AI Based Diabetes Prediction System

Problem Definition:

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

Design Thinking Process

* **Problem Understanding and Definition**: Understand the problem of diabetes outcome prediction and its significance in healthcare. Define the objectives and desired outcomes.
* **Data Collection and Exploration**: Gather the diabetes dataset and perform initial data exploration to understand its structure and characteristics.
* **Data Preprocessing**: Clean the data by handling missing values and outliers. Apply techniques like imputation to deal with missing data. Address class imbalance using SMOTE.
* **Feature Extraction**: Select relevant features using a Particle Swarm Optimization (PSO) algorithm. The selected features are based on their importance to the prediction task.
* **Model Selection**: Choose appropriate machine learning algorithms for the classification task. Two models are considered: a Convolutional Neural Network (CNN) and a Tabular-PFN (TabPFN) classifier.
* **Model Training**: Train the selected models using the preprocessed data. Define callbacks for model training, such as early stopping and learning rate reduction.
* **Model Evaluation**: Evaluate the performance of both the CNN and TabPFN models using appropriate evaluation metrics, such as accuracy and other classification metrics.

Phases of Development

1. **Data Collection and Acquisition**:

* Downloading dataset from the kaggle

1. **Data Exploration and Understanding**:
   * Explore the dataset to gain insights into its structure and content.
   * Identify potential challenges and opportunities within the data.
2. **Data Preprocessing**:
   * Handle missing values through imputation.
   * Address outliers and noisy data points.
   * Apply class imbalance techniques like SMOTE.
   * Normalize or standardize features as needed.
3. **Feature Extraction**:
   * Utilize a Particle Swarm Optimization (PSO) algorithm to select the most relevant features for the classification task.
   * Create feature vectors for model training(for 2dCNN).
4. **Model Selection**:
   * Choose machine learning algorithms suitable for the problem. In this project, a Convolutional Neural Network (CNN) and Tabular-PFN Classifier are considered.
5. **Model Training**:
   * Train the selected models using the preprocessed data.
   * Monitor training progress and make adjustments as needed.
6. **Model Evaluation**:
   * Assess the performance of the models using a set of evaluation metrics.
   * Analyze and interpret model results to understand their strengths and weaknesses.
7. **Ensemble Methods**:
   * Explore innovative techniques, such as ensemble methods, to combine the predictions of both the CNN and TabPFN models.
   * Assess the impact of ensemble techniques on overall model performance.

Dataset:- <https://www.kaggle.com/datasets/mathchi/diabetes-data-set>

Context:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

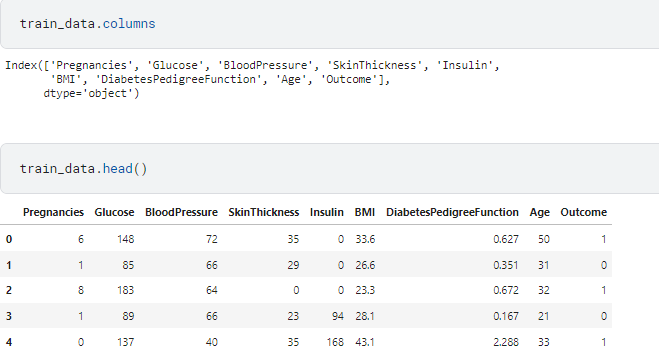
Content:

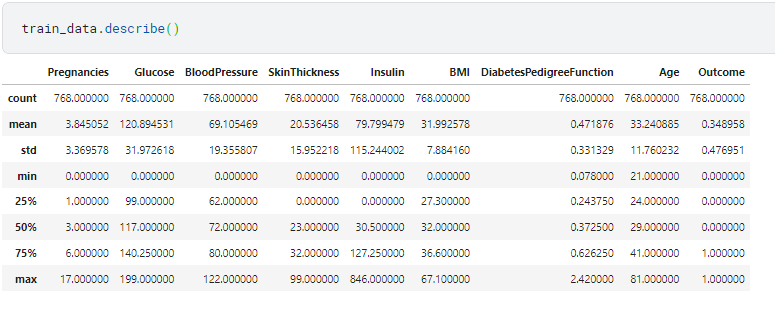
Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

