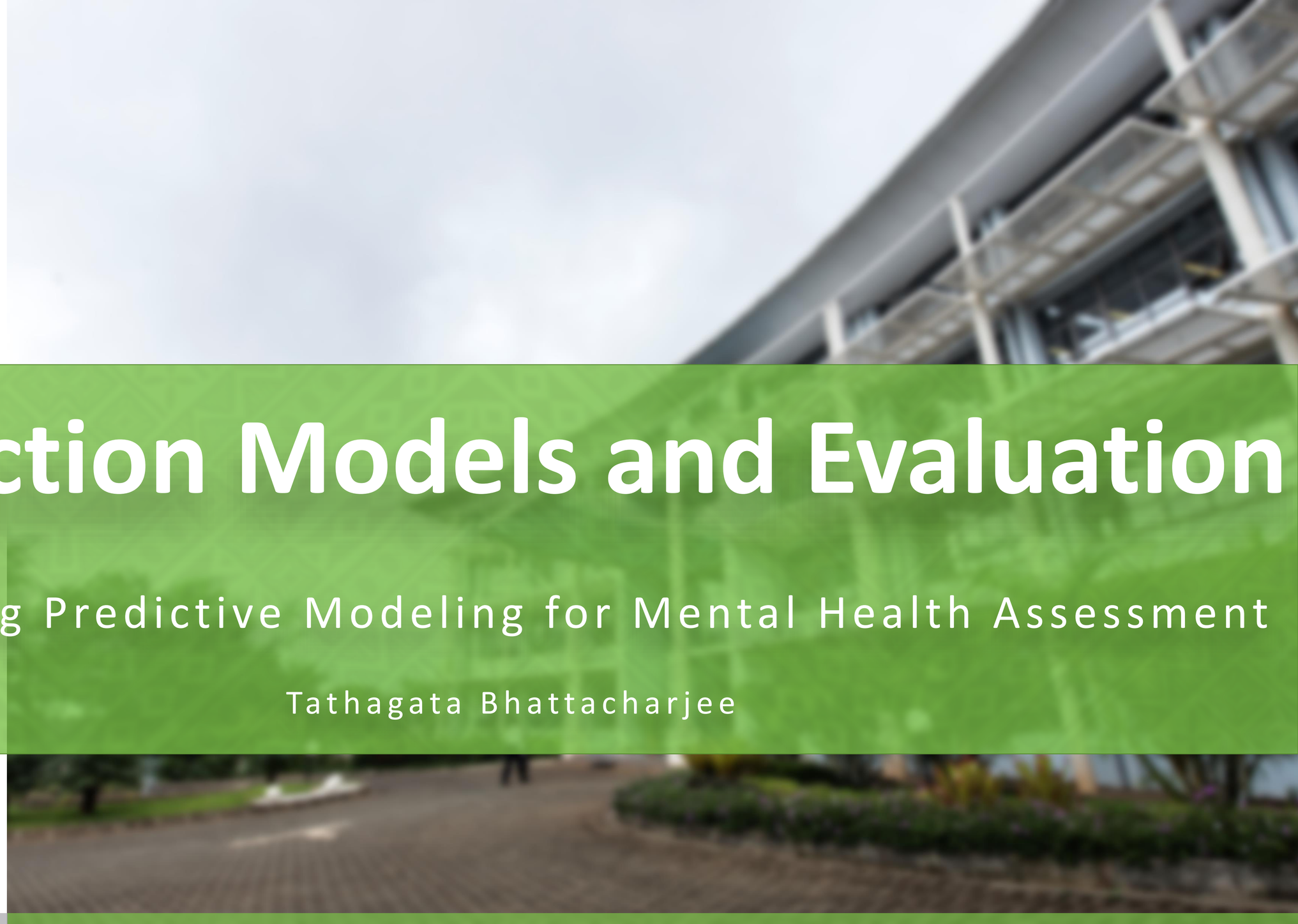
African Population and Health Research Center

Transforming lives in Africa through research.



