)
```

```
111                     fig.savefig(  # saving the
    jpg format with info.
112                         jpg_path
113                         + patient
114                         + "/"
115                         + types
116                         + "/"
117                         + patient
118                         + types
119                         + ".jpg",
120                         dpi=300,
121                     )
122             except:
123                 continue
124
125     """
126     <<best_view>> finds the best postion view
    in each direction of sagittal,
127     coronal and axial. It is done by counting
    all the non-zero cell and storing the
128     maximum value for each direction.
129
130     Args:
131     img : NIFTI image as an input
132
133     return --> best_postions as a list containg
     the best postion view for visualization
134     """
135
136     def best_view(self, img):
137
138         img_data = img.get_fdata()
139         count_sag, count_axi, count_cor = [],
    [], []
```

```
140
141         for i in range(0, img.shape[0]):
142             count_sag.append(np.count_nonzero(
    img_data[i, :, :]))
143
144         for j in range(0, img.shape[1]):
145             count_cor.append(np.count_nonzero(
    img_data[:, j, :]))
146
147         for k in range(0, img.shape[2]):
148             count_axi.append(np.count_nonzero(
    img_data[:, :, k]))
149
150         position_cor = np.argmax(count_cor)
151         position_sag = np.argmax(count_sag)
152         position_axi = np.argmax(count_axi)
153
154         return [position_sag, position_cor,
    position_axi]
155
156       """ <<image_info>> function extracts the
    volume and size of images + single
157         vocxel in each one and store it in a
    csv file.
158       """
159
160       def image_info(self):
161
162         df = pd.DataFrame(
163             columns=[
164                 "image",
165                 "voxel volume",
166                 "voxel size",
167                 "image volume",
```

```
                "image size",
            ]
        )

        for patient in self.patientID:
            try:
                for types in self.sqtypes_task2
:
                    # img = nib.load(
                    #     train_path_nifti
                    #     + patient
                    #     + "/"
                    #     + patient
                    #     + '_'
                    #     + types
                    #     + ".nii.gz"
                    # )
                    img = nib.load(
                        os.path.join(
                            self.
train_path_nifti,
                            patient + "/" +
types + "/" + patient + types + ".nii",
                        )
                    )

                    voxel_size = list(img.
header.get_zooms())
                    voxel_volume = np.prod(img.
header["pixdim"][1:4])

                    image_shape = list(img.
shape)
```

```
195                     voxel_count = np.
    count_nonzero(img.get_data())
196                     image_volume = voxel_volume
     * voxel_count
197                     image_size = (
198                         image_shape[0] *
    voxel_size[0],
199                         image_shape[1] *
    voxel_size[1],
200                         image_shape[2] *
    voxel_size[2],
201                     )
202 
203                     df = df.append(
204                         {
205                             "image": patient +
    types,
206                             "voxel volume":
    voxel_volume,
207                             "voxel size":
    voxel_size,
208                             "image volume":
    image_volume,
209                             "image size":
    image_size,
210                         },
211                         ignore_index=True,
212                     )
213             except:
214                 continue
215 
216         df.to_csv("
    imageInfo_resampled_task2_data.csv", index=
    False)
```

libraries_pre.py

Pyradiomics class **separates** the different areas of the segmented images and **generates** a csv file containing all the dataset and segmented images directory. The primary function is the **feature extractor** which gets all the radiomics features as a csv file:

```
"""
libraries : Pyradiomics
"""
import os
import collections
import csv
import logging
import SimpleITK as sitk
import radiomics
from radiomics import featureextractor
import glob
import numpy as np

""" path_result : storing the result in a
  specific directory
    seg_path : different segmented images
    patientID : list of patiend's ID - Task 1/2
    train_task1 : NIFTI images for task 1
  dataset
    sequence_types : sequence type of mMRI
  pictures
"""


class Pyradiomics:
    def __init__(self, path_result, seg_path,
  train_task1, sequence_types, patientID):
```

```
24
25          self.path_result = r"...[path to store
    the result]"
26          self.seg_path = r"... [path to
    segmented dataset]"
27          self.train_task1 = r"path to task 1
    dataset"
28          self.sequence_types = ["flair", "t1", "
    t1ce", "t2"]
29          self.patientID = os.listdir(
30              r"... [path to list of patient's ID
     of task 1/2]"
31          )
32
33      """ <<generate_csv>> function generate a
    csv file with two columns: Image and Mask
34          directory of different sequence type
    and different segmented brain
35          tumor will be added as a new row to the
     csv file.
36
37          outcome:
38              create a csv file with original
    image and different masks.
39      """
40
41      def generate_csv(self):
42          with open(
43              os.path.join(self.path_result, "
    radiomics_features_task1.csv"),
44              "a",
45              newline="",
46          ) as csvfile:
47
```

```
48              # creating the column heads
49              writer = csv.writer(csvfile)
50              writer.writerow(["Image", "Mask"])
51
52              # filling each cell with the path
    of image and mask
53              try:
54                  for patient in self.patientID:
55                      for types in self.
    sequence_types:
56
57                          img = os.path.join(
58                              self.train_task1,
59                              patient + "\\" +
    patient + "_" + types + ".nii.gz",
60                          )
61
62                          mask = os.path.join(
63                              self.train_task1,
    patient + "\\" + patient + "_seg.nii.gz"
64                          )
65                          mask_ED = os.path.join(
66                              self.train_task1,
67                              patient + "\\" +
    patient + "_seg_ED.nii.gz",
68                          )
69                          mask_ET = os.path.join(
70                              self.train_task1,
71                              patient + "\\" +
    patient + "_seg_ET.nii.gz",
72                          )
73                          mask_NCR = os.path.join
    (
74                              self.train_task1,
```

```
75                                  patient + "\\" +
    patient + "_seg_NCR.nii.gz",
76                              )
77
78                              writer.writerow([img] +
     [mask])
79                              writer.writerow([img] +
     [mask_ED])
80                              writer.writerow([img] +
     [mask_ET])
81                              writer.writerow([img] +
     [mask_NCR])
82                      except:
83                          pass
84
85      """ Separating the segmented images of task
     1 into four parts:
86          1. image with the whole tumor (labes :
    1+2+4)
87          2. image with necrotic (NCR) parts of
    the tumor (labes : 1)
88          3. image with peritumoral edematous/
    invaded tissue (ED) (labes : 2)
89          4. image with enhancing tumor (ET) (
    labes : 4)
90          5. image with tumor core (COR) (labels
    : 1+4)
91      """
92
93      def separate_seg(self):
94          #%% loop through the input path folder
    to separate different area
95          #   and relable it to 1 for feature
    extraction
```

