**3 ML Features and RFE**

- GLCM (Gray-Level Co-occurrence Matrix): Analyses texture by calculating how often pairs of pixels

with specific values and in a specified spatial relationship occur in an image, useful in image

classification.

- LBP (Local Binary Patterns): A texture[function of spatial variation of brightness,spatial arrangement of colours or intensities, spatial distribution of intensity levels in a neighborhood] descriptor that summarizes local structures in images by comparing each pixel with its neighbourhood.

- HOG (Histogram of Oriented Gradients): Captures the edge directions and distributions in localized

portions of an image, commonly employed in human detection within images.

- RFE (Recursive Feature Elimination): A feature selection method that fits a estimator model and removes the weakest feature (or features) until the specified number of features is reached, enhancing model accuracy.

**6 ML Classifiers**

- Logistic Regression: A statistical model that in its basic form uses a logistic function to model a

binary dependent variable, widely used for binary classification tasks.

- Random Forest: An ensemble learning method for classification and regression that operates by

constructing multiple decision trees and outputting the mode of their predictions, providing

robustness.

- SVM (Support Vector Machine): A supervised learning model that uses SVM finds the optimal hyperplane that best separates data points belonging to different classes in a high-dimensional space; it's effective in high-dimensional spaces. SVM can handle linear and non-linear classification tasks by using different types of kernels (e.g., linear, polynomial, radial basis function) to map the input data into a higher-dimensional feature space where the classes are more easily separable.

- XGBoost (eXtreme Gradient Boosting): An optimized distributed gradient boosting library designed

to be highly efficient, flexible, and portable.

- Adaboost (Adaptive Boosting): A boosting algorithm that can be used with many types of data and

classifiers, it focuses on classification problems and aims to convert a set of weak classifiers into a

strong one. AdaBoost iteratively trains a sequence of weak classifiers, where each subsequent classifier focuses on the examples that were misclassified by the previous ones.

- Histogram based Grading Boosting: A type of Gradient Boosting that uses histograms to summarize the gradient information during the boosting process. The architecture of HGB involves constructing histograms over the feature space for each feature dimension. The histograms are built based on the gradient values of the loss function with respect to the feature values. These histograms represent the distribution of gradient information across different regions of the feature space

**6 CNN Models**

- VGG16: A deep convolutional network trained on ImageNet dataset for image recognition that proved that depth is a critical component for good performance. The deep convolutional layers of VGG models are effective at extracting hierarchical features from input images. As the layers progress, they capture increasingly abstract features. [13 convolutional layers, 3 maxpooling layers]

- VGG19: Similar to VGG16 but with three more convolutional layers, this network has more depth

which can lead to better feature learning. These pretrained models have learned rich feature representations from a diverse range of images, which can be fine-tuned or used as feature extractors for transfer learning tasks. This is particularly advantageous when working with limited datasets or when computational resources are limited.

[16 convolutional layers, 3 maxpooling layers]

- ResNet152V2: A deep residual network that uses skip connections or shortcuts to jump over some

layers, helps to solve the vanishing gradient problem and allows for very deep networks.

- InceptionV3: An advanced CNN with inception modules that apply convolutions of different sizes

simultaneously, improving computational efficiency and model performance on image data.

- MobileNet: MobileNet is a lightweight, less complex model based on depthwise separable convolutions which significantly reduce the number of parameters without losing performance.

- DenseNet121: A CNN where each layer is directly connected to every other layer in a feed-forward

fashion, strengthens feature propagation and encourages feature reuse, which significantly reduces

the number of parameters.

**Architecture breakdown**

VGG19: This is a convolutional neural network (CNN) architecture that consists of 19 layers, including convolutional layers and max-pooling layers[MaxPooling: is a down-sampling operation commonly used in convolutional neural networks (CNNs). MaxPooling operates on each feature map separately and reduces its size while retaining the most important information.]. It's commonly used for image classification tasks.

Flatten Layer: This layer is used to convert the multi-dimensional feature maps from the convolutional layers into a one-dimensional feature vector. This flattening step is necessary to connect the convolutional part of the CNN with the fully connected layers.

dense\_l (with Dropout): This appears to be a fully connected (dense) layer with a dropout regularization applied. Dropout is a technique used to prevent overfitting in neural networks by randomly setting a fraction of input units to zero during training.

dense\_2 (with Dropout): Another fully connected layer with dropout regularization.

dense\_3: Fully connected layer.

dense\_4 (SoftMax): The final fully connected layer with a softmax activation function. Softmax is commonly used in multi-class classification problems as it converts the output of the network into probability distributions over multiple classes, making it suitable for predicting the class probabilities.

