Diabetes Prediction Using Machine Learning Models

Chetan Gowda

February 24, 2025

Abstract

The increasing prevalence of diabetes presents a global health challenge. Early prediction of diabetes can lead to timely medical intervention, reducing the impact of the disease. This report explores the application of machine learning models—K-Nearest Neighbors (KNN), Random Forest (RF), and Support Vector Machines (SVM)—to predict the likelihood of diabetes based on a set of medical attributes. We use a publicly available diabetes dataset and apply various preprocessing techniques, including feature engineering and scaling. The models are evaluated based on their accuracy, precision, recall, and F1-score. The results show that Random Forest outperforms the other models in terms of accuracy and other metrics.

Contents

1. Introduction

Diabetes is a chronic medical condition where the body fails to regulate blood glucose (sugar) levels effectively. Over time, this can lead to complications such as cardiovascular disease, kidney failure, and nerve damage. Early detection is crucial to managing and preventing these complications. This project leverages machine learning techniques to predict the likelihood of a person having diabetes based on multiple features, including age, body mass index (BMI), blood pressure, insulin levels, and others.
The aim of this report is to demonstrate the process of data preprocessing, model training, hyperparameter tuning, evaluation, and comparison of different machine learning models. We explore three different models—K-Nearest Neighbors (KNN), Random Forest (RF), and Support Vector Machine (SVM)—and evaluate their performance on a diabetes dataset.

2. Data Preprocessing

Data preprocessing is a crucial step in the machine learning pipeline. In this section, we detail the steps taken to prepare the dataset for analysis, which include loading the data, cleaning, feature engineering, and scaling.

2.1 Loading the Dataset

The dataset used in this project is the Pima Indians Diabetes Database, which contains several medical attributes related to diabetes diagnosis. The dataset is loaded using the pandas library.

```
import pandas as pd
data = pd.read_csv("Downloads/ML_Activity/diabetes.csv")
```

2.2 Feature Engineering

Feature engineering helps enhance model performance by creating new features that may better capture relationships in the data. In this project, two new features are created:

- `BMI_squared`: The square of BMI, capturing non-linear effects of body mass index.
- `Age_Glucose_interaction`: The interaction between age and glucose, assuming a potential joint effect on diabetes risk.

The code for feature engineering is as follows:

```
data['BMI_squared'] = data['BMI'] ** 2
data['Age_Glucose_interaction'] = data['Age'] * data['
    Glucose']
```

2.3 Target and Features Separation

The target variable `Outcome` is the indicator of whether the individual is diabetic (1) or not (0). The features (input variables) are stored in `x`, and the target variable is stored in `y`.

```
y = data.Outcome
x_before = data.drop(["Outcome"], axis=1)
```

2.4 Feature Scaling

Since some machine learning algorithms, such as KNN and SVM, are sensitive to the scale of the data, it is important to scale the features. StandardScaler is used to scale the features to have a mean of 0 and a standard deviation of 1.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x = scaler.fit_transform(x_before)
```

2.5 Train-Test Split

The dataset is split into a training set (80% of the data) and a test set (20% of the data) using the `train_test_split` function.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2, random_state=1)
```

3. Model Selection and Hyperparameter Tuning

Three machine learning models are explored in this project: K-Nearest Neighbors (KNN), Random Forest (RF), and Support Vector Machines (SVM). We also perform hyperparameter tuning to optimize each model.

3.1 K-Nearest Neighbors (KNN)

KNN is a simple, instance-based learning algorithm where the classification of an instance depends on the majority class among its nearest neighbors. The key hyperparameter in KNN is `n_neighbors`, which defines the number of neighbors to consider.
We use GridSearchCV to find the best values for `n_neighbors` and the distance metric (`p`). The model is initialized with the 'distance' weight, meaning closer neighbors are given higher weight.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