```
96
97          for patient in self.patientID:
98
99              # reading the original image
100             img = sitk.ReadImage(
101                 os.path.join(
102                     self.train_task1,
103                     patient + "/" + patient + "
    _seg.nii.gz",
104                 )
105             )
106
107             # getting the image data
108             img_data = sitk.GetArrayFromImage(
    img)
109
110             # relabel the whole tumor to 1
111             img_whole_data = np.where((img_data
     != 0), 1, img_data)
112
113             # keeping the nerotic part of the
    tumor and removing the rest
114             img_NCR_data = np.where((img_data
    != 1), 0, img_data)
115
116             # extracting the edema (label 2)
    and relabel it to 1
117             img_ED_data = np.where(
118                 (np.where((img_data != 2), 0,
    img_data) == 2),
119                 1,
120                 np.where((img_data != 2), 0,
    img_data),
121             )
```

```
122
123             # extracting the enhacing part of
    the tumor (label 4) and relabel it to 1
124             img_ET_data = np.where(
125                 (np.where((img_data != 4), 0,
    img_data) == 4),
126                 1,
127                 np.where((img_data != 4), 0,
    img_data),
128             )
129
130             # extracting the core of the tumor
    (label 1 & 4) and relabel it to 1
131             img_COR_data = (
132                 np.where((img_data == 2), 0,
    img_data)
133                 & np.where((img_data == 1), 1,
    img_data)
134                 & np.where((img_data == 4), 1,
    img_data)
135             )
136
137             # getting the metadata of the
    original image and assign it to
138             # new segmented area as NIFTI file
139             img_whole = sitk.GetImageFromArray(
    img_whole_data)
140             img_whole.CopyInformation(img)
141             img_NCR = sitk.GetImageFromArray(
    img_NCR_data)
142             img_NCR.CopyInformation(img)
143             img_ED = sitk.GetImageFromArray(
    img_ED_data)
144             img_ED.CopyInformation(img)
```

```
145             img_ET = sitk.GetImageFromArray(
    img_ET_data)
146             img_ET.CopyInformation(img)
147             img_COR = sitk.GetImageFromArray(
    img_COR_data)
148             img_COR.CopyInformation(img)
149 
150             # saving all the nifti files in
    output path
151             sitk.WriteImage(
152                 img_whole,
153                 os.path.join(
154                     self.train_task1, patient +
     "/" + patient + "_seg_whole.nii.gz"
155                 ),
156             )
157 
158             sitk.WriteImage(
159                 img_NCR,
160                 os.path.join(
161                     self.train_task1, patient +
     "/" + patient + "_seg_NCR.nii.gz"
162                 ),
163             )
164 
165             sitk.WriteImage(
166                 img_ED,
167                 os.path.join(
168                     self.train_task1, patient +
     "/" + patient + "_seg_ED.nii.gz"
169                 ),
170             )
171 
172             sitk.WriteImage(
```

```
173                 img_ET,
174                 os.path.join(
175                     self.train_task1, patient +
    "/" + patient + "_seg_ET.nii.gz"
176                 ),
177             )
178
179             sitk.WriteImage(
180                 img_COR,
181                 os.path.join(
182                     self.train_task1, patient +
    "/" + patient + "_seg_COR.nii.gz"
183                 ),
184             )
185
186     """ <<feature_extraction>> function uses
    the csv file which contains the
187         directory to segmented images and
    differemt masks in order to extract
188         radiomics features of each nifti image.
189
190         outcome:
191             create a csv file with all the
    features related to origanl images (and
192             its filters)
193     """
194
195     def feature_extraction():
196
197         os.chdir(r"... [path to save the
    pyradiomics results]")
198         outPath = r"... [path to save the
    pyradiomics results]"
199
```

```
200         filescsv = glob.glob("
    radiomics_features_task1.csv")
201 
202         # filescsv=glob.glob('Radiomics_*.csv')
203         for inFile in filescsv[:]:
204             inputCSV = os.path.join(outPath,
    inFile)
205             outputFilepath = os.path.join(
    outPath, "Results_" + inFile)
206             progress_filename = os.path.join(
    outPath, "pyrad_log.txt")
207             params = os.path.join(outPath, "
    exampleSettings", "Params.yaml")
208 
209             # Configure logging
210             rLogger = logging.getLogger("
    radiomics")
211 
212             # Set logging level
213             # rLogger.setLevel(logging.INFO)  #
     Not needed, default log level of logger is
    INFO
214 
215             # Create handler for writing to log
     file
216             handler = logging.FileHandler(
    filename=progress_filename, mode="w")
217             handler.setFormatter(
218                 logging.Formatter("%(levelname)
    s:%(name)s: %(message)s")
219             )
220             rLogger.addHandler(handler)
221 
```

```
222             # Initialize logging for batch log
    messages
223             logger = rLogger.getChild("batch")
224 
225             # Set verbosity level for output to
     stderr (default level = WARNING)
226             radiomics.setVerbosity(logging.INFO
    )
227 
228             logger.info("pyradiomics version: %
    s", radiomics.__version__)
229             logger.info("Loading CSV")
230 
231             flists = []
232             try:
233                 with open(inputCSV, "r") as
    inFile:
234                     cr = csv.DictReader(inFile,
     lineterminator="\n")
235                     flists = [row for row in cr
    ]
236             except Exception:
237                 logger.error("CSV READ FAILED",
     exc_info=True)
238 
239             logger.info("Loading Done")
240             logger.info("Patients: %d", len(
    flists))
241 
242             if os.path.isfile(params):
243                 extractor = featureextractor.
    RadiomicsFeatureExtractor(params)
244             else:  # Parameter file not found,
    use hardcoded settings instead
```

```
245                 settings = {}
246                 # settings['binWidth'] = 25
247                 # settings['
    resampledPixelSpacing'] = [0.75, 0.75, 1]  #
     [3,3,3]
248                 # settings['interpolator'] =
    sitk.sitkBSpline
249                 # settings['correctMask'] =
    True
250                 settings["geometryTolerance"] =
     1
251                 # settings['enableCExtensions']
     = True
252
253                 extractor = featureextractor.
    RadiomicsFeatureExtractor(**settings)
254                 # extractor.enableInputImages(
    wavelet= {'level': 2})
255
256                 #  logger.info('Enabled input
    images types: %s', extractor.
    enabledImageTypes)
257                 #  logger.info('Enabled features: %
    s', extractor.enabledFeatures)
258                 #  logger.info('Current settings: %
    s', extractor.settings)
259
260                 headers = None
261
262                 for idx, entry in enumerate(flists,
     start=1):
263
264                     logger.info(
```

