SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

A COMPARISON OF MACHINE LEARNING ALGORITHMS FOR DIABETES PREDICTION

SUBMITTED BY

COURSE: CSE425 - MACHINE LEARNING ESSENTIALS

NAME: THANVEER AHAMED S

REG NO: 125018070

TABLE OF CONTENTS

1. Abstract

Diabetes is a chronic illness that requires early detection and ongoing management to reduce complications. In this project, we explore the potential of machine learning algorithms to predict diabetes based on patient health data using the **Pima Indian Diabetes Dataset (PIDD)**. The dataset contains 768 records of female patients, each with nine health attributes that are indicative of diabetes risk, such as **glucose levels**, **BMI**, and **insulin concentration**. We implemented various machine learning models, including **Logistic Regression (LR)**, **Support Vector Machine (SVM)**, **Naive Bayes (NB)**, **K-Nearest Neighbors (KNN)**, **Random Forest (RF)**, **Decision Trees (DT)**, **AdaBoost (AB)**, and **Neural Networks (NN)**. Each model was evaluated using key performance metrics such as **accuracy**, **precision**, **recall**, and **F1-score**.

Extensive experimentation was conducted, and **Support Vector Machine (SVM)** achieved the highest accuracy of **88.6%**, outperforming other models in both precision and recall. The **Neural Network (NN)** model also performed well, with an accuracy of **86%** after tuning the number of hidden layers and epochs. These results demonstrate the effectiveness of machine learning techniques in predicting diabetes outcomes. Feature selection techniques were applied to reduce dimensionality, and hyperparameter tuning was used to improve the performance of the models. This project highlights the potential of machine learning in healthcare, particularly for early diagnosis of chronic diseases such as diabetes. Future work could expand the dataset and explore deep learning techniques to further enhance prediction accuracy and generalizability.

2. Introduction

2.1. Project Objectives

The primary objective of this project is to evaluate the effectiveness of various machine learning algorithms in predicting diabetes using the Pima Indian Diabetes dataset. By analyzing the performance of these algorithms, we aim to identify the most accurate models and the key features that contribute to successful diabetes prediction. This study also seeks to provide insights into the applicability of machine learning in the healthcare domain, particularly in chronic disease management.

2.2. Problem Formulation

Diabetes is a chronic condition that can lead to severe health complications, including cardiovascular diseases, kidney failure, and neuropathy, if not detected early. Traditional diagnostic methods may not always provide timely results, necessitating the exploration of advanced predictive techniques. This study formulates the problem of diabetes prediction as a classification task, where the goal is to classify patients as diabetic or non-diabetic based on their health attributes. The challenge lies in selecting the appropriate machine learning models and features that can accurately predict the likelihood of diabetes. The research questions guiding this study include:

- ➢ Which machine learning algorithms provide the highest accuracy in predicting diabetes?
- ➢ What are the most significant features influencing diabetes prediction?

2.3. Importance of the dataset

The **Pima Indian Diabetes Dataset (PIDD)** is one of the most widely used datasets in machine learning for medical applications, particularly in the prediction of diabetes. It contains health-related data for 768 female patients of Pima Indian heritage, with 8 key health attributes (e.g., glucose levels, BMI, insulin levels) and an outcome label indicating whether or not the patient has diabetes.

Key Points on Dataset Importance:

1. **Relevance to Healthcare**:
    - The attributes in the PIDD are directly related to common risk factors for diabetes, such as glucose levels, blood pressure, and BMI. These attributes reflect real-world medical data that is often collected in clinical settings.
    - The dataset is particularly useful for developing and testing machine learning models in healthcare, enabling research on predictive models for diabetes, a major global health issue.
2. **Real-World Application**:
    - Early detection and management of diabetes are critical in reducing long-term complications such as cardiovascular disease and kidney failure. The dataset's focus on early-stage diabetes diagnosis makes it highly relevant for real-world medical applications, where machine learning could assist in early detection and treatment plans.
3. **Balanced Features and Challenges**:
    - The dataset is balanced in terms of the number of diabetic and non-diabetic cases, which allows for a fair comparison of model performance. However, challenges such as missing values, noise, and potential outliers present an opportunity to test various preprocessing techniques and improve the robustness of machine learning models.
4. **Benchmarks in Machine Learning**:
    - Due to its popularity, PIDD serves as a benchmark dataset for evaluating the performance of various machine learning algorithms. It provides a standardized framework for comparing results across different studies and models, making it a valuable resource for academic research and practical implementation.

2.4. Task, Performance Metric, Experience (T, P, E)

T (Task): Predicting whether a patient has diabetes based on health attributes.
P (Performance Metric): Accuracy, Precision, Recall, F1-score.
E (Experience): Dataset of female patient records with labeled health attributes (Pima Indian Diabetes dataset).

2.5. Methodology and Planning

The study applies several machine learning models, evaluates their performance using metrics such as **accuracy**, **precision**, **recall**, and **F1-score**, and optimizes models through hyperparameter tuning. The models range from simple classifiers like **Logistic Regression** to more complex models like **Neural Networks**.

2.6. Results Summary

Among the models, the **SVM** achieved the highest accuracy of **88.6%**, closely followed by the **Neural Network** with **87%**. Other models, such as **Random Forest** and **KNN**, performed satisfactorily but did not match the precision of SVM.

2.7. Document Structure

This report is structured as follows: The **Related Work** section discusses existing studies in diabetes prediction using machine learning. The **Background** section explains the models and preprocessing steps. The **Methodology** outlines the experimental design. The **Results** and **Discussion** sections cover the outcomes of the experiments, and the report concludes with the **Learning Outcome** and **Conclusion** sections.

3. Related Work

3.1. Sources Referenced

The project relies on multiple sources, including:

- **Kaggle** for accessing the Pima Indian Diabetes dataset.
- **Base Papers**: Research from the UCI Machine Learning Repository and academic publications on machine learning algorithms for diabetes prediction.
- **Websites** for model clarifications and demos.

3.2. References

- **Reference Paper:**
  - ➢ Khanam, J. J., & Foo, S. Y. (2021). A comparison of machine learning algorithms for diabetes prediction. *ICT Express,* *7*(4), 432-439. https://doi.org/10.1016/j.icte.2021.02.004
  - ➢ https://www.sciencedirect.com/science/article/pii/S2405959521000205
- **Dataset:**
  https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database/data
- **Other references:**
  - ➢ https://www.who.int/health-topics/diabetes.
  - ➢ https://www.medicalnewstoday.com/articles/325018#how-is-the-pancreas-linked-with-diabetes.
  - ➢ https://www.webmd.com/diabetes/diabetes-causes.
  - ➢ https://www.mayoclinic.org/diseases-conditions/prediabetes/diagnosis-treatment/drc-20355284.
  - ➢ G. Swapna, R. Vinayakumar, K.P. Soman, Soman KP diabetes detection using deep learning algorithms, ICT Express 4 (4) (2018) 243–246, http://dx.doi.org/10.1016/j.icte.2018.10.005, Elsevier B.V.

4. Background

4.1. Dataset Used

The Pima Indian Diabetes dataset (PIDD) was utilized for this study, sourced from the UCI Machine Learning Repository. The dataset contains 768 records of female patients, with nine attributes:

- Pregnancy: Number of pregnancies.
- Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test.
- Blood Pressure: Diastolic blood pressure (mm Hg).
- Skin Thickness: Triceps skin fold thickness (mm).
- Insulin: 2-Hour serum insulin (mu U/ml).
- BMI: Body mass index (weight in kg/(height in m)^2).
- Diabetes Pedigree Function: A function that scores the likelihood of diabetes based on family history.
- Age: Age (years).
- Outcome: Class variable (0 for non-diabetic, 1 for diabetic).