* Pregnancies: Number of times pregnant
* Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
* BloodPressure: Diastolic blood pressure (mm Hg)
* SkinThickness: Triceps skin fold thickness (mm)
* Insulin: 2-Hour serum insulin (mu U/ml)
* BMI: Body mass index (weight in kg/(height in m)^2)
* DiabetesPedigreeFunction: Diabetes pedigree function
* Age: Age (years)
* Outcome: Class variable (0 or 1)

Data exploration:

Data exploration is a crucial initial step to understand the dataset's structure, uncover patterns, and identify relevant features for building a predictive model. In the case of the diabetes dataset





Data Split: Features and Target Variable

One of the critical steps in preparing the dataset for model building is the splitting of the data into features (x\_train) and the target variable (y\_train).

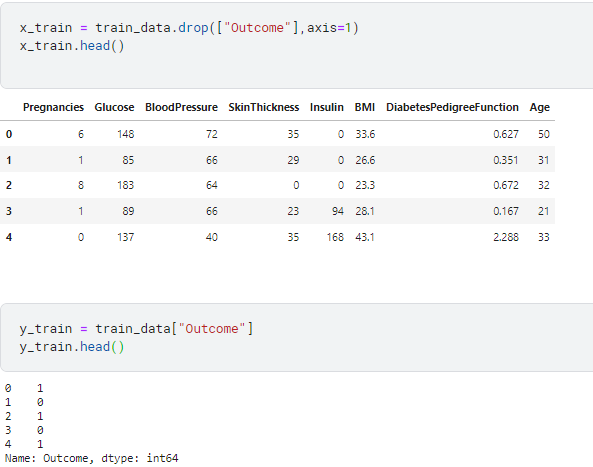
* Features (x\_train):

The features, often denoted as x\_train, represent the independent variables used as input to the machine learning model. In the context of the diabetes dataset, these features correspond to measurements and characteristics of the patients, such as "Pregnancies," "Glucose," "BloodPressure," "SkinThickness," "Insulin," "BMI," "DiabetesPedigreeFunction," and "Age."

Features are used to make predictions or classifications based on their values.

* Target Variable (y\_train):

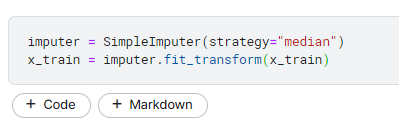
The target variable, denoted as y\_train, represents the dependent variable or the outcome we want to predict. In this dataset, the target variable is "Outcome."



Missing Value Imputation

In the dataset, it is common to encounter missing values in various columns. To handle missing data, we employ the SimpleImputer class to impute missing values with a specific strategy. In this case, we use the strategy of filling missing values with the median value for each respective

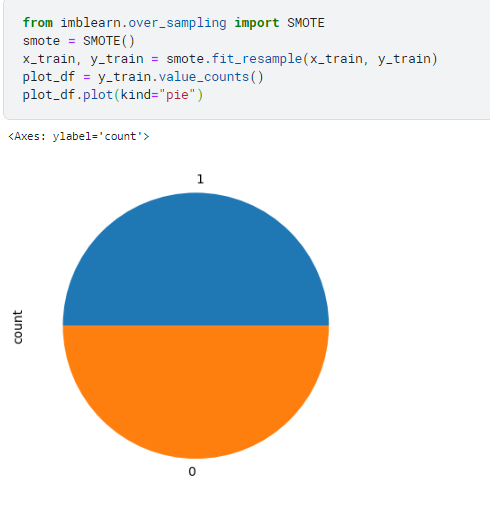
Why Impute with Median?

