EXPERIMENT 5

Shravanya Andhale
D15A-30

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

THEORY:

1. Dataset Source

Dataset: CEEW India Residential Energy Survey Microdata

Source: Council on Energy, Environment and Water

The dataset contains household-level information on electricity access, billing practices, infrastructure quality, appliance ownership, and socio-economic characteristics.

2. Dataset Description

Target Variable:

q609_prepaid_meter_int

Data Cleaning:

Retained only responses 0 and 1

Removed 99 and missing values

Binary Target Created:

Prepaid_Interest = 1 if interested

Prepaid_Interest = 0 if not interested

Test Set Distribution:

Class 0: 1152

Class 1: 286

Predictor Variables Used:

asset_index_1

q208_priminc_earner_edu

q202_resp_age

q213_no_members

q302_grid_hrs_no

q308_grid_voltage_low_app

q326_satis_electricity

q314_a_online_pay_ever_yn

q401_bee_star_label_heard_yn

These features capture structural, infrastructure, satisfaction, and digital behaviour dimensions.

3. Mathematical Formulation of SVM

Support Vector Machine constructs an optimal separating hyperplane.

For linearly separable data:

Minimize: $\mathbf{1/2\ \|w\|^2}$

Subject to: $\mathbf{y_i\ (w \cdot x_i + b) \geq 1}$

For non-linearly separable data:

Slack variables allow margin violations

Kernel function transforms data into higher-dimensional space

RBF Kernel used:

$$\mathbf{K(x_i, x_j) = \exp(-\gamma\ \|x_i - x_j\|^2)}$$

Hyperparameters:

C controls margin flexibility

Gamma controls influence radius of support vectors

Class imbalance handled using class_weight='balanced'.

4. Algorithm Limitations

Sensitive to hyperparameter selection

Computationally intensive with larger datasets

Requires feature scaling

Does not provide direct feature importance

Performance influenced by class imbalance

5. Methodology / Workflow

1. Import required libraries
2. Load dataset
3. Clean and filter target variable
4. Create binary classification target
5. Select relevant predictors
6. Handle missing values
7. Perform stratified train-test split
8. Apply StandardScaler
9. Perform baseline logistic regression for signal check
10. Implement SVM with RBF kernel
11. Tune hyperparameters using GridSearchCV
12. Evaluate performance using multiple classification metrics
13. Compare SVM with Decision Tree, Random Forest, and KNN

6. Performance Analysis

SVM Results

Test Accuracy: 0.6551

Balanced Accuracy: 0.6165

ROC-AUC: 0.6423

Confusion Matrix:

[[784 368]
[128 158]]

Classification Insights:

Recall for interested households: 0.55

Precision for interested households: 0.30

Model detects more than half of interested households

Balanced accuracy indicates moderate discriminatory power

Model Comparison Using ROC-AUC

Decision Tree: 0.5893

Random Forest: 0.6729

KNN: 0.6019

SVM: 0.6423

Interpretation:

Random Forest achieved highest discrimination

SVM outperformed Decision Tree and KNN

Nonlinear structure exists in the dataset

Structural and infrastructural variables moderately explain prepaid interest

7. Hyperparameter Tuning

Kernel: RBF

C tested: 1, 10

Gamma tested: 'scale', 0.1

Cross-validation: 5-fold

Optimization metric: ROC-AUC

Hyperparameter tuning improved performance beyond baseline logistic regression AUC of 0.6091.

OUTPUT:

SVM achieved moderate classification performance.

Balanced detection of minority class was achieved using class_weight='balanced'.

Random Forest achieved the highest overall discrimination among compared models.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score,accuracy_score, confusion_matrix, classification_report
```

```python
df = pd.read_csv("CEEW - IRES Data.csv", low_memory=False)
print("Dataset shape:", df.shape)
```

```
Dataset shape: (14851, 517)
```

```python
df_svm = df.copy()

df_svm = df_svm[df_svm['q609_prepaid_meter_int'].isin([0, 1])]

df_svm['Prepaid_Interest'] = np.where(
    df_svm['q609_prepaid_meter_int'] == 1, 1, 0
)

print("Target Distribution:")
print(df_svm['Prepaid_Interest'].value_counts())
```

