**Multi-Disease Prediction System using Machine Learning**

**1. Abstract:**

The healthcare industry is currently experiencing a significant imbalance, with a high patient-to-doctor ratio, posing challenges to effective patient care. In the age of healthcare digitization, using machine learning (ML) to forecast diseases has become essential for accurately identifying and effectively treating a variety of medical disorders. This paper examines how to use Django and machine learning to build a reliable multi-disease prediction system for diseases such as stroke, depression in students, and diabetes. Based on user input features, the system is trained to categorize and forecast the likelihood of various diseases using real-world medical datasets. Both healthcare experts and non-technical individuals can profit from the system thanks to the Django framework's smooth user interface. The paper also discusses crucial procedures including data preprocessing, model selection, training, and deployment, emphasizing how crucial recall and dependability are for predictive health solutions. By offering prompt medical action, it seeks to increase awareness of how these technologies can transform early diagnosis, lower healthcare costs, and ultimately improve patient outcomes.

**2. Introduction:**

Artificial intelligence (AI) and machine learning (ML) have become revolutionary technologies in today's quickly changing healthcare environment, with the potential to greatly enhance patient care and medical diagnostics. Disease prediction is one of the most exciting and important uses of machine learning in healthcare, where algorithms are trained to identify trends in medical data and forecast the probability of different illnesses. Different ML models can be trained on a disease dataset and it could be used to predict the probability of a person getting that disease, eliminating the need of a doctor.

Stroke, Diabetes, and Depression in students are among the leading health challenges globally. Early detection and prevention are vital to reduce morbidity and mortality. Building strong, dependable, and intuitive systems that can manage several illnesses at once is the difficult part, though. This is where machine learning-powered multi-disease prediction systems developed using Django are useful.

The best platform for implementing machine learning models and giving users an easy-to-use interface for making predictions is Django, a robust Python web framework. We can construct an application that enables people and healthcare practitioners to make better decisions regarding health risks by fusing the predictive power of machine learning with the user-friendly web development platform Django.

The Multidisease Prediction System demonstrates the potential of leveraging machine learning to address critical health challenges. The modular nature of the system allows scalability to include additional diseases in the future. From data collection and model training to deployment, this paper will examine the steps involved in developing a multi-disease prediction system and the possible benefits it could offer in terms of bettering early detection, improving healthcare outcomes, and lowering treatment costs. The technological challenges of developing this system, the potential of machine learning in healthcare, and the concern to raise awareness among people about these diseases will be covered in subsequent parts.

This project also includes awareness articles about the diseases the models are being made for. These blogs/articles include information about how to identify those diseases, what are the symptoms, what are the preventive measures for that disease, and if a person has already contracted the disease, how to keep it in control and prevent further damage to oneself.

**3. Literature Survey**

The usage of Machine Learning to predict diseases has brought upon a great deal of improvement in the healthcare industry, considering today’s really high patient-to-doctor ratio. Although many previous papers have discussed upon this topic of developing and using ML models to predict diseases, many of them just talked about how the models are trained for different diseases, and not how a user can access them. And those papers which did talk about a user interface like a web application, there seems to be a lack of usage of better libraries and frameworks to develop those web applications. This literature survey reviews existing research on disease prediction using ML, more specifically focused on diseases like stroke, diabetes, and student depression. This section also highlights the gaps in previous researches and how they can be overcome in this paper.

The method proposed by K. Gaurav et al. [1] includes the usage of various algorithms and models like Random Forest (RF), Support Vector Machine (SVM), and Long Short-Term Memory (LSTM), etc. but a particular disease on which these models were developed and tested on is not discussed. And this particular paper [1] does not also discuss about how an end-user will be able to use the model and interact with, for example an interactive web application. The gaps in this paper [1] can be filled by tailoring models to particular disease datasets, and developing an interactive web interface for the ease of user to access the models.

The second method proposed in the paper by K. Reshma et al. [2] shows a multi-disease prediction system and it employs the usage of a web application developed using the streamlit library, allowing the user to access multiple diseases on the same web application. The models which have been discussed in this paper include Support Vector Classifier (SVC), and Logistic Regression. The diseases for which these models have been trained are Diabetes (uses the SVC model), Heart Disease (uses the Logistic Regression model), and Parkinson’s disease (uses the SVC model) with accuracies of 79%, 85%, and 89% respectively. Although the current paper doesn’t include models for Heart Disease and Parkinson’s Disease, the model for Diabetes which is discussed in the paper [2] though an improvement from the previous models made for the same disease, this still could use some improvement as its accuracy is just 79%.

