جامعة أم القرى
UMM AL-QURA UNIVERSITY

MACHINE LEARNING (1)

Assignment 2

Introduction

The objective of this assignment is to apply cross-validation for model selection using the Diabetes Dataset from sklearn.datasets. We will explore the dataset, preprocess the data, and compare Logistic Regression and KNN classification models.

Using k-fold cross-validation, we will evaluate model performance based on accuracy and metrics such as precision, recall, F1 score, and ROC-AUC. Finally, we will tune the best model's hyperparameters to enhance its predictive accuracy.

Overview of the Diabetes Dataset

The Diabetes Dataset is a well-known dataset available in the sklearn.datasets module
It contains information about various health indicators that are used to predict the onset of diabetes in patients.

The dataset consists of 10 numerical features, including:

- **Age**
- **Sex**
- **Body mass index (BMI): Measure of body fat**
- **Blood pressure measurements: Average blood pressure**
- **Blood serum measurements (e.g., cholesterol levels)**

Data Exploration and Preprocessing

```
# explore the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   age                   442 non-null    float64
 1   sex                   442 non-null    float64
 2   bmi                   442 non-null    float64
 3   bp                    442 non-null    float64
 4   s1                    442 non-null    float64
 5   s2                    442 non-null    float64
 6   s3                    442 non-null    float64
 7   s4                    442 non-null    float64
 8   s5                    442 non-null    float64
 9   s6                    442 non-null    float64
 10  Diabetes Progression  442 non-null    float64
dtypes: float64(11)
memory usage: 38.1 KB
```

The command data.info() was executed to display a summary of the DataFrame, including the number of entries, data types, and non-null counts for each feature, to ensure the data is clean.

```
# Create a binary target variable
# where values above the mean of the the data target are 1, and others are 0
threshold = data['Diabetes Progression'].mean()

y = (y_target > threshold).astype(int)
data['Binary target'] = y
```

I calculated the mean of the target variable (Diabetes Progression), using the mean value as a threshold to categorize the target variable into two classes.

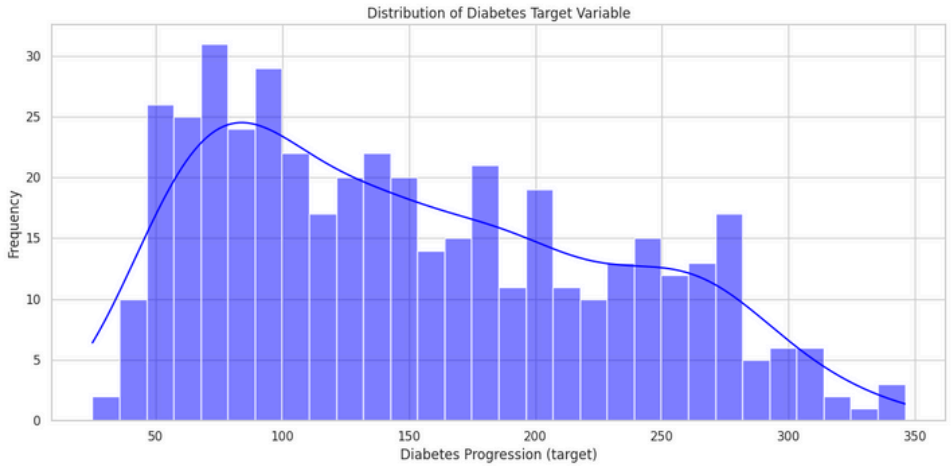
where values above the mean were classified as 1 (high diabetes progression), and values below or equal to the mean were classified as 0 (low diabetes progression).

```
# Calculate basic statistics
statistics = data.describe().rename(index={'50%': 'median'})
statistics
```