knn = KNeighborsClassifier(weights='distance')
param_grid_knn = {
    'n_neighbors': range(1, 21),
    'p': [1, 2]
}
grid_search_knn = GridSearchCV(knn, param_grid_knn, cv=5,
   scoring='accuracy', n_jobs=-1, verbose=1)
grid_search_knn.fit(x_train, y_train)
best_knn = grid_search_knn.best_estimator_
```

3.2 Random Forest (RF)

Random Forest is an ensemble learning method that uses multiple decision trees to improve prediction accuracy. The hyperparameters tuned include:

- `n_estimators`: The number of trees in the forest.
- `max_depth`: The maximum depth of each tree.
- `min_samples_split`: The minimum number of samples required to split an internal node.
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node.

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(random_state=1)
param_grid_rf = {
    'n_estimators': [100, 200],
```

```
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
grid_search_rf = GridSearchCV(rf_model, param_grid_rf, cv
   =5, scoring='accuracy', n_jobs=-1, verbose=1)
grid_search_rf.fit(x_train, y_train)
best_rf = grid_search_rf.best_estimator_
```

3.3 Support Vector Machines (SVM)

SVM is a powerful model for binary classification. The hyperparameters tuned include:

- `C`: Regularization parameter.
- `kernel`: The kernel function ('linear' or 'rbf').
- `gamma`: Kernel coefficient.

```
from sklearn.svm import SVC

svm_model = SVC(random_state=1)
param_grid_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
grid_search_svm = GridSearchCV(svm_model, param_grid_svm,
   cv=5, scoring='accuracy', n_jobs=-1, verbose=1)
grid_search_svm.fit(x_train, y_train)
best_svm = grid_search_svm.best_estimator_
```

4. Model Evaluation and Results

The performance of each model is evaluated using the following metrics:

- **Accuracy**: The percentage of correctly predicted instances.
- **Precision**: The proportion of true positives among all positive predictions.
- **Recall**: The proportion of true positives among all actual positives.
- **F1-Score**: The harmonic mean of precision and recall.

4.1 Confusion Matrix

A confusion matrix is a useful tool for evaluating the performance of a classification model. It provides a summary of the prediction results, showing true positives, false positives, true negatives, and false negatives.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm_knn = confusion_matrix(y_test, y_pred_knn)
cm_rf = confusion_matrix(y_test, y_pred_rf)
cm_svm = confusion_matrix(y_test, y_pred_svm)

plt.figure(figsize=(12, 4))

# KNN
plt.subplot(1, 3, 1)
sns.heatmap(cm_knn, annot=True, fmt="d", cmap="Blues",
    xticklabels=["Healthy", "Diabetic"], yticklabels=["
    Healthy", "Diabetic"])
plt.title("Confusion Matrix for KNN")
plt.xlabel("Predicted")
plt.ylabel("Actual")

# Random Forest
plt.subplot(1, 3, 2)
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Blues",
    xticklabels=["Healthy", "Diabetic"], yticklabels=["
    Healthy", "Diabetic"])
plt.title("Confusion Matrix for Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")

# SVM
plt.subplot(1, 3, 3)
sns.heatmap(cm_svm, annot=True, fmt="d", cmap="Blues",
    xticklabels=["Healthy", "Diabetic"], yticklabels=["
    Healthy", "Diabetic"])
plt.title("Confusion Matrix for SVM")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

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

4.2 Classification Reports

The classification reports for each model provide detailed metrics like precision, recall, and F1-score. Below is the code for generating classification reports:

```
from sklearn.metrics import classification_report

print("Classification Report for KNN:\n",
    classification_report(y_test, y_pred_knn))
print("Classification Report for Random Forest:\n",
    classification_report(y_test, y_pred_rf))
print("Classification Report for SVM:\n",
    classification_report(y_test, y_pred_svm))
```

4.3 Comparison of Model Performance

The following table compares the performance of each model across different metrics.

Model	Accuracy	Precision	Recall	F1-Score
KNN	0.74	0.73	0.77	0.75
Random Forest	0.78	0.79	0.80	0.79
SVM	0.75	0.75	0.75	0.75

Table 1: Comparison of Model Performance

5. Conclusion

This project demonstrates the use of machine learning models—K-Nearest Neighbors, Random Forest, and Support Vector Machines—to predict diabetes based on a variety of health-related features. Through preprocessing, feature engineering, and model tuning, we have achieved reasonable performance in terms of accuracy and other metrics.

Among the models tested, Random Forest performed the best, with the highest accuracy and F1-score. However, all models showed promising results, suggesting that machine learning can be an effective tool for early diabetes prediction.