Overfitting:

Overfitting occurs when a machine learning model learns the training data too well, to the extent that it captures noise or random fluctuations in the data as if they were genuine patterns.

Essentially, the model memorizes the training data instead of learning to generalize from it.

As a result, an overfitted model performs very well on the training data but poorly on unseen data (validation or test data).

Overfitting often happens when the model is too complex relative to the amount and quality of training data available.

Generalization:

Generalization refers to the ability of a machine learning model to perform well on unseen data.

A model that generalizes well is able to correctly predict outcomes for data it hasn't seen before, meaning it has learned the underlying patterns in the data rather than memorizing specific instances.

Generalization is the ultimate goal in machine learning because it indicates that the model has learned the true relationship between inputs and outputs and can apply that knowledge to new, unseen examples.

**Hyperparameters**

- Epochs: Refers to one cycle through the full training dataset, more epochs can mean better

learning, but also risk overfitting if not managed with techniques like early stopping.

- Loss: The objective that a model's training algorithm seeks to minimize, different loss functions can

be used depending on the nature of the problem, like mean squared error for regression.

Binary cross entropy - a loss function used in machine learning and deep learning to measure the difference between predicted binary outcomes and actual binary labels. It quantifies the dissimilarity between probability distributions, aiding model training by penalizing inaccurate predictions.

- Learning rate: A hyperparameter that controls how much to change the model in response to the

estimated error each time the model weights are updated, a critical parameter that can affect model

performance and training speed.

- Classifier: A type of machine learning algorithm that makes predictions or classifications based on

the input data, examples include logistic regression and neural networks.

Softmax - changes values into probabilities which is good for binary classes

- Train-test split: The process of dividing a dataset into subsets for training and testing, which allows

for the evaluation of a model on unseen data to assess its predictive performance.

- Augmented images: Enlarged dataset through transformations like rotation, translation, and flipping

of images, helps the model generalize better by simulating various perspectives and conditions.

**Estimator for RFE**

Linear Models:

Linear models, such as Linear Regression, Logistic Regression and SVC are often used with RFE. These models provide coefficients for each feature, making it easy to rank and select features based on their feature importance. [\_coef\_, \_feature*\_*importance\_]

Tree-Based Models:

Random Forests, and Gradient Boosted Trees are suitable for RFE because they inherently provide feature importances.

**Performance Metrics**

Accuracy:

Accuracy measures the overall correctness of the model by considering both true positives and true negatives. In your case, the model is correct about 89% of the time.

Precision:

Precision, also known as positive predictive value, is the ratio of correctly predicted positive observations to the total predicted positives. In your case, when the model predicts a positive class, it is correct 89% of the time.

Sensitivity (Recall) or True Positive Rate:

Sensitivity (or recall) is the ratio of correctly predicted positive observations to the all observations in the actual positive class. In your case, the model correctly identifies 89% of the actual positive instances.

Specificity:

Specificity is the ratio of correctly predicted negative observations to all observations in the actual negative class. In your case, the model correctly identifies 89% of the actual negative instances.

Confusion Matrix:

The confusion matrix provides a detailed breakdown of the model's predictions:

