Sri Sivasubramaniya Nadar College of Engineering, Chennai
(An Autonomous Institution Affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	Semester	VI
Subject Code & Name	UCS2612 – Machine Learning Algorithms Laboratory		
Academic Year	2025–2026 (Even)	Batch	2023–2027

Keerthana R CSE-B 3122235001066

Experiment 4: Binary Classification using Linear and Kernel-Based Models

Objective

To classify emails as spam or ham (non-spam) using Logistic Regression and Support Vector Machine (SVM) classifiers, perform hyperparameter tuning, and analyze the effect of different kernels and regularization techniques on classification performance.
Specific Goals:

1. Implement Logistic Regression with L1 and L2 regularization
2. Implement SVM with multiple kernels (Linear, Polynomial, RBF, Sigmoid)
3. Perform hyperparameter tuning using Grid Search
4. Compare probabilistic vs margin-based classifiers
5. Analyze bias-variance trade-off
6. Evaluate models using multiple performance metrics
7. Perform k-fold cross-validation for robust evaluation

Dataset Description

Spambase Dataset

Source: UCI Machine Learning Repository / Kaggle
Purpose: Email spam detection
Size: 4,601 emails
Features: 57 numerical features + 1 binary target
Classes: 0 = Ham (non-spam), 1 = Spam

Feature Categories

1. **Word Frequency (48 features):** Percentage of specific words (e.g., ”free”, ”money”, ”business”)
2. **Character Frequency (6 features):** Percentage of characters: `;`, `(`, `[`, `!`, `$`, `#`
3. **Capital Letter Statistics (3 features):**
   - Average length of capital letter sequences
   - Longest capital letter sequence
   - Total number of capital letters

Class Distribution

- Ham (Class 0): 2,788 emails (60.6%)
- Spam (Class 1): 1,813 emails (39.4%)
- Moderately imbalanced dataset

Mathematical and Theoretical Foundation

1. Logistic Regression

Logistic Regression is a probabilistic linear classifier that models the probability of class membership using the sigmoid function.

1.1 Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

where $z = \mathbf{w}^T\mathbf{x} + b$ is the linear combination of features.

1.2 Probability Model

The probability that a sample $\mathbf{x}$ belongs to class 1:

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}} \tag{2}$$

1.3 Decision Rule

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1|\mathbf{x}) \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

1.4 Loss Function

Logistic Regression minimizes the log-loss (binary cross-entropy):

$$\text{Loss} = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)] \tag{4}$$

where $\hat{p}_i = P(y = 1|\mathbf{x}_i)$ is the predicted probability.

1.5 Regularization

L1 Regularization (Lasso):

$$\text{Loss}_{\text{L1}} = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)] + \frac{1}{C}\|\mathbf{w}\|_1 \tag{5}$$

- Encourages sparsity (some coefficients $\rightarrow 0$)
- Performs automatic feature selection
- Useful when many features are irrelevant

L2 Regularization (Ridge):

$$\text{Loss}_{\text{L2}} = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right] + \frac{1}{2C}\|\mathbf{w}\|^2 \tag{6}$$

- Shrinks all coefficients proportionally
- Does not perform feature selection
- Better for correlated features

1.6 Hyperparameter C

C is the inverse of regularization strength:

- **Small C** (e.g., 0.01): Strong regularization, simpler model, higher bias
- **Large C** (e.g., 100): Weak regularization, complex model, higher variance

2. Support Vector Machine (SVM)

SVM is a margin-based classifier that finds the optimal hyperplane separating classes by maximizing the margin.

2.1 Linear SVM

For linearly separable data, SVM finds the hyperplane:

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{7}$$

that maximizes the margin:

$$\text{Margin} = \frac{2}{\|\mathbf{w}\|} \tag{8}$$

2.2 Optimization Problem

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \tag{9}$$

subject to:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{10}$$

where ξ_i are slack variables allowing misclassification.

2.3 SVM Kernels

For non-linearly separable data, SVM uses the kernel trick to map data to higher dimensions.