The method discussed in the paper by Sathya Sundaram M. et al. [3] developed a model for stroke prediction. In this paper [3] various models such as Random Forest, Decision Tree, and Logistic Regression have been trained and tested upon the stroke dataset and Decision Tree has been chosen as the final model which shows an accuracy of 95.5%. Although the performance of this model is pretty good, the first gap in this research papers is that the user interface has been developed using tkinter which may provide a clean interface, but cannot be deployed and thus does not allow for people from all over the world to access it. This can be solved by developing a web application for the user interface which can be deployed on the cloud, allowing for a wide range of users from different parts of the world to access the ML model. The application discussed in the paper [3] can be further improved by including multiple models for various diseases, essentially making it a multi-disease prediction system.

Just like the paper by K. Gaurav et al. [1], this paper by Dr. Nitish Das et al. [4] Developed ML models on the dataset from Kaggle which has 41 diseases as the target, i.e., the models are trained on the dataset so that they predict one among the 41 unique possible diseases in the given dataset. The problem with this kind of system [4] is that its lack of ease of use by the user, in the way that it requires the user to enter 132 input values to get a prognosis. And another problem with the model trained on this dataset would be that the model outputs any one of the 41 possible target label, meaning that if a user is absolutely healthy, the model is likely to falsely show that the user may have a disease. Although the model is really accurate, it still is not tailored to predicting one disease (binary classification). Models trained to predict specific diseases would have the chance to see and learn from datasets with more number of records, whereas the model in the paper [4] was trained on just over 4,900 records.

The next paper by Kumar Bibhuti B. Singh et al. [5] discuss a multi-disease prediction system for the diseases such as Heart Disease, Parkinson’s Disease, and Diabetes. Multiple models have been trained upon the datasets for these diseases. Although Heart Disease and Parkinson’s disease are out of the scope for this paper, the models tested to predict Diabetes could use some improvement as none of the tested models reached over 80% accuracy. Plus the diabetes model does require the user to visit a doctor or at least a medical professional to get some of the input features such as Insulin, etc. and does not let the user to self-diagnose.

The research paper by S. Yadav et al. [6] discussed models for diseases like Breast Cancer, Asthma, Brain Cancer, Liver Disease, Heart Disease, Infection, Diabetes, Kidney Disease, Lung Cancer, Alzheimer’s, Anemia, Thyroid, Parkinson’s, Alopecia, Gastric Cancer, and COVID-19. This paper [6] compiled the work done by various authors and compared the models developed by them. It has no mention of how the user interacts with and accesses the models, for example a web application.

Another improvement which could be made to the applications made in the aforementioned papers is the addition of awareness articles about the mentioned diseases in respective research papers into the applications/user interfaces, which the user can read and gain useful knowledge on how to identify the disease, how to prevent the disease, etc.

**4. Methodology**

This study discusses how a system has been built using ML to predict stroke, depression in students, and diabetes. The methodology explains in detail the steps taken to design and implement a reliable and user-friendly system. Various key tools and techniques were employed to make this possible, and to make sure that the system delivers accurate and meaningful results. The stages in the development of this system are as follows:

**4.1: Data Collection**

The datasets used in this system to train and test the predictive Machine Learning models were sourced from Kaggle.

The dataset used for the first disease, i.e., the stroke prediction dataset [8] is sourced from Kaggle and it is available in a .csv file format. The size of this file is about 320 KB. This dataset has 5,110 records and it has 12 columns namely:

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Column Name** | **Description** |
| 1. | id | Unique identifier |
| 2. | gender | “Male” or “Female” |
| 3. | age | Age of the patient |
| 4. | hypertension | 0 if the patient doesn’t have hypertension, 1 if the patient has hypertension |
| 5. | heart\_disease | 0 if the patient doesn’t have heart disease, 1 if the patient has heart disease |
| 6. | ever\_married | “No” or “Yes” |
| 7. | work\_type | “children”, “Govt\_job”, “Never\_worked”, “Private”, or “Self-employed” |
| 8. | Residence\_type | “Rural” or “Urban” |
| 9. | avg\_glucose\_level | Average glucose level in the blood |
| 10. | bmi | Body mass index |
| 11. | smoking\_status | “formerly smoked”, “never smoked”, or “smokes” |
| 12. | stroke (target) | 1 if the patient had a stroke, 0 if he not |