Imputing missing values with the median is a common strategy because it is robust and less affected by outliers. It provides a reasonable estimate of the central value for a feature, even in the presence of extreme values. This helps to maintain the integrity of the dataset and is particularly important for features with missing data.feature.

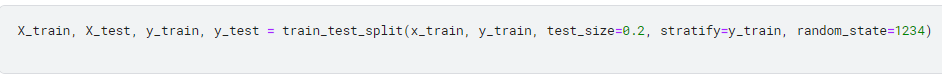
Handling Class Imbalance

In many real-world datasets, the distribution of classes in the target variable may be imbalanced, with one class significantly outnumbering the other. To ensure a balanced representation of both classes and prevent model bias towards the majority class, we employ the Synthetic Minority Over-sampling Technique (SMOTE).

Addressing class imbalance is essential to avoid model bias and achieve accurate and fair predictions. In a highly imbalanced dataset, the model may have a tendency to predict the majority class, ignoring the minority class, which could lead to inaccurate results, especially in medical and healthcare applications like diabetes prediction. By applying SMOTE, we ensure that the model is trained on a more equitable dataset, leading to improved model performance.



Train-Test Split

Splitting the dataset into training and testing sets is a fundamental practice to assess the performance and generalization of machine learning models. In this project, we utilize the train\_test\_split function to accomplish this task while taking into account the class distribution.

Feature Selection with Particle Swarm Optimization (PSO)

In this project, we employ a Particle Swarm Optimization (PSO) algorithm to perform feature selection. PSO is a computational optimization technique inspired by the social behavior of birds and fish. It is used to find the subset of features that maximizes a specific fitness function, which is typically tied to the model’s performance.

**Importance of Feature Selection:**

Feature selection is crucial for several reasons:

* It reduces dimensionality, which can lead to simpler and faster models.
* It improves model interpretability by focusing on the most informative features.
* It can enhance model performance by excluding irrelevant or redundant features.

In this project, PSO is used to identify the most relevant features, contributing to the creation of an effective predictive model for diabetes detection. The selected feature subset is crucial for model accuracy and interpretability.

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

Dataset Reshaping (CNN)

* Original Data Structure:

The original data, represented by x\_train and x\_test, typically has a tabular structure with rows and columns where each row corresponds to an example (e.g., a patient) and each column represents a feature (e.g., measurements or characteristics of the patient).

* Reshaping for CNN:

To make the data compatible with a CNN, it is reshaped into a multi-dimensional array. In this project, the reshaping is performed to create a 4D array, often in the form (number of examples, height, width, channels).

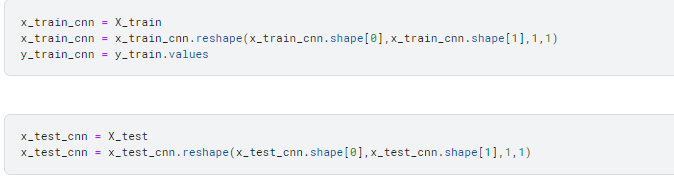
The 4D array is created by considering that the data represents images or spatial information, where:

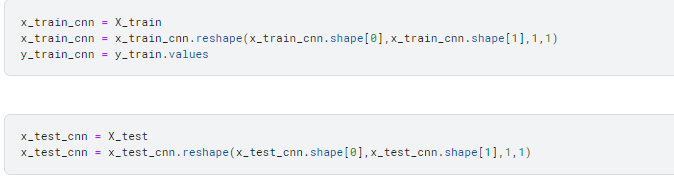
number of examples corresponds to the number of data points (e.g., patients).

height and width represent the spatial dimensions (e.g., image height and width).

channels can be used to represent color channels or additional dimensions, but in this case, it's often set to 1 for grayscale images or single-channel data.

* Training and Testing Sets:

The reshaped training dataset is stored in x\_train\_cnn, and the reshaped testing dataset is stored in x\_test\_cnn. These variables are now suitable for training and testing CNN models.



