**Paper Link:** [Here](F://Academic/4th%20Year/7th%20Semester/Sessional/CSE%204000%20Project%20or%20Thesis/Paper/Dola%20Mam/Diabetes%20Prediction%20Using%20Ensembling.pdf)

**Paper Name: Diabetes Prediction Using Ensembling of Different Machine Learning Classifiers**

**Proposed Method:** We propose a new pipeline for diabetes prediction from the PIMA Indians Diabetes dataset. Prepro cessing, in the proposed pipeline, is the heart of achieving the state-of-the-art result, which consists of outlier rejection, f illing missing values, data standardization, feature selec tion, and K-fold cross-validation. We consider the mean value in the missing position of attribute rather than median value, as it has a more central tendency toward the mean of that attribute distribution. The folding of the dataset for cross-fold validation is performed carefully to preserve the percentage of class proportion, as same as in the original dataset. Different ML classifiers (k-nearest Neighbour (k NN), RF, DT, NB, AB, and XGBoost (XB)) and MLP were implemented in our proposed pipeline. We apply the grid search technique for selecting the number of hidden layers, number of neurons in each hidden layer, activation function, neuron initializer, batch size, learning rate, epoch, percentage of dropped neurons, loss function, an optimizer of MLP and hyperparameters of ML models. Extensive experiments are performed on different combinations of preprocessing and ML classifiers for maximizing the AUC of diabetes predic tion under the same experimental conditions and dataset. The best ML classifier is then set as a baseline model to evaluate our proposed classifier quantitatively for the prediction of diabetes precisely. Moreover, we propose an ensembling classifier by the combination of the ML models for boosting the diabetes prediction. To ensemble the ML models, soft weighted voting is employed, where the weight for the in dividual model was estimated from the respective AUC. The AUCof the ML model is chosen as the weight of that model for voting ensembling rather than accuracy since AUC is unbiased to the class distribution. Extensive experiments on different combinations of the ML models are accomplished for searching the best ensemble classifier where the best performing preprocessing from the previous experiments is employed. The organization of the remaining paper is as follows: Section II presents the dataset, proposed methodology, and evaluation metrics. In section III, the different experimental results are reported with the interpretation. Finally, the paper is concluded with future works in section IV.

**[Q1] How grid search technique can be used for selecting the number of hidden layers?**

* Certainly! Grid search is a powerful technique for hyperparameter tuning in machine learning. When it comes to selecting the number of hidden layers in a neural network, grid search can be quite useful. Let’s dive into how you can apply grid search for this purpose:

1. **Understanding Hidden Layers:**

In a neural network, the hidden layers are the intermediary layers between the input layer and the output layer.

The number of hidden layers and the number of neurons (units) in each layer significantly impact the model’s performance.

1. **Grid Search for Hidden Layers:**

To determine the optimal number of hidden layers, follow these steps:

Define a Range: Decide on a range of possible hidden layer configurations. For instance, consider values from 1 to 5 hidden layers.

Create a Grid: Set up a grid of hyperparameters to explore. In this case, the hyperparameter of interest is the number of hidden layers.

Train and Evaluate Models: For each configuration (number of hidden layers), train a neural network using cross-validation (e.g., K-fold cross-validation).

Evaluate Performance: Use a performance metric (e.g., accuracy, mean squared error) to assess how well each model generalizes.

Select the Best Configuration: The configuration with the best performance (lowest error or highest accuracy) is the optimal choice.

1. **Example Python Code:** (Python)

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid

param\_grid = {

'hidden\_layer\_sizes': [(10,), (20,), (30,), (40,), (50,)], # Vary the number of hidden layers

'activation': ['relu'], # You can explore other activation functions

'solver': ['adam'], # Choose an appropriate solver

}

# Create the model

model = MLPClassifier(max\_iter=1000, random\_state=42)

# Perform grid search

grid\_search = GridSearchCV(model, param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(X\_train, y\_train) # X\_train and y\_train are your training data

# Get the best configuration

best\_hidden\_layers = grid\_search.best\_params\_['hidden\_layer\_sizes']

print(f"Best hidden layers configuration: {best\_hidden\_layers}")

1. Considerations:

Keep in mind that deeper networks (more hidden layers) may require more data to prevent overfitting.

Grid search helps you find the optimal configuration, but it’s essential to validate the chosen model on a separate test set.

Remember that grid search allows you to systematically explore different hyperparameter combinations, making it a valuable tool for selecting the right architecture for your neural network. 🌟