Name : Manaswi Kabadi
Class : D15C
Roll no. : 27
Batch : B

EXP - 5

Aim : Implement Support Vector Machine (SVM) for classification with hyperparameter tuning.

Dataset Source : https://www.kaggle.com/datasets/wenruliu/adult-income-dataset

Dataset Description :

The Adult Income Dataset (Census Income dataset) is a real-world dataset derived from the 1994 U.S. Census. The objective is to predict whether an individual's annual income exceeds $50,000 based on demographic and employment-related features. It is well suited for binary classification using SVM, as it contains a large number of samples and multiple features that require proper preprocessing and feature scaling.

- File format: CSV
- Total records: ~48,000
- Number of attributes: 14
- Problem type: Binary classification

Dataset Attributes:

Column Name	Data Type	Description
age	Numerical (Integer)	Age of the client (years)
job	Categorical	Type of job
marital	Categorical	Marital status
education	Categorical	Education level
balance	Numerical	Average yearly balance (euros)
housing	Categorical	Housing loan status
loan	Categorical	Personal loan status
duration	Numerical	Duration of last contact (seconds)
campaign	Numerical	Number of contacts during campaign
y	Categorical (Binary)	Target variable (Subscription: Yes/No)

Target Variables Used in the Experiment

- **income**
    - 0 → Income less than or equal to 50K
    - 1 → Income greater than 50K

The target variable is used to classify individuals based on their annual income level.

Theory

Support Vector Machine (SVM) is a supervised machine learning algorithm primarily used for classification and, in some cases, regression. The main objective of SVM is to find an optimal decision boundary, called a hyperplane, that best separates data points belonging to different classes.Unlike other classifiers that focus only on minimizing classification error, SVM aims to maximize the margin, which is the distance between the hyperplane and the nearest data points from each class. These nearest data points are known as support vectors, and they play a crucial role in defining the position of the hyperplane.

SVM can handle both:

- Linearly separable data using a linear hyperplane, and
- Non-linearly separable data using the kernel trick, which maps data into a higher-dimensional space where separation becomes possible.

In this experiment, SVM is applied to the Adult Income dataset to classify whether an individual earns more than 50K or less than or equal to 50K based on demographic and employment-related features.

Mathematical Formulation of the Algorithm :

For a linearly separable dataset, SVM tries to find a hyperplane defined as:

$$w \cdot x + b = 0$$

Where:

- w is the weight vector
- x is the input feature vector
- b is the bias

The goal is to maximize the margin while correctly classifying all data points:

$$y_i(w \cdot x_i + b) \geq 1$$

Where:

- xi is the i-th data point
- yi∈{−1,+1} is the class label

Algorithm Limitations :

- **High Computational Cost**
  Support Vector Machines require significant computational resources, especially when using non-linear kernels such as the RBF kernel. Training time increases rapidly with large datasets like the Adult Income dataset.**Condition of limitation:** Performance slows down when the number of samples is very large.
- **Sensitivity to Feature Scaling**
  SVM is highly sensitive to the scale of input features. Features with larger numeric ranges can dominate the decision boundary if scaling is not applied.**Condition of limitation:** Without proper feature scaling, SVM may produce inaccurate results.
- **Difficulty in Choosing Hyperparameters**
  The performance of SVM depends heavily on hyperparameters such as C, kernel, and gamma. Improper selection of these parameters can lead to underfitting or overfitting. **Condition of limitation:** Requires careful tuning using techniques like GridSearchCV.
- **Limited Interpretability**
  Unlike Decision Trees or Linear Regression, SVM models do not provide easily interpretable rules or coefficients, making it difficult to explain predictions.**Condition of limitation:** Not suitable when model transparency is required.
- **Memory Intensive for Large Datasets**
  SVM stores support vectors in memory, which can be expensive for large datasets. **Condition of limitation:** High memory usage when many support vectors are selected.

Methodology / Workflow :

Data Collection

The Adult Income dataset was loaded from a CSV file. The dataset contains demographic and employment-related information used to predict whether an individual's annual income exceeds $50,000.

Target Identification

The target variable income was identified and converted into binary form, where 0 represents income less than or equal to 50K and 1 represents income greater than 50K. All remaining columns were treated as input features.