Selected Algorithm:

* 2DCNN:

2DCNN, or two-dimensional convolutional neural network, is a type of deep learning model that is specifically designed for processing images. It is composed of a series of convolutional layers, which can extract spatial features from the input image. These features can then be used for a variety of tasks, such as image classification, object detection, and image segmentation.

2DCNNs are typically trained using a supervised learning approach. This means that the model is given a set of training images, each of which is labelled with its corresponding class or segmentation. The model then learns to extract the features from the images that are most relevant to the task at hand.

* TabPFN:

TabPFN, or Tabular Prior-Data Fitted Network, is a type of neural network that is specifically designed for tabular data. It is a modified version of the original PFN architecture, which has been shown to be very effective for tabular classification tasks.

TabPFN has two key modifications that make it well-suited for tabular data:

* Attention masks: TabPFN uses slight adjustments to the attention masks in the PFN architecture, which results in shorter inference times.
* Zero-padding: TabPFN can handle datasets with varying numbers of features by zero-padding the shorter features.

TabPFN is also trained on synthetic datasets that are generated using principles from causal reasoning and simplicity. This gives TabPFN a prior knowledge of tabular data, which allows it to learn more quickly and efficiently.

TabPFN has been shown to achieve state-of-the-art results on a variety of tabular classification benchmarks. It is also very fast and efficient, making it ideal for real-time applications.

Why this Algorithm?

* If the data can be structured to resemble images or spatial information, a 2DCNN is a good choice. It's excellent at capturing spatial features and can learn complex patterns within the data, potentially offering state-of-the-art performance.
* On the other hand, if you're working with the original tabular data format, TabPFN is tailored for this type of data. It efficiently handles structured data, incorporates attention mechanisms for feature selection, and can adapt to varying numbers of features. Its prior knowledge of tabular data can also help it learn quickly and effectively.

CNN Model Architecture:

The Convolutional Neural Network (CNN) model is designed with several layers for feature extraction and classification. The architecture is defined as follows:

1. **Convolutional Layers**: The model starts with two convolutional layers. The first layer has 64 filters with a kernel size of (3, 1), and the second layer has 128 filters with the same kernel size. These layers extract spatial features from the input time series data.
2. **Batch Normalization**: Batch normalization is applied after each convolutional layer to improve training stability and accelerate convergence.
3. **Global Average Pooling**: Following the convolutional layers, a global average pooling layer is used to reduce the spatial dimensions and extract essential features.
4. **Dense Layers**: Three dense layers follow, with decreasing units (128, 64, 32). Each dense layer is activated with ReLU and includes dropout with a dropout rate of 0.5 to prevent overfitting.
5. **Output Layer**: The final layer is a dense layer with a single unit and sigmoid activation, which outputs the classification probability.

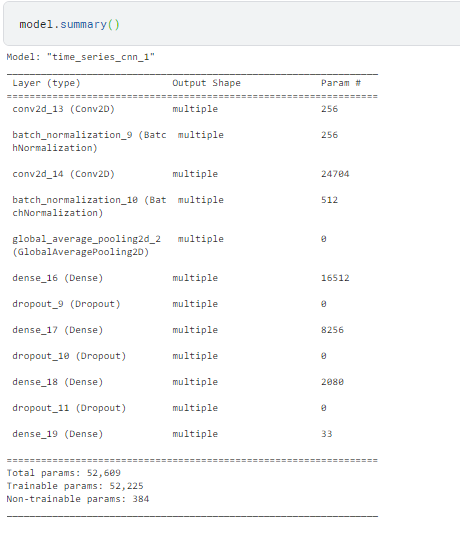
Python:



Model Summary

The summary of the CNN model, an overview of the network's architecture, including layer types, output shapes, and the number of trainable parameters.