Table 1: Features in the stroke prediction dataset [8]

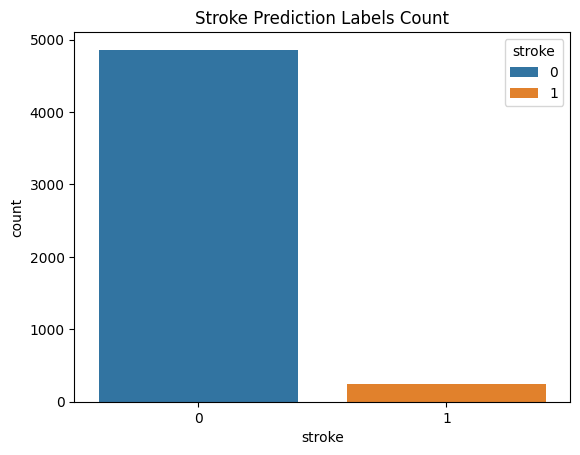


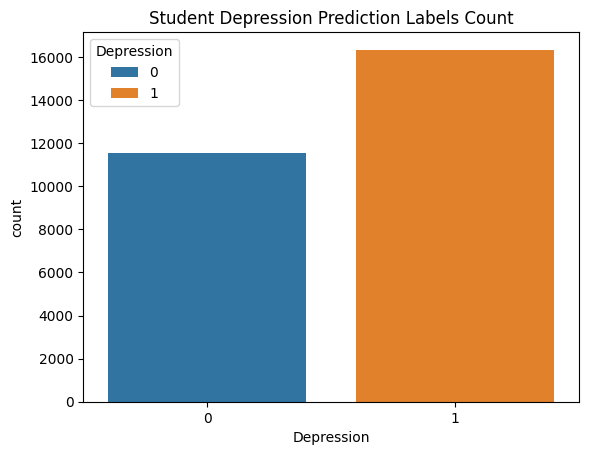
Fig. 1: Countplot of the stroke target feature

The dataset has 4,861 records for those patients who don’t have stroke (label 0) and 249 records for those patients who do have stroke (label 1).

The second dataset for predicting student depression, the student depression dataset [7] is sourced from Kaggle as well and is available in a .csv file format whose size is 1.7 MB and has 27,901 records with 18 columns which are:

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Column Name** | **Description** |
| 1. | id | Unique identifier |
| 2. | Gender | “Male” or “Female” |
| 3. | Age | Age of the student |
| 4. | City | City of the student |
| 5. | Profession | Profession |
| 6. | Academic Pressure | Rating from 0 through 5 describing how much pressure a student feels academically (0 being no pressure and 5 being the highest) |
| 7. | Work Pressure | Rating from 0 through 5 describing how much pressure a student feels work wise (0 being no pressure and 5 being the highest) |
| 8. | CGPA | CGPA of a student (Max: 10) |
| 9. | Study Satisfaction | Rating from 0 through 5 describing how satisfied a student feels study wise (14 pt0 being no satisfaction and 5 being the highest) |
| 10. | Job Satisfaction | Rating from 0 through 5 describing how satisfied a student feels job wise (0 being no satisfaction and 5 being the highest) |
| 11. | Sleep Duration | Less than 5 hours, 5-6 hours, 7-8 hours, or More than 8 hours |
| 12. | Dietary Habits | Unhealthy, Moderate, or Healthy |
| 13. | Degree | The student’s degree |
| 14. | Suicidal Thoughts | Ever had suicidal thoughts? Yes or No |
| 15. | Study Hours | How many hours a day a student studies for |
| 16. | Financial Stress | Rating from 0 through 5 describing how stressed a student feels financially (0 being no stress and 5 being the highest) |
| 17. | Family History of Mental Illness | Does the student’s family have any history of mental illness? Yes or No |
| 18. | Depression (target) | 1 if the student has depression, 0 if not |