```
265                     "(%d/%d) Processing Patient
    (Image: %s, Mask: %s)",
266                     idx,
267                     len(flists),
268                     entry["Image"],
269                     entry["Mask"],
270                 )
271 
272                 imageFilepath = entry["Image"]
273                 maskFilepath = entry["Mask"]
274                 label = entry.get("Label", None
    )
275 
276                 if str(label).isdigit():
277                     label = int(label)
278                 else:
279                     label = None
280 
281                 if (imageFilepath is not None)
    and (maskFilepath is not None):
282                     featureVector = collections
    .OrderedDict(entry)
283                     featureVector["Image"] = os
    .path.basename(imageFilepath)
284                     featureVector["Mask"] = os.
    path.basename(maskFilepath)
285 
286                     try:
287                         featureVector.update(
288                             extractor.execute(
    imageFilepath, maskFilepath, label)
289                         )
290 
```

```
                            with open(
    outputFilepath, "a") as outputFile:
                                writer = csv.writer
    (outputFile, lineterminator="\n")
                                if headers is None:
                                    headers = list(
    featureVector.keys())
                                    writer.writerow
    (headers)

                                row = []
                                for h in headers:
                                    row.append(
    featureVector.get(h, "N/A"))
                                writer.writerow(row
    )
                        except Exception:
                            logger.error("FEATURE
    EXTRACTION FAILED", exc_info=True)
```

libraries_pyradiomics.py

DataProcess class tries to generate a clean dataset before applying any Machine Learning classification method. **numeric_data** will keep only numerical data type and remove the other unusable types. The other functions such as **mean_conf** or **make_confusion_matrix** used to get confusion matrix in proper way through different nested folds algorithms. **top_features** functions and also **low_variance** considered for choosing best and the most informative features.

```
"""
libraries : data processing and applying ML
    algorithms
"""
import os
import pandas as pd
import numpy as np
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from xgboost import XGBClassifier
from sklearn.feature_selection import
  VarianceThreshold
import matplotlib.pyplot as plt
from sklearn.model_selection import
  StratifiedKFold
import seaborn as sns
from sklearn.feature_selection import RFE
from sklearn.feature_selection import RFECV

""" separated_extracted_data_list: list of
  separated csv file names
    separated_extracted_data_path: path of
  separated csv file names
"""

class DataProcess:
    def __init__(
        self, separated_extracted_data_list,
  separated_extracted_data_path, save_path
    ):

        self.separated_extracted_data_list = os
  .listdir(
            r"... [path to list of separateed
  csv files name]"
        )
        self.separated_extracted_data_path = (
            r"...[path to directory of the
  separated csv files]"
        )
```

```
32          self.save_path = r"... [path to store
    the result]"
33
34      def numeric_data(self, file_name):
35
36          """<<numeric_data>>: this function help
     to eliminate the columns which does not
    have
37          numeric type of data:
38
39          parameter --> file (csv)
40          return --> output dataset with only
    numerical data type
41          """
42
43          df_raw = pd.read_csv(self.
    separated_extracted_data_path + file_name)
44
45          # excluding the image IDs
46          df = df_raw.drop(["Image"], axis=1)
47
48          # excluding all the object types of
    data + adding back the ID column
49          df_numeric = df.select_dtypes(exclude="
    object")
50          df_numeric.insert(0, "Image", df_raw["
    Image"])
51
52          return df_numeric
53
54      def low_variance(self, file_name):
55
56          """
```

```
57          <<low_variance>>: this function check
    feathurs' variance and based on
58          the defined threshold, remov the low
    variance features which give less
59          information about the data (the numeric
     data that has obtained.)
60          """
61
62          # getting numerical data with
    numeric_data function
63          df_numeric = DataProcess.numeric_data(
    file_name)
64
65          # Removing both constant and quasi-
    constant features
66          var_thr = VarianceThreshold(threshold
    =0.25)
67          var_thr.fit(df_numeric)
68
69          concol = [
70              column
71              for column in df_numeric.columns
72              if column not in df_numeric.columns
    [var_thr.get_support()]
73          ]
74          df_numeric = df_numeric.drop(concol,
    axis=1)
75
76          # save the new updated dataset
77          df_numeric.to_csv(os.path.join(self.
    save_path, "cleaned_" + file_name))
78
79      def mean_conf(self, confusion_matrix):
80
```

```
81          """
82          <<mean_conf>> simply gets the mean of
    each element in confiusion matrixs
83          which are the outcome in each k-subset
    cross validaion.
84
85          return --> mean of all confision
    matrics
86          """
87          # empty lists to fill up every elements
     of the different confusion matrics
88          e1, e2, e3, e4 = [], [], [], []
89          for i in range(0, len(confusion_matrix)
    ):
90              e1.append(confusion_matrix[i
    ][0][0])
91              e2.append(confusion_matrix[i
    ][0][1])
92              e3.append(confusion_matrix[i
    ][1][0])
93              e4.append(confusion_matrix[i
    ][1][1])
94          # getting mean of each element
95          mean_matrix = [
96              [round(np.mean(e1)), round(np.mean(
    e2))],
97              [round(np.mean(e3)), round(np.mean(
    e4))],
98          ]
99          return mean_matrix
100
101       def make_confusion_matrix(cf, group_names,
    categories, title):
102         """
```

```
103         This function will make a pretty plot
    of an sklearn Confusion Matrix cm using
104         a Seaborn heatmap visualization.
105 
106         Parameters
107         ---------
108         cf:            confusion matrix to be
    passed in
109         group_names:   List of strings that
    represent the labels row by row to be shown
    in each square.
110         categories:    List of strings
    containing the categories to be displayed on
     the x,y axis.
111         cmap:          Colormap of the values
    displayed from matplotlib.pyplot.cm.
112                        See http://matplotlib.
    org/examples/color/colormaps_reference.html
113         title:         Title for the heatmap.
114 
115         Returns
116         -------
117         None
118         """
119 
120         group_labels = ["{}\n".format(value)
    for value in group_names]
121 
122         group_percentages = [
123             "{:.2f}".format(value) for value in
     cf.flatten() / np.sum(cf)
124         ]
125         group_counts = [f"{round(559*value)}\n"
     for value in cf.flatten()]
```