Prediction Models and Evaluation

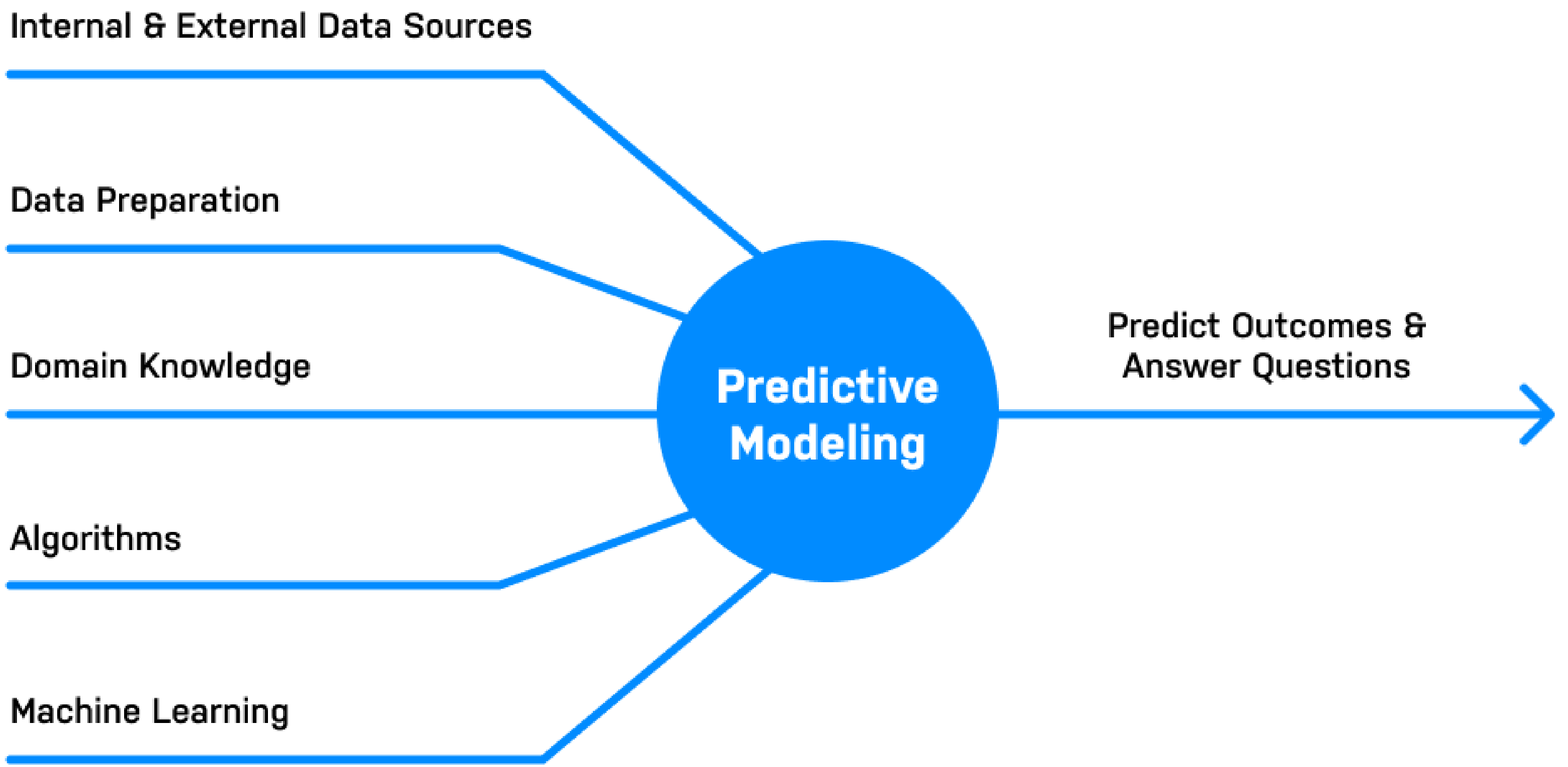
Leveraging Predictive Modeling for Mental Health Assessment

