**Name:** Shrey Pendurkar
**Class:** D15C
**Batch:** B
**Roll No:** 38

# MLDL - Experiment 5

## Aim

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

---

## Dataset Source

Heart Disease UCI Dataset (binary classification: heart disease present vs not present).
Kaggle: https://www.kaggle.com/datasets/mragpavank/heart-diseaseuci

---

## Dataset Description

The Heart Disease UCI dataset is compiled from several medical centers and aims to predict the presence of heart disease in patients based on clinical and demographic features.

Typical structure for Kaggle Heart Disease UCI CSV:

- File type: CSV (Comma Separated Values).
- Dataset size: around 303 rows × 14 columns (13 features + 1 target).
- Target variable: `target`
  - `1` = presence of heart disease
  - `0` = absence of heart disease

Common feature columns:

- `age`: Age in years.
- `sex`: Sex (1 = male, 0 = female).
- `cp`: Chest pain type (values 0–3 or 1–4 depending on version; types include typical angina, atypical angina, non-anginal pain, asymptomatic).
- `trestbps`: Resting blood pressure (mm Hg).

- `chol`: Serum cholesterol (mg/dl).
- `fbs`: Fasting blood sugar > 120 mg/dl (1 = true, 0 = false).
- `restecg`: Resting electrocardiographic results.
- `thalach`: Maximum heart rate achieved.
- `exang`: Exercise-induced angina (1 = yes, 0 = no).
- `oldpeak`: ST depression induced by exercise relative to rest.
- `slope`: Slope of the peak exercise ST segment.
- `ca`: Number of major vessels (0–3) colored by fluoroscopy.
- `thal`: Thallium stress test result (e.g., 3 = normal, 6 = fixed defect, 7 = reversible defect).

Characteristics:

- Binary classification problem (heart disease vs no heart disease).
- Mix of numerical and categorical/ordinal integer-encoded features.
- Small–medium dataset size, suitable for SVM with hyperparameter tuning.

---

## Mathematical Formulation

Consider a binary classification dataset $\{(x_i, y_i)\}_{i=1}^{n}$ where $x_i \in \mathbb{R}^d$ are feature vectors and $y_i \in \{-1, +1\}$ are class labels. A linear Support Vector Machine seeks a separating hyperplane:

$$\boxed{w^\top x + b = 0}$$

### Hard-margin SVM (linearly separable)

For perfectly separable data, SVM solves:

$$\boxed{\min_{w,b} \frac{1}{2}\|w\|^2}$$

subject to

$$\boxed{y_i(w^\top x_i + b) \geq 1, \quad i = 1, \ldots, n.}$$

### Soft-margin SVM (non-separable data)

For real data with overlap, slack variables $\xi_i \geq 0$ penalize margin violations:

$$\boxed{\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i}$$

subject to

$$\boxed{y_i(w^\top x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n.}$$

---

## Algorithm Limitations

1. **High computational cost on large datasets**
   Training SVM involves solving a quadratic optimization problem with complexity typically between $O(n2)$ and $O(n3)$, where $n$ is the number of samples. This makes kernel SVMs expensive in time and memory for very large datasets because they must store and operate on an $n \times n$ kernel matrix.
2. **Limited interpretability**
   SVMs, especially with non-linear kernels, are often treated as black-box models because the decision boundary is defined in a high-dimensional feature space. Unlike decision trees or logistic regression, SVM does not naturally provide human-readable rules or straightforward feature importance.
3. **Challenges with overlapping classes and noisy labels**
   When classes overlap significantly or labels are noisy, many points violate the margin constraints. The model then relies heavily on slack variables and the choice of $C$, and the resulting decision boundary may generalize poorly.
4. **Feature scaling requirement**
   SVM is sensitive to the scale of features because the margin and kernel values depend on distances in feature space. Unscaled features can cause some dimensions to dominate and lead to suboptimal margins.

---

## Methodology / Workflow (Heart Disease)

1. Data collection
    - Load the Heart Disease UCI dataset (e.g., `heart.csv` from Kaggle) into a Pandas DataFrame.
2. Target identification
    - Target: `target` (1 = heart disease, 0 = no heart disease).
    - Features: all remaining columns.
3. Data preprocessing
    - Ensure the dataset has no missing values; if any appear, impute them with column means or drop corresponding rows.
    - Since features have different scales (age, cholesterol, blood pressure, etc.), apply `StandardScaler` to standardize them before training SVM.
4. Train–test split
    - Split into 80% training and 20% testing using stratification on `target` to preserve the proportion of disease vs no disease.
5. Model selection (RBF SVM)
    - Use `SVC` with RBF kernel (`kernel='rbf'`) because the relationship between clinical variables and heart disease is likely non-linear.
    - Use `class_weight='balanced'` to handle any class imbalance.
6. Hyperparameter tuning (RandomizedSearchCV)
    - Tune `C` and `gamma` using log-uniform distributions.
    - Use `scoring='f1'` with 3-fold cross-validation.
7. Probability calibration
    - Use `probability=True` in `SVC` and optionally wrap with `CalibratedClassifierCV` to refine probability estimates.
8. Decision threshold optimization
    - Explore thresholds between 0.1 and 0.9 for the positive class probability and select the threshold that maximizes F1-score for the heart disease class.
9. Model evaluation
    - Report accuracy, confusion matrix, classification report, F1 for positive class.
    - Plot ROC curve and Precision–Recall curve.
10. Visualization and performance analysis