Data Preprocessing

Categorical attributes such as workclass, education, occupation, and marital status were encoded into numerical values. Missing values were handled to ensure consistency in training. Feature scaling was applied using StandardScaler, as SVM is sensitive to feature magnitudes.

Train–Test Split

The dataset was divided into 80% training data and 20% testing data using stratified sampling to maintain class distribution across both sets.

Model Training (Baseline SVM)

An initial Support Vector Machine classifier was trained using default hyperparameters to establish baseline performance.

Hyperparameter Tuning

Hyperparameters such as C, kernel, and gamma were optimized using GridSearchCV. Cross-validation was performed to identify the parameter combination that produced the best classification accuracy.

Final Model Training

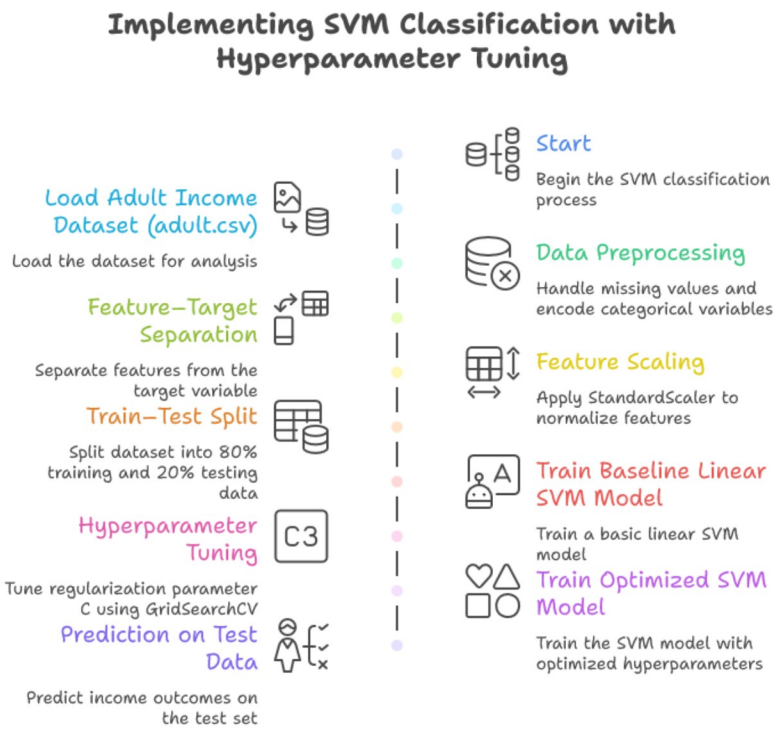
The SVM model was retrained using the optimal hyperparameters obtained from GridSearchCV.

Prediction

The optimized SVM model was used to predict income classes on the test dataset.

Model Evaluation

The performance of the final model was evaluated using accuracy, confusion matrix, precision, recall, and F1-score to assess classification effectiveness.



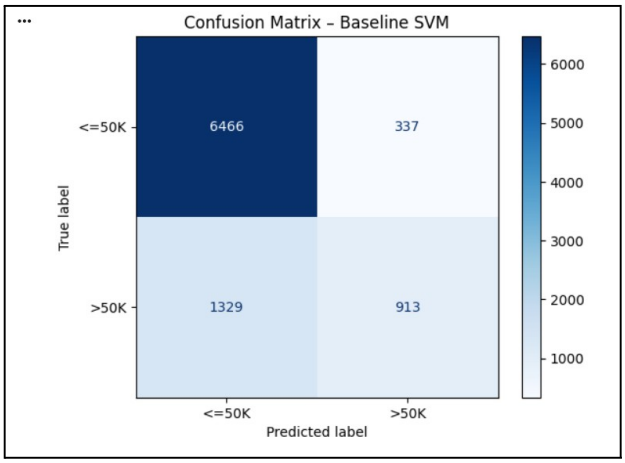
Code :