Linear Kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j \tag{11}$$

Polynomial Kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma\mathbf{x}_i^T\mathbf{x}_j + r)^d \tag{12}$$

RBF (Radial Basis Function) Kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2) \tag{13}$$

Sigmoid Kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma\mathbf{x}_i^T\mathbf{x}_j + r) \tag{14}$$

2.4 SVM Hyperparameters

C (Regularization Parameter):

- **Small C**: Wide margin, more misclassifications, higher bias
- **Large C**: Narrow margin, fewer misclassifications, higher variance

γ (Kernel Coefficient):

- **Small γ**: Smooth decision boundary, higher bias
- **Large γ**: Complex decision boundary, higher variance
- **'scale'**: $\gamma = \frac{1}{n_{\text{features}} \times \text{Var}(X)}$
- **'auto'**: $\gamma = \frac{1}{n_{\text{features}}}$

3. Performance Metrics

For binary classification with TP, TN, FP, FN:

Accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{15}$$

Precision:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{16}$$

Recall:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{17}$$

F1 Score:

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{18}$$

Preprocessing Steps

1. Data Loading and Exploration

```python
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,
    cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report)
import time
import warnings
warnings.filterwarnings('ignore')

# Load dataset
df = pd.read_csv('spambase.csv')

# Display basic information
```

```python
print("Dataset Shape:", df.shape)
print("\nFirst few rows:")
print(df.head())
print("\nDataset Info:")
print(df.info())
print("\nMissing Values:")
print(df.isnull().sum().sum())
print("\nClass Distribution:")
print(df['spam'].value_counts())
```

Listing 1: Loading the Spambase Dataset

2. Feature Standardization

```python
# Separate features and target
X = df.drop('spam', axis=1)
y = df['spam']

# Split data (80-20)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Standardize features (critical for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"Training set: {X_train.shape}")
print(f"Test set: {X_test.shape}")
print(f"Class distribution in training: {y_train.value_counts()}")
```

Listing 2: Feature Scaling

3. Exploratory Data Analysis

```python
# Class distribution
plt.figure(figsize=(8, 5))
y.value_counts().plot(kind='bar', color=['green', 'red'], alpha=0.7)
plt.title('Class Distribution: Ham vs Spam')
plt.xlabel('Class (0=Ham, 1=Spam)')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()

# Feature correlation with target
correlations = X.corrwith(y).sort_values(ascending=False)
print("Top 10 features correlated with spam:")
print(correlations.head(10))

# Correlation heatmap (top features)
plt.figure(figsize=(10, 8))
top_features = correlations.abs().nlargest(15).index
corr_matrix = df[list(top_features) + ['spam']].corr()
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Heatmap of Top Features')
plt.tight_layout()
plt.show()
```

Listing 3: EDA Visualizations

Implementation Details

1. Baseline Logistic Regression

```python
# Train baseline Logistic Regression (L2, default C=1)
print("="*60)
print("Baseline Logistic Regression")
print("="*60)

start_time = time.time()
lr_baseline = LogisticRegression(random_state=42, max_iter=1000)
lr_baseline.fit(X_train_scaled, y_train)
lr_baseline_time = time.time() - start_time

# Predictions
y_pred_lr_baseline = lr_baseline.predict(X_test_scaled)

# Evaluation
print(f"\nBaseline Logistic Regression Results:")
print(f"Training Time: {lr_baseline_time:.4f}s")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr_baseline):.4f}")
print(f"Precision: {precision_score(y_test, y_pred_lr_baseline):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_lr_baseline):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred_lr_baseline):.4f}")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_lr_baseline))
```