```
Target Distribution:
Prepaid_Interest
0    9533
1    2069
Name: count, dtype: int64
```

```python
predictors = [
    'asset_index_1',
    'q208_priminc_earner_edu',
    'q202_resp_age',
    'q213_no_members',
    'q302_grid_hrs_no',
    'q308_grid_voltage_low_app',
    'q326_satis_electricity',
    'q314_a_online_pay_ever_yn',
    'q401_bee_star_label_heard_yn'
]

df_model = df_svm[predictors + ['Prepaid_Interest']].dropna()

print("Final modeling shape:", df_model.shape)
print("Final class distribution:")
print(df_model['Prepaid_Interest'].value_counts())
```

```
Final modeling shape: (7186, 10)
Final class distribution:
Prepaid_Interest
0    5759
1    1427
Name: count, dtype: int64
```

```python
X = df_model[predictors]
y = df_model['Prepaid_Interest']

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_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)
```

```
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_s, y_train)

y_prob = log_reg.predict_proba(X_test_s)[:,1]

print("Baseline Logistic AUC:", roc_auc_score(y_test, y_prob))
```

```
Baseline Logistic AUC: 0.6091989607614607
```

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    roc_auc_score,
    roc_curve,
    precision_recall_curve,
    average_precision_score
)
import matplotlib.pyplot as plt
import seaborn as sns
```

```
svm = SVC(
    kernel='rbf',
    probability=True,
    class_weight='balanced',
    random_state=42
)

param_grid = {
    'C': [1, 10],
    'gamma': ['scale', 0.1]
}

grid_search = GridSearchCV(
    svm,
    param_grid,
    cv=5,
    scoring='roc_auc',
    n_jobs=-1
)

grid_search.fit(X_train_s, y_train)
```

```
▸ GridSearchCV

▸ best_estimator_:
  SVC

  ▸ SVC
```

```
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation AUC:", grid_search.best_score_)
```

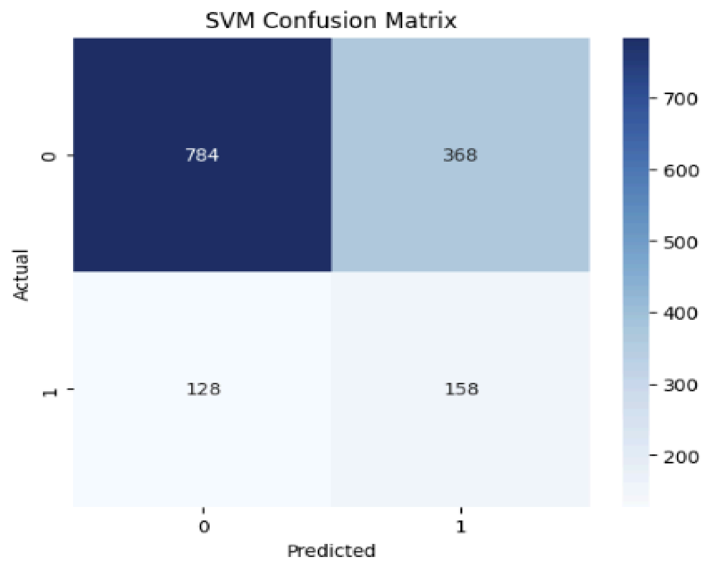
```
Best Parameters: {'C': 1, 'gamma': 'scale'}
Best Cross-Validation AUC: 0.6411147268515057
```

```python
best_svm = grid_search.best_estimator_

y_pred = best_svm.predict(X_test_s)
y_prob = best_svm.predict_proba(X_test_s)[:, 1]
```