```
126
127         box_labels = [
128             f"{v1}{v2}{v3}".strip()
129             for v1, v2, v3 in zip(group_labels,
     group_counts, group_percentages)
130         ]
131         box_labels = np.asarray(box_labels).
    reshape(cf.shape[0], cf.shape[1])
132
133         # Accuracy is sum of diagonal divided
    by total observations
134         accuracy = np.trace(cf) / float(np.sum(
    cf))
135
136         # Metrics for Binary Confusion Matrices
137         precision = cf[1, 1] / sum(cf[:, 1])
138         recall = cf[1, 1] / sum(cf[1, :])
139         f1_score = 2 * precision * recall / (
    precision + recall)
140         stats_text = "\n\nAccuracy={:0.3f}\
    nPrecision={:0.3f}\nRecall={:0.3f}\nF1 Score
    ={:0.3f}".format(
141             accuracy, precision, recall,
    f1_score
142         )
143
144         # Make the heatmap visualization
145         plt.figure(figsize=None)
146         sns.heatmap(
147             cf,
148             annot=box_labels,
149             fmt="",
150             cmap="Blues",
151             cbar=True,
```

```
152             xticklabels="auto",
153             yticklabels=categories,
154             vmin=0,
155             vmax=1,
156         )
157 
158         plt.ylabel("True label")
159         plt.xlabel("Predicted label" +
    stats_text)
160 
161         plt.title(title)
162 
163     def correlation_map(df, list_im_feat,
    save_path=None, title=None):
164         """
165         Parameters
166         ----------
167         df: DataFrame
168             dataframe as an input
169         list_im_feat: List
170             set of features name extracted from
     a dataframe (by feature selectors)
171         save_path: str
172             directory to save the output as a
    JPG (default is None)
173         title: str
174             title for the heatmap (default is
    None.)
175 
176         Returns
177         -------
178         None.
179 
180         """
```

```
181         # dividing the whole dataframe into the
    small part containing only top features
182         selected_df = df[list_im_feat].copy()
183 
184         fig, ax = plt.subplots(figsize=(15, 12)
    )
185 
186         # plotting correlation heatmap
187         dataplot = sns.heatmap(
188             selected_df.corr(), cmap="YlGnBu",
    annot=True, vmin=-1, vmax=1
189         )
190 
191         if save_path != None and title != None:
192             plt.title(title)
193             plt.savefig(save_path, dpi=300,
    bbox_inches="tight")
194 
195         # displaying heatmap
196         plt.show()
197 
198     def top_features_XGB(self, df, number_feat,
    target_var):
199 
200         """
201         This function finds common top i
    features (by XGBoost classifier) and
202         returns them as a list of strings that
    are the features' names.
203 
204 
205         Parameters
206         ----------
207             df: DataFrame
```

```
208                 dataframe as an input
209             number_feat: int
210                 number of features needed to be
    ranked
211             target_var: str
212                 target variable
213
214         Returns
215         -------
216             im_feat: list
217                 list of i top-ranked features
218         """
219
220         # defining the target value and
    separate it
221         y = df[target_var]
222         X = df.drop([target_var], axis=1)
223
224         kf = KFold(n_splits=5, shuffle=True)
225         for train_index, test_index in kf.split
    (X):
226             X_train, X_test = X.iloc[
    train_index, :], X.iloc[test_index, :]
227             y_train, y_test = y.iloc[
    train_index], y.iloc[test_index]
228
229             # declare parameters
230             params = {
231                 "objective": "binary:logistic",
232                 "max_depth": 4,
233                 "alpha": 10,
234                 "learning_rate": 1.0,
235                 "n_estimators": 100,
236             }
```

```
237
238             # instantiate the classifier
239             xgb_clf = XGBClassifier(**params)
240
241             # fit the classifier to the
    training data
242             xgb_clf.fit(X_train, y_train)
243
244             # list of features name
245             feat_names = list(X_train.columns)
246
247             feats = {}  # a dict to hold
    feature_name: feature_importance
248             for feature, importance in zip(
    feat_names, xgb_clf.feature_importances_):
249                 feats[feature] = importance  #
    add the name/value pair
250             # appending the dictionary of
    features with their scores by each k subset
251             feats.update({x: y for x, y in
    feats.items() if y != 0})
252
253         # sort the features based on their
    importance
254         im_feat = sorted(feats.items(), key=
    lambda feats: feats[1], reverse=True)[
255             :number_feat
256         ]
257         # im_feat.sort(key = lambda x: x[1],
    reverse=True)
258         im_feat = [item for sublist in im_feat
    for item in sublist]
259         im_feat = [elm for elm in im_feat if
    isinstance(elm, str)]
```

```
260
261         # the list of most i-th top ranked
    features
262         return im_feat
263
264       def top_features_RFE(self, df, number_feat,
     target_var):
265         """
266         This function finds top features using
    the Recursive Feature Elimination
267         approach and returns a list of str.
268
269         Parameter
270         ---------
271         df: DataFrame
272             dataframe as an input
273         number_feat: int
274             number of features needed to be
    ranked
275         target_var: str
276             target variable
277
278         Return
279         ------
280         best_feat: list
281             list of top features' name
282
283         """
284
285         # defining the target value and
    separate it
286         y = df[target_var]
287         X = df.drop([target_var], axis=1)
288
```

```
289         best_feat = []  # list of faetures' name
290 
291         rfe = RFE(
292             estimator=DecisionTreeClassifier(), step=1, n_features_to_select=number_feat
293         )
294 
295         # fit RFE
296         rfe.fit(X, y)
297 
298         # get the score for the top selected features
299         feature_importance = rfe.estimator_.feature_importances_
300         # sort the reanking out with its index number (first one is the best feature)
301         feature_importance_sorted = sorted(
302             enumerate(feature_importance), key=lambda x: x[1], reverse=True
303         )
304         # extract the index of ranking among the top features
305         top_n_idx = [idx for idx, _ in feature_importance_sorted[:]]
306 
307         # based on index get the name of the features
308         top_n_feat_idx = [rfe.get_support(1)[i] for i in top_n_idx]
309 
310         for item in top_n_feat_idx:
311             best_feat.append(X.iloc[:, int(item)].name)
```

```
312
313         return best_feat[:number_feat]
314
315     def top_features_RFECV(self, df,
    number_feat, target_var):
316         """
317         This function finds top features using
    the Recursive Feature Elimination
318         in a cross-validation loop to find the
    optimal number of features and returns
319         them as a list of str.
320
321         Parameter
322         ---------
323         df: DataFrame
324             dataframe as an input
325         number_feat: int
326             number of features needed to be
    ranked
327         target_var: str
328             target variable
329
330         Return
331         ------
332         best_feat: list
333             list of top features' name
334
335         """
336
337         # defining the target value and
    separate it
338         y = df[target_var]
339         X = df.drop([target_var], axis=1)
340
```