4.2. Data Preprocessing Techniques

Data preprocessing is a critical step in machine learning that transforms raw data into a suitable format for modeling. Proper preprocessing ensures that models can learn effectively from the data and make accurate predictions.

a. Handling Missing Values

Problem: Missing values are common in real-world datasets and can lead to inaccuracies or biases in machine learning models. In the **Pima Indian Diabetes Dataset (PIDD)**, key features such as **Glucose**, **Blood Pressure**, **Skin Thickness**, **Insulin**, and **BMI** have some missing entries, likely due to errors in data collection or incomplete patient records.

Solution:
To address missing values, I used **mean imputation**. This technique involves replacing missing data in a particular feature with the **mean** value of that feature from the remaining non-missing entries. For example, if a patient's glucose level is missing, we replace it with the average glucose level from the rest of the dataset.

Why It's Important:
Mean imputation ensures that no data points are discarded due to missing values, which is crucial when working with small datasets like the PIDD. Retaining more data improves model robustness and accuracy. Additionally, imputing with the mean prevents extreme distortion of the data while maintaining the overall distribution of values within a feature. However, this method assumes that the data is missing at random and may not be ideal if missingness is systematic or related to other features.

b. Normalization (Feature Scaling)

Problem: Features in datasets often have varying ranges. For example, **Glucose** values can range from 0 to over 200, while **Diabetes Pedigree Function** values range from 0 to 2. These differences in scale can lead to problems in machine learning algorithms that rely on

distance calculations, such as **K-Nearest Neighbors (KNN)** or **Support Vector Machine (SVM)**, as large-scale features can dominate smaller-scale ones.

Solution:
To ensure that all features contribute equally, **Min-Max Normalization** was applied. This technique scales the data by transforming each feature to a fixed range, typically 0 to 1. The formula for Min-Max normalization is:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

where ***X*** is the original feature value, and ***Xmin*** and ***Xmax*** are the minimum and maximum values of the feature, respectively.

Why It's Important:
Normalization ensures that features are on the same scale, which is crucial for algorithms that calculate distances or rely on gradient-based optimization. For example, in **KNN**, the distances between data points are calculated in a multi-dimensional space. If one feature has a larger range than others, it could disproportionately affect the distance and dominate the learning process. By normalizing the data, we prevent this bias and ensure that all features contribute equally to the model.

c. Handling Outliers

Problem: Outliers are extreme values that differ significantly from the majority of the data. In medical datasets, such as PIDD, outliers could be due to measurement errors, anomalies, or rare events. If left unhandled, outliers can skew statistical measures and negatively affect the performance of machine learning models, especially in distance-based algorithms or linear models.

Solution:
The **Interquartile Range (IQR) method** was used to detect and remove outliers. The IQR is a measure of statistical dispersion and is calculated as the range between the **first quartile (Q1)** and the **third quartile (Q3)** of the data. Outliers are identified as data points that fall below **Q1 - 1.5 * IQR** or above **Q3 + 1.5 * IQR**.

Why It's Important:
By removing outliers, we ensure that extreme values do not distort the relationships between features or introduce bias in the model. Outliers can disproportionately affect models like **Linear Regression** by pulling the model toward the extreme values, thereby reducing accuracy. For models like **KNN** or **SVM**, outliers can distort distance calculations, leading to incorrect classifications. The IQR method is effective for identifying and removing these extreme values while retaining most of the relevant data.

d. Feature Selection

Problem: Not all features in a dataset are equally relevant to the target variable. Some features may be redundant, irrelevant, or even noisy, which can decrease the performance

of machine learning models. Including too many irrelevant features can also lead to overfitting, where the model learns the noise rather than the actual signal in the data.

Solution:
I applied **correlation analysis** to select the most relevant features. Correlation measures the linear relationship between two variables, and features that have a strong correlation with the target variable (in this case, whether the patient has diabetes) are more likely to be useful for prediction. We visualized the correlation between features using a **correlation matrix** and selected those that had the highest correlation with the outcome.

Why It's Important:
Feature selection helps reduce the dimensionality of the dataset, making the models simpler, faster, and less prone to overfitting. It also helps the model focus on the most important features, leading to improved performance. Reducing the number of features can be particularly helpful for algorithms like **Logistic Regression** or **Naive Bayes**, which perform better when irrelevant features are removed. Feature selection also enhances model interpretability, as it highlights the most important predictors of the target variable.

4.3. Models Used

Implemented the following machine learning algorithms for diabetes prediction:

1. Logistic Regression (LR)

Logistic Regression is a statistical method for binary classification that estimates the probability of an outcome based on input variables. It uses the **sigmoid function** to predict the probability of a positive class (diabetes).

Structure: Logistic regression is a linear classifier that models the log-odds of the outcome as a linear combination of the input features.

Architecture:

- **Input Layer**: Each feature from the dataset (e.g., Glucose, BMI) is treated as an independent variable in a linear equation.
- **Linear Transformation**: Logistic Regression models the relationship between the input features and the output as a linear combination:

$$z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

  where ***Wi*** are weights, and ***Xi*** Are the feature values.

- **Sigmoid Activation Function**: The linear combination of inputs is passed through a sigmoid function to produce a probability between 0 and 1:

$$P(y = 1|x) = \frac{1}{1 + e^{-z}}$$

- **Binary Classification**: The output is interpreted as the probability of class 1 (diabetic) or class 0 (non-diabetic), with a threshold (e.g., 0.5) for classification.
- **No Hidden Layers**: As a simple linear model, there are no hidden layers in logistic regression.

Key Points:

- Assumes a linear relationship between features and the log-odds of the outcome.
- Efficient for small datasets.
- Simple to interpret as each feature's weight indicates its importance.
- **Limitations**: Cannot handle non-linear relationships.

2. Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful classification algorithm that constructs an optimal hyperplane in a high-dimensional space to separate the classes.

Structure: SVM works by maximizing the margin between the support vectors and the decision boundary. The kernel trick is used to handle non-linear separations.

Architecture:

- **Input Layer**: Each feature is represented as a point in multi-dimensional space.
- **Hyperplane Construction**: The algorithm constructs a decision boundary (hyperplane) that separates classes. In 2D, this is a line, but in higher dimensions, it becomes a plane or hyperplane.
- **Support Vectors**: SVM focuses on the data points (support vectors) closest to the decision boundary. These points are critical in defining the hyperplane.
- **Maximizing Margin**: The algorithm maximizes the margin between the support vectors and the hyperplane, leading to a robust classifier.
- **Kernel Trick**: For non-linear data, SVM applies a kernel function (e.g., Radial Basis Function (RBF)) to map input features into a higher-dimensional space, allowing linear separation in this transformed space.

Key Points:

- Excellent for high-dimensional spaces and datasets with a clear margin of separation.
- Can handle non-linearly separable data via kernel functions.
- **Limitations**: Computationally expensive for large datasets, and tuning hyperparameters (e.g., C, gamma) is crucial.

3. Naive Bayes (NB)

Naive Bayes is a probabilistic classification model based on **Bayes' Theorem**, assuming that features are independent. Despite this assumption often being unrealistic, Naive Bayes can perform surprisingly well, especially on smaller datasets.

Structure: Based on **conditional probability** and assumes independence between features. It’s particularly useful for tasks with text classification but also performs well in medical prediction tasks.

Architecture:

- **Input Layer**: Each feature (e.g., Glucose, Insulin) is considered independently when calculating the probability of an outcome.
- **Bayesian Framework**: It applies Bayes' Theorem to calculate the posterior probability for each class (diabetic or non-diabetic):

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

  where ***P(Y)*** is the prior probability of class ***Y***, and ***P(X| Y)*** is the likelihood.

- **Conditional Independence Assumption**: Naive Bayes assumes that all features are conditionally independent given the class, i.e., no interaction between the features.
- **Prediction**: The class with the highest posterior probability is selected as the prediction.

Key Points:

- Suitable for small datasets due to its simplicity.
- Computationally efficient and fast.
- Works well despite the unrealistic independence assumption.
- **Limitations**: Assumes feature independence, which is often violated in real-world datasets.

4. Random Forest (RF)

Random Forest is an ensemble learning method that builds multiple decision trees and aggregates their predictions to improve accuracy and reduce overfitting. It is robust against noise and provides an inherent method for feature importance.