Listing 4: Baseline Logistic Regression

2. Logistic Regression with Hyperparameter Tuning

```python
# Define parameter grid
param_grid_lr = {
    'penalty': ['l1', 'l2'],
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga']
}

# Grid Search with 5-fold CV
print("\n" + "="*60)
print("Logistic Regression Hyperparameter Tuning")
print("="*60)

grid_lr = GridSearchCV(
    LogisticRegression(random_state=42, max_iter=1000),
    param_grid_lr,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

start_time = time.time()
grid_lr.fit(X_train_scaled, y_train)
lr_tuning_time = time.time() - start_time

print(f"\nBest Parameters: {grid_lr.best_params_}")
print(f"Best CV Accuracy: {grid_lr.best_score_:.4f}")
print(f"Tuning Time: {lr_tuning_time:.4f}s")

# Best model
lr_best = grid_lr.best_estimator_
y_pred_lr_best = lr_best.predict(X_test_scaled)

# Evaluation
print(f"\nOptimized Logistic Regression Results:")
print(f"Test Accuracy: {accuracy_score(y_test, y_pred_lr_best):.4f}")
print(f"Precision: {precision_score(y_test, y_pred_lr_best):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_lr_best):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred_lr_best):.4f}")
```

Listing 5: Tuning Logistic Regression

3. SVM with Different Kernels

```python
# Test different kernels
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
svm_results = {}

print("\n" + "="*60)
print("SVM Kernel Comparison")
print("="*60)

for kernel in kernels:
    print(f"\nTraining SVM with {kernel} kernel...")

    start_time = time.time()
    if kernel == 'poly':
        svm = SVC(kernel=kernel, degree=3, random_state=42)
    else:
        svm = SVC(kernel=kernel, random_state=42)

    svm.fit(X_train_scaled, y_train)
    train_time = time.time() - start_time

    # Predictions
    y_pred = svm.predict(X_test_scaled)

    # Store results
    svm_results[kernel] = {
        'model': svm,
        'accuracy': accuracy_score(y_test, y_pred),
        'precision': precision_score(y_test, y_pred),
        'recall': recall_score(y_test, y_pred),
        'f1': f1_score(y_test, y_pred),
        'time': train_time
    }

    print(f"Accuracy: {svm_results[kernel]['accuracy']:.4f}")
    print(f"F1 Score: {svm_results[kernel]['f1']:.4f}")
    print(f"Training Time: {train_time:.4f}s")
```

Listing 6: SVM Kernel Comparison

4. SVM Hyperparameter Tuning

```python
# Focus on RBF kernel (typically best for spam detection)
param_grid_svm = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto'],
    'kernel': ['rbf']
}

print("\n" + "="*60)
print("SVM (RBF) Hyperparameter Tuning")
print("="*60)

grid_svm = GridSearchCV(
    SVC(random_state=42),
    param_grid_svm,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

start_time = time.time()
grid_svm.fit(X_train_scaled, y_train)
svm_tuning_time = time.time() - start_time

print(f"\nBest Parameters: {grid_svm.best_params_}")
print(f"Best CV Accuracy: {grid_svm.best_score_:.4f}")
print(f"Tuning Time: {svm_tuning_time:.4f}s")

# Best model
svm_best = grid_svm.best_estimator_
y_pred_svm_best = svm_best.predict(X_test_scaled)

# Evaluation
print(f"\nOptimized SVM Results:")
print(f"Test Accuracy: {accuracy_score(y_test, y_pred_svm_best):.4f}")
print(f"Precision: {precision_score(y_test, y_pred_svm_best):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_svm_best):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred_svm_best):.4f}")
```

Listing 7: Tuning SVM (RBF Kernel)

5. K-Fold Cross-Validation

```python
from sklearn.model_selection import cross_validate

# Define models to evaluate
models = {
    'Logistic Regression': lr_best,
    'SVM (RBF)': svm_best
}

# Perform 5-fold CV
print("\n" + "="*60)
print("5-Fold Cross-Validation")
print("="*60)

cv_results = {}
for name, model in models.items():
    print(f"\nEvaluating {name}...")

    cv_scores = cross_validate(
        model, X_train_scaled, y_train,
        cv=5,
        scoring=['accuracy', 'precision', 'recall', 'f1'],
        return_train_score=True
    )

    cv_results[name] = {
        'test_accuracy': cv_scores['test_accuracy'],
        'test_precision': cv_scores['test_precision'],
        'test_recall': cv_scores['test_recall'],
        'test_f1': cv_scores['test_f1']
    }

    print(f"Accuracy: {cv_scores['test_accuracy'].mean():.4f} (+/- {cv_scores['test_accuracy'].std():.4f})")
    print(f"Precision: {cv_scores['test_precision'].mean():.4f} (+/- {cv_scores['test_precision'].std():.4f})")
    print(f"Recall: {cv_scores['test_recall'].mean():.4f} (+/- {cv_scores['test_recall'].std():.4f})")
    print(f"F1 Score: {cv_scores['test_f1'].mean():.4f} (+/- {cv_scores['test_f1'].std():.4f})")
```