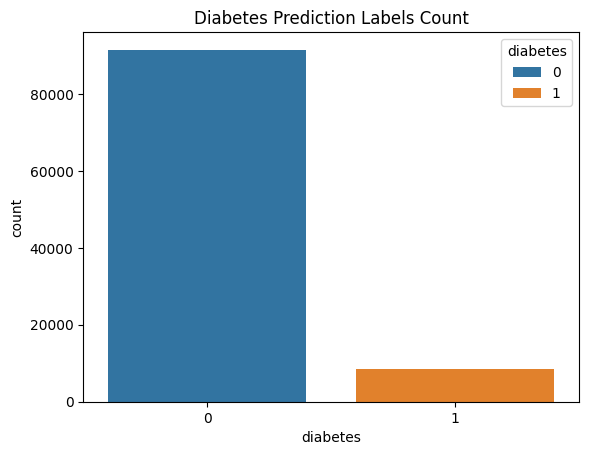
Table 2: Features in the student depression prediction dataset [7]

  
Fig. 2: Countplot of the depression target labels

The third dataset used for this system is to train and test the diabetes prediction model. This too is sourced from Kaggle, the Diabetes prediction dataset [9]. This is a .csv file and its size is 3.8 MB, with 100,000 records and 9 columns namely:

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Column Name** | **Description** |
| 1. | gender | “Male” or “Female” |
| 2. | age | Age of the patient |
| 3. | hypertension | 0 if the patient doesn’t have hypertension, 1 if the patient has hypertension |
| 4. | heart\_disease | 0 if the patient doesn’t have heart disease, 1 if the patient has heart disease |
| 5. | smoking\_history | “never”, “former”, or “current” |
| 6. | bmi | Body Mass Index |
| 7. | HbA1c\_level | Hemoglobin A1c level |
| 8. | blood\_glucose\_level | Glucose level in the blood |
| 9. | diabetes | 0 if the patient doesn’t have diabetes, 1 if they do |

Table 3: Features in the diabetes prediction dataset [9]

Fig. 3: Countplot of the diabetes target labels

**4.2: Data Preprocessing**

Several data preprocessing techniques have been applied on the datasets collected to handle missing values, for feature encoding, etc. First of all the dataset is read into a Pandas dataframe using the pandas.read\_csv(“dataset\_name.csv”) method and then the .info() method is applied on the dataframe to get some useful information about the columns in the dataset, such as the datatype of the values in the columns, number of non-null values, etc.

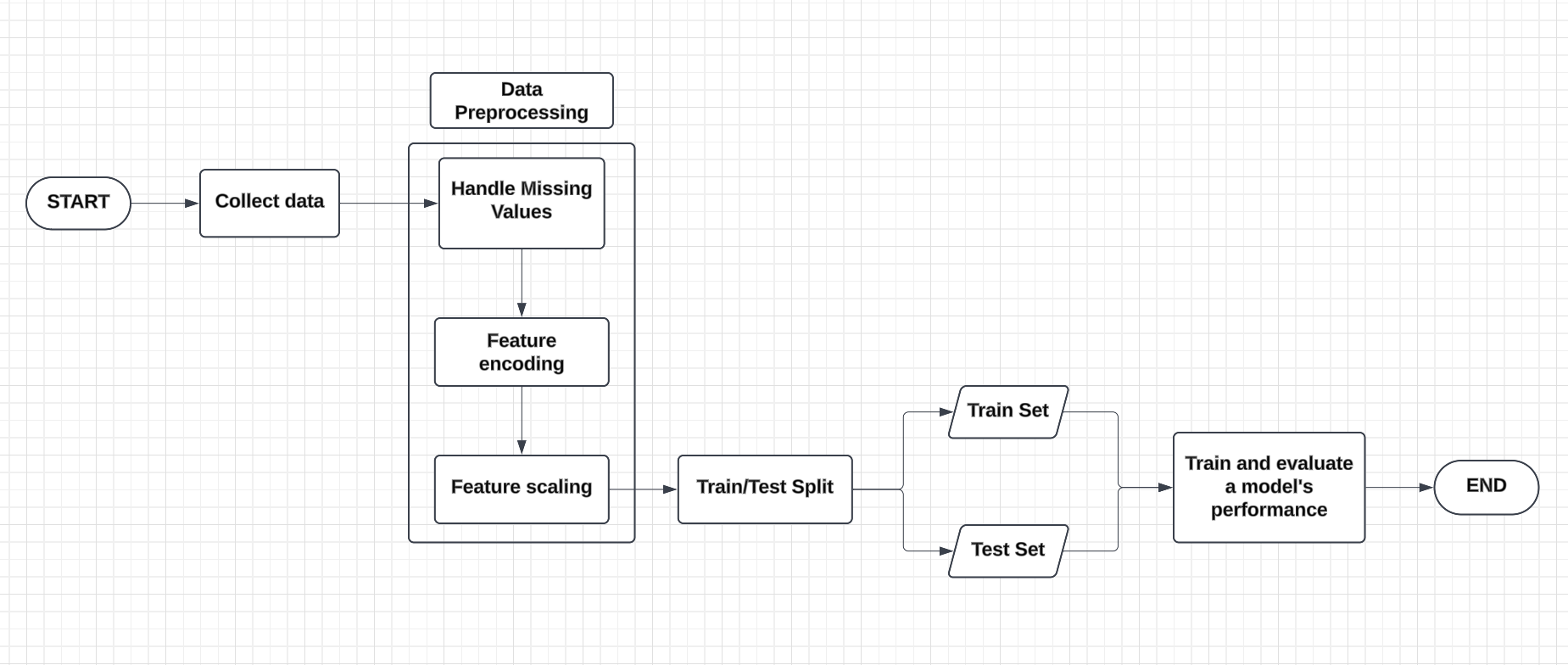
The first dataset to be preprocessed is the stroke dataset [8]. First of all the irrelevant column which is the “id” column can be dropped from the dataframe. The second step in this process is applying feature encoding to all the columns which have categorical data. This is due to the fact that machine learning models can only learn from numerical values. Label encoding can be used as the feature encoding technique for the columns “gender”, “ever\_married”, “work\_type”, “Residence\_type”, and “smoking\_status”. This step converts the categorical data in those columns into numerical data so that the ML model can learn them. The next step is to handle missing values in the “bmi” column. This can be done using the mean imputer of sklearn library. Now the dataset can be split up into the train set on which the models will be trained, and the test set on which the models will be evaluated. This is done in the ratio of 80% and 20%. After this step, the training set can be oversampled using SMOTE (Synthetic Minority Oversampling Technique) to artificially inflate the 1 class in the target variable. Then feature scaling can be applied on the features of the both train and the test datasets. For this, standardization technique can be used. Now the train set is ready to train the models.