```
341         best_feat = []  # list of faetures'
    name
342
343         # define RFECV
344         rfecv = RFECV(
345             estimator=DecisionTreeClassifier(),
346             cv=StratifiedKFold(5),
347             scoring="accuracy",
348             min_features_to_select=number_feat,
349         )
350
351         # fit RFECV
352         rfecv.fit(X, y)
353
354         # get the score for the top selected
    features
355         feature_importance = rfecv.estimator_.
    feature_importances_
356         # sort the reanking out with its index
    number (first one is the best feature)
357         feature_importance_sorted = sorted(
358             enumerate(feature_importance), key=
    lambda x: x[1], reverse=True
359         )
360         # extract the index of ranking among
    the top features
361         top_n_idx = [idx for idx, _ in
    feature_importance_sorted[:]]
362
363         # based on index get the name of the
    features
364         top_n_feat_idx = [rfecv.get_support(1)[
    i] for i in top_n_idx]
365         for item in top_n_feat_idx:
```

```
366                 best_feat.append(X.iloc[:, int(item
    )].name)
367
368             return best_feat[:number_feat]
```

libraries_data.py

```
1  #%% main part
2  """ Splitting the dataset and applying k-fold
       cross validation
3      Feature selection by XGBoost method
4      Fitting different model: SVM,
       LogisticRegression, Random forest, NN
5      Changing the numbere of features to see the
        idieal number
6  """
7  # defining a new empty dataframe to fill with
       different metrics
8  metrics = pd.DataFrame(
9      columns=[
10             "features_number",
11             "mean_accuracy_NN",
12             "std_accuracy_NN",
13             "mean_f1score_NN",
14             "std_f1score_NN",
15             "confusion_NN",
16             "mean_accuracy_SVM",
17             "std_accuracy_SVM",
18             "mean_f1score_SVM",
19             "std_f1score_SVM",
20             "confusion_SVM",
21             "mean_accuracy_LR",
22             "std_accuracy_LR",
23             "mean_f1score_LR",
24             "std_f1score_LR",
```

```
        "confusion_LR",
        "mean_accuracy_MLP",
        "std_accuracy_MLP",
        "mean_f1score_MLP",
        "std_f1score_MLP",
        "confusion_MLP",
    ]
)

# defining a new empty dataframe for filling
  best parameters in each iteration
parameters = pd.DataFrame(
    columns=[
        "features_number",
        "Nearest Neighbor",
        "Support Vector Machine",
        "Logistic Regresion",
        "Multi-layer Perceptron",
        "Best Selected Features",
    ]
)

# copy the dataframe for mean and std of
  metrics
train_metrics = metrics.copy()
test_metrics = metrics.copy()

# reading and splitting the edataset into train
   and test
df = pd.read_csv("/content/drive/MyDrive/data/
  all_best_data.csv")

# getting the top features
```

```
list_im_feat = top_features_XGB(df, 20, "MGMT_value")
print("check here:", list_im_feat)

# defining the target value and separate it
y = df["MGMT_value"]
X = df.drop(["MGMT_value", "Unnamed: 0"], axis=1)


# dataset with best features
X = X[list_im_feat].copy()

# splitting the whole dataset into train (80%) and test (20%)
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.2, random_state=0)

# transform data: final test
scaler = MinMaxScaler()
X_ts = scaler.fit_transform(X_ts)

# convert the test set to the dataframe in order to use it in the while loop
X_ts = pd.DataFrame(X_ts, columns=X_tr.columns)

# applying k-fold cross validation (K=10) --> outer loop
cv_outer = KFold(n_splits=10, shuffle=True)

# iteration over number of features i
i = 20
while i != 0:

```

```
82      # defining performance metrics lists for
    training
83      conf_NN_tr, conf_SVM_tr, conf_LR_tr,
    conf_MLP_tr = [], [], [], []
84      acc_NN_tr, acc_SVM_tr, acc_LR_tr,
    acc_MLP_tr = [], [], [], []
85      f1_NN_tr, f1_SVM_tr, f1_LR_tr, f1_MLP_tr =
    [], [], [], []
86
87      # defining performance metrics lists for
    test
88      conf_NN_ts, conf_SVM_ts, conf_LR_ts,
    conf_MLP_ts = [], [], [], []
89      acc_NN_ts, acc_SVM_ts, acc_LR_ts,
    acc_MLP_ts = [], [], [], []
90      f1_NN_ts, f1_SVM_ts, f1_LR_ts, f1_MLP_ts =
    [], [], [], []
91
92      # defining best paramters list to store for
     traing\test
93      best_par_NN, best_par_SVM, best_par_LR,
    best_par_MLP = [], [], [], []
94
95      # configuring thee cross-validation outer
    loop
96      for train_index, test_index in cv_outer.
    split(X_tr):
97          X_train, X_test = X_tr.iloc[train_index
    , :], X_tr.iloc[test_index, :]
98          y_train, y_test = y_tr.iloc[train_index
    ], y_tr.iloc[test_index]
99
100         # configuring the cross-validation
    procedure (inner loop)
```

```
101         cv_inner = KFold(n_splits=5, shuffle=
    True, random_state=1)
102 
103         # keep the most top i ranked features
104         X_train = X_train[list_im_feat[:i]].
    copy()
105         X_test = X_test[list_im_feat[:i]].copy
    ()
106         X_ts = X_ts[list_im_feat[:i]].copy()
107 
108         # transform data
109         X_train = scaler.fit_transform(X_train)
110         X_test = scaler.fit_transform(X_test)
111 
112         #%% Nearst neighbor:
113         # Create and train the
    KNeighborsClassifier on the train\test set
114         model_NN = KNeighborsClassifier()
115 
116         # Set up possible values of parameters
    to optimize over
117         parameters_NN = {
118             "n_neighbors": [3, 5, 11, 19],
119             "weights": ["ubiform", "distance"],
120             "metric": ["euclidean", "manhattan"
    ],
121         }
122 
123         # define search
124         classifier_NN = GridSearchCV(
125             model_NN, parameters_NN, scoring="
    accuracy", cv=cv_inner, refit=True
126         )
127 
```