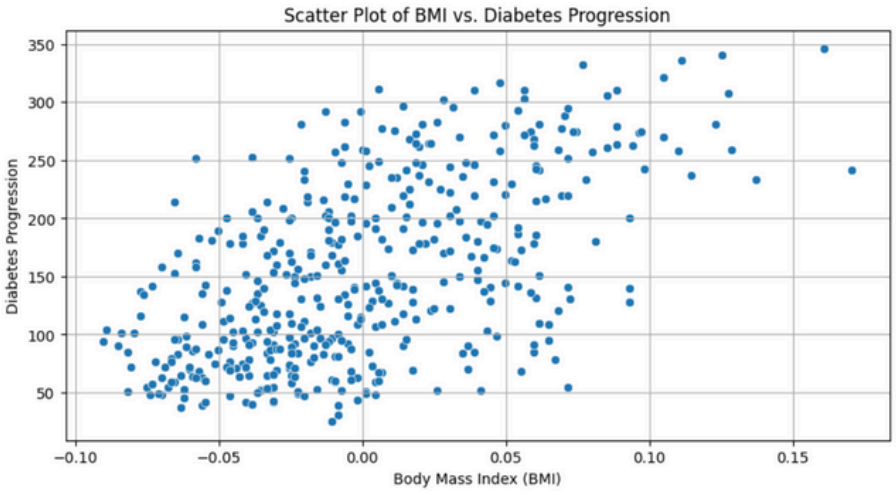
Finally, I calculated basic statistics for the dataset using data.describe(). This function provides a statistical summary of the numerical features
I also renamed the index from '50%' to 'median'

Data Visualizations

To better understand the data distribution, I created two visualizations: a histogram of BMI and a scatter plot between age and BMI.



The histogram indicates how many individuals in the dataset have a specific range of diabetes progression. This helps to identify whether most of the data points cluster around certain values or if they are spread out.



The scatter plot shows the likelihood of developing diabetes increases with a higher BMI.

Elevated BMI is often associated with excess body fat, particularly around the abdomen, which can lead to insulin resistance—a key factor in the development of diabetes.