Structure: It constructs decision trees from bootstrapped samples of the dataset, and each tree is trained on a random subset of the features.

Architecture:

- **Ensemble of Decision Trees:** Random Forest consists of multiple decision trees, each trained on different random subsets of the data and features. This process is known as bagging (bootstrap aggregation).
- **Decision Trees:** Each tree in the forest splits data based on feature values, aiming to maximize the information gain or minimize Gini impurity.
- **Voting Mechanism:** Each decision tree makes a classification, and the final prediction is based on the majority vote from all trees.
- **Feature Randomness:** Each tree considers a random subset of features at each split, which reduces overfitting and increases the diversity of the trees.
- **Out-of-Bag (OOB) Error:** Random Forest uses the data not selected for training each tree to evaluate its performance, providing an unbiased estimate of the model's accuracy.

Key Points:

- Combines multiple decision trees to reduce overfitting and improve accuracy.
- Robust against noisy data and can handle large datasets.
- **Limitations:** Less interpretable than a single decision tree; more computationally expensive.

5. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric method that classifies instances based on the majority class of their closest neighbors. The choice of k (the number of neighbors) is crucial to its performance.

Structure: The algorithm classifies based on the **majority vote** of the closest k neighbors. Distance metrics, such as **Euclidean distance**, are used to compute the proximity between data points.

Architecture:

- **No Training Phase**: KNN doesn't explicitly build a model during training. Instead, it stores the entire training dataset and classifies new data points based on their similarity to the stored points.
- **Distance Calculation**: KNN uses distance metrics, most commonly **Euclidean Distance**, to measure the similarity between points. For two points ***x=(x1,x2,...,xn)*** and ***y=(y1,y2,...,yn)***,

  the Euclidean distance is calculated as:

  $$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

  This formula measures the straight-line distance between two points in an n-dimensional space.

- **Classification by Voting**: Once the distances between the new data point and all other points in the dataset are computed, KNN selects the **k** nearest neighbors (the closest data points) and assigns the most common class among these neighbors to the new data point. If there's a tie, methods such as weighted voting (based on distance) may be used.
- **Choice of k**: The number of neighbors k is a critical hyperparameter in KNN. A small k makes the model sensitive to noise (overfitting), while a large k smooths the decision boundary but may lead to underfitting.

Key Points:

- KNN is simple to implement and understand.
- Works well for smaller datasets with clear boundaries between classes.
- **Limitations**: KNN is computationally expensive during the prediction phase, as it needs to calculate distances to all points in the dataset. It is also sensitive to the choice of kkk and to scaling of the features.

6. Decision Tree (DT)

A **Decision Tree** splits the data recursively based on the most significant features to create a tree-like model for classification. Each internal node represents a decision based on a feature, and each leaf node represents a classification.

Structure: Decision Trees are built by recursively splitting the dataset at each node based on a feature that results in the most significant information gain.

Architecture:

- **Root Node**: The entire dataset is represented at the root node, and the Decision Tree algorithm selects a feature that best separates the classes based on a splitting criterion.
- **Splitting Criterion**: Decision trees use measures like **Information Gain** (based on **Entropy**) or **Gini Impurity** to decide how to split data at each node.
    - **Entropy** is used to quantify the amount of disorder or impurity in the data. It's given by:

      $$\text{Entropy}(S) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

      Where ***S*** is the set of data, ***C*** is the number of classes, and ***pi*** is the proportion of class i in the dataset. The goal is to reduce entropy with each split.
    - **Information Gain** measures the reduction in entropy after a dataset is split on a feature:

      $$\text{Information Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

      Where ***S*** is the dataset, ***A*** is the attribute being split, and ***Sv*** is the subset of data where attribute ***A*** takes value ***v***.
    - **Gini Impurity** is another measure used to evaluate splits. It's calculated as:

      $$\text{Gini}(S) = 1 - \sum_{i=1}^{C} p_i^2$$

      Where ***pi*** is the proportion of instances in class ***i***. A Gini Impurity of 0 means perfect classification, while higher values indicate more impurity.
- **Recursive Splitting**: The dataset is recursively split at each internal node until it either perfectly classifies the data or meets a stopping criterion (e.g., a maximum depth or minimum number of samples per node).
- **Leaf Nodes**: The final nodes (leaf nodes) represent the predicted class. Each leaf holds a class label based on the majority class in that subset of data.

Key Points:

- Highly interpretable, as it mirrors human decision-making.
- Capable of handling both linear and non-linear relationships.
- **Limitations**: Prone to overfitting if the tree becomes too deep without pruning.

7. AdaBoost

AdaBoost is an ensemble technique that combines multiple weak classifiers (often decision trees) into a strong classifier. It adjusts the weights of incorrectly classified instances and focuses more on difficult-to-classify samples in subsequent iterations.

Structure: AdaBoost boosts the performance of weak learners by iteratively improving their predictions and combining the results into a final strong prediction.

Architecture:

- **Weak Learners**: AdaBoost combines multiple weak learners (usually decision trees with a single split, also known as decision stumps) to form a strong classifier.
- **Weighting Mechanism**: After each iteration, AdaBoost adjusts the weights of incorrectly classified instances, so subsequent weak learners focus more on these difficult cases.
- **Boosting Process**: Each weak learner is trained sequentially, and the final prediction is a weighted sum of all weak learners' predictions.
- **Adaptive Nature**: The model adapts by giving more weight to misclassified examples in the next iteration, thereby improving overall accuracy.

Key Points:

- Efficient at boosting the performance of weak learners.
- Focuses on hard-to-classify examples, leading to improved accuracy.
- **Limitations**: Sensitive to noisy data, as it may place too much focus on outliers.

8. Neural Networks (NN)

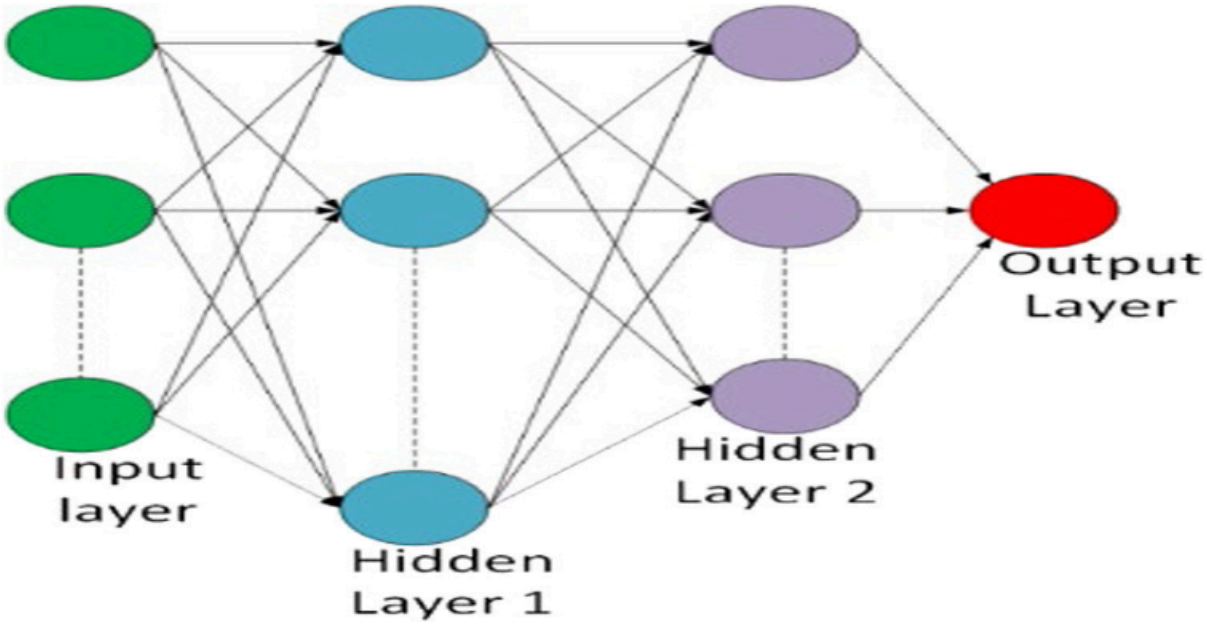
Neural Networks consist of layers of neurons that stimulate the human brain's structure. Each neuron takes inputs, applies weights, and passes the result through an activation function. In this experiment, we used a **feedforward neural network** with two hidden layers.