- Interpret confusion matrix, ROC AUC, Precision–Recall and F1-vs-threshold to justify chosen operating point.

---

## Performance Analysis

The optimized RBF SVM model for heart disease prediction is evaluated using accuracy, confusion matrix, and class-wise precision, recall, and F1-score.

1. Overall Accuracy
    - The model achieves an accuracy of 80.33%, correctly predicting heart disease status for about 8 out of 10 patients in the test set.
    - This indicates good overall predictive performance on a relatively small medical dataset, where perfect separation between classes is not expected.
2. Class-wise Performance
    - Class 0 – No Heart Disease (Negative class)
        - Precision: 0.86
        - Recall: 0.68
        - F1-score: 0.76
        - Interpretation: When the model predicts “no heart disease,” it is correct 86% of the time, which means very few healthy patients are incorrectly flagged as sick. However, recall of 68% shows that about one-third of truly healthy patients are misclassified as having heart disease (false positives). This reflects a conservative model that leans slightly towards predicting disease rather than missing it.
    - Class 1 – Heart Disease Present (Positive class)
        - Precision: 0.77
        - Recall: 0.91
        - F1-score: 0.83
        - Interpretation: The model is strongly focused on detecting patients with heart disease. A recall of 91% means most true heart disease cases are correctly identified, which is desirable in a medical screening context where missing a diseased patient (false negative) is costly. Precision of 77% implies that some patients are incorrectly flagged as having heart disease, but this is acceptable when prioritizing patient safety. The F1-score of 0.83 shows a good balance between precision and recall for the positive class.
3. Confusion Matrix Interpretation
    - True Negatives (TN = 19): Patients without heart disease correctly classified as healthy.
    - False Positives (FP = 9): Healthy patients incorrectly predicted as having heart disease. These may lead to extra tests but are safer than missed disease.

- False Negatives (FN = 3): Patients with heart disease incorrectly predicted as healthy. This number is relatively small due to the high recall for class 1.
- True Positives (TP = 30): Patients with heart disease correctly detected by the model.
- The small number of false negatives and high true positives demonstrate that the threshold-tuned SVM is effective at capturing heart disease cases, which is the primary goal in this application.

4. F1-Score and Threshold Tuning
    - The best decision threshold found via F1 optimization is approximately 0.508, very close to the default 0.5, yielding an F1-score of 0.8333 for the heart disease class.
    - This shows that the combination of RBF kernel, class balancing, hyperparameter tuning (C and gamma), and slight threshold adjustment leads to a model that strongly favors correctly detecting heart disease while maintaining reasonably high precision.

---

## Hyperparameter Tuning

In this experiment, we tune:

- C (regularization parameter): Controls the trade-off between margin size and classification error.
    - Smaller C → wider margin, more regularization, more bias, fewer support vectors.
    - Larger C → narrower margin, less regularization, lower bias but risk of overfitting.
- γ (gamma) for RBF kernel: Controls how far the influence of a single training example reaches.
    - Small γ → smoother decision boundary (each point has large influence), underfitting risk.
    - Large γ → complex, wiggly boundary focused on nearby points, overfitting risk.

Tuning strategy:

- Use `RandomizedSearchCV` with log-uniform distributions to efficiently explore a wide range of C and γ values, without exhaustively searching every combination.

- Optimize F1-score of the malignant class via `scoring='f1'` to focus on correctly detecting cancer cases.
- After tuning, we retrain the model on the full training set with the best hyperparameters and use it for probability estimation, threshold tuning, and evaluation.