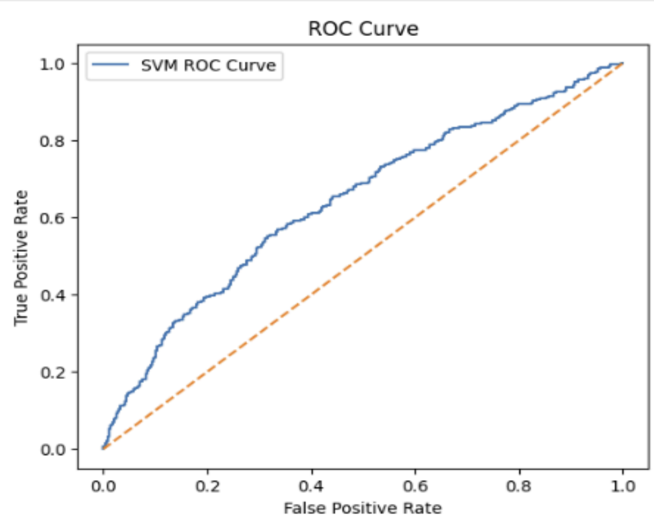
```python
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    roc_auc_score,
    balanced_accuracy_score
)

print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Balanced Accuracy:", balanced_accuracy_score(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("Test ROC-AUC:", roc_auc_score(y_test, y_prob))
```

```
Test Accuracy: 0.655076495132128
Balanced Accuracy: 0.6165015540015539

Confusion Matrix:
[[784 368]
 [128 158]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.68      0.76      1152
           1       0.30      0.55      0.39       286

    accuracy                           0.66      1438
   macro avg       0.58      0.62      0.57      1438
weighted avg       0.75      0.66      0.69      1438

Test ROC-AUC: 0.642321654040404
```

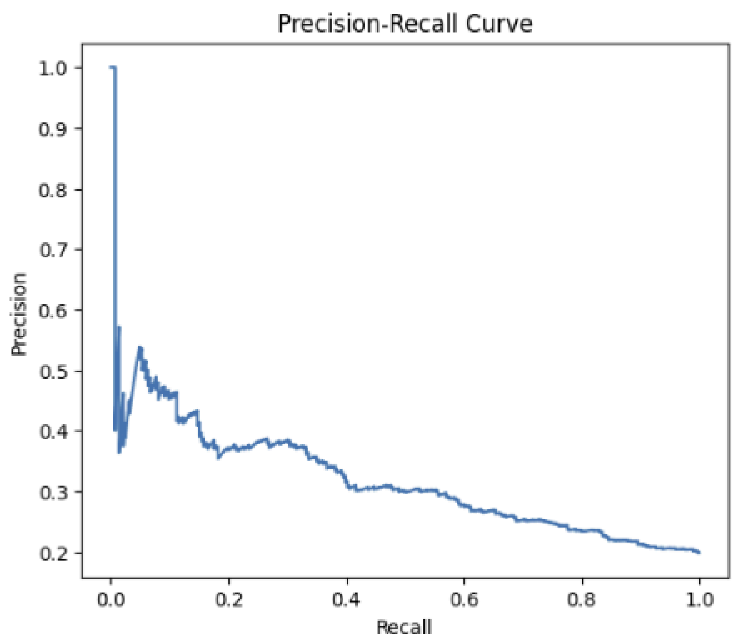
```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("SVM Confusion Matrix")
plt.show()
```




```python
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(y_test, y_prob)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label="SVM ROC Curve")
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```



```python
from sklearn.metrics import precision_recall_curve, average_precision_score

precision, recall, _ = precision_recall_curve(y_test, y_prob)

plt.figure(figsize=(6,5))
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.show()

print("Average Precision Score:", average_precision_score(y_test, y_prob))
```



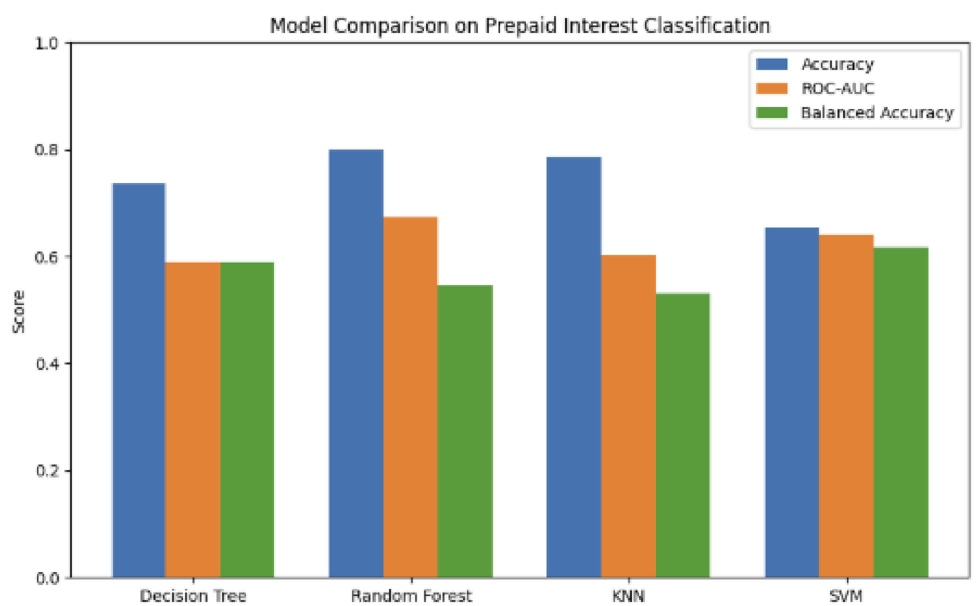