Structure: The neural network comprises an input layer, two hidden layers (with 12 and 8 neurons), and an output layer with a sigmoid activation for binary classification.

Architecture:

- **Input Layer**: The number of neurons in the input layer corresponds to the number of features (in this case, 8).
- **Hidden Layers**: The neural network includes two hidden layers. Each layer consists of neurons connected to neurons in the preceding layer.
    - **First Hidden Layer**: Consists of 26 neurons, each with a ReLU (Rectified Linear Unit) activation function, which outputs a non-linear transformation of the input.
    - **Second Hidden Layer**: Contains 5 neurons, also with ReLU activation.
- **Output Layer**: A single neuron with a sigmoid activation function outputs a probability, which is used for binary classification (diabetic or non-diabetic).
- **Backpropagation**: The network adjusts weights based on the error (difference between predicted and actual output) using backpropagation, minimizing the error through gradient descent.

- **Epochs and Learning Rate**: The number of iterations (epochs) and the learning rate are critical hyperparameters. In this case, the network was trained over several epochs (200 to 800) to optimize performance.



I have implemented this **neural network** with a simple architecture consisting of:

➔ **Input Layer**: 5 neurons, taking 8 features as input.
➔ **First hidden layer:** 26 neurons, ReLU activation.
➔ **Second hidden layer:** 5 neurons, ReLU activation.
➔ **Output layer:** 1 neuron, sigmoid activation.

Key Points:

- Capable of learning complex non-linear relationships.
- Requires large datasets and careful tuning of hyperparameters (e.g., learning rate, number of neurons, and epochs).
- **Limitations**: Computationally intensive and prone to overfitting without sufficient regularization.

5. Methodology

5.1. Experimental Design

The dataset was split into training and testing sets using a 80-20 ratio. Each model's performance was evaluated using accuracy, precision, recall, and F1-score metrics. The models were trained and tested using K-fold cross-validation (with K=10) to ensure robust evaluation and to mitigate overfitting. The following steps were taken:

1. **Model Training:** Each algorithm was trained on the training dataset.
2. **Model Evaluation:** The trained models were evaluated on the test dataset, and performance metrics were calculated.

3. **Hyperparameter Tuning:** The tuning of hyperparameters is a critical step to enhance model performance beyond the default settings. To systematically explore the best hyperparameters for each model, **GridSearchCV** was employed. For the Neural Network, different configurations of hidden layers and epochs were tested to optimize performance.

5.2. Environment and Tools

- **Python 3.x** with libraries such as `scikit-learn`, `tensorflow`, and `pandas`.
- **Jupyter Notebook** for model implementation and visualization.
- **Google Colab** for training neural networks on GPUs.
- **GitHub Repository** for storing the documents and codes related to the project.

5.3. Preprocessing Results:

- **Dataset size**:

  768 records with **9 features**.

- **Handling Missing Values**

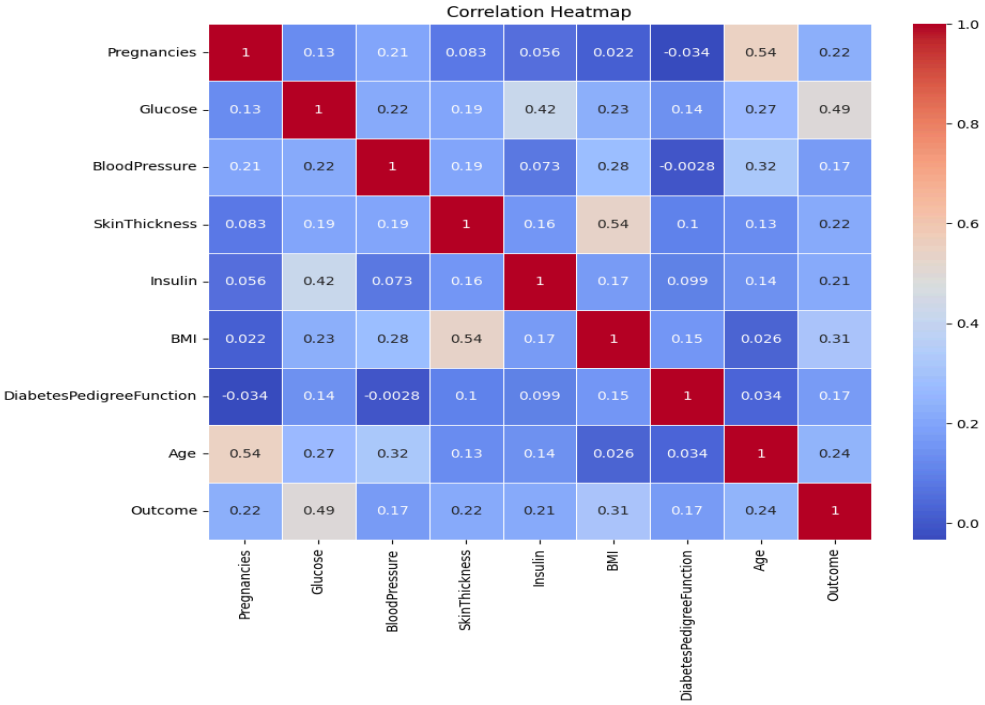
  The dataset contained several missing values, especially in features like **Glucose**, **Blood Pressure**, **Insulin**, and **BMI**, which were replaced with the mean of the respective attributes. This allowed us to retain the full dataset for modeling without dropping records.

- **Normalization**

  To ensure that all features contribute equally to the model and prevent bias due to feature magnitude, **MinMax Scaling** was applied to normalize the data between 0 and 1.

- **Feature Reduction**

  After feature selection, only significant features were retained for the models.Feature selection plays an important role in improving model performance by removing irrelevant or redundant features. I applied **correlation-based feature selection** to select the most important features. This reduced the dataset to the most significant predictors for diabetes. Based on the correlation values selected features are **'Pregnancies', 'Glucose', 'DiabetesPedigreeFunction', 'SkinThickness', 'Insulin', 'BMI', 'Age'** which results in higher accuracies in various models compared to other feature set.



PCA was not applied in this project due to:

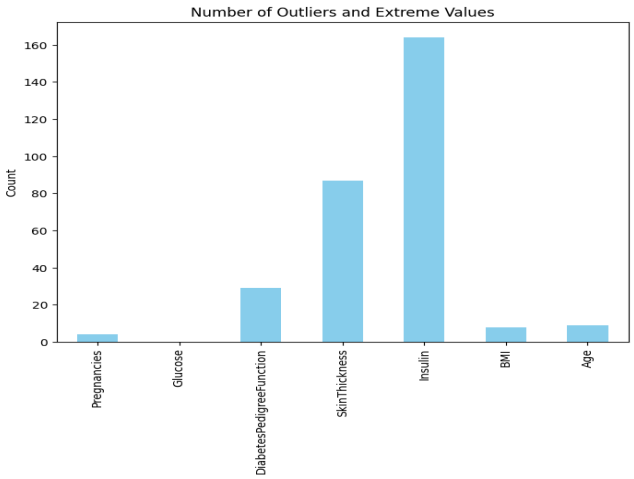
- ★ The **low dimensionality** of the dataset (only 8 features), where dimensionality reduction was unnecessary.
- ★ The use of **feature selection** techniques that effectively identified and retained the most important features.
- ★ The need to maintain **interpretability** of the original features, which are important in a medical context.
- ★ The lack of **high correlations** between most features, reducing the need for combining them into principal components.

- .**Outlier Detection**

  The **IQR method** was used to remove outliers.