```
128         # execute search
129         result_NN = classifier_NN.fit(X_train,
    y_train)
130 
131         # get the best performing model fit on
    the whole training\test set + save the best
    parameters
132         best_model_NN = result_NN.
    best_estimator_
133         best_par_NN.append(classifier_NN.
    best_params_)
134 
135         # make a prediction on the validation
    set and then check model performance (train)
136         y_pred_NN = best_model_NN.predict(
    X_train)
137 
138         acc_NN_tr.append(accuracy_score(y_train
    , y_pred_NN))
139         conf_NN_tr.append(confusion_matrix(
    y_train, y_pred_NN, normalize="all"))
140         f1_NN_tr.append(f1_score(y_train,
    y_pred_NN))
141 
142         # make a prediction on the validation
    set and then check model performance (test)
143         y_pred_NN = best_model_NN.predict(X_ts)
144 
145         acc_NN_ts.append(accuracy_score(y_ts,
    y_pred_NN))
146         conf_NN_ts.append(confusion_matrix(y_ts
    , y_pred_NN, normalize="all"))
147         f1_NN_ts.append(f1_score(y_ts,
    y_pred_NN))
```

```
148
149         #%% Support Vector Machine:
150         # build the SVM classifier and train it
     on the entire training\test data set
151         model_SVM = SVC()
152
153         # Set up possible values of parameters
    to optimize over
154         parameters_SVM = {
155             "C": [0.1, 1, 10, 100, 1000],
156             "gamma": [1, 0.1, 0.01, 0.001,
    0.0001],
157             "kernel": ["rbf", "poly", "sigmoid"
    ],
158         }
159
160         # define search
161         classifier_SVM = GridSearchCV(
162             model_SVM, parameters_SVM, scoring=
    "accuracy", cv=cv_inner, refit=True
163         )
164
165         # execute search
166         result_SVM = classifier_SVM.fit(X_train
    , y_train)
167
168         # get the best performing model fit on
    the whole training\test set + save the best
    parameters
169         best_model_SVM = result_SVM.
    best_estimator_
170         best_par_SVM.append(classifier_SVM.
    best_params_)
171
```

```
172         # get predictions on the test set and
    store the performance metrics (train)
173         y_pred_SVC = best_model_SVM.predict(
    X_train)
174 
175         acc_SVM_tr.append(accuracy_score(
    y_train, y_pred_SVC))
176         conf_SVM_tr.append(confusion_matrix(
    y_train, y_pred_SVC, normalize="all"))
177         f1_SVM_tr.append(f1_score(y_train,
    y_pred_SVC))
178 
179         # get predictions on the test set and
    store the performance metrics (test)
180         y_pred_SVC = best_model_SVM.predict(
    X_ts)
181 
182         acc_SVM_ts.append(accuracy_score(y_ts,
    y_pred_SVC))
183         conf_SVM_ts.append(confusion_matrix(
    y_ts, y_pred_SVC, normalize="all"))
184         f1_SVM_ts.append(f1_score(y_ts,
    y_pred_SVC))
185 
186         #%% Logistic Regession:
187         # build the classifier and fit the
    model
188         model_LR = LogisticRegression()
189 
190         # Set up possible values of parameters
    to optimize over
191         parameters_LR = {
192             "penalty": ["none", "l1", "l2", "
    elasticnet"],
```

```
193             "C": [0.001, 0.009, 0.01, 0.09, 1,
    5, 10, 25, 50, 75, 100],
194             "solver": ["newton-cg", "lbfgs", "
    liblinear"],
195         }
196 
197         # define search
198         classifier_LR = GridSearchCV(
199             model_LR, parameters_LR, scoring="
    accuracy", cv=cv_inner, refit=True
200         )
201 
202         # execute search
203         result_LR = classifier_LR.fit(X_train,
    y_train)
204 
205         # get the best performing model fit on
    the whole training set + save the best
    parameters
206         best_model_LR = result_LR.
    best_estimator_
207         best_par_LR.append(classifier_LR.
    best_params_)
208 
209         # prediction and store performance
    metrics (train)
210         y_pred_LR = best_model_LR.predict(
    X_train)
211 
212         acc_LR_tr.append(accuracy_score(y_train
    , y_pred_LR))
213         conf_LR_tr.append(confusion_matrix(
    y_train, y_pred_LR, normalize="all"))
```

```
214         f1_LR_tr.append(f1_score(y_train,
    y_pred_LR))
215
216         # prediction and store performance
    metrics (test)
217         y_pred_LR = best_model_LR.predict(X_ts)
218
219         acc_LR_ts.append(accuracy_score(y_ts,
    y_pred_LR))
220         conf_LR_ts.append(confusion_matrix(y_ts
    , y_pred_LR, normalize="all"))
221         f1_LR_ts.append(f1_score(y_ts,
    y_pred_LR))
222
223         #%% Neural Network:
224         # create a MLPClassifier and fit the
    model
225         model_MPL = MLPClassifier(
226             solver="lbfgs", alpha=1e-5,
    hidden_layer_sizes=(6,), random_state=1
227         )
228
229         # Set up possible values of parameters
    to optimize over
230         parameters_MLP = {
231             "batch_size": [256],
232             "momentum": [0.9, 0.99],
233             "learning_rate_init": [0.001, 0.01,
     0.1],
234             "solver": ["adam"],
235             "alpha": [0.0001, 0.05],
236             "learning_rate": ["constant", "
    adaptive"],
237         }
```

```
238
239         # define search
240         classifier_MLP = GridSearchCV(
241             model_MPL, parameters_MLP, scoring=
    "accuracy", cv=cv_inner, refit=True
242         )
243
244         # execute search
245         result_MLP = classifier_MLP.fit(X_train
    , y_train)
246
247         # get the best performing model fit on
    the whole training set + save the best
    parameters
248         best_model_MLP = result_MLP.
    best_estimator_
249         best_par_MLP.append(classifier_MLP.
    best_params_)
250
251         # prediction and store preformance
    metrics (train)
252         y_pred_NN = best_model_MLP.predict(
    X_train)
253
254         acc_MLP_tr.append(accuracy_score(
    y_train, y_pred_NN))
255         conf_MLP_tr.append(confusion_matrix(
    y_train, y_pred_NN, normalize="all"))
256         f1_MLP_tr.append(f1_score(y_train,
    y_pred_NN))
257
258         # prediction and store preformance
    metrics (test)
```