Model Selection Using Cross-Validation for Classification

Split the data into training and testing sets (80% training, 20% testing).

```
[162] # Split the dataset 80% for training and 20% for testing
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

I chose Logistic Regression and K-Nearest Neighbors (KNN) as the models for this classification task.

- **Logistic Regression: A linear model commonly used for binary classification tasks.**
- **K-Nearest Neighbors (KNN): A non-linear model that classifies data points based on their proximity to neighboring points.**

Cross-Validation Implementation

K-fold cross-validation (k=5) was used to evaluate the models' performance.
The Results are:

- **Logistic Regression achieved a mean accuracy of 73.1%.**
- **KNN achieved a mean accuracy of 69.9%.**

Both models were trained on the training set using the entire 80% of the training data.

Performance Metrics
For both models, the following performance metrics were calculated:

1. **Accuracy: The proportion of correct predictions.**
2. **Precision: The proportion of true positives among all positive predictions.**
3. **Recall: The proportion of actual positives that were correctly predicted.**
4. **F1 Score: The harmonic mean of precision and recall, which balances the two.**
5. **ROC-AUC: Measures how well the model distinguishes between classes, with higher values indicating better performance.**

The Results are:
Model Comparison:

- **Logistic Regression:**
  - **Cross-validation accuracy: 0.7311 (73.1%)**
  - **Precision: 0.95 (95%)**
  - **Recall: 0.4871 (48.7%)**
  - **F1 Score: 0.6441 (64.4%)**
  - **ROC-AUC: 0.8174 (81.7%)**
- **K-Nearest Neighbors (KNN):**
  - **Cross-validation accuracy: 0.6998 (69.9%)**
  - **Precision: 0.7407 (74.1%)**
  - **Recall: 0.5128 (51.3%)**
  - **F1 Score: 0.6061 (60.6%)**
  - **ROC-AUC: 0.7477 (74.8%)**

The Logistic Regression model performed the best during cross-validation unlike KNN theat shows weaker performance in distinguishing between the classes.

Based on the cross-validation results Logistic Regression is the best model as it has the highest overall accuracy, precision, F1 scores and ROC-AUC. however, recall is lower for Logistic Regression. This means it is missing a significant number of positive instances compared to KNN.

Hyperparameter Tuning

After selecting Logistic Regression as the best model, to improve the performance of the Logistic Regression model, GridSearchCV was used to tune the hyperparameters. Specifically, the regularization strength (C) and the solver type were optimized.
The result is:
Best Hyperparameters:

- **C = 10, solver = 'liblinear'**

The tuned Logistic Regression model was tested again, and performance metrics were compared with the original model.

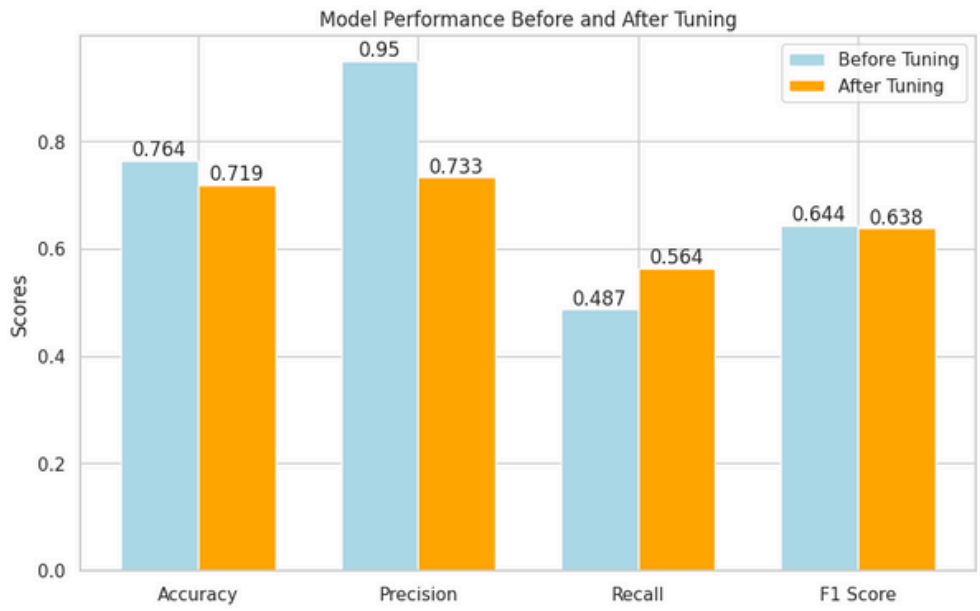
Results With 80-20 Data Split Before Tuning:Before Tuning:

- **Accuracy: 0.7640 (76.4%)**
- **Precision: 0.95 (95%)**
- **Recall: 0.4871 (48.7%)**
- **F1 Score: 0.6441 (64.4%)**

After Tuning:

- **Accuracy: 0.7191 (71.9%)**
- **Precision: 0.7333 (73.3%)**
- **Recall: 0.5641 (56.4%)**
- **F1 Score: 0.6377 (63.8%)**

To visualize the performance of the model, a bar plot was created comparing metrics such as accuracy, precision, recall, and F1 score for Logistic Regression both before and after tuning



Hyperparameter Tuning with 70-30 split

it is possible that the accuracy after hyperparameter tuning could be lower than the accuracy before tuning, and this can happen due to several reasons one of them is Overfitting:
Before tuning, the default hyperparameters may result in a model that generalizes reasonably well to both training and test sets, even though the model may not be very fine-tuned.

After tuning, hyperparameters could lead to overfitting the training data if the model becomes too complex.

To solve this problem:
During hyperparameter tuning (e.g., using GridSearchCV), having a larger test set can help fine-tune the model more accurately.
With more test data, you can identify overfitting earlier and optimize hyperparameters like regularization strength or the number of neighbors in KNN to ensure the model generalizes better.

The study (Trade-off between training and testing ratio in machine learning for medical image processing) shows underscores the importance of selecting an optimal train-test split ratio

"for training and how much is for testing. Typical proportions include a split of 70 percent for training and 30 percent for testing, or 80 percent for training and 20 percent for testing. However, determining the optimal ratio hinges on factors such as dataset size, complexity, and the intricacy of the model.
((A larger training set allows the model to learn more effectively)), while a good-sized testing set ensures a fair evaluation of the model's effectiveness. The right balance between the training and testing sets depends on what you want the machine learning project to achieve, how much data you have, and how complex your model is. The goal is to find a ratio that allows the model to learn well while also ensuring it can be tested accurately."

Reference link

This study shows that splits like 70-30 allow for a better balance between training and evaluation, ensuring that the model is learning more effectively, not overfitting to a small test set, which is particularly crucial when tuning hyperparameters.

So in this part the dataset is split into 70% for training and 30% for testing, instead of the earlier 80-20 split, to address possible overfitting and ensure better model evaluation.