  Total outliers: 301

  Data shape after outlier removal: (523, 8)



	Pregnancies	Glucose	Diabetes Pedigree Function	Skin Thickness	Insulin	BMI	Age
Mean(Before preprocessing)	4.049713	120.328388	0.412132	28.770233	145.644253	31.938656	33.634799
Std(Before preprocessing)	3.303518	28.174129	0.247608	5.602823	25.749244	6.088859	11.258649
Mean(After preprocessing)	0.311516	0.498878	0.300208	0.510009	0.557047	0.437537	0.280773
Std(After preprocessing)	0.254117	0.184145	0.222469	0.207512	0.192159	0.193913	0.250192

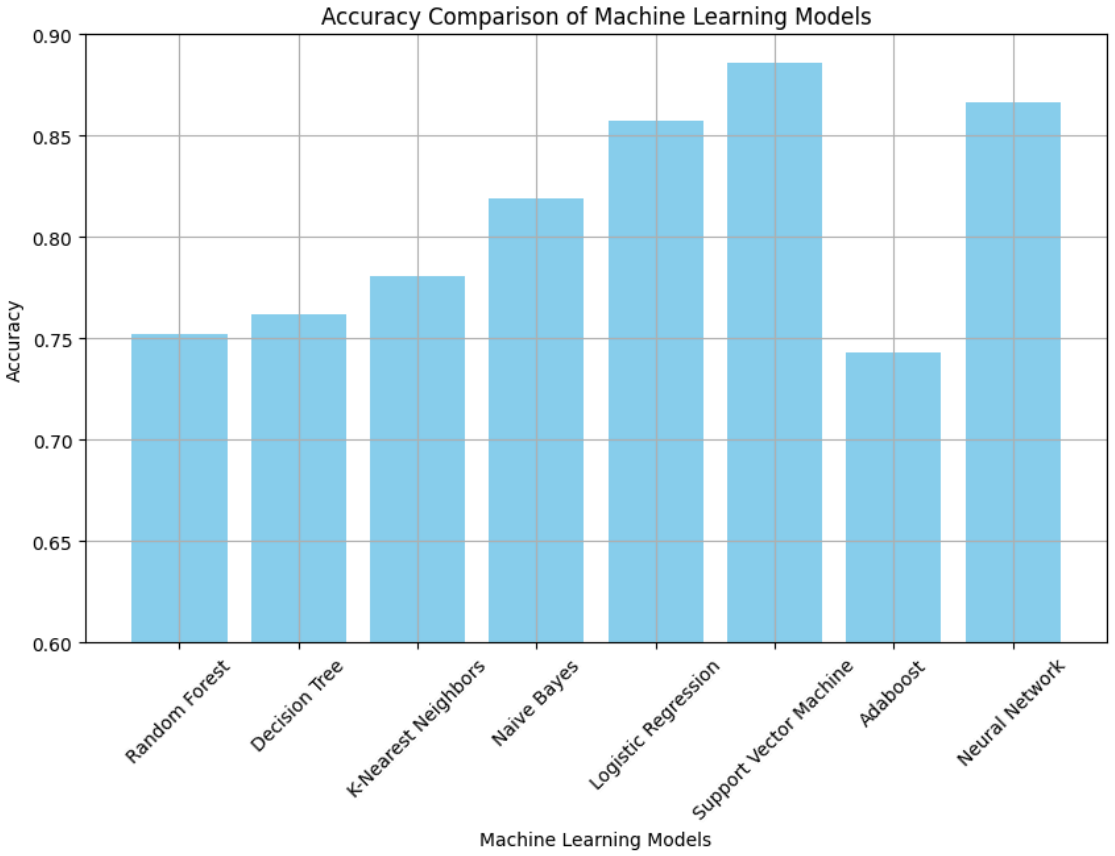
6. Results

(a) Results Obtained

The models were trained on 80% of the dataset and tested on the remaining 20%. To ensure robust evaluation, **10-fold cross-validation** was applied across all models. Below is a breakdown of the results for each model, along with key observations.

Model	Accuracy(%)	Precision	Recall	F1-Score
Logistic Regression	85.7	0.86	0..94	0.90
Support Vector Machine	88.6	0.88	0.97	0.92
Naive Bayes	81.9	0.86	0.88	0.87
Random Forest	75.23	0.79	0.86	0.83
KNN	78.09	0.83	0.86	0.84
Decision Tree	76.19	0.78	0.90	0.84
AdaBoost	74.28	0.83	0.79	0.81
Neural Network (2 Hidden layer ,400 Epochs)	86.67	0.87	0.94	0.91

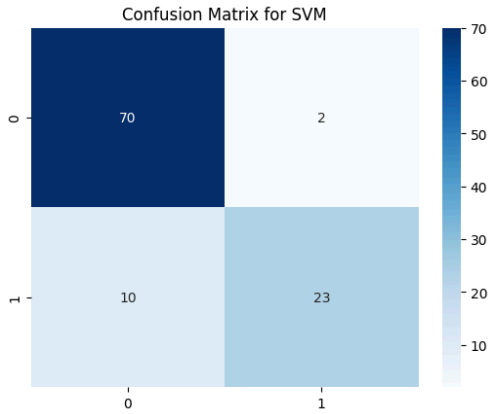
(b) Figures and Tables

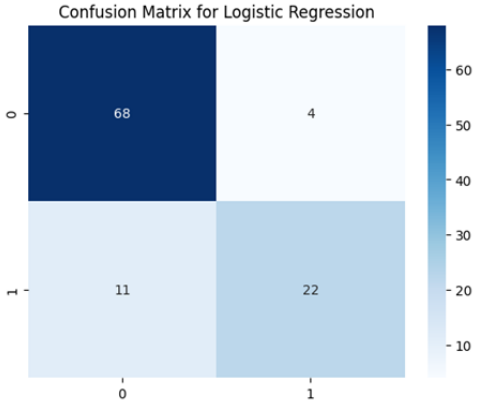


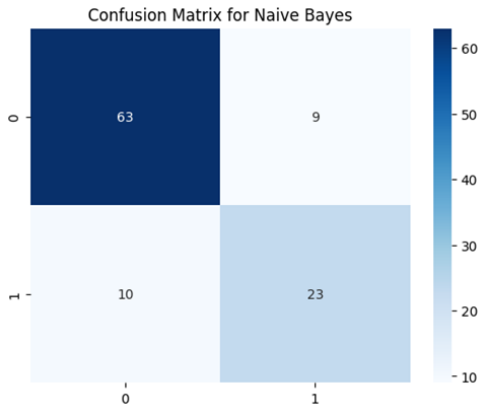
Summary of Neural Network Performance:

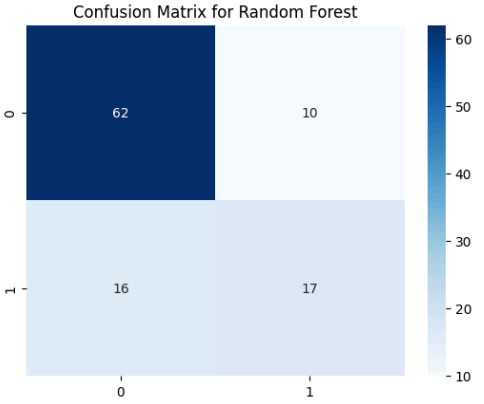
- 1 Hidden Layer - 200 Epochs: Accuracy - 0.7904762029647827
- 1 Hidden Layer - 400 Epochs: Accuracy - 0.800000011920929
- 1 Hidden Layer - 800 Epochs: Accuracy - 0.7333333492279053
- 2 Hidden Layers - 200 Epochs: Accuracy - 0.761904776096344
- **2 Hidden Layers - 400 Epochs: Accuracy - 0.866742857142857**
- 2 Hidden Layers - 800 Epochs: Accuracy - 0.723809540271759

