MACHINE LEARNING

March 20, 2025

Name : KARAMAGI HENRY ATUHAIRE
Registration Number : 2023/U/MMU/BCS/00523

1 Initial Model

[158]:
```
# Importing the libraries

import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix
from joblib import parallel_backend

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix
```

1. Loading the Iris dataset

[159]:
```
# loading the dataset
df = pd.read_csv('iris.csv')
df.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',␣
↪'species']
```

[160]:
```
# Features
X = df.iloc[:, :-1].values

# Target
y = df.iloc[:, -1].values
```

2. Preprocessing the Dataset : Scaling the Features

[161]:
```
# Scaling the features
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
```

3. Split the Dataset into Training and Testing Sets

[162]:
```
# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X_scaled,  y,   test_size=0.
↪3,  random_state=42)
```

4. Training a Logistic Regression Classifier on the Training Set

[163]:
```
# The original model
original_model =   LogisticRegression(max_iter=200)
original_model.fit(X_train,    y_train)
```

[163]:
```
LogisticRegression(max_iter=200)
```

5. Model Evaluation

[164]:
```
# Model Evaluation
# make prediction
y_pred_original = original_model.predict(X_test)
accuracy_original = accuracy_score(y_test, y_pred_original)


# Calculating evaluation metrics for the original model
accuracy = accuracy_score(y_test, y_pred_original)
precision = precision_score(y_test, y_pred_original, average='weighted')
recall = recall_score(y_test, y_pred_original, average='weighted')
f1 = f1_score(y_test, y_pred_original, average='weighted')

# Results
print("Original Model Metrics")
print(f"Accuracy   :   {accuracy:.2f}")
print(f"Precision  :   {  precision:.2f}")
print(f"Recall     :   { recall:.2f}")
print(f"F1 Score   :   {f1:.2f}")
```

```
Original Model Metrics
Accuracy   :   0.91
Precision  :   0.92
Recall     :   0.91
F1 Score   :   0.91
```

2 Cross Validation of the Original model

[165]:
```python
cv_scores = cross_val_score(original_model,   X_scaled,y, cv=5)
print("Cross Validation with the Original Model")
print(f"C-V Scores: {cv_scores}")
print(f"Average C-V Accuracy: {cv_scores.mean():.2f}")
```

```
Cross Validation with the Original Model
C-V Scores: [0.96666667 1.         0.93333333 0.9        1.        ]
Average C-V Accuracy: 0.96
```

3 Optimization

3.0.1 Option 1: Parallel Processing

[166]:
```python
with parallel_backend('threading', n_jobs=-1):  # using all available CPU cores
    # Enabling parallel processing
    parallel_model = LogisticRegression(max_iter=200, n_jobs=-1)

    # fitting the data
    parallel_model.fit(X_train, y_train)


# Calculating evaluation metrics for the parallel processing model
accuracy_parallel = accuracy_score(y_test, y_pred_parallel)
precision_parallel = precision_score(y_test, y_pred_parallel, average='weighted')
recall_parallel = recall_score(y_test, y_pred_parallel, average='weighted')
f1_parallel = f1_score(y_test, y_pred_parallel, average='weighted')
conf_matrix_parallel = confusion_matrix(y_test, y_pred_parallel)

# Results
print("\nParallel Processing Model Metrics")
print(f"Accuracy   :    {accuracy_parallel:.2f}")
print(f"Precision  :    {precision_parallel:.2f}")
print(f"Recall     :    {recall_parallel:.2f}")
print(f"F1 Score   :    {f1_parallel:.2f}")
```

```
Parallel Processing Model Metrics
Accuracy   :    1.00
Precision  :    1.00
Recall     :    1.00
F1 Score   :    1.00
```

3.0.2 Option 2: Efficient Algorithm using Solver

[167]:
```python
# Using the solver paramaeter
efficient_model = LogisticRegression(solver='saga', max_iter=200)
efficient_model.fit(X_train, y_train)
```

```python
# Calculate evaluation metrics for the efficient algorithm model
accuracy_efficient = accuracy_score(y_test, y_pred_efficient)
precision_efficient = precision_score(y_test, y_pred_efficient,␣
↪average='weighted')
recall_efficient = recall_score(y_test, y_pred_efficient, average='weighted')
f1_efficient = f1_score(y_test, y_pred_efficient, average='weighted')
conf_matrix_efficient = confusion_matrix(y_test, y_pred_efficient)

# Print the results
print("\nSolver Model Metrics")
print(f"Accuracy   : {accuracy_efficient:.2f}")
print(f"Precision  : {precision_efficient:.2f}")
print(f"Recall     : {recall_efficient:.2f}")
print(f"F1 Score   : {f1_efficient:.2f}")
```

```
Solver Model Metrics
Accuracy   : 1.00
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00
```

3.0.3 Option 3: Sparse Matrices

[168]:
```python
# Converting to a sparse matrix
X_sparse = csr_matrix(X_scaled)
X_train_sparse, X_test_sparse, y_train, y_test = train_test_split(X_sparse, y,␣
↪test_size=0.3, random_state=42)

sparse_model = LogisticRegression(max_iter=200)
sparse_model.fit(X_train_sparse, y_train)

# Calculate evaluation metrics for the sparse matrices model
accuracy_sparse = accuracy_score(y_test, y_pred_sparse)
precision_sparse = precision_score(y_test, y_pred_sparse, average='weighted')
recall_sparse = recall_score(y_test, y_pred_sparse, average='weighted')
f1_sparse = f1_score(y_test, y_pred_sparse, average='weighted')
conf_matrix_sparse = confusion_matrix(y_test, y_pred_sparse)

# Print the results
print("\nSparse Matrices Model Metrics")
print(f"Accuracy   : {accuracy_sparse:.2f}")
print(f"Precision  : {precision_sparse:.2f}")
print(f"Recall     : {recall_sparse:.2f}")
print(f"F1 Score   : {f1_sparse:.2f}")
```

```
Sparse Matrices Model Metrics
```

```
Accuracy   : 1.00
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00
```