```
Average Precision Score: 0.320117805604908
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, balanced_accuracy_score

# -------- Decision Tree --------
dt = DecisionTreeClassifier(
    class_weight='balanced',
    random_state=42
)

dt.fit(X_train_s, y_train)

dt_pred = dt.predict(X_test_s)
dt_prob = dt.predict_proba(X_test_s)[:,1]

dt_acc = accuracy_score(y_test, dt_pred)
dt_auc = roc_auc_score(y_test, dt_prob)
dt_bal_acc = balanced_accuracy_score(y_test, dt_pred)


# -------- Random Forest --------
rf = RandomForestClassifier(
    n_estimators=200,
    class_weight='balanced',
    random_state=42
)

rf.fit(X_train_s, y_train)

rf_pred = rf.predict(X_test_s)
rf_prob = rf.predict_proba(X_test_s)[:,1]

rf_acc = accuracy_score(y_test, rf_pred)
rf_auc = roc_auc_score(y_test, rf_prob)
rf_bal_acc = balanced_accuracy_score(y_test, rf_pred)


# -------- KNN --------
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train_s, y_train)

knn_pred = knn.predict(X_test_s)
knn_prob = knn.predict_proba(X_test_s)[:,1]
```

```
knn_acc = accuracy_score(y_test, knn_pred)
knn_auc = roc_auc_score(y_test, knn_prob)
knn_bal_acc = balanced_accuracy_score(y_test, knn_pred)


# -------- SVM (already trained) --------
svm_acc = accuracy_score(y_test, y_pred)
svm_auc = roc_auc_score(y_test, y_prob)
svm_bal_acc = balanced_accuracy_score(y_test, y_pred)


print("Decision Tree AUC:", dt_auc)
print("Random Forest AUC:", rf_auc)
print("KNN AUC:", knn_auc)
print("SVM AUC:", svm_auc)
```

```
Decision Tree AUC: 0.5893095619658121
Random Forest AUC: 0.6729388233294484
KNN AUC: 0.6019009202602952
SVM AUC: 0.642321654040404
```

```python
import matplotlib.pyplot as plt
import numpy as np

models = ['Decision Tree', 'Random Forest', 'KNN', 'SVM']

accuracy = [dt_acc, rf_acc, knn_acc, svm_acc]
roc_auc = [dt_auc, rf_auc, knn_auc, svm_auc]
balanced_acc = [dt_bal_acc, rf_bal_acc, knn_bal_acc, svm_bal_acc]

x = np.arange(len(models))
width = 0.25

plt.figure(figsize=(10,6))

plt.bar(x - width, accuracy, width, label='Accuracy')
plt.bar(x, roc_auc, width, label='ROC-AUC')
plt.bar(x + width, balanced_acc, width, label='Balanced Accuracy')

plt.xticks(x, models)
plt.ylabel("Score")
plt.title("Model Comparison on Prepaid Interest Classification")
plt.ylim(0,1)
plt.legend()
plt.show()
```



CONCLUSION:

- Prepaid meter interest shows moderate structural predictability.
- Nonlinear interactions between socio-economic and infrastructure variables influence adoption interest.
- Machine learning models demonstrate that prepaid adoption behaviour is partially explainable but not strongly separable using observable features alone.
- Random Forest performs best, indicating the importance of feature interactions in behavioural classification tasks.