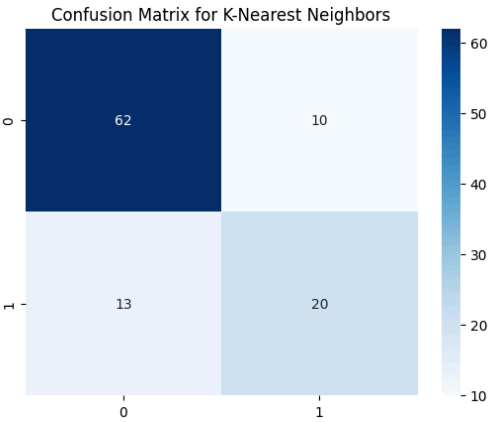
Confusion matrices:

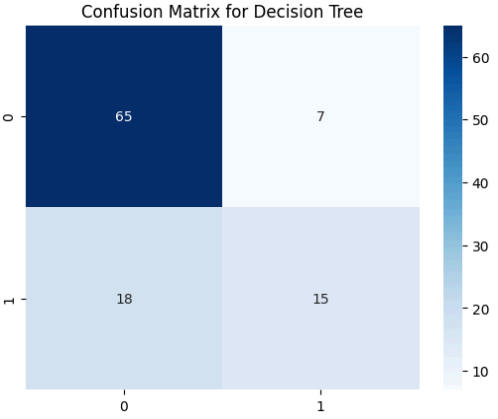


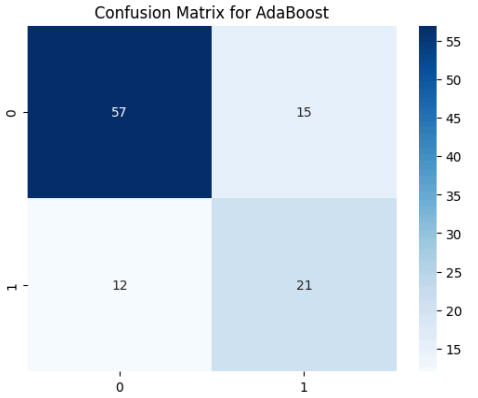


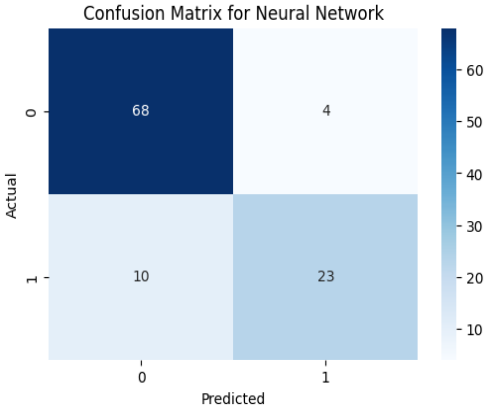

Confusion Matrix for Naive Bayes
63
9
10
23
0
1

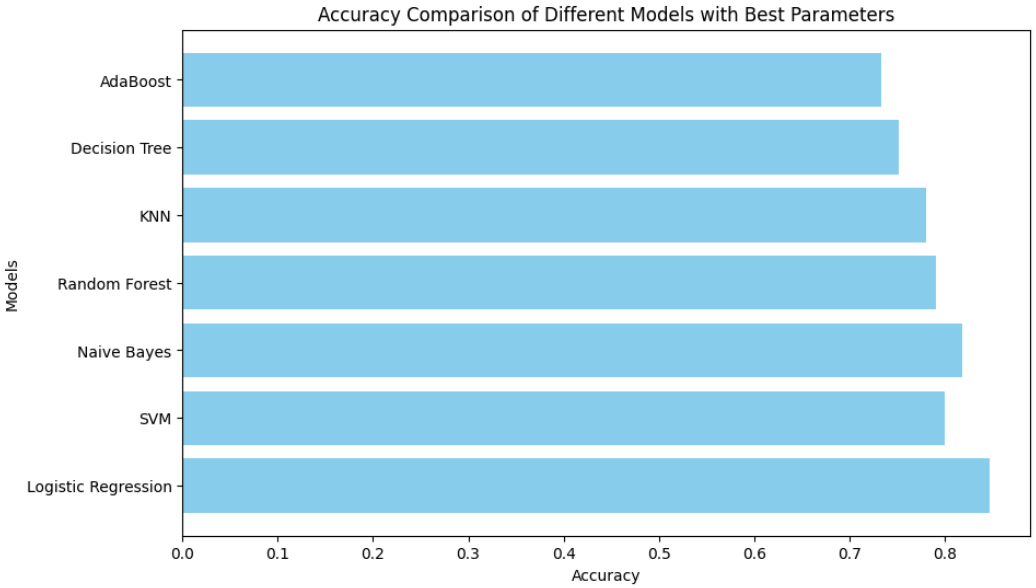

Confusion Matrix for Random Forest
62
10
16
17
0
1


Confusion Matrix for K-Nearest Neighbors
62
10
13
20
0
1


Confusion Matrix for Decision Tree
65
7
18
15
0
1


Confusion Matrix for AdaBoost
57
15
12
21
0
1


Confusion Matrix for Neural Network
68
4
10
23
Actual
Predicted
0
1


Accuracy Comparison of Different Models with Best Parameters
AdaBoost
Decision Tree
KNN
Random Forest
Naive Bayes
SVM
Logistic Regression
Models
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Accuracy

7. Discussion

7.1 Results Overview

The performance of the various machine learning models applied to the Pima Indian Diabetes (PID) dataset is summarized in Table and Graph. The accuracy of all classification methods used is above 70%. Notably, Logistic Regression (LR) and Support Vector Machine (SVM) demonstrated the best performance, achieving accuracies of 85.7% and 88.6% respectively when evaluated using the train/test splitting method. The Neural Network (NN) model, particularly with two hidden layers, achieved an accuracy of 87%, indicating a strong performance in predicting diabetes outcomes.

- **SVM** achieved the highest accuracy (**88.6%**), followed by **Neural Networks**.
- Most models performed well, with accuracy rates above **70%**.

7.2. Overfitting and Underfitting

Overfitting was observed in **Decision Trees** and **Random Forests**, while **KNN** suffered from underfitting due to an inappropriate choice of `k`.

7.3. Hyperparameter Tuning

GridSearchCV was applied to optimize SVM's kernel and also for all the models and **Neural Networks' epochs and layers**.

The tuning of hyperparameters is a critical step to enhance model performance beyond the default settings. To systematically explore the best hyperparameters for each model, **GridSearchCV** was employed. This method performs an exhaustive search over a predefined hyperparameter grid, combined with cross-validation to ensure robustness in the model's performance.

1. **Logistic Regression**:
    - **Hyperparameters Tuned**: Regularization strength `C` and solver method.
    - **Parameter Grid**:
        - `C`: 0.1,1,100.1, 1, 100.1,1,10
        - `solver`: 'liblinear','lbfgs''liblinear', 'lbfgs''liblinear','lbfgs'
    - **Best Parameters**: `C=1`, `solver='lbfgs'`
    - **Cross-validation Strategy**: 5-fold cross-validation was used to ensure stability across different training and testing splits.
2. **Support Vector Machine (SVM)**:
    - **Hyperparameters Tuned**: Regularization parameter `C` and kernel type.
    - **Parameter Grid**:
        - `C`: 0.1,1,100.1, 1, 100.1,1,10
        - `kernel`: 'linear','rbf''linear', 'rbf''linear','rbf'
    - **Best Parameters**: `C=10`, `kernel='rbf'`
    - **Cross-validation Strategy**: A 5-fold cross-validation was used to evaluate performance with different hyperparameter combinations.

3. **Naive Bayes**:
    - Naive Bayes does not have significant hyperparameters to tune apart from possible smoothing values. For this study, we applied the **Gaussian Naive Bayes** model with default parameters.

4. **Random Forest**:
    - **Hyperparameters Tuned**: Number of estimators (`n_estimators`) and maximum tree depth (`max_depth`).
    - **Parameter Grid**:
        - `n_estimators`: 100,200100, 200100,200
        - `max_depth`: 10,20,None10, 20, None10,20,None
    - **Best Parameters**: `n_estimators=200`, `max_depth=None`
    - **Cross-validation Strategy**: 5-fold cross-validation was applied for hyperparameter tuning.