True Positive (TP): 34 instances

True Negative (TN): 34 instances

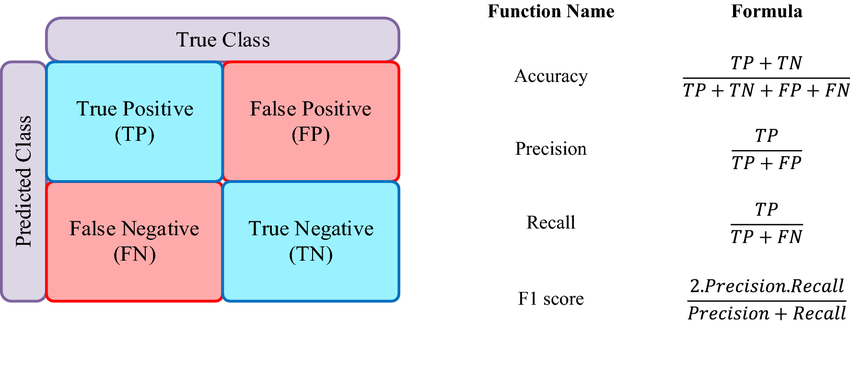
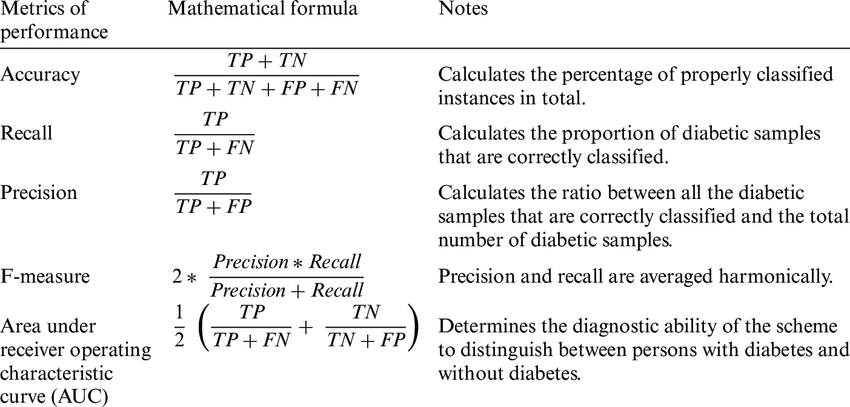
False Positive (FP): 4 instances

False Negative (FN): 4 instances

F1 Score:

The F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall. In your case, it is 89%.

AUC Score:

The AUC (Area Under the ROC Curve) score measures the area under the Receiver Operating Characteristic (ROC) curve. It provides an aggregate measure of the model's ability to discriminate between the positive and negative classes. A score of 0.96 indicates good performance.

**Code explanation**

1. this code segment

1. Importing necessary libraries:

- `numpy` as `np`: For numerical operations on arrays.

- `cv2`: For image processing using OpenCV.

- `os`: For operating system related operations, like directory listing.

- `tensorflow`: For deep learning functionalities.

- `ImageDataGenerator`, `preprocess\_input` from `tensorflow.keras.preprocessing.image`: For image data preprocessing.

- `drive` from `google.colab`: For mounting Google Drive.

2. Mounting Google Drive:

- Mounts Google Drive to access files.

3. Load dataset and preprocess:

- Define paths to directories containing training and testing images for healthy and schizophrenia affected brains.

4. Function `load\_dataset`:

- Takes four arguments: paths to healthy and schizophrenia directories for training and testing.

- Initializes empty lists to store images and labels for training and testing.

- Loops through images in the healthy and schizophrenia directories for training and testing:

- Loads each image using `load\_img` from `tensorflow.keras.preprocessing.image` and resizes it to

(224, 224) pixels.

- Converts the image to an array using `img\_to\_array`.

- Appends the image array to the respective `images\_train` or `images\_test` list.

- Appends the label (0 for healthy, 1 for schizophrenia) to the respective `labels\_train` or `labels\_test` list.

- Returns numpy arrays for images and labels for training and testing.

5. Calling that function `load\_dataset` with paths to load the dataset.

This code sets up the dataset for a deep learning model by loading images from specified directories,

resizing them, and storing them as arrays along with their labels. The dataset is then used for training and testing a model to classify healthy and schizophrenia-affected brains.

2. This code segment prepares the dataset for training a deep learning model using VGG19 architecture for image classification. Here's a breakdown of the code:

1. Importing necessary libraries:

- `os`: For operating system related operations.

- `time`: For measuring time taken for training.

- `numpy` as `np`: For numerical operations on arrays.

- `tensorflow`: For deep learning functionalities.

- `layers` from `tensorflow.keras`: For building neural network layers.

- `ImageDataGenerator`, `preprocess\_input` from `tensorflow.keras.preprocessing.image`: For image data preprocessing.

- `VGG19` from `tensorflow.keras.applications.vgg19`: Pre-trained VGG19 model.

- `ModelCheckpoint’ from `tensorflow.keras.callbacks`: For model callbacks.

- `Model`, `Sequential` from `tensorflow.keras.models`: For defining the model architecture.

- `train\_test\_split` from `sklearn.model\_selection`: For splitting the dataset into training and testing

sets.

- Various metrics from `sklearn.metrics`: For evaluating the model.

- `to\_categorical` from `tensorflow.keras.utils`: For converting labels to categorical format.

