Diabetes Prediction using Machine Learning Models

CHETAN GOWDA

February 24, 2025

1. Introduction

This project aims to predict whether an individual has diabetes based on medical attributes such as BMI, age, and glucose levels, among others. We employ three different machine learning models: K-Nearest Neighbors (KNN), Random Forest, and Support Vector Machine (SVM). The objective is to assess their performance, fine-tune their hyperparameters, and evaluate the models based on various metrics such as accuracy, precision, recall, and F1-score.

2. Libraries and Dependencies

The following libraries are used in this project:

- `pandas`: For data manipulation and reading CSV files.
- `matplotlib.pyplot`: For data visualization.
- `numpy`: For numerical computations.
- `sklearn.model_selection`: Contains functions for splitting the data and hyperparameter tuning.
- `sklearn.preprocessing`: For scaling the features.
- `sklearn.neighbors, sklearn.ensemble, sklearn.svm`: Contain machine learning models (KNN, Random Forest, and SVM).

- `sklearn.metrics`: For evaluating the model's performance.
- `seaborn`: For advanced visualization like heatmaps.

3. Data Preprocessing

3.1 Loading the Dataset

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

This command reads the diabetes dataset from the given path into a pandas DataFrame.

3.2 Feature Engineering

To improve model performance, new features are created based on existing features. For example:

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

- `BMI_squared`: A new feature created by squaring the BMI value, capturing non-linear relationships between BMI and diabetes.
- `Age_Glucose_interaction`: This feature is the product of `Age` and `Glucose` to capture any joint effect between the two on diabetes.

3.3 Preparing Features and Target Variable

The target variable `y` (outcome) is separated from the features `x`:

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

3.4 Feature Scaling

Feature scaling is performed using `StandardScaler` to ensure that each feature has a mean of 0 and a variance of 1:

```
scaler = StandardScaler()
x = scaler.fit_transform(x_before)
```

3.5 Train-Test Split

The data is split into training and testing sets using an 80-20 split:

```
x_train, x_test, y_train, y_test = train_test_split(x,
    y, test_size=0.2, random_state=1)
```

This ensures that the model is trained on 80% of the data and evaluated on the remaining 20%.

4. Model Implementation and Hyperparameter Tuning

4.1 K-Nearest Neighbors (KNN)

KNN is initialized, and a hyperparameter tuning process is conducted using `GridSearchCV` to find the best number of neighbors and distance metric:

```
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_
```

4.2 Random Forest

Random Forest model is initialized, and hyperparameter tuning is performed for parameters such as the number of trees, maximum depth, and more:

```
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_
```

4.3 Support Vector Machine (SVM)

SVM is initialized, and hyperparameter tuning is performed for parameters such as regularization and kernel:

```
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_
```

4.4 Model Predictions

The models are used to predict the target values on the test data:

```
y_pred_knn = best_knn.predict(x_test)
y_pred_rf = best_rf.predict(x_test)
y_pred_svm = best_svm.predict(x_test)
```

5. Evaluation and Visualization

5.1 Confusion Matrix

Confusion matrices are plotted to evaluate the models' performance. The following code generates the confusion matrix for KNN:

```
cm_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(6, 6))
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")
plt.show()
```

This code is repeated for the Random Forest and SVM models, generating their respective confusion matrices.

5.2 Classification Report

A classification report is generated for each model. The following code prints the classification report for KNN:

```
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))
```

6. Conclusion

This project demonstrates the process of building, fine-tuning, and evaluating machine learning models for diabetes prediction. By comparing the KNN, Random Forest, and SVM models using confusion matrices and classification reports, we are able to assess their performance in terms of various metrics. The use of hyperparameter tuning via `GridSearchCV` allows for optimal model performance. Future work could involve further model optimization and the exploration of additional algorithms for improved prediction.