```
# Split the dataset into 70% for training and 30% for testing
X_train_70, X_test_30, y_train_70, y_test_30 = train_test_split(X, y, test_size=0.3, random_state=42)
```

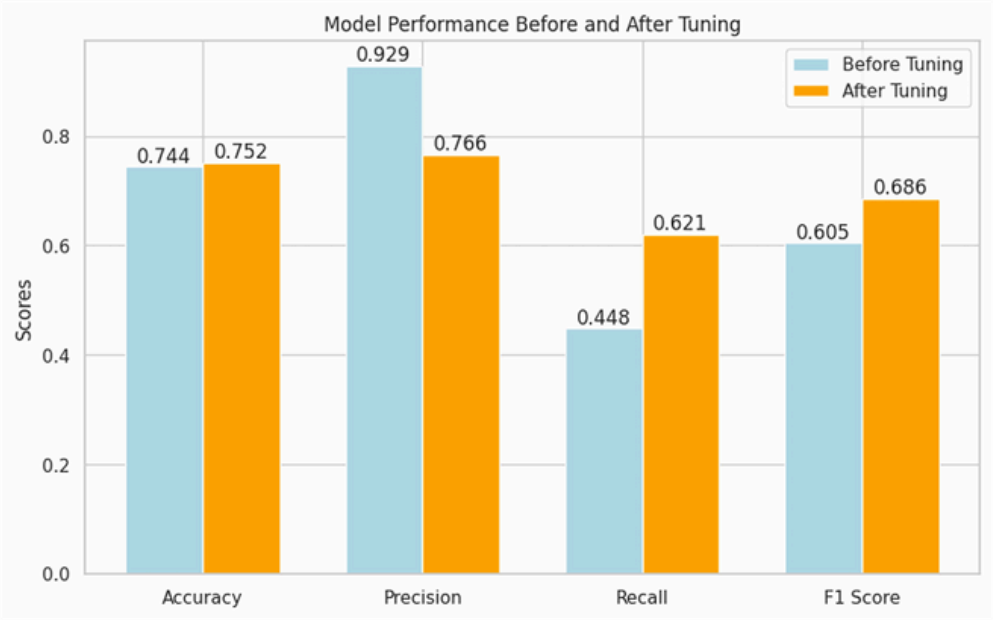
Using the same hyperparameter tuning process as before, GridSearchCV is applied on Logistic Regression with the 70-30 data split. This process ensures that the best hyperparameters are selected based on accuracy scores from 5-fold cross-validation.

The best hyperparameters are selected, and the tuned model is evaluated on the 30% test set using common metrics like accuracy, precision, recall, and F1 score.
The results for 70-30 Data Split are
Before Tuning:

- **Accuracy: 0.744 (74.4%)**
- **Precision: 0.929 (92.9%)**
- **Recall: 0.448 (44.8%)**
- **F1 Score: 0.605 (60.5%)**

After Tuning:

- **Accuracy: 0.7523 (75.2%)**
- **Precision: 0.7727 (77.3%)**
- **Recall: 0.5641 (56.4%)**
- **F1 Score: 0.6512 (65.1%)**



In result :

- **80-20 Split After tuning, accuracy slightly decreased from 76.4% to 71.9%. However, recall improved significantly from 48.7% to 56.4%, meaning the tuned model became better at identifying positive cases of diabetes progression.**
- **Precision dropped from 95% to 73.3%, indicating that the model became less accurate in predicting the positive cases, even though it detected more of them (higher recall). The F1 score remained relatively stable.**

however

- **70-30 Split Before tuning, accuracy was 74.4%, which increased slightly to 75.2% after tuning. The recall also improved significantly from 44.8% to 56.4%, meaning the model was better at detecting positive cases after tuning.**
- **Although precision decreased from 92.9% to 77.3%, this decrease was compensated by the better recall. The F1 score improved from 60.5% to 65.1%, indicating a better balance between precision and recall after tuning.**

In summary
the 70-30 split resulted in a more balanced model after tuning, with only a small improvement in accuracy but better overall performance in terms of recall and F1 score. Meanwhile, the 80-20 split saw a greater drop in accuracy but improved recall significantly.

Analysis and Reflection

1. Provide a description of the Diabetes dataset, including the dependent and independent variables.

1. Dependent Variable (Target):

Diabetes Progression (y_target or Diabetes Progression column):
This is the dependent variable or the target that we aim to predict.
It represents a quantitative measure of diabetes progression after one year, based on various clinical, physiological, and demographic variables.

2. Independent Variables (Features):

The dataset includes 10 independent variables (features) represent various factors that can influence diabetes progression:

1. **Age (age):**
   - **The age of the patient, normalized.**
2. **Sex (sex):**
   - **The sex of the patient, normalized (0 for female, 1 for male).**
3. **Body Mass Index (BMI, bmi):**
   - **A measure of body fat based on the patient's weight in relation to their height.**
4. **Blood Pressure (bp):**
   - **Mean arterial blood pressure**
5. **Serum Measurement 1 (s1):**
   - **This is a normalized measure of total cholesterol levels**
6. **Serum Measurement 2 (s2):**
   - **A normalized measure of low-density lipoproteins (LDL), also known as "bad cholesterol"**