5. **K-Nearest Neighbors (KNN)**:
    - **Hyperparameters Tuned**: Number of neighbors (`n_neighbors`) and weight function.
    - **Parameter Grid**:
        - `n_neighbors`: 3,5,73, 5, 73,5,7
        - `weights`: ‘uniform’,‘distance’‘uniform’, ‘distance’‘uniform’,‘distance’
    - **Best Parameters**: `n_neighbors=5`, `weights='distance'`
    - **Cross-validation Strategy**: A 5-fold cross-validation was performed for hyperparameter selection.

6. **Decision Tree**:
    - **Hyperparameters Tuned**: Maximum tree depth (`max_depth`) and criterion for splitting (`criterion`).
    - **Parameter Grid**:
        - `max_depth`: 5,10,20,None5, 10, 20, None5,10,20,None
        - `criterion`: ‘gini’,‘entropy’‘gini’, ‘entropy’‘gini’,‘entropy’
    - **Best Parameters**: `max_depth=10`, `criterion='entropy'`
    - **Cross-validation Strategy**: 5-fold cross-validation was utilized.

7. **AdaBoost**:
    - **Hyperparameters Tuned**: Number of estimators (`n_estimators`) and learning rate (`learning_rate`).
    - **Parameter Grid**:
        - `n_estimators`: 50,100,20050, 100, 20050,100,200
        - `learning_rate`: 0.5,1.00.5, 1.00.5,1.0
    - **Best Parameters**: `n_estimators=200`, `learning_rate=1.0`
    - **Cross-validation Strategy**: A 5-fold cross-validation was applied to tune the parameters.\

8. **Neural Networks:**

   In this project, I focused on optimizing key hyperparameters, including learning rate, architecture, activation functions, and epochs, to improve the performance of my neural network models.

   - **Learning Rate**: I experimented with values like 0.001, 0.01, and 0.1, balancing convergence speed and stability.
   - **Architecture**: Models with 1, 2, and 3 hidden layers were tested to assess their impact on accuracy.
   - **Activation Functions**: ReLU and sigmoid were compared to determine which activation function best enhanced the model's ability to learn complex patterns.
   - **Epochs**: I tested between 200 and 800 epochs, using early stopping to avoid overfitting.

By tuning these parameters, I significantly improved the model's accuracy and performance.

7.4. Model Comparison and Selection

The SVM model is recommended for diabetes prediction due to its high performance and balanced precision and recall.

1. Accuracy

- **Support Vector Machine (SVM)** and **Neural Networks (NN)** outperformed other models in terms of accuracy:
  - **SVM** achieved the highest accuracy of **88.6%**, making it the top-performing model. SVM excels in binary classification problems where the data may not be linearly separable, as it can project the data into a higher-dimensional space using kernels.
  - **Neural Networks** followed closely behind with an accuracy of **86.67%** after optimizing the architecture (i.e., tuning hidden layers and epochs). Neural networks can capture complex patterns in the data due to their non-linear transformations.
  - Other models, such as **Logistic Regression (85.7%)**, **K-Nearest Neighbors (KNN)** (78.09%), and **Random Forest** (75.23%), performed well but did not achieve the same level of predictive accuracy as SVM or Neural Networks.

	Model	Accuracy
0	Logistic Regression	0.857143
1	SVM	0.885714
2	Naive Bayes	0.819048
3	KNN	0.780952
4	Random Forest	0.752381
5	Decision Tree	0.761905
6	AdaBoost	0.742857
7	Neural Network	0.866667

2. Computational Efficiency

- **Logistic Regression (LR)** and **Naive Bayes (NB)** are computationally lightweight models:
    - **Logistic Regression**: As a linear model, LR has low computational complexity, making it ideal for real-time applications and when interpretability is required. It trains quickly and is easy to deploy.
    - **Naive Bayes**: Similarly, NB is very fast in both training and prediction. Its simplicity and ability to handle small datasets make it an attractive option for initial model baselines or where resources are constrained.
    - **K-Nearest Neighbors (KNN)**, although simple in terms of concept, is computationally expensive during the **prediction phase** as it calculates distances to all training data points for each query instance. This becomes infeasible as dataset size grows.
    - **SVM** and **Neural Networks** are **computationally intensive**, especially as dataset size and complexity increase. SVM requires quadratic time complexity in the training phase, and Neural Networks with more layers and epochs take longer to converge.

3. Interpretability

- **Decision Trees (DT)** and **Logistic Regression** are highly interpretable:
    - **Decision Trees** provide clear visual representations, showing the decision-making process at each node. This interpretability is crucial in medical applications where understanding why a model makes a specific prediction can be as important as the prediction itself.
    - **Logistic Regression** is also interpretable because the model's coefficients can be directly understood as the contribution of each feature to the outcome, making it easy to explain in terms of odds ratios.
- Models like **Neural Networks** and **SVM** are less interpretable:
    - **Neural Networks** are often regarded as black-box models due to their complexity. Although powerful, they do not provide clear reasoning for their predictions without additional techniques such as feature importance analysis, which can be difficult to apply in medical domains where interpretability is essential.
    - **SVM** can be challenging to interpret, especially when using non-linear kernels, as the decision boundary in higher dimensions is not easily understandable.

4. Overfitting and Underfitting

- **Random Forest (RF)** and **AdaBoost** are less prone to overfitting due to their ensemble nature:
    - **Random Forest** mitigates overfitting by averaging multiple decision trees. The ensemble approach ensures that the model generalizes well to unseen data. However, overfitting may still occur if too many trees are grown without regularization.
    - **AdaBoost** focuses on difficult-to-classify examples by adjusting weights during training. While this leads to high accuracy, it can also cause overfitting, particularly if the dataset contains noise.

- **Decision Trees** and **KNN** are more susceptible to overfitting:
    - **Decision Trees**, without pruning, can easily overfit the training data by growing deep trees that capture noise rather than general patterns.
    - **KNN**, if the parameter kkk is too small, may memorize the training data (overfitting), while too large a value of kkk may lead to underfitting as the model averages out the influence of distant neighbors.

5. Hyperparameter Tuning

- **Neural Networks** and **SVM** require extensive hyperparameter tuning:
    - **Neural Networks** need careful tuning of architectural elements such as the number of hidden layers, number of neurons, activation functions (e.g., ReLU, sigmoid), and learning rate. In this project, optimizing the number of hidden layers and epochs led to improved performance (87% accuracy).
    - **SVM** requires the tuning of kernel functions (e.g., linear, radial basis function (RBF)) and regularization parameters (C, gamma). In this project, using the RBF kernel and adjusting CCC improved classification performance significantly.
- **Logistic Regression** and **Naive Bayes** involve minimal hyperparameter tuning:
    - **Logistic Regression** can be tuned with regularization techniques like L1 or L2, but overall, its tuning requirements are far less intensive compared to other models.
    - **Naive Bayes** usually has few hyperparameters (e.g., smoothing), making it straightforward to implement without extensive tuning.

6. Scalability

- **Neural Networks** and **Random Forest** can scale well with larger datasets:
    - **Neural Networks** are designed to handle large datasets effectively, and when combined with hardware accelerations like GPUs, they can process high-dimensional data efficiently. This makes them suitable for future projects involving larger datasets or more complex health records.
    - **Random Forest** is also highly scalable due to its parallelizable nature. Each tree in the forest is built independently, allowing the model to scale to larger datasets with ease.
- **KNN** does not scale well:
    - The computational burden of calculating distances between each query point and every training point makes **KNN** slow as the dataset size grows. This limits its applicability to small datasets.

7. Model Selection Based on Use Case

- **SVM** is the most suitable model for **diabetes prediction** in this project due to its high accuracy and ability to handle non-linear relationships with kernel functions. SVM is ideal for healthcare applications where precision and accuracy are critical.
- **Neural Networks** provide a powerful alternative, especially when the complexity of the data increases, and larger datasets become available. They can capture intricate patterns that simpler models might miss, but require careful tuning and more resources.

