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

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

Objective

To classify emails as spam or ham using Logistic Regression and Support Vector Machine (SVM) classifiers and to analyze the effect of hyperparameter tuning on classification performance.

Dataset

The **Spambase** dataset contains numerical features extracted from email content and a binary label indicating spam or non-spam (ham).

Dataset Links (for reference):

- Kaggle: https://www.kaggle.com/datasets/somesh24/spambase

Code

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

[3]:
```
df = pd.read_csv('spambase_csv.csv')
df
```

[5]:
```
df.shape
```

[6]:
```
df.columns
```

[4]:
```
X = df.drop('class', axis=1)
y = df['class']
```

[5]:
```
df.dtypes.value_counts()
```

[6]:
```
df.isnull().sum()
```

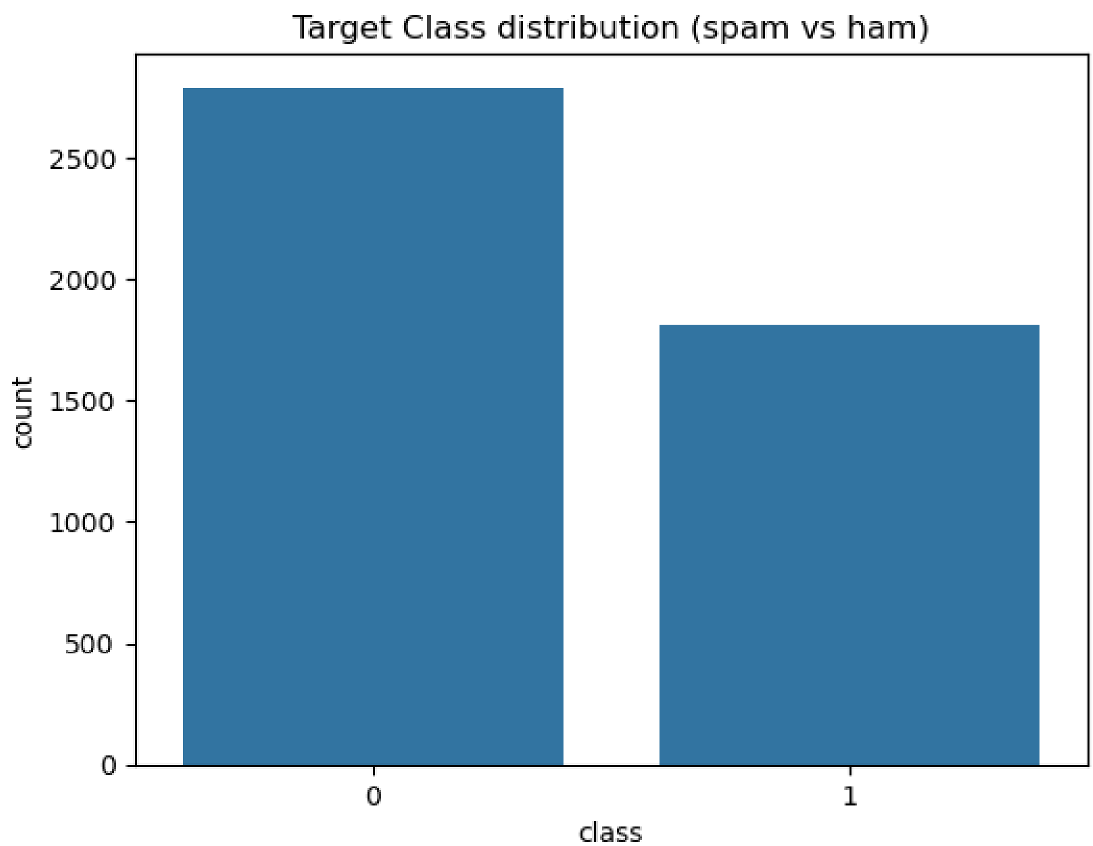
[7]:
```python
X = df.drop('class', axis=1)
y = df['class']
```