The second dataset which is the student depression dataset [7] can be preprocessed in a similar way as the previous one. Firstly, all the irrelevant columns which don’t contribute to the prediction can be dropped, such as “id”, “City”, “Profession”, “Work Pressure”, and “Job Satisfaction”. Now label encoding can be applied on the columns “Gender”, “Sleep Duration”, “Dietary Habits”, “Degree”, “Suicidal thoughts”, and “Family History of Mental Illness”. This dataset [7] has some missing values in the “Financial Stress” column which could be handled using mode (most frequent) imputation. Now the dataset can be split into train and test datasets in the ratio 80% and 20% respectively. Finally the standardization technique can be applied for feature scaling, and now the train and test sets are ready to be used.

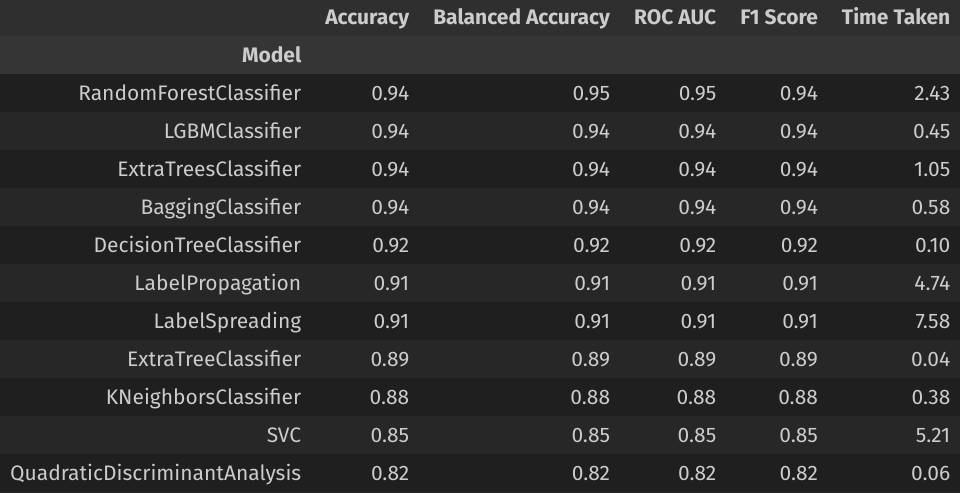
The final dataset to be preprocessed is the diabetes dataset [9]. For this, first of all the irrelevant columns are analyzed and dropped from the dataset, which is the “HbA1c\_level”. The second step is label encoding which can be applied on the columns “gender”, and “smoking\_history”. This dataset [9] has no missing values out of all 100,000 records so the step of handling the missing values can be ignored completely. The dataset can now be split into the train and the test sets in the ratio of 80% and 20%, and finally those two sets can be standardized to bring the values of all the features on to the same scale, which is a normal distribution with a mean 0 and a standard deviation of 1. The train set can be used to train the models, and test set can be used to evaluate their performance.