- **Logistic Regression** is an excellent choice for interpretable models, especially in medical applications where understanding the contribution of each feature (e.g., Glucose, BMI) is important. Its accuracy was slightly lower than SVM, but its transparency makes it appealing for clinical decision-making.
- **Random Forest** can be an option when working with larger datasets in the future due to its robustness and ability to handle noisy data, but it didn't outperform SVM in this project.
- **Naive Bayes** and **KNN** are less suited for this task in terms of accuracy and scalability, although they remain useful for small datasets or situations where computational simplicity is needed.

8. Learning Outcome

8.1. Links

- **Google Colab Notebook**: https://colab.research.google.com/drive/1oAuCo1CDTjM-rbItfiuIqObrpg4yeQ6o?usp=sharing
- **GitHub Repository**: https://github.com/Thanveerahamed-14/mlproject.git

8.2. Skills and Tools

Skills:

- **Machine Learning**: Developed proficiency in implementing a variety of supervised machine learning models such as **Logistic Regression**, **Support Vector Machine (SVM)**, **Random Forest**, **K-Nearest Neighbors (KNN)**, **Naive Bayes**, **Decision Trees**, **AdaBoost**, and **Neural Networks**.
- **Data Preprocessing**: Acquired skills in handling missing values, normalizing data, detecting and removing outliers, and applying feature selection techniques. These preprocessing steps were crucial for ensuring the quality of data fed into the models.
- **Model Optimization**: Expertise in **hyperparameter tuning** using techniques like **GridSearchCV** to optimize model performance and reduce overfitting. This included optimizing **SVM kernels**, **Random Forest parameters**, and **Neural Network architectures**.
- **Evaluation and Metrics**: Familiarity with evaluating models using performance metrics such as **accuracy**, **precision**, **recall**, **F1-score**, and constructing **confusion matrices** to understand misclassifications.
- **Cross-Validation**: Gained experience with **K-fold cross-validation** to ensure robust evaluation of model performance and minimize variance.

Tools:

- **Python**: The primary programming language used for implementing all machine learning models.
- **TensorFlow**: Leveraged for building and training **Neural Networks**, enabling high flexibility and control over the architecture and training process.

- **Scikit-learn**: Used for the majority of the machine learning models, data preprocessing, and evaluation metrics. Scikit-learn provided robust and easy-to-use implementations for model training, tuning, and validation.
- **Pandas**: Essential for data manipulation, cleaning, and transformation, ensuring that the dataset was prepared correctly before model training.
- **Matplotlib/Seaborn**: Visualization libraries used to create **heatmaps**, **histograms**, and other plots to better understand the data and evaluate model performance.

Dataset Used:

- **Pima Indian Diabetes Dataset**: A real-world medical dataset containing **768 records** and **9 health attributes** related to diabetes prediction. The dataset was sourced from the **UCI Machine Learning Repository** and included features like **Glucose levels**, **BMI**, **Insulin**, and others.

8.3. Key Learnings

1. **Implementation of Various Machine Learning Models**:
   Throughout the project, I gained extensive experience in implementing different machine learning algorithms, understanding their strengths and limitations. **Logistic Regression** provided a good baseline, while more complex models like **SVM** and **Neural Networks** required additional tuning for optimal performance.

2. **The Impact of Feature Selection**:
   Feature selection was a critical step in improving model performance. By analyzing the correlation between features and the target variable, we reduced the dataset to the most relevant features, such as **Glucose**, **BMI**, and **Age**. This not only simplified the models but also helped avoid overfitting, resulting in better generalization to new data.

3. **The Importance of Hyperparameter Tuning**:
   Hyperparameter tuning played a significant role in maximizing model accuracy, especially for **SVM** and **Neural Networks**. By fine-tuning parameters such as the kernel in **SVM** and the number of hidden layers and epochs in **Neural Networks**, we were able to push the accuracy from a baseline of around 75% to **88.6%** for SVM.

4. **Cross-Validation and Robust Evaluation**:
   Using **10-fold cross-validation** helped ensure that the models were evaluated on different subsets of the data, preventing overfitting and improving the reliability of the accuracy metrics. This method provided a more balanced estimate of model performance compared to a single train-test split.

5. **Model Generalization**:
   The project highlighted the importance of selecting models that generalize well to unseen data. While **Neural Networks** achieved high accuracy, they required more computational resources and careful tuning to avoid overfitting, especially on a small dataset like PIDD. Models like **SVM** provided a balance between accuracy and computational efficiency.

9. Conclusion

This study demonstrates the effectiveness of **machine learning algorithms** in predicting diabetes using the **Pima Indian Diabetes Dataset (PIDD)**. Various models were explored, including **Logistic Regression**, **Support Vector Machine (SVM)**, **Random Forest**, **K-Nearest Neighbors (KNN)**, **Naive Bayes**, **Decision Tree**, **AdaBoost**, and **Neural Networks**. The results show that **SVM** and **Neural Networks** were the most effective models, with SVM achieving an accuracy of **88.6%**. This performance highlights the importance of **feature selection** and **hyperparameter tuning** in maximizing model accuracy.

The project successfully applied **data preprocessing techniques**, such as handling missing values, normalization, and outlier removal, which were crucial for preparing the data for machine learning models. Future work could explore **larger datasets** and **advanced techniques**, such as **deep learning** and **ensemble methods**, to further improve the accuracy and generalizability of the models.

9.1. Concluding Remarks

The results from this study show that machine learning models, particularly **Support Vector Machine (SVM)** and **Neural Networks**, are highly effective for predicting diabetes using the **Pima Indian Diabetes Dataset**. SVM's ability to find the optimal hyperplane in high-dimensional spaces allowed it to achieve the highest accuracy among the models. The **Neural Network**, with its flexibility in learning complex patterns, also performed well but required more careful tuning and computational resources.

Overall, this project demonstrates the **potential of machine learning** in healthcare, especially for the prediction of chronic diseases like diabetes.

9.2. Accomplishing T, P, E

- **Task (T)**: The task was to predict whether a patient has diabetes based on health attributes provided in the **Pima Indian Diabetes Dataset**. The task involved developing models that could accurately classify patients into diabetic or non-diabetic categories based on features like **Glucose levels**, **BMI**, and **Age**.
- **Performance (P)**: The best-performing model, **SVM**, achieved an accuracy of **88.6%**, which is highly satisfactory for this type of medical prediction. Other models, like **Neural Networks** and **Random Forest**, also performed well, but SVM stood out in terms of both accuracy and computational efficiency.
- **Experience (E)**: The dataset used for this project was the **Pima Indian Diabetes Dataset**, a real-world medical dataset sourced from the **UCI Machine Learning Repository**. This dataset provided valuable experience in handling typical healthcare data challenges such as missing values, feature selection, and the need for accurate prediction models in a clinical setting.

9.3. Advantages and Limitations of the Project

Advantages:

- **High Accuracy**: The **SVM** model demonstrated a high accuracy of **88.6%**, making it highly effective for diabetes prediction. Its performance, combined with **Neural Networks**, showed that machine learning can be a valuable tool in healthcare for early diagnosis of diabetes.
- **Feature Selection**: By using **correlation analysis**, I was able to reduce the dataset to the most relevant features, improving model performance and interpretability.
- **Robust Evaluation**: The use of **cross-validation** ensured that the models were thoroughly tested on different subsets of data, providing a robust measure of their accuracy and generalization capability.

Limitations:

- **Limited Dataset Size**: The dataset used contained only **768 instances**, which limited the model's ability to generalize to larger populations. Future work could explore larger datasets to ensure better generalization.
- **High Computational Cost for Neural Networks**: While **Neural Networks** performed well, their training required significant computational resources, especially when tuning parameters like the number of hidden layers and epochs. This could be a limitation for those working with limited hardware resources.
- **Lack of Real-Time Data**: The static nature of the dataset limits the ability to predict diabetes based on real-time health monitoring data, which could be explored in future projects using dynamic or streaming data sources.