7. **Serum Measurement 3 (s3):**
   - **A normalized measure of high-density lipoproteins (HDL), or "good cholesterol."**
8. **Serum Measurement 4 (s4):**
   - **A normalized measure of triglyceride levels, a type of fat in the blood**
9. **Serum Measurement 5 (s5):**
   - **A normalized measure of blood sugar (blood glucose level)**
10. **Serum Measurement 6 (s6):**

- **A normalized measure of insulin level, which indicates how the body is regulating blood sugar.**

2. Which classification model performed best during cross-validation? Discuss any challenges you faced in selecting the best model.

The Logistic Regression model performed the best during cross-validation compared to the K-Nearest Neighbors (KNN) classifier:

- **Higher Accuracy: Logistic Regression consistently achieved higher cross-validation accuracy compared to KNN (73.1% vs. 69.9%).**
- **Better Precision: Logistic Regression demonstrated significantly higher precision (95%) than KNN (74.1%), meaning it made fewer false-positive predictions.**
- **ROC-AUC Advantage: The ROC-AUC score of Logistic Regression (81.7%) indicates that it performed better in distinguishing between positive and negative classes compared to KNN (74.8%).**

Challenges in Selecting the Best Model:

Balancing Precision and Recall:

- **While Logistic Regression had higher precision, its recall (48.7%) was lower than KNN's recall (51.3%). This means that Logistic Regression was better at correctly identifying true positives (precision), but it missed a significant number of positive cases (lower recall).**

3. Did hyperparameter tuning improve the model's performance? Why do you think this is the case?

Yes, hyperparameter tuning did improve the model's performance in some aspects, but the improvement was limited to some Performance Metricsthe and the split ratio:

1. **Improved Recall:**

- **After tuning, the recall improved in both splits (from 48.7% to 56.4% in the 80-20 split and from 44.8% to 56.4% in the 70-30 split). This means the model became better at identifying more positive cases of diabetes progression, which is important for reducing false negatives.**

1. **Impact of Split Ratio (70-30 vs. 80-20):**

- **The 70-30 split provided a larger test set, allowing for a more robust evaluation of the model's performance. With more test data, the model can show a more realistic performance, which could explain why the tuned model's accuracy slightly improved in the 70-30 split, compared to the 80-20 split where accuracy dropped. The model had a larger set of examples to generalize on, which helped optimize the balance between precision and recall.**

Conclusion:
Hyperparameter tuning did improve the model's performance in terms of recall and F1 score, particularly in the 70-30 split. This improved the model's ability to detect positive cases (higher recall) and provided a better balance between precision and recall.
However, the overall accuracy decreased slightly in the 80-20 split, indicating a trade-off between model flexibility and generalization, possibly due to overfitting.
The 70-30 split provided a more balanced evaluation with a slight improvement in accuracy after tuning, but may not capture the full range of patterns in the data, which can reduce its predictive power and generalization ability.