Tathagata Bhattacharjee

Contents

Predictive Model

- Prediction model or a predictive model, is a mathematical algorithm or statistical technique that is trained on historical data to make predictions about future or unseen data
- Widely used in fields such as finance, healthcare, marketing, manufacturing, etc.
- Plays a crucial role in decision-making processes, risk assessment, resource allocation, and optimization based on data-driven insights
- Key characteristics of prediction models:
  1) **Learning from Data:** Prediction models learn patterns and relationships from historical data through training. It analyzes features (input variables) and their corresponding target variable (the variable to be predicted) to understand their relation
  2) **Generalization:** Once trained, prediction models can generalize their learned patterns to make predictions on new data.
  3) **Types of Predictions:** Prediction models can be used for various types of predictions
     a) Classification (assigning categories or labels to input data)
     b) Regression (predicting numerical values)
     c) Clustering (grouping similar data points together)
     d) Time-series forecasting (predicting future values based on historical trends)
  4) **Evaluation:** Prediction models are evaluated based on their predictive performance using metrics such as accuracy, precision, recall, F1-score, mean squared error (MSE), etc.
  5) **Iterative Improvement:** Prediction models often undergo iterative improvement processes, where they are trained, evaluated, and refined multiple times to enhance their predictive accuracy and generalization capabilities
  6) **Deployment:** Once a prediction model has been trained and validated, it can be deployed in real-world applications to make predictions on new data


Internal & External Data Sources
Data Preparation
Domain Knowledge
Algorithms
Machine Learning
Predictive Modeling
Predict Outcomes & Answer Questions

Machine Learning

Machine learning (ML) is a subset of artificial intelligence (AI)

It focuses on developing algorithms and statistical models that enable computers to learn from and make predictions or decisions based on data without being explicitly programmed

Learning from Data: ML algorithms learn patterns and relationships from large amounts of data. The more data they are exposed to, the better they can learn and generalize to new data

Types of Learning:

Supervised Learning: Algorithms learn from labeled data, where each example is paired with the correct output. It involves predicting an output variable from input variables.

Unsupervised Learning: Algorithms learn from unlabeled data, finding hidden patterns or structures within the data.

Reinforcement Learning: Learn to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties.

Model Representation: ML models can be represented in various forms, including decision trees, neural networks, support vector machines, k-nearest neighbors, and more

Training and Evaluation: ML models are trained on a portion of the data called the **training set**. They are then evaluated on another portion of the data called the test set to assess their performance. Evaluation metrics include accuracy, precision, recall, F1-score, and mean squared error, among others.

Challenges in Machine Learning

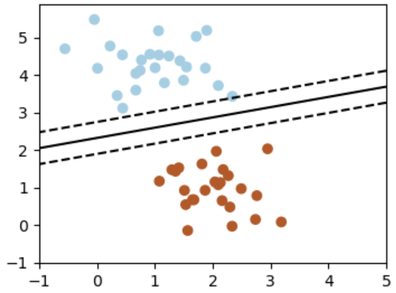
Data Quality: ML models heavily rely on data quality. Poor quality or biased data can lead to inaccurate predictions or biased outcomes

Interpretability: Some ML models, especially deep learning models, are often considered black boxes, making it challenging to interpret their decisions

Scalability: Training and deploying ML models at scale can be computationally expensive and require specialized infrastructure

Ethical Considerations: ML models may perpetuate or amplify biases present in the data. Ensuring fairness, transparency, and accountability in ML systems is crucial.

Classification



It is the process of arranging things in groups or classes according to their resemblances

It is a supervised learning method where the goal is to categorize input data points into one of several predefined classes or categories based on their features

PROCESS:

Data Collection: Gather labeled data where each data point is associated with a class label indicating its category.

Data Preprocessing: Clean and preprocess the data, including handling missing values, encoding categorical variables, and scaling numerical features.

Model Selection: Choose a classification algorithm suitable for the problem, such as logistic regression, decision trees, random forests, support vector machines, or neural networks.

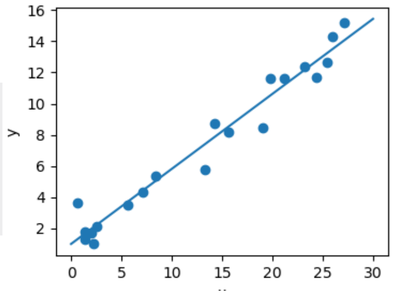
Model Training: Train the selected model on the labeled training data to learn the patterns and relationships between features and class labels.

Model Evaluation: Evaluate the trained model's performance using metrics like accuracy, precision, recall, F1-score, and confusion matrix on a separate test dataset.

Model Tuning: Fine-tune the model parameters or try different algorithms to improve performance.