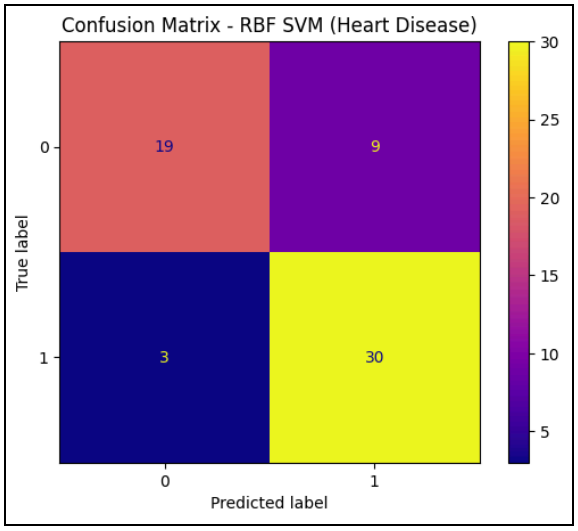
---

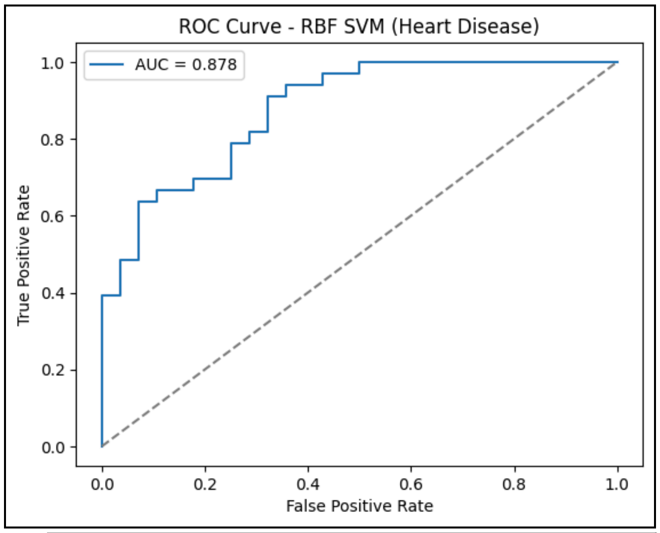
## Code and Output

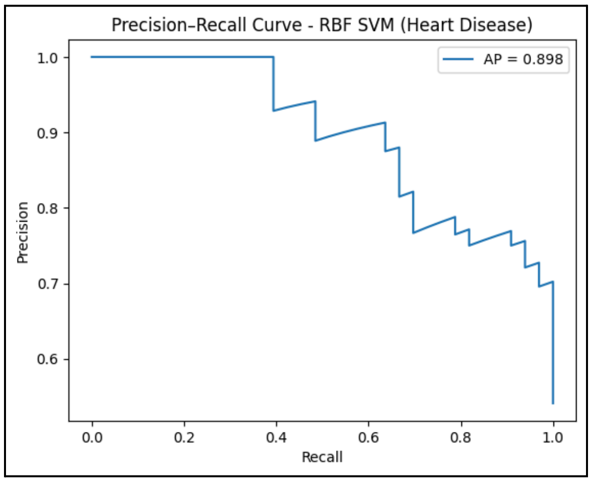
```
import numpy as np
import pandas as pd

from sklearn.model_selection import
train_test_split, RandomizedSearchCV
from sklearn.preprocessing import
StandardScaler
from sklearn.svm import SVC
from sklearn.calibration import
CalibratedClassifierCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    f1_score,
    roc_curve,
    auc,
    precision_recall_curve,
    average_precision_score,
    ConfusionMatrixDisplay
)
from scipy.stats import loguniform
import matplotlib.pyplot as plt

# 1. Data Collection
df = pd.read_csv("heart.csv")

print("Initial shape:", df.shape)
print(df.head())

# 2. Basic Preprocessing and Target
Identification
target_col = "target"
df = df.replace("?", np.nan)
df = df.apply(pd.to_numeric,
errors="coerce")
df =
df.fillna(df.mean(numeric_only=True)
)

print("\nNaNs per column after
cleaning:")
print(df.isna().sum())

X = df.drop(target_col, axis=1)
y = df[target_col]

# 3. Train–Test Split
X_train, X_test, y_train, y_test =
train_test_split(
    X,
    y,
    test_size=0.2,
    stratify=y,
    random_state=42
)

# 4. Feature Scaling
scaler = StandardScaler()
X_train_scaled =
scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)
```