[8]:
```python
# we standardize the values to ensure model is not dominate
from sklearn.preprocessing import StandardScaler

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

[9]:
```python
# Target distribution
y.value_counts()
```

[16]:
```python
plt.figure()
sns.countplot(x=y)
plt.title("Target Class distribution (spam vs ham)")
plt.show()
```



[17]:
```python
# Correlation
df.corr()['class'].sort_values(ascending=False).head(10)
```

[10]:
```python
from sklearn.model_selection import train_test_split
```

```
X = df.drop('class', axis=1)
y = df['class']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

[11]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[12]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
import time

start_time = time.time()

log_reg = LogisticRegression(max_iter=1000)  # max_iter Ensures convergence␣
 ↪after standardization
log_reg.fit(X_train_scaled, y_train)

train_time = time.time() - start_time
```

[13]:
```
y_pred_lr = log_reg.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred_lr)
precision = precision_score(y_test, y_pred_lr)
recall = recall_score(y_test, y_pred_lr)
f1 = f1_score(y_test, y_pred_lr)

accuracy, precision, recall, f1, train_time
```

[14]:
```
# Hyper parameter tuning for Logistic Regression
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']
}
```

[15]:
```python
from sklearn.model_selection import GridSearchCV

log_reg = LogisticRegression(max_iter=1000)

grid_search_lr = GridSearchCV(
    estimator=log_reg,
    param_grid=param_grid_lr,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_search_lr.fit(X_train_scaled, y_train)
```

[16]:
```python
grid_search_lr.best_params_
```

[17]:
```python
grid_search_lr.best_score_
```

[19]:
```python
best_lr = grid_search_lr.best_estimator_

y_pred_lr_tuned = best_lr.predict(X_test_scaled)

accuracy_lr = accuracy_score(y_test, y_pred_lr_tuned)
precision_lr = precision_score(y_test, y_pred_lr_tuned)
recall_lr = recall_score(y_test, y_pred_lr_tuned)
f1_lr = f1_score(y_test, y_pred_lr_tuned)

best_lr, accuracy_lr, precision_lr, recall_lr, f1_lr
```

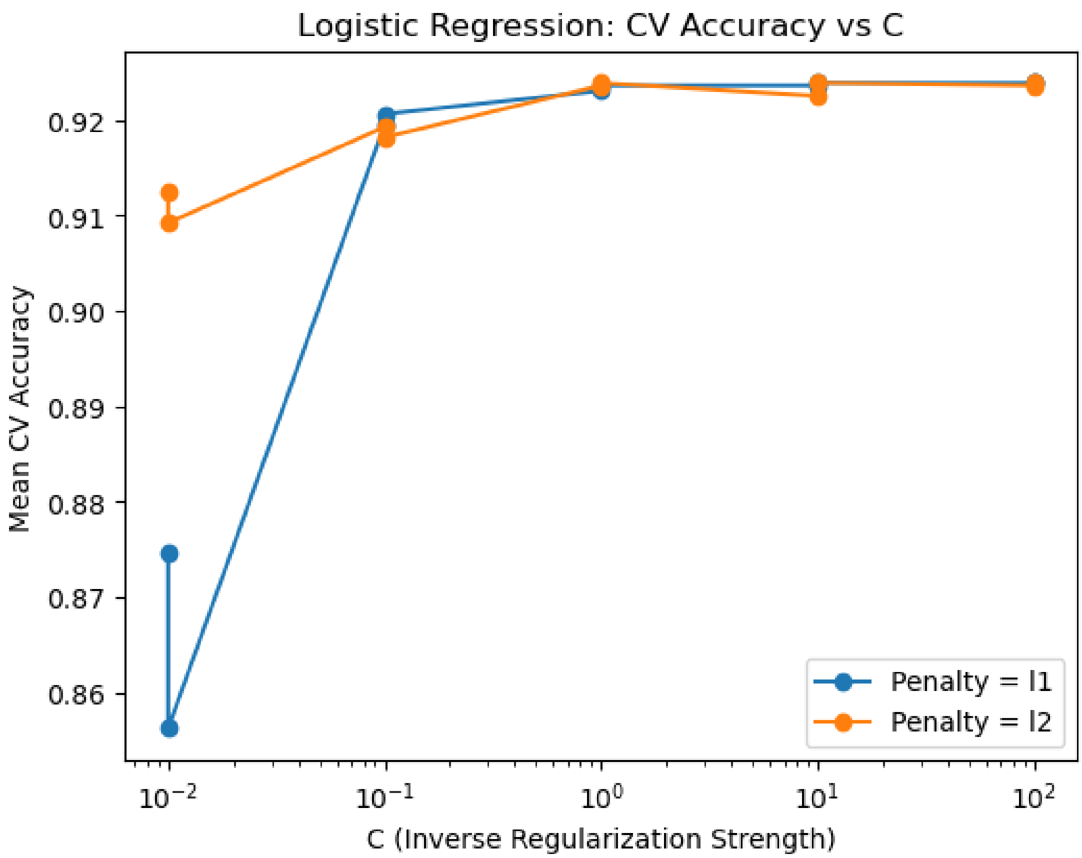
[38]:
```python
lr_results = pd.DataFrame(grid_search_lr.cv_results_)