Prediction: Use the trained model to make predictions on new, unseen data by feeding input features and obtaining predicted class labels.

Regression



It is a statistical technique that relates a dependent variable to one or more independent (explanatory) variables

It can show whether changes observed in the dependent variable are associated with changes in one or more of the explanatory variables

Types of Regression:

Linear Regression: The simplest form of regression, where the relationship between the variables is assumed to be linear. Example: The weight of the person is linearly related to their height. So, a linear relationship between the height and weight of the person. Accordingly, as we increase the height, the weight of the person will also increase.

Polynomial Regression: Extends linear regression to capture non-linear relationships. Is one of the ML algorithms for making predictions. Example: Used to predict the spread rate of infectious diseases

Multiple Regression: Considers multiple independent variables to predict the dependent variable. The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear. Example: To predict the price of a house based on its size, number of bedrooms, location, and other factors

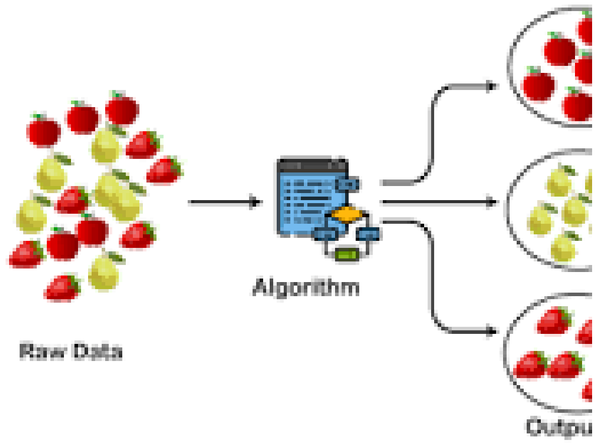
Logistic Regression: Used for binary classification tasks, predicting the probability of an event occurring in relationships between two data factors. Example: Predicting if an incoming email is spam or not spam, or predicting if a credit card transaction is fraudulent or not fraudulent

Clustering

Clustering is an unsupervised learning technique used to group similar data points into clusters based on their features

The aim is to partition the data such that data points within the same cluster are more similar to each other than to those in other clusters

Example: Group together patients with high glucose levels and more age, potentially indicating a more severe progression of diabetes that will require insulin therapy



Time-series forecasting

Time-series forecasting is a technique used to predict future values based on historical time-ordered data points

It involves analyzing patterns, trends, and seasonality within the time series to make accurate predictions about future values

Example: Data on patient admissions and readmissions, including timestamps, patient demographics, medical conditions, treatments, and outcomes can help forecast future readmission rates for different time-periods, and help identify periods with predicted high readmission rates and implement preventive measures, such as care coordination, patient education, and follow-up interventions. Will also help in resource allocation strategically to address potential areas of concern and improve patient outcomes

scikit-learn

scikit-learn: Simplifying Machine Learning in Python

It is a powerful and user-friendly machine-learning library in Python that provides a wide range of tools for building and deploying machine-learning models

scikit-learn and sklearn refer to the same machine-learning library in Python

The name "scikit-learn" is the official name of the library, but it is commonly abbreviated as "sklearn" for convenience in code

scikit-learn seamlessly integrates with other popular Python libraries such as NumPy, SciPy, Pandas, and Matplotlib, enabling to leverage their functionalities for data manipulation, scientific computing, and visualization

A Python Program to Build a Prediction Model

```
# Importing necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.metrics import classification_report, accuracy_score

from sklearn.metrics import confusion_matrix
```

This code snippet imports necessary libraries from scikit-learn and pandas and sets up an environment for building and evaluating a machine learning model using a RandomForestClassifier for classification tasks

1. **pandas (imported as pd):** Used for data manipulation and analysis.
2. **train_test_split from sklearn.model_selection:** Used to split the dataset into training and testing subsets.
3. **RandomForestClassifier from sklearn.ensemble:** Random Forest classifier model from scikit-learn, which is an ensemble learning method used for classification.
4. **OneHotEncoder from sklearn.preprocessing:** Used for one-hot encoding categorical features.
5. **ColumnTransformer from sklearn.compose:** Used to apply transformers to columns of an array or pandas DataFrame.
6. **SimpleImputer from sklearn.impute:** Used to handle missing values in the dataset.
7. **classification_report and accuracy_score from sklearn.metrics:** Used to evaluate the performance of the classifier.
8. **confusion_matrix from sklearn.metrics:** Used to evaluate the performance of the classifier by calculating the confusion matrix.