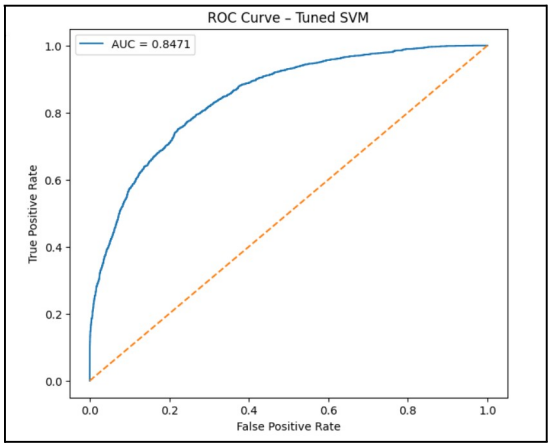
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn.preprocessing import
LabelEncoder, StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import (
    accuracy_score,
classification_report,
    confusion_matrix,
ConfusionMatrixDisplay,
    roc_curve, auc,
precision_recall_curve, f1_score
)
df = pd.read_csv("adult.csv")
df.replace("?", np.nan,
inplace=True)
df.dropna(inplace=True)
le = LabelEncoder()
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] =
le.fit_transform(df[col])
X = df.drop("income", axis=1)
y = df["income"]
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
random_state=42, stratify=y
)
scaler = StandardScaler()
X_train_scaled =
scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)
svm_baseline = LinearSVC(C=1,
max_iter=5000)
svm_baseline.fit(X_train_scaled,
y_train)

y_pred_base =
svm_baseline.predict(X_test_scaled)

print("===== BASELINE SVM
PERFORMANCE =====")
print(f"Accuracy:
{accuracy_score(y_test,
y_pred_base):.4f}")
print("\nClassification Report:\n")
print(classification_report(y_test,
y_pred_base))

# Confusion Matrix (Baseline)
cm_base = confusion_matrix(y_test,
y_pred_base)
disp_base = ConfusionMatrixDisplay(
    confusion_matrix=cm_base,
    display_labels=["<=50K", ">50K"]
)

disp_base.plot(cmap="Blues")
plt.title("Confusion Matrix -
Baseline SVM")
plt.show()
param_grid = {
    'C': [0.01, 0.1, 1, 10]
}

grid = GridSearchCV(
    LinearSVC(max_iter=5000),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)
```

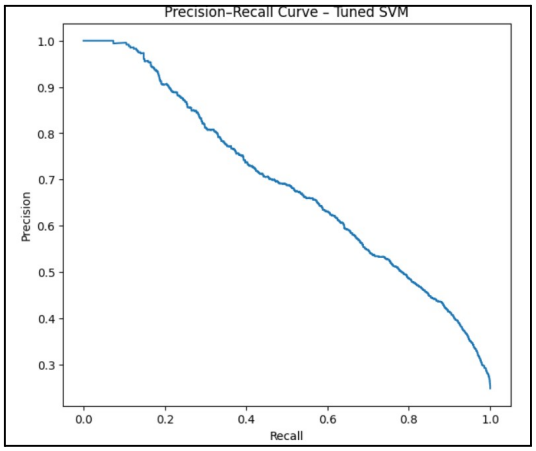
```
grid.fit(X_train_scaled, y_train)

print("\n===== BEST PARAMETERS
=====")
print(grid.best_params_)
print(f"Best Cross-Validation
Accuracy: {grid.best_score_:.4f}")

best_svm = grid.best_estimator_

y_pred =
best_svm.predict(X_test_scaled)

print("\n===== TUNED SVM PERFORMANCE
=====")
print(f"Accuracy:
{accuracy_score(y_test,
y_pred):.4f}")
print("\nClassification Report:\n")
print(classification_report(y_test,
y_pred))

# Confusion Matrix (Tuned)
cm = confusion_matrix(y_test,
y_pred)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=["<=50K", ">50K"]
)

disp.plot(cmap="Greens")
plt.title("Confusion Matrix - Tuned
SVM")
plt.show()

y_scores =
best_svm.decision_function(X_test_sc
aled)
fpr, tpr, _ = roc_curve(y_test,
y_scores)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(7, 6))
plt.plot(fpr, tpr, label=f"AUC =
{roc_auc:.4f}")
plt.plot([0, 1], [0, 1],
linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Tuned SVM")
plt.legend()
plt.show()
precision, recall, pr_thresholds =
precision_recall_curve(y_test,
y_scores)