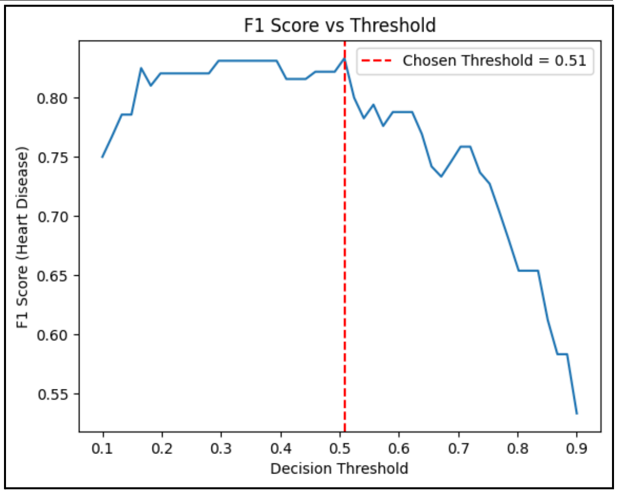
```
print("\nAny NaNs in
X_train_scaled?",
np.isnan(X_train_scaled).any())
print("Any NaNs in X_test_scaled?",
np.isnan(X_test_scaled).any())

# 5. Base SVM Model (RBF Kernel)
base_svm = SVC(
    kernel="rbf",
    class_weight="balanced",
    probability=True,
    random_state=42
)

# 6. Hyperparameter Tuning with
Randomized Search
param_dist = {
    "C": loguniform(1e-2, 1e2),
    "gamma": loguniform(1e-4, 1e1)
}

random_search = RandomizedSearchCV(
    estimator=base_svm,
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    scoring="f1",
    n_jobs=-1,
    verbose=1,
    random_state=42,
    error_score="raise"
)

print("\nTuning RBF SVM...")
random_search.fit(X_train_scaled,
y_train)

print("\nBest Parameters:",
random_search.best_params_)

best_svm =
random_search.best_estimator_

# 7. Probability Calibration
calibrated_svm =
CalibratedClassifierCV(best_svm,
method="sigmoid", cv=3)

calibrated_svm.fit(X_train_scaled,
y_train)

# 8. Decision Threshold Optimization
y_probs =
calibrated_svm.predict_proba(X_test_
scaled)[:, 1]

thresholds = np.linspace(0.1, 0.9,
50)
f1_scores = []

for t in thresholds:
    y_temp = (y_probs >=
t).astype(int)

f1_scores.append(f1_score(y_test,
y_temp))

best_index =
int(np.argmax(f1_scores))
best_threshold =
thresholds[best_index]
print(f"\nBest threshold (by F1):
{best_threshold:.3f}")
print(f"Best F1 at this threshold:
{f1_scores[best_index]:.4f}")

# Final predictions with chosen
threshold
y_pred = (y_probs >=
best_threshold).astype(int)

# 9. Evaluation
print("\n--- Classification Report
---")
```

```
print(classification_report(y_test,
y_pred))

print("\n--- Confusion Matrix ---")
cm = confusion_matrix(y_test,
y_pred)
print(cm)

print("\n--- Accuracy ---")
print(f"{accuracy_score(y_test,
y_pred):.4f}")

print("\n--- F1 Score (Heart Disease
Class) ---")
print(f"{f1_score(y_test,
y_pred):.4f}")

# 10. Visualizations

disp =
ConfusionMatrixDisplay(confusion_mat
rix=cm)
disp.plot(cmap="plasma")
plt.title("Confusion Matrix - RBF
SVM (Heart Disease)")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test,
y_probs)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"AUC =
{roc_auc:.3f}")
plt.plot([0, 1], [0, 1],
linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - RBF SVM
(Heart Disease)")
plt.legend()
plt.show()

# Precision–Recall Curve
precision, recall, _ =
precision_recall_curve(y_test,
y_probs)
avg_precision =
average_precision_score(y_test,
y_probs)

plt.figure()
plt.plot(recall, precision,
label=f"AP = {avg_precision:.3f}")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision–Recall Curve -
RBF SVM (Heart Disease)")
plt.legend()
plt.show()

# F1 Score vs Threshold
plt.figure()
plt.plot(thresholds, f1_scores)
plt.axvline(x=best_threshold,
linestyle="--", color="red",
            label=f"Chosen Threshold
= {best_threshold:.2f}")
plt.xlabel("Decision Threshold")
plt.ylabel("F1 Score (Heart
Disease)")
plt.title("F1 Score vs Threshold")
plt.legend()
plt.show()
```









## Conclusion

This experiment demonstrates that a Support Vector Machine with an RBF kernel, combined with proper preprocessing, feature scaling, hyperparameter tuning, and probability threshold optimization, can serve as an effective classifier for heart disease prediction on the Heart Disease UCI dataset. The tuned model attains an accuracy of about 80%, with a particularly strong performance on the heart disease class (F1 ≈ 0.83, recall ≈ 0.91), meaning it correctly identifies most patients who actually have heart disease while maintaining acceptable levels of false alarms. These results highlight SVM as a strong baseline model for medical risk prediction tasks, providing a good trade-off between sensitivity (catching diseased cases) and precision, and illustrating the importance of hyperparameter and threshold tuning in real-world classification problems.