1.Project Report:

Title-diabetes detection project

Course-vityarthi project

Student name-prakhar paliwal

Faculty name- DR Monika Vyas

Registration no-25BCE10523

Instituition- vit Bhopal university

Year-2025

1.5.INDEX

Sr. No.	Section
1	Problem Statement
2	Objectives
3	System Design (Flowchart & Use Case)
4	Working & Explanation
5	Full Source Code
6	Output Screenshots
7	Conclusion
8	References

Pima Indian Diabetes Prediction using Support Vector Machine (SVM)

2. Introduction

This project is my first real look at how machine learning can be used in the real world, specifically in the field of predictive healthcare. The main goal was to create a strong classification model that could predict how likely it was that a person had diabetes based on a few important diagnostic factors.

I used the Pima Indian Diabetes Dataset, which is a well-known benchmark dataset that has information on female patients. The Support Vector Machine (SVM) is a powerful supervised learning algorithm that is the best tool for this classification task. The main part of this project was going through the whole machine learning process: cleaning and preparing raw data, standardizing it (a very important step), training the model, and finally, testing it on completely new test data to see how well it worked.

3. Problem Statement

The goal for this project is to detect diabetes in patients having symptons and detect these using machine learning model which helps prevent further risk and is a early indication for it.

4. working and explaination

This uses large level machine learning modes and train on dataset and like sugar level, insulin,fat percentage etc and thus helps users to predict the diabetes in them and achieves this through libraries like numpy,panda, sctlearn etc.

5. project source code

In [26]:
```
# Accuracy score on the test data

X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [27]:
```
print('Accuracy of test data: ', test_data_accuracy)
```

```
Accuracy of test data:  0.7727272727272727
```

Making a Predictive System

In [31]:
```
input_data = (1,85,66,29,0,26.6,0.351,31)


# Changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array as we are predicting for one instance. Our model is defined
# for 768 data points. So, we need to reshape it to make it to predict for one.

input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# Standardize the input data
std_data = scaler.transform(input_data_reshaped)

prediction = classifier.predict(std_data)

if prediction[0] == 0:
    print('Person is not Diabetic.')
else:
    print('Person is Diabetic.')
```

```
Person is not Diabetic.
```

In [16]:
```
X = standardized_data
# Y remains the labels
```

In [17]:
```
# splitting the data into train and test dataset

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, stratify = Y, random_state = 2)
```

In [18]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

Training the model

In [21]:
```
classifier = svm.SVC(kernel = 'linear')

# SVC means Support Vector Classifier
# We are using linear model
```

In [22]:
```
# training the support vector machine classifier

classifier.fit(X_train, Y_train)
```

Out[22]:
```
SVC(kernel='linear')
```

Model Evaluation

Accuracy Score

In [23]:
```
# Accuracy score on the training data

X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [24]:
```
print('Accuracy of training data: ', training_data_accuracy)
```

```
Accuracy of training data:  0.7866449511400652
```

In [11]:

```
diabetes_dataset.groupby('Outcome').mean()
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Outcome								
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

In [12]:

```
# Separating the data and label

X = diabetes_dataset.drop(columns = 'Outcome', axis = 1)
Y = diabetes_dataset['Outcome']
```

Data Standardization

In [13]:

```
scaler = StandardScaler()
```

In [14]:

```
# scaler.fit_transform() does two things:
# 1. It fits all the inconsistent data with our standard scaler function.
# 2. Based on above, we are transforming all the data to a common range.
standardized_data = scaler.fit_transform(X)
```

In [15]:

```
print(standardized_data)

# Here, it can be verified that all these values are in the range of 0 and 1. This will help the model to make the better pr
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

In [8]:
```
# number of rows and columns in the dataframe
diabetes_dataset.shape
```

Out[8]: (768, 9)

In [9]:
```
# Getting the statistical measures of the data
diabetes_dataset.describe()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [10]:
```
# Total number of diabetic and non-diabetic people
diabetes_dataset['Outcome'].value_counts()
```

Out[10]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

0 --> Non-diabetic 1 --> Diabetic

Importing the dependencies

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler  # Will be used to standardize the data to a common range
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Data Collection and Analysis

PIMA Diabetes Dataset (Females only)

In [6]:
```
# Loading the diabetes dataset to a pandas dataframe

diabetes_dataset = pd.read_csv('/content/diabetes dataset.csv')
```

In [7]:
```
# Printing first 5 rows of the dataset

diabetes_dataset.head()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

6.5. Design Decisions & Rationale

Decision	Rationale (Why I chose this)
Support Vector Classifier (SVC)	I chose the SVC because it is known to be effective in high-dimensional spaces and for finding clear, robust boundaries between classes, which is exactly what we need for binary classification.
StandardScaler Usage	This was the most crucial decision. Without standardization, the features with larger scales (like Insulin or Glucose) would completely dominate the features with smaller scales (like Diabetes Pedigree Function), leading to a biased and poor model.
kernel = 'linear'	I started with a linear kernel for simplicity. Since I am a beginner, it's easier to interpret the model and check if the classes are linearly separable before moving to more complex non-linear kernels like RBF.
stratify = Y in Splitting	The dataset is imbalanced (500 non-diabetic vs. 268 diabetic). Using stratify=Y ensures that both the training and testing sets maintain this same ratio, giving us a more honest evaluation of the model's true performance.

6.7. Implementation Details

The entire project was implemented in **Python 3.x** and developed within a **Jupyter Notebook**.

The key libraries and their roles are:

Library	Role in Project	Specific Functions Used

Pandas	Data loading and manipulation (DataFrame creation).	pd.read_csv(), .head(), .shape, .drop()
NumPy	Numerical computations and array handling for prediction.	np.asarray(), .reshape()
Scikit-learn	The machine learning framework.	StandardScaler, train_test_split, svm.SVC, accuracy_score

8. conclusion

This projects demonstrates the use of ai to successfully tackle a real life problem.The diabetes detector covers data storage,input validation and statistics. It is simple,lightweight and usefull for daily productivity.

9. References

1. **Pima Indian Diabetes Dataset:** Sourced from the UCI Machine Learning Repository (commonly accessed via Kaggle).
2. **Scikit-learn Official Documentation:** Used extensively for understanding the parameters and usage of StandardScaler, train_test_split, and svm.SVC.
3. **Pandas Library Guide:** Referenced for data manipulation commands and handling the DataFrame structure.
4. **NumPy Reference:** Used to understand array reshaping for single-point prediction.