Load Dataset

mh_data = pd.read_csv("synthetic_depression_data_training.csv")

pd.read_csv: This is a function from the pandas library (imported as pd). It is used to read data from a CSV (Comma Separated Values) file and create a DataFrame, which is a tabular data structure in pandas.

"synthetic_depression_data_training.csv": This is the filename of the CSV file containing the dataset. The code assumes that the CSV file is located in the current directory. If the file is located in a different directory, you need to provide the full or relative path to the file.

mh_data =: This part of the code assigns the DataFrame created by pd.read_csv to the variable data. This allows you to access and manipulate the dataset using the variable name data.

Selecting relevant features and target variable

features = ['age', 'sex', 'income_t1', 'employment_t1', 'stress_t1', 'mh_score_t1']

target = 'depression_t1'

X = mh_data[features]

y = mh_data[target]

features = [...]: This line defines a list of feature names. These are the variables/columns from the dataset that will be used as input to the machine learning model. In this case, the features include 'age', 'sex', 'income_t1', 'employment_t1', 'stress_t1', and 'mh_score_t1'.

target = 'depression_t1': This line specifies the target variable. The target variable is the variable/column that the model will attempt to predict. In this case, the target variable is 'depression_t1'.

X = mh_data[features]: This line creates a new DataFrame X containing only the columns specified in the features list. It selects the features from the original dataset mh_data.

y = mh_data[target]: This line creates a new Series y containing the values of the target variable. It selects the target variable from the original dataset mh_data.

Handling missing values (if any)

imputer = SimpleImputer(strategy='most_frequent')

X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

imputer = SimpleImputer(strategy='most_frequent'): This line creates an instance of the SimpleImputer class from scikit-learn, which is used to handle missing values in the data. The strategy='most_frequent' argument specifies that missing values should be imputed using the most frequent value along each column.

X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns): This line applies the imputation process to the DataFrame X, which contains the input features. The fit_transform method of the SimpleImputer object fits the imputer to the data (calculates the most frequent value for each column) and transforms the data by replacing missing values with the most frequent value in each column. The resulting array is then converted back into a DataFrame using pd.DataFrame. The columns=X.columns argument ensures that the column names of the original DataFrame are retained in the new DataFrame X.

Label Encoding for categorical variables

label_encoder = LabelEncoder()

X['sex'] = label_encoder.fit_transform(X['sex'])

X['employment_t1'] = label_encoder.fit_transform(X['employment_t1'])

X['stress_t1'] = label_encoder.fit_transform(X['stress_t1’])

label_encoder = LabelEncoder(): This line creates an instance of the LabelEncoder class from scikit-learn. LabelEncoder is used to encode categorical variables as integer labels.

X['sex'] = label_encoder.fit_transform(X['sex']): This line encodes the categorical variable 'sex' in the DataFrame X using the fit_transform method of the LabelEncoder object. The fit_transform method fits the label encoder to the unique categories in the 'sex' column and transforms the categories into integer labels. These integer labels are then assigned back to the 'sex' column in the DataFrame X.

X['employment_t1'] = label_encoder.fit_transform(X['employment_t1']): Similarly, this line encodes the categorical variable 'employment_t1' in the DataFrame X using the same label encoder object.

X['stress_t1'] = label_encoder.fit_transform(X['stress_t1']): Likewise, this line encodes the categorical variable 'stress_t1' in the DataFrame X using the same label encoder object.

Splitting the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

This code snippet splits the dataset into training and testing sets

train_test_split: This is a function from scikit-learn's model_selection module. It is used to split a dataset into training and testing subsets.

X: This represents the input features (independent variables) of the dataset. It is typically a DataFrame or an array-like object.

y: This represents the target variable (dependent variable) of the dataset. It is typically a Series or an array-like object.

test_size=0.2: This parameter specifies the proportion of the dataset to include in the testing split. In this case, it is set to 0.2, which means 20% of the data will be used for testing, and the remaining 80% will be used for training.

random_state=42: This parameter sets the random seed for reproducibility. It ensures that the data is split in the same way each time the code is run, which is useful for obtaining consistent results.