```
259         y_pred_NN = best_model_MLP.predict(X_ts
    )
260 
261         acc_MLP_ts.append(accuracy_score(y_ts,
    y_pred_NN))
262         conf_MLP_ts.append(confusion_matrix(
    y_ts, y_pred_NN, normalize="all"))
263         f1_MLP_ts.append(f1_score(y_ts,
    y_pred_NN))
264 
265       # storing result of evaluation metrics in
    dataframe for furthre anlysis
266       # trainging results
267       train_data_to_store = {
268           "features_number": f"{i}",
269           "mean_accuracy_NN": np.mean(acc_NN_tr),
270           "std_accuracy_NN": np.std(acc_NN_tr),
271           "mean_f1score_NN": np.mean(f1_NN_tr),
272           "std_f1score_NN": np.std(f1_NN_tr),
273           "confusion_NN": mean_conf(conf_NN_tr),
274           "mean_accuracy_SVM": np.mean(acc_SVM_tr
    ),
275           "std_accuracy_SVM": np.std(acc_SVM_tr),
276           "mean_f1score_SVM": np.mean(f1_SVM_tr),
277           "std_f1score_SVM": np.std(f1_SVM_tr),
278           "confusion_SVM": mean_conf(conf_SVM_tr)
    ,
279           "mean_accuracy_LR": np.mean(acc_LR_tr),
280           "std_accuracy_LR": np.std(acc_LR_tr),
281           "mean_f1score_LR": np.mean(f1_LR_tr),
282           "std_f1score_LR": np.std(f1_LR_tr),
283           "confusion_LR": mean_conf(conf_LR_tr),
284           "mean_accuracy_MLP": np.mean(acc_MLP_tr
    ),
```

```
285             "std_accuracy_MLP": np.std(acc_MLP_tr),
286             "mean_f1score_MLP": np.mean(f1_MLP_tr),
287             "std_f1score_MLP": np.std(f1_MLP_tr),
288             "confusion_MLP": mean_conf(conf_MLP_tr)
    ,
289         }
290
291         # test results
292         test_data_to_store = {
293             "features_number": f"{i}",
294             "mean_accuracy_NN": np.mean(acc_NN_ts),
295             "std_accuracy_NN": np.std(acc_NN_ts),
296             "mean_f1score_NN": np.mean(f1_NN_ts),
297             "std_f1score_NN": np.std(f1_NN_ts),
298             "confusion_NN": mean_conf(conf_NN_ts),
299             "mean_accuracy_SVM": np.mean(acc_SVM_ts
    ),
300             "std_accuracy_SVM": np.std(acc_SVM_ts),
301             "mean_f1score_SVM": np.mean(f1_SVM_ts),
302             "std_f1score_SVM": np.std(f1_SVM_ts),
303             "confusion_SVM": mean_conf(conf_SVM_ts)
    ,
304             "mean_accuracy_LR": np.mean(acc_LR_ts),
305             "std_accuracy_LR": np.std(acc_LR_ts),
306             "mean_f1score_LR": np.mean(f1_LR_ts),
307             "std_f1score_LR": np.std(f1_LR_ts),
308             "confusion_LR": mean_conf(conf_LR_ts),
309             "mean_accuracy_MLP": np.mean(acc_MLP_ts
    ),
310             "std_accuracy_MLP": np.std(acc_MLP_ts),
311             "mean_f1score_MLP": np.mean(f1_MLP_ts),
312             "std_f1score_MLP": np.std(f1_MLP_ts),
313             "confusion_MLP": mean_conf(conf_MLP_ts)
    ,
```