**4.3: Training the models**

The second stage in developing this system after the data has been preprocessed is training multiple machine learning models on the preprocessed training datasets, and evaluate their performance scores (mainly recall, as the models being made by us are disease prediction models, and it would be really useful if the recall scores of the models are high, thus reducing the number of false negatives, i.e. Type-II errors made by the models), and among the multiple trained models, the best performing one is chosen as the final model for the prediction of that particular disease.

  
Fig. 4: Process showing the steps to train and test an ML model

We have employed the usage of the “lazypredict” library to train multiple models on a training dataset and get their performance scores in a nice table representation. First of all the LazyClassifier class is to be imported from the lazypredict.Supervised module. Then an instance of that LazyClassifier class is created, say “clf”. After this, the preprocessed train and test sets are passed into the clf.fit() method which returns “models” and “predictions”. On outputting the “models”, it shows various models used by the LazyClassifier and their performance scores. From this table of models, the models which seem to perform good are selected for that particular disease prediction, their hyperparameters are tuned to get them to spit out even better recall scores. This same step is followed for all the diseases to get an idea of which models would perform good on that particular disease and choose those models to further try to improve their performance, and then those tuned models are further tested for their performance scores again, and finally the best one among those is chosen for that particular disease prediction, and that model would be ready to be consumed in the web application.

Fig. 5: Some of the models used by the LazyClassifier on the stroke prediction train set

For the stroke prediction task, first of all the LazyClassifier is applied on the train and test sets which were obtained from the previous data preprocessing step. From this, we got to know that the Random Forest Classifier, LGBM (Light Gradient Boost Machine) Classifier, Gradient Boost Classifier, and the Extra Trees Classifier models seemed to show promising results. So these models were picked from the plethora of models shown by LazyClassifier. Now these models are trained again on the train set for stroke prediction which contained random 80% records from the original raw dataset, and a point to be noted is that this train set is already preprocessed, meaning that all the missing values have been handled, categorical data has been encoded into numerical values, and all the features have been standardized (feature scaling). After training these models on the train set, they are evaluated for their performance scores against the test set. These performance scores include Recall and Accuracy.

|  |  |  |
| --- | --- | --- |
| **Model** | **Recall Score** | **Accuracy Score** |
| Light Gradient Boost Machine Classifier | 94.73% | 94.14% |
| Extra Tree Classifier | 93.59% | 90.59% |
| Gradient Boost Classifier | 88.84% | 87.09% |
| Random Forest Classifier | 95.66% | 94.59% |

Table 4: Table showing the performance scores of the models trained for stroke prediction

From this the best performing models are the Random Forest Classifier and the Light Gradient Boost Machine Classifier. Either one of them can be chosen as the final model for the stroke prediction task.

The second model to be made is for the student depression prediction task. Similar to the previous task, to train a model for this task, firstly the LazyClassifier is applied on to the preprocessed train and test sets obtained from the data preprocessing step. The train set for this task has 22,320 random records from the original dataset which is about 80% of it, and then the models are tested against 5,581 random records from the test set which are the rest 20% of the original dataset. Out of all the models available, and trained and tested by the LazyClassifier, only four seemed to show some good performance scores, which are Linear Discriminant Analysis (LDA) Classifier, Gaussian Naive Bayes (NB) Classifier, K-Nearest Neighbors (KNN) Classifier, and Support Vector Classifier (SVC).

|  |  |  |
| --- | --- | --- |
| **Model** | **Recall Score** | **Accuracy Score** |
| Linear Discriminant Analysis | 88.39% | 83.69% |
| Gaussian Naive Bayes | 85.29% | 82.89% |
| K-Nearest Neighbors | 86.1% | 81.35% |
| Support Vector Classifier | 87.55% | 83.57% |

Table 5: Table showing the performance scores of the models trained for student depression prediction

From these scores, it is apparent that the Linear Discriminant Analysis Classifier, and the Support Vector Classifier perform the best among all the tested models with similar scores, so either one can be used for the prediction of student depression.