Python:



Model Training

The model is compiled and trained with the following settings:

* **Optimizer**: The Adam optimizer is used, which is a popular optimization algorithm for training neural networks. It adapts the learning rate during training to optimize model parameters efficiently.
* **Loss Function**: Binary cross-entropy loss is employed as the loss function. This loss function is commonly used for binary classification tasks, such as predicting diabetes (0 or 1). It measures the dissimilarity between the predicted probabilities and the actual class labels.
* **Evaluation Metric**: The accuracy metric is chosen as the evaluation metric. Accuracy measures the proportion of correctly classified instances in the validation dataset. It is a fundamental metric for classification tasks and provides an easily interpretable measure of model performance.
* **Training Details**: The model is trained for 300 epochs, which represents the number of complete passes through the training dataset. During each epoch, the model learns from the training data and adjusts its parameters to minimize the loss. This process is repeated for 300 epochs to ensure the model has the opportunity to converge and improve its performance.
* **Batch Size**: A batch size of 128 is used. This means that the training dataset is divided into batches of 128 examples, and the model's parameters are updated after processing each batch. A suitable batch size balances training efficiency and memory usage.

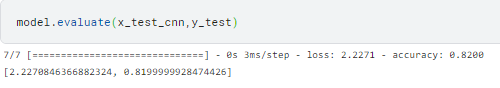
Python:



Model Evaluation

We evaluate the model's performance on the training data. The output includes the loss and accuracy.

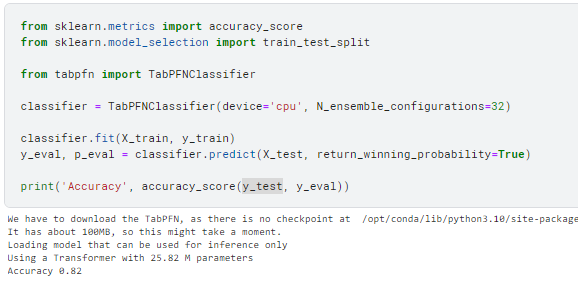
Python:



Model Training and Evaluation with TabPFN

The model trained and evaluated using the TabPFN library is configured with 32 ensemble configurations, indicating that it employs an ensemble learning approach with multiple configurations. After training, the model's accuracy is assessed.

Python:



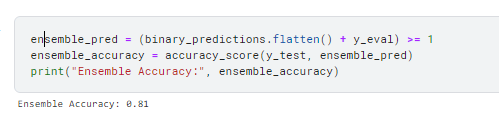
Innovative Techniques

Ensemble of Models

Ensembling models is a powerful technique used to enhance classification accuracy and make more robust predictions. In this project, an ensemble of models is created by combining the predictions from two different models: the CNN model and the TabPFN classifier. The ensemble decision is made by taking a majority vote among these models. Here's how it works:

1. **Multiple Models**: Two distinct models are trained for the diabetes prediction task:
   * **CNN Model**: This model is designed to capture spatial features in the data and is particularly suitable for scenarios where the data can be treated as images or spatial information.
   * **TabPFN Classifier**: TabPFN is tailored for tabular data and is optimized for structured datasets like the original diabetes dataset.
2. **Individual Predictions**: Each of the two models makes individual predictions on the test data. These predictions represent the model's belief about whether a given patient has diabetes or not.
3. **Majority Vote**: To create an ensemble, the individual predictions from both models are combined by taking a majority vote. In a binary classification task like diabetes prediction, this typically means that the class predicted by the majority of models is taken as the final prediction for each data point.
4. **Improving Robustness**: Ensembling models in this way can enhance the robustness of predictions. It reduces the risk of making decisions based on the idiosyncrasies of a single model and is particularly valuable when different models have different strengths and weaknesses.

Python:



Conclusion

This project aimed to develop a machine learning model for the classification of diabetes outcomes, with a focus on improving prediction accuracy and contributing to early diabetes detection and risk assessment. This project followed a structured design thinking process, consisting of problem understanding, data collection, preprocessing, feature extraction, model selection, training, evaluation, and innovative techniques.