- `matplotlib.pyplot` as `plt`: For plotting graphs.

2. Splitting the dataset:

- `X\_train\_dl\_split` and `y\_train\_dl\_split` are the training images and labels, respectively.

- `X\_test\_dl\_split` and `y\_test\_dl\_split` are the testing images and labels, respectively.

- `to\_categorical` is used to convert the labels to a categorical format.

This code sets up the dataset for training a deep learning model using VGG19 architecture. The dataset is split into training and testing sets, and labels are converted to a categorical format. The model can now be defined and trained using this prepared dataset.

3. This code segment builds and compiles a deep learning model using the VGG19 architecture for image classification. Here's a breakdown of the code:

1. `print\_layer\_trainable` function:

- Iterates through all layers in the base model (VGG19) and prints whether each layer is trainable or not, along with its name.

2. Build and compile the model:

- `base\_model`: Loads the pre-trained VGG19 model with weights from ImageNet, excluding the top (classification) layers, and sets the input shape to (224, 224, 3).

- Freezes all layers in the base model (sets `trainable` attribute to False).

- Unfreezes the last 15 layers for fine-tuning by setting their `trainable` attribute to True.

- `model`: Defines the full model architecture using a Sequential model.

- Rescales input images to the range [0, 1].

- Appends the base model (VGG19) to the model.

- Adds a Flatten layer to flatten the output from the base model.

- Adds Dense layers with ReLU activation and Dropout for regularization.

- Adds a Dense output layer with softmax activation for binary classification.

- Compiles the model using the Adam optimizer, binary crossentropy loss, and accuracy metric.

3. Model Checkpoint:

- `tl\_checkpoint\_h5` and `tl\_checkpoint\_tf` are ModelCheckpoint callbacks that monitor the

validation accuracy and save the best weights to files in HDF5 and TensorFlow formats, respectively.

This code segment sets up a deep learning model for binary image classification using transfer learning with VGG19. The model is compiled and ready for training.

4. This code segment sets up ImageDataGenerator instances for data augmentation and creates generators for training and validation data. Here's a breakdown of the code:

1. `train\_datagen`:

- Creates an `ImageDataGenerator` object for data augmentation.

- Specifies various augmentation options:

- `rotation\_range`: Degree range for random rotations.

- `width\_shift\_range` and `height\_shift\_range`: Fraction of total width or height for random horizontal or vertical shifts.

- `shear\_range`: Shear intensity in radians.

- `zoom\_range`: Range for random zoom.

- `horizontal\_flip`: Randomly flips images horizontally.

- `fill\_mode`: Strategy for filling in newly created pixels.

2. `train\_test\_split`:

- Splits the original training data (`X\_train\_dl\_split`, `y\_train\_dl\_split`) into training and validation sets

(`X\_train`, `X\_val\_dl\_split`, `y\_train`, `y\_val\_dl\_split`) using a 70-30 split ratio.

3. `train\_generator` and `val\_generator`:

- Create generator objects using the `flow` method of `train\_datagen`.

- `train\_generator` generates augmented images and labels for training.

- `val\_generator` generates augmented images and labels for validation.

These generators will be used to train the model with augmented images, which can improve the model's performance and generalization.

5. This code segment calculates the confusion matrix, specificity, and recall, and then plots the training

history (accuracy and validation accuracy) of the model. Here's a breakdown of the code:

1. Calculating confusion matrix, specificity, and recall:

- `y\_pred\_classes`: Predicted classes obtained by taking the argmax of `y\_pred\_dl` along axis 1.

- `y\_true\_classes`: True classes obtained by taking the argmax of `y\_test\_dl\_split` along axis 1.

- `conf\_matrix`: Confusion matrix computed using `confusion\_matrix` from `sklearn.metrics`.

- `specificity`: Calculated as the true negative rate (TN / (TN + FP)).

- `recall`: Calculated as the true positive rate or sensitivity (TP / (TP + FN)).

2. Plotting training history:

- Creates a figure with a size of (12, 8).

- Plots training accuracy and validation accuracy against epochs.

- Labels the x-axis as 'Epochs' and the y-axis as 'Accuracy'.

- Sets the title of the plot as 'Accuracy vs. Epochs'.

- Adds a legend to differentiate between training and validation accuracy.

These visualizations and metrics help in understanding the performance and behavior of the model

during training and evaluation.