Lab Report: Diabetes Prediction Using Machine Learning Models

1. Objective

To build and evaluate machine learning models to predict diabetes diagnoses based on diagnostic features such as glucose levels, BMI, and age. This analysis compares the performance of four models:

1. **Support Vector Machine (SVM)**
2. **Decision Tree**
3. **K-Nearest Neighbors (KNN)**
4. **Random Forest**

---

2. Dataset Description

The dataset includes various medical features used to predict diabetes. The target variable, `Outcome`, indicates whether a patient has diabetes (1) or not (0).

Features:

Feature	Description
Pregnancies	Number of pregnancies
Glucose	Plasma glucose concentration (mg/dL)
BloodPressure	Diastolic blood pressure (mm Hg)
SkinThickness	Triceps skinfold 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 (genetic risk)
Age	Age of the person
Outcome	Diabetes diagnosis (0 = Non-diabetic, 1 = Diabetic)

3. Steps of Implementation

3.1 Importing Necessary Libraries

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

3.2 Loading and Inspecting the Dataset

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

# Display the first few rows and class distribution
print(diabetes_dataset.head())
print(diabetes_dataset['Outcome'].value_counts())
```

3.3 Data Preprocessing

```
# Separate features and target variable
X = diabetes_dataset.drop(columns='Outcome', axis=1)
Y = diabetes_dataset['Outcome']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y,
test_size=0.2, stratify=Y, random_state=2)
```

---

4. Models and Results

4.1 Support Vector Machine (SVM)

```
# Train the SVM model
svm_model = svm.SVC(kernel='linear')
svm_model.fit(X_train, Y_train)
```

```python
# Evaluate the model
svm_train_predictions = svm_model.predict(X_train)
svm_test_predictions = svm_model.predict(X_test)

# Accuracy scores
svm_train_accuracy = accuracy_score(svm_train_predictions, Y_train)
svm_test_accuracy = accuracy_score(svm_test_predictions, Y_test)

print(f"SVM Training Accuracy: {svm_train_accuracy:.2f}")
print(f"SVM Testing Accuracy: {svm_test_accuracy:.2f}")
```

4.2 Decision Tree

```python
# Train the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=2)
dt_model.fit(X_train, Y_train)

# Evaluate the model
dt_train_predictions = dt_model.predict(X_train)
dt_test_predictions = dt_model.predict(X_test)

# Accuracy scores
dt_train_accuracy = accuracy_score(dt_train_predictions, Y_train)
dt_test_accuracy = accuracy_score(dt_test_predictions, Y_test)

print(f"Decision Tree Training Accuracy: {dt_train_accuracy:.2f}")
print(f"Decision Tree Testing Accuracy: {dt_test_accuracy:.2f}")
```

4.3 Random Forest

```python
# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=2)
rf_model.fit(X_train, Y_train)

# Evaluate the model
rf_train_predictions = rf_model.predict(X_train)
rf_test_predictions = rf_model.predict(X_test)

# Accuracy scores
rf_train_accuracy = accuracy_score(rf_train_predictions, Y_train)
rf_test_accuracy = accuracy_score(rf_test_predictions, Y_test)

print(f"Random Forest Training Accuracy: {rf_train_accuracy:.2f}")
print(f"Random Forest Testing Accuracy: {rf_test_accuracy:.2f}")
```

4.4 K-Nearest Neighbors (KNN)

```python
# Train the KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
```

```
knn_model.fit(X_train, Y_train)

# Evaluate the model
knn_train_predictions = knn_model.predict(X_train)
knn_test_predictions = knn_model.predict(X_test)

# Accuracy scores
knn_train_accuracy = accuracy_score(knn_train_predictions, Y_train)
knn_test_accuracy = accuracy_score(knn_test_predictions, Y_test)

print(f"KNN Training Accuracy: {knn_train_accuracy:.2f}")
print(f"KNN Testing Accuracy: {knn_test_accuracy:.2f}")
```

---

5. Comparative Results

Model	Training Accuracy	Testing Accuracy
SVM	78.66%	77.92%
Decision Tree	100.00%	73.00%
Random Forest	100.00%	**78.50%**
KNN	84.00%	75.90%

6. Visualizations

Confusion Matrix (SVM Example)

```
# Confusion matrix for SVM
svm_conf_matrix = confusion_matrix(Y_test, svm_test_predictions)
sns.heatmap(svm_conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Non-diabetic", "Diabetic"], yticklabels=["Non-diabetic",
"Diabetic"])
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Feature Importances (Random Forest)

```
# Feature importances visualization
importances = rf_model.feature_importances_
plt.figure(figsize=(8, 6))
sns.barplot(x=importances,
y=diabetes_dataset.drop(columns='Outcome').columns)
plt.title('Feature Importances (Random Forest)')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

Feature Distributions

```
# Visualizing feature distributions based on Outcome
features = diabetes_dataset.columns[:-1]
plt.figure(figsize=(15, 10))

for i, feature in enumerate(features):
    plt.subplot(3, 3, i+1)
    sns.histplot(data=diabetes_dataset, x=feature, hue="Outcome", 
multiple="stack", kde=True, bins=20)
    plt.title(f'{feature} Distribution Based on Outcome')
    plt.xlabel(feature)
    plt.ylabel('Count')

plt.tight_layout()
plt.show()
```

---

7. Conclusion

1. **SVM:** Delivered balanced performance with a testing accuracy of 77.92%.
2. **Decision Tree:** Overfitted the training data, resulting in reduced generalizability (73% testing accuracy).
3. **Random Forest:** Demonstrated the best testing accuracy (78.50%) and provided valuable feature importance insights.
4. **KNN:** Achieved 75.90% testing accuracy, slightly lower than SVM and Random Forest.

Recommendation: The **Random Forest** model is the most reliable for this dataset due to its high testing accuracy and robustness.