Listing 8: 5-Fold Cross-Validation

Visualizations

1. Confusion Matrices

```
from sklearn.metrics import ConfusionMatrixDisplay

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Logistic Regression
ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred_lr_best, ax=axes[0], cmap='Blues'
)
axes[0].set_title(f'Logistic Regression\nAccuracy: {accuracy_score(y_test, y_pred_lr_best):.4f}')
axes[0].grid(False)

# SVM
ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred_svm_best, ax=axes[1], cmap='Greens'
)
```

```python
axes[1].set_title(f'SVM (RBF)\nAccuracy: {accuracy_score(y_test, y_pred_svm_best):.4f}')
axes[1].grid(False)

plt.tight_layout()
plt.show()
```

Listing 9: Confusion Matrix Visualization

2. ROC Curves

```python
from sklearn.metrics import roc_curve, auc

# Get probability predictions
y_prob_lr = lr_best.decision_function(X_test_scaled)
y_prob_svm = svm_best.decision_function(X_test_scaled)

# Compute ROC curves
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob_lr)
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_prob_svm)

roc_auc_lr = auc(fpr_lr, tpr_lr)
roc_auc_svm = auc(fpr_svm, tpr_svm)

# Plot
plt.figure(figsize=(10, 6))
plt.plot(fpr_lr, tpr_lr, linewidth=2,
         label=f'Logistic Regression (AUC = {roc_auc_lr:.3f})')
plt.plot(fpr_svm, tpr_svm, linewidth=2,
         label=f'SVM RBF (AUC = {roc_auc_svm:.3f})')
plt.plot([0, 1], [0, 1], 'k--', linewidth=2, label='Random Classifier')

plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curves Comparison', fontsize=14)
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Listing 10: ROC Curve Comparison

3. Performance Comparison

```python
# Comparison data
models_names = ['Logistic Regression', 'SVM (RBF)']
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']

lr_scores = [
    accuracy_score(y_test, y_pred_lr_best),
    precision_score(y_test, y_pred_lr_best),
    recall_score(y_test, y_pred_lr_best),
    f1_score(y_test, y_pred_lr_best)
]

svm_scores = [
    accuracy_score(y_test, y_pred_svm_best),
    precision_score(y_test, y_pred_svm_best),
    recall_score(y_test, y_pred_svm_best),
    f1_score(y_test, y_pred_svm_best)
]

# Plot
x = np.arange(len(metrics))
width = 0.35

fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(x - width/2, lr_scores, width, label='Logistic Regression', alpha=0.8)
ax.bar(x + width/2, svm_scores, width, label='SVM (RBF)', alpha=0.8)

ax.set_xlabel('Metrics', fontsize=12)
ax.set_ylabel('Score', fontsize=12)
ax.set_title('Performance Metrics Comparison', fontsize=14)
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend(fontsize=11)
ax.grid(True, alpha=0.3, axis='y')
ax.set_ylim([0.85, 1.0])

plt.tight_layout()
plt.show()
```

Listing 11: Model Comparison Bar Chart

Performance Results

Hyperparameter Tuning Results

Table 1: Hyperparameter Tuning Summary

Model	**Search**	**Best Parameters**	**Best CV Acc**
Logistic Regression	Grid Search	penalty='l2', C=10, solver='saga'	0.9293
SVM	Grid Search	kernel='rbf', C=10, gamma='scale'	0.9315

Analysis:

- Logistic Regression optimal with L2 regularization and C=10
- SVM RBF kernel optimal with C=10 and gamma='scale'
- Both models achieved ¿93% CV accuracy
- Grid Search explored 20 combinations for LR, 8 for SVM