4 Comparison of the Performance of the models

[169]:
```
print("\nModel Performance Comparison")
print(f"Original Model Accuracy  : {accuracy_original:.2f}")
print(f"Parallel Model Accuracy  : {accuracy_parallel:.2f}")
print(f"Solver Model Accuracy    : {accuracy_efficient:.2f}")
print(f"Sparse Model Accuracy    : {accuracy_sparse:.2f}")
```

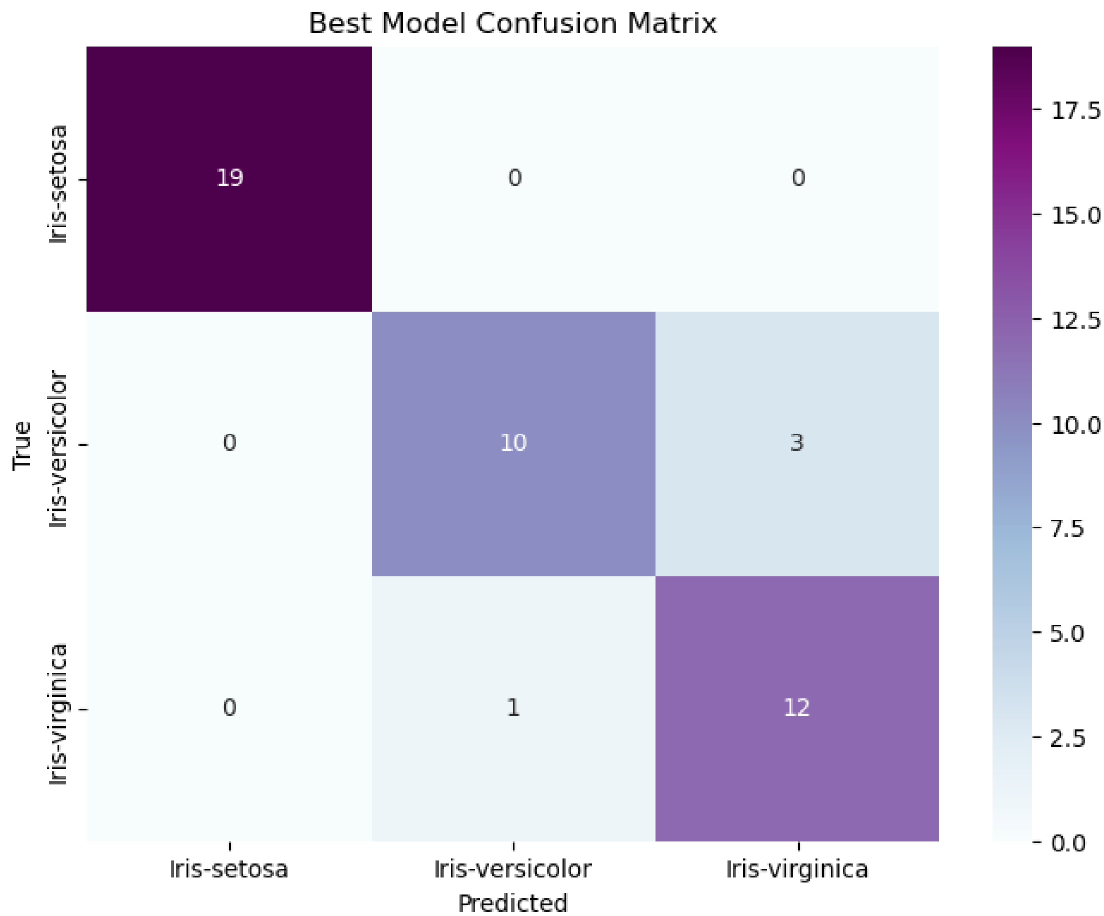
```
Model Performance Comparison
Original Model Accuracy  : 0.91
Parallel Model Accuracy  : 1.00
Solver Model Accuracy    : 1.00
Sparse Model Accuracy    : 1.00
```

5 Visualization

[170]:
```
# 1. selecting the model with the highest accuracy
best_model = parallel_model if accuracy_parallel >= max(accuracy_original,␣
 ↪accuracy_efficient, accuracy_sparse) else \
             efficient_model if accuracy_efficient >= max(accuracy_original,␣
 ↪accuracy_parallel, accuracy_sparse) else \
             sparse_model if accuracy_sparse >= max(accuracy_original,␣
 ↪accuracy_parallel, accuracy_efficient) else \
             original_model

y_pred_best = best_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred_best)
```

[171]:
```
# 2. Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True,fmt='d',   cmap='BuPu',
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Best Model Confusion Matrix')
plt.show()
```



6 Cross Validation of the Best model

[172]:
```
cv_scores = cross_val_score(best_model,   X_scaled,y, cv=5)
print("Cross Validation with the Best Model")
print(f"C-V Scores: {cv_scores}")
print(f"Average C-V Accuracy: {cv_scores.mean():.2f}")
```

```
Cross Validation with the Best Model
C-V Scores: [0.96666667 1.         0.93333333 0.9        1.        ]
Average C-V Accuracy: 0.96
```

[ ]:
```
```