plt.figure(figsize=(7, 6))
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve -
Tuned SVM")
plt.show()
f1_scores = []
for thresh in pr_thresholds:
    y_pred_thresh = (y_scores >=
thresh).astype(int)

f1_scores.append(f1_score(y_test,
y_pred_thresh))

plt.figure(figsize=(7, 6))
plt.plot(pr_thresholds, f1_scores)
plt.xlabel("Decision Threshold")
plt.ylabel("F1 Score")
plt.title("F1 Score vs Threshold -
Tuned SVM")
plt.show()
```

OUTPUT :

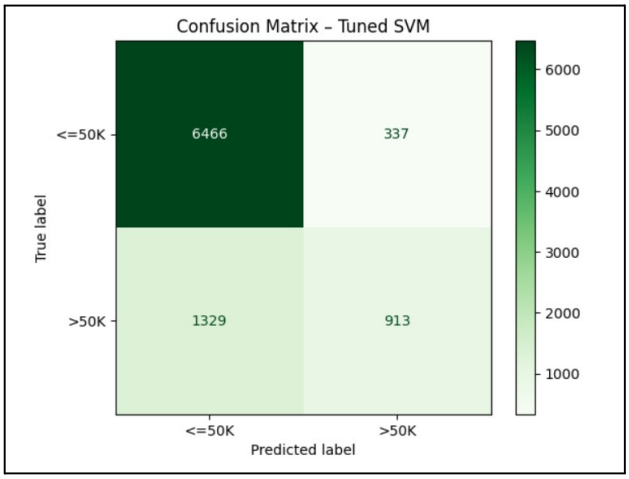
```
===== BASELINE SVM PERFORMANCE =====
Accuracy: 0.8158

Classification Report:

              precision    recall  f1-score   support

           0       0.83      0.95      0.89      6803
           1       0.73      0.41      0.52      2242

    accuracy                           0.82      9045
   macro avg       0.78      0.68      0.70      9045
weighted avg       0.80      0.82      0.80      9045
```








```
Classification Report:

              precision    recall  f1-score   support

           0       0.83      0.95      0.89      6803
           1       0.73      0.41      0.52      2242

    accuracy                           0.82      9045
   macro avg       0.78      0.68      0.70      9045
weighted avg       0.80      0.82      0.80      9045
```



Performance Analysis :

The performance of the Support Vector Machine (SVM) classifier was evaluated using accuracy, confusion matrix metrics (precision, recall, F1-score) for both the baseline and tuned models on the Adult Income dataset.

Baseline SVM Performance

The baseline Linear SVM model achieved an **accuracy of 81.58%**.

From the classification report:

- **Class 0 (Income ≤ 50K):**
  Precision = **0.83**, Recall = **0.95**, F1-score = **0.89**
  This indicates that the model is highly effective at identifying individuals earning less than or equal to 50K.

- **Class 1 (Income > 50K):**
  Precision = **0.73**, Recall = **0.41**, F1-score = **0.52**
  The lower recall shows that a significant number of high-income individuals are misclassified as low-income.

The macro-average F1-score of 0.70 highlights the impact of class imbalance, while the weighted F1-score of 0.80 reflects stable overall performance.

Hyperparameter-Tuned SVM Performance

After hyperparameter tuning using GridSearchCV, the optimal value of the regularization parameter was found to be C = 1, with a best cross-validation accuracy of 82.15%.
The tuned SVM model achieved the same test accuracy of 81.58% as the baseline model. The precision, recall, and F1-scores for both classes remained unchanged, indicating that hyperparameter tuning did not significantly alter the model's predictive behavior.
This outcome suggests that the baseline model was already well-optimized for this dataset and that further regularization did not provide additional improvement.

Conclusion :

In this experiment, a Support Vector Machine (SVM) classifier was implemented on the Adult Income dataset to predict whether an individual's income exceeds $50,000. The model achieved an accuracy of 81.58%, demonstrating good overall classification performance. Hyperparameter tuning confirmed that the baseline model was already optimal, as no significant improvement was observed. The results highlight that SVM performs well on large datasets but is affected by class imbalance, emphasizing the importance of evaluating class-wise metrics along with accuracy.