X_train, X_test, y_train, y_test: These are the outputs of the train_test_split function. They represent the training and testing subsets of the input features (X) and target variable (y). X_train and y_train contain the training data, while X_test and y_test contain the testing data.

Training the Random Forest Classifier

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)

RandomForestClassifier: RandomForestClassifier is a class from scikit-learn's ensemble module. It implements a random forest classifier, which is an ensemble learning method that builds multiple decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

n_estimators=100: This parameter specifies the number of trees in the random forest. In this case, we're setting it to 100, meaning that 100 decision trees will be built during the training process.

random_state=42: This parameter sets the random seed for reproducibility. It ensures that the random initialization of the model's parameters is the same each time the code is run, which helps in obtaining consistent results.

model.fit(X_train, y_train): This line of code trains the random forest classifier model using the fit method. It takes two main arguments:

X_train: The input features (independent variables) of the training dataset.

y_train: The target variable (dependent variable) of the training dataset.

The **fit** method fits the model to the training data by learning patterns from the input features (X_train) and their corresponding target labels (y_train). The model adjusts its parameters (i.e., the decision trees) to minimize the prediction error.

Making predictions on the test set

y_pred = model.predict(X_test)

model.predict(X_test): This line of code uses the predict method of the trained model (model) to make predictions on the test dataset (X_test).

X_test: This represents the input features (independent variables) of the test dataset. It contains the data that the model hasn't seen during training, and we want to evaluate the model's performance on this unseen data.

y_pred: This is the variable where the predicted values are stored. After calling model.predict(X_test), y_pred will contain the predicted target labels (or values) corresponding to the input features in X_test.

Evaluating the model: Accuracy

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)

accuracy_score(y_test, y_pred): This function computes the accuracy of the model's predictions by comparing the predicted values (y_pred) with the true target values (y_test). It returns the accuracy score, which is the proportion of correctly predicted samples over the total number of samples.

The accuracy score indicates how often the model's predictions match the actual labels in the test set.

Evaluating the model: Classification Report

print("Classification Report:")

print(classification_report(y_test, y_pred))

classification_report(y_test, y_pred): This function generates a comprehensive classification report that includes various evaluation metrics such as precision, recall, F1-score, and support for each class.

The classification report provides insights into the model's performance across different classes, making it useful for multi-class classification tasks.

Precision

Precision:

Precision measures the proportion of true positive predictions (correctly predicted positive instances) among all instances predicted as positive by the model.

It is calculated as the ratio of true positives to the sum of true positives and false positives.

Precision quantifies the model's ability to avoid false positives, indicating how reliable positive predictions are.

Mathematically, precision is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall (Sensitivity):

Recall measures the proportion of true positive predictions among all actual positive instances in the dataset.

It is calculated as the ratio of true positives to the sum of true positives and false negatives.

Recall quantifies the model's ability to identify all positive instances, indicating how sensitive the model is to detecting positives.

Mathematically, recall is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1-Score

F1-score is the harmonic mean of precision and recall.

It provides a balance between precision and recall, considering both false positives and false negatives.

F1-score reaches its best value at 1 (perfect precision and recall) and worst at 0.

F1-score is a useful metric when there is an imbalance between the classes in the dataset.

Mathematically, F1-score is defined as:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Support

Support refers to the number of occurrences of each class in the true target labels.

It indicates the number of samples in the test set that belong to each class.

Support provides context for interpreting precision, recall, and F1-score, especially in the case of imbalanced datasets.

Summary:

precision measures the model's ability to make correct positive predictions,

recall measures its ability to identify all positive instances

F1-score combines both precision and recall into a single metric

Support provides the frequency of each class in the dataset, aiding in the interpretation of other metrics, particularly in imbalanced classification problems

Evaluating the model: Confusion Matrix

print("Confusion Matrix:")

print(confusion_matrix(y_test, y_pred))

The confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It presents a summary of the predictions made by the model versus the actual labels in a tabular format.