plt.figure()

for penalty in lr_results['param_penalty'].unique():
    subset = lr_results[lr_results['param_penalty'] == penalty]
    plt.plot(
        subset['param_C'],
        subset['mean_test_score'],
        marker='o',
        label=f'Penalty = {penalty}'
    )

plt.xscale('log')
plt.xlabel('C (Inverse Regularization Strength)')
plt.ylabel('Mean CV Accuracy')
plt.title('Logistic Regression: CV Accuracy vs C')
plt.legend()
```

```
plt.show()
```



[26]:
```
# SVM
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
import time
```

[21]:
```
start = time.time()

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_scaled, y_train)

time_linear = time.time() - start
y_pred_linear = svm_linear.predict(X_test_scaled)

acc_linear = accuracy_score(y_test, y_pred_linear)
f1_linear = f1_score(y_test, y_pred_linear)
```

[22]:
```python
# Polynonmial kernel

start = time.time()

svm_poly = SVC(kernel='poly', degree=3)
svm_poly.fit(X_train_scaled, y_train)

time_poly = time.time() - start
y_pred_poly = svm_poly.predict(X_test_scaled)

acc_poly = accuracy_score(y_test, y_pred_poly)
f1_poly = f1_score(y_test, y_pred_poly)
```

[23]:
```python
# RBF kernel

start = time.time()

svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train_scaled, y_train)

time_rbf = time.time() - start
y_pred_rbf = svm_rbf.predict(X_test_scaled)

acc_rbf = accuracy_score(y_test, y_pred_rbf)
f1_rbf = f1_score(y_test, y_pred_rbf)
```

[24]:
```python
# sigmoid kernel

start = time.time()

svm_sigmoid = SVC(kernel='sigmoid')
svm_sigmoid.fit(X_train_scaled, y_train)

time_sigmoid = time.time() - start
y_pred_sigmoid = svm_sigmoid.predict(X_test_scaled)

acc_sigmoid = accuracy_score(y_test, y_pred_sigmoid)
f1_sigmoid = f1_score(y_test, y_pred_sigmoid)
```

[25]:
```python
print("Linear   :", acc_linear, f1_linear, time_linear)
print("Poly     :", acc_poly, f1_poly, time_poly)
print("RBF      :", acc_rbf, f1_rbf, time_rbf)
print("Sigmoid  :", acc_sigmoid, f1_sigmoid, time_sigmoid)
```

[40]:
```python
# Hyperparameter Tuning for SVM

param_grid_svm = {
```

```python
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3]    # used only for polynomial kernel
}
```

[41]:
```python
# grid search with k=5 folds
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

svm = SVC()

grid_search_svm = GridSearchCV(
    estimator=svm,
    param_grid=param_grid_svm,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_search_svm.fit(X_train_scaled, y_train)
```

[42]:
```python
grid_search_svm.best_params_
```

[42]:
```
{'C': 10, 'degree': 2, 'gamma': 'scale', 'kernel': 'rbf'}
```