The final prediction model to be developed for this system is the diabetes prediction. A similar process is followed to develop a model for this too as the previous two models. The LazyClassifier is applied on to the diabetes prediction train set and test set. The train set for this task has 80,000 records (80% of the original raw dataset) and the test set for this task has 20,000 records (20% of the original raw dataset). From the different models trained and tested by the LazyClassifier, only the Random Forest Classifier seemed to show some promising and somewhat accurate results with moderate performance scores. The recall score of this model has been further brought up by lowering the prediction threshold for the positive label (1) to 0.2. This means that if the prediction probability for the class 1 is anything greater than or equal to 0.2, the model outputs a positive label prediction. Usually by default this threshold would be set to 0.5 meaning that the probability has to be greater than or equal to 50% for the model to spit out a positive label prediction.

|  |  |  |
| --- | --- | --- |
|  | **Recall Score** | **Accuracy Score** |
| **Before the threshold adjustment** | 71.97% | 82.26% |
| **After the threshold adjustment** | 91.02% | 95.95% |

Table 6: Table showing the significance of threshold adjustment for the diabetes prediction model

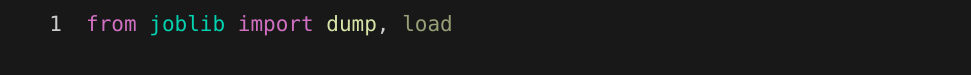
This model along with the previous two models are ready to be saved, so that their performance scores can be fixed, and after that they would be ready to be integrated into the web application and used to make predictions.

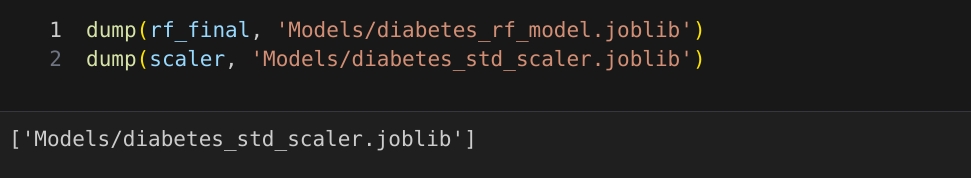
**4.4: Saving the models**

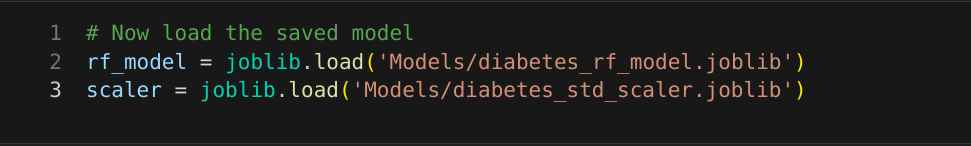
The models which have been trained successfully can now be saved into .joblib files using the “joblib” library in Python. Saving models using “joblib” is much more faster and efficient than using the “pickle” library to save models into a .pkl file format. The .joblib file is a serialized file format and is used to save and load Python objects. This saving process can be done using the joblib.dump(“filename.joblib”) method and what it does is it essentially saves the trained models (instances of their respective classes) into a .joblib file. Each model and standard scaler can be saved into different .joblib files.

These .joblib files can be loaded wherever we want using the joblib.load(“filename.joblib”) method, and the models and the standard scalers for each model can be accessed. An example process of loading and saving the models is shown below:

First import the methods from the joblib library:

Then save the models and the scalers with a desired file name and into a desired directory:

  
Now the saved models can be loaded and consumed wherever needed:

  
**4.5: Developing a user interface**

Now that the models for the three diseases, namely stroke, diabetes, and student depression have been successfully trained with good performance scores, and have been saved into .joblib files, they’re ready to be loaded in a user interface and ready to make predictions by taking input from the user. This user interface can be a web application with the following components:

1. Backend
2. Frontend
3. Database

The backend of a web application is the part of the application which handles all the logic, data processing, and communication between the frontend and the database or the server. This specific application is developed in Django using Python as the backend language. Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It is widely used to build secure and scalable web applications.