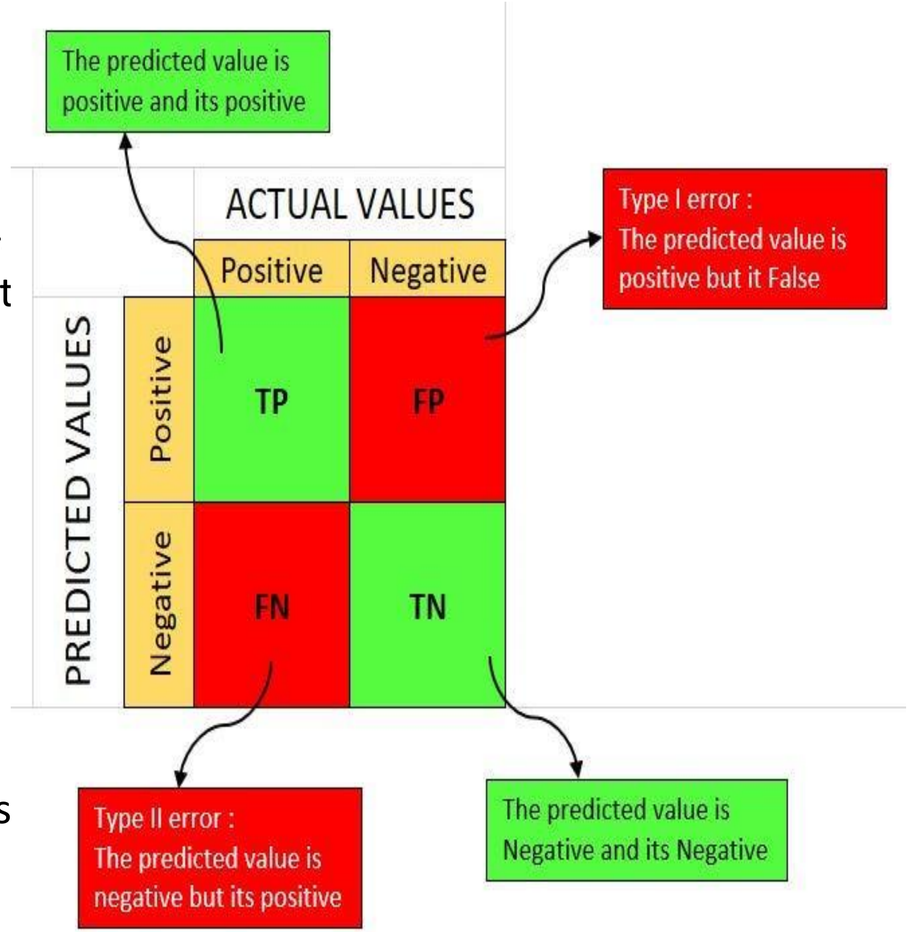
In the context of the confusion matrix:

True Positives (TP): These are the instances that were correctly predicted as positive by the model.

True Negatives (TN): These are the instances that were correctly predicted as negative by the model.

False Positives (FP): These are the instances that were incorrectly predicted as positive by the model (i.e., the model predicted positive, but the actual label was negative).

False Negatives (FN): These are the instances that were incorrectly predicted as negative by the model (i.e., the model predicted negative, but the actual label was positive).



Write predictions to a CSV file

```
predictions_df = pd.DataFrame({'Actual_Depression': y_test, 'Predicted_Depression': y_pred})
predictions_df.to_csv('predictions.csv', index=False)
print("Predictions written to 'predictions.csv' file successfully.")
```

This code segment is responsible for creating a DataFrame containing the actual depression values (y_test) and the corresponding predicted depression values (y_pred). It then saves this DataFrame to a CSV file named 'predictions.csv'. Finally, it prints a confirmation message indicating that the predictions have been successfully written to the CSV file.

predictions_df = pd.DataFrame({'Actual_Depression': y_test, 'Predicted_Depression': y_pred}): This line creates a DataFrame called predictions_df with two columns: 'Actual_Depression' and 'Predicted_Depression'. The 'Actual_Depression' column contains the true depression values from the test set (y_test), while the 'Predicted_Depression' column contains the depression values predicted by the model (y_pred).

predictions_df.to_csv('predictions.csv', index=False): This line saves the predictions_df DataFrame to a CSV file named 'predictions.csv'. The index=False parameter ensures that the index of the DataFrame is not included in the CSV file.

African Population and
Health Research Center

Transforming lives in Africa through research.

THANK YOU

@aphrc

www.aphrc.org