```
314     }
315
316     # store best parametrs
317     par_to_store = {
318         "features_number": f"{i}",
319         "Nearest Neighbor": best_par_NN,
320         "Support Vector Machine": best_par_SVM,
321         "Logistic Regresion": best_par_LR,
322         "Multi-layer Perceptron": best_par_MLP,
323         "Best Selected Features": list_im_feat
    [:i],
324     }
325
326     train_metrics = train_metrics.append(
    train_data_to_store, ignore_index=True)
327     test_metrics = test_metrics.append(
    test_data_to_store, ignore_index=True)
328     parameters = parameters.append(par_to_store
    , ignore_index=True)
329
330     # reducing number of features for next
    iteration
331     i -= 1
332
333 # save the data as a csv file
334 train_metrics.to_csv("... [path to save
    training result]")
335 test_metrics.to_csv("... [path to save tst
    result]")
336 parameters.to_csv("... [path to save best
    hypreparametre]")
```

libraries_result.py

Bibliography

[1] M. Tovi, MR imaging in cerebral gliomas analysis of tumour tissue components, Acta Radiologica. Supplementum, vol. 384, pp. 1, 1993.

[2] S. Lapointe, A. Perry, N.A. Butowski, Primary brain tumours in adults, Lancet (London, England) 392 (10145) (2018) 432–446.

[3] M.E. Hegi, A.C. Diserens, T. Gorlia, M.F. Hamou, N. de Tribolet, M. Weller, J.M. Kros, J.A. Hainfellner, W. Mason, L. Mariani, J.E. Bromberg, P. Hau, R.O. Mirimanoff, J.G. Cairncross, R.C. Janzer, R. Stupp, MGMT gene silencing and benefit from temozolomide in glioblastoma, The New England journal of medicine 352 (10) (2005) 997–1003.

[4] E.H. Bell, P. Zhang, B.J. Fisher, D.R. Macdonald, J.P. McElroy, G.J. Lesser, J. Fleming, A.R. Chakraborty, Z. Liu, A.P. Becker, D. Fabian, K.D. Aldape, L.S. Ashby, M. Werner-Wasik, E.M. Walker, J.P. Bahary, Y. Kwok, H.M. Yu, N.N. Laack, C.J. Schultz, H.J. Gray, H.I. Robins, M.P. Mehta, A. Chakravarti, Association of MGMT Promoter Methylation Status With Survival Outcomes in Patients With High-Risk Glioma Treated With Radiotherapy and Temozolomide: An Analysis From the NRG Oncology/RTOG 0424 Trial, JAMA oncology (2018).

[5] W. Wick, C. Hartmann, C. Engel, M. Stoffels, J. Felsberg, F. Stockhammer, M.C. Sabel, S. Koeppen, R. Ketter, R. Meyermann, M. Rapp, C. Meisner, R.D. Kortmann, T. Pietsch, O.D. Wiestler, U. Ernemann, M. Bamberg, G. Reifenberger, A. von Deimling, M. Weller, NOA-04 randomized phase III trial of sequential radiochemotherapy of anaplastic glioma with procarbazine, lomustine, and vincristine or temozolomide, Journal of clinical oncology : official journal of the American Society of Clinical Oncology 27 (35) (2009) 5874–5880.

[6] T. Gorlia, M.J. van den Bent, M.E. Hegi, R.O. Mirimanoff, M. Weller, J.G. Cairncross, E. Eisenhauer, K. Belanger, A.A. Brandes, A. Allgeier, D. Lacombe, R. Stupp, Nomograms for predicting survival of patients with newly diagnosed glioblastoma: prognostic factor analysis of EORTC and NCIC trial 26981-22981/ CE.3, The Lancet. Oncology 9 (1) (2008) 29–38.

[7] Cui Y, Tha KK, Terasaka S, Yamaguchi S, Wang J, Kudo K, et al. Prognostic

imaging biomarkers in glioblastoma: development and independent validation on the basis of multiregion and quantitative analysis of MR images. Radiology 2016;278(2):546–53.

[8] Rios Velazquez E, Meier R, Dunn Jr W, et al. Fully automatic GBM segmentation in the TCGA-GBM dataset: prognosis and correlation with VASARI features. Sci Rep 2015;5:16822.

[9] Gutman DA, Dunn Jr WD, Grossmann P, et al. Somatic mutations associated with MRI-derived volumetric features in glioblastoma. Neuroradiology 2015;57(12):1227–37.

[10] Rios Velazquez E, Meier R, Dunn Jr W, et al. Fully automatic GBM segmentation in the TCGA-GBM dataset: prognosis and correlation with VASARI features. Sci Rep 2015;5:16822.

[11] Akbarzadeh, Omid. "Evaluating Latency in a 5G Infrastructure for Ultralow Latency Applications - Webthesis." webthesis.biblio.polito.it. www.webthesis.biblio.polito.it/id/eprint/22652.

[12] Keshavarz, S., R. Keshavarz, and A. Abdipour, "Compact active duplexer based on CSRR and interdigital loaded microstrip coupled lines for LTE application," Progress In Electromagnetics Research C, Vol. 109, 27-37, 2021. doi:10.2528/PIERC20112307

[13] Khosravi MR, Samadi S, Akbarzadeh O (2017) Determining the optimal range of angle tracking radars. In: IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI 2017), pp 3132–3135

[14] S. Keshavarz, H. M. Kadry and D. L. Sounas, "Four-port Spatiotemporally Modulated Circulator with Low Modulation Frequency," 2021 IEEE Texas Symposium on Wireless and Microwave Circuits and Systems (WMCS), 2021, pp. 1-4, doi: 10.1109/WMCS52222.2021.9493276.

[15] Hamzehei, Sahand. "Gateways and Wearable Tools for Monitoring Patient Movements in a Hospital Environment - Webthesis.", www.webthesis.biblio.polito.it/id/eprint/22711.

[16] www.med.upenn.edu/cbica/brats2021/

[17] Vasileios G. Kanas, Evangelia I. Zacharaki, Ginu A. Thomas, Pascal O. Zinn, Vasileios Megalooikonomou, Rivka R. Colen, Learning MRI-based classification models for MGMT methylation status prediction in glioblastoma, Computer Methods and Programs in Biomedicine, Volume 140, 2017, Pages 249-257, ISSN 0169-2607,

[18] Mahdie Jajroudi, Milad Enferadi, Amir Azar Homayoun, Reza Reiazi, MRI-based machine learning for determining quantitative and qualitative characteristics affecting the survival of glioblastoma multiforme, Magnetic Resonance Imaging, Volume 85, 2022, Pages 222-227, ISSN 0730-725X.

[19] M. Patel, J. Zhan, K. Natarajan, R. Flintham, N. Davies, P. Sanghera, J. Grist, V. Duddalwar, A. Peet, V. Sawlani, Machine learning-based radiomic

evaluation of treatment response prediction in glioblastoma, Clinical Radiology, Volume 76, Issue 8, 2021, Pages 628.e17-628.e27, ISSN 0009-9260.

[20] www.pyradiomics.readthedocs.io/en/latest/features.html

[21] Monique Meuschke, Laura A. Garrison, Noeska N. Smit, Benjamin Bach, Sarah Mittenentzwei, Veronika Weiß, Stefan Bruckner, Kai Lawonn, Bernhard Preim, Narrative medical visualization to communicate disease data,Computers & Graphics, Volume 107, 2022, Pages 144-157,ISSN 0097-8493.

[22] "Radiomics." Radiomics, www.radiomics.io/pyradiomics.html.

[23] Majeed Alneamy JS, A Hameed Alnaish Z, Mohd Hashim SZ, Hamed Alnaish RA. Utilizing hybrid functional fuzzy wavelet neural networks with a teaching learning-based optimization algorithm for medical disease diagnosis. Comput Biol Med. 2019;112:103348.

[24] Das DK, Dutta PK. Efficient automated detection of mitotic cells from breast histological images using deep convolution neutral network with wavelet decomposed patches. Comput Biol Med. 2019;104:29–42. doi: 10.1016/j.compbiomed.2018.11.001.

[25] Ganeshan B, Hosur A, Skogen K, Tasker F, Dizdarevic S, Miles KA. Multi-parametric FDG PET-CT in thoracic malignancies: opportunities for combined prognostic imaging biomarkers. Presented at: UK Radiological Congress 2012, Manchester, UK.

[26] R.O. Sinnott, H. Duan, Y. Sun, Chapter 15 - A Case Study in Big Data Analytics: Exploring Twitter Sentiment Analysis and the Weather, Editor(s): Rajkumar Buyya, Rodrigo N. Calheiros, Amir Vahid Dastjerdi, Big Data, Morgan Kaufmann, 2016, Pages 357-388.

[27] www.javatpoint.com/multi-layer-perceptron-in-tensorflow

[28] www.scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[29] Chen S, Xu Y, Ye M, Li Y, Sun Y, Liang J, Lu J, Wang Z, Zhu Z, Zhang X, Zhang B. Predicting MGMT Promoter Methylation in Diffuse Gliomas Using Deep Learning with Radiomics. J Clin Med. 2022 Jun 15;11(12):3445. doi: 10.3390/jcm11123445. PMID: 35743511; PMCID: PMC9224690.

[30] www.kaggle.com/code/juliojuse/single-model-0-747-2dcnn-inference

[31] www.kaggle.com/code/juliojuse/single-model-0-747-2dcnn-inference

[32] Li, Y., Wei, D., Liu, X., Fan, X., Wang, K., Li, S., Zhang, Z., Ma, K., Qian, T., Jiang, T., Zheng, Y. and Wang, Y., 2022. Molecular Subtyping of Diffuse Gliomas Using Magnetic Resonance Imaging: Comparison and Correlation Between Radiomics and Deep Learning. Beijing Neurosurgical Institute, Capital Medical University;7 No. 119 South Fourth Ring West Road;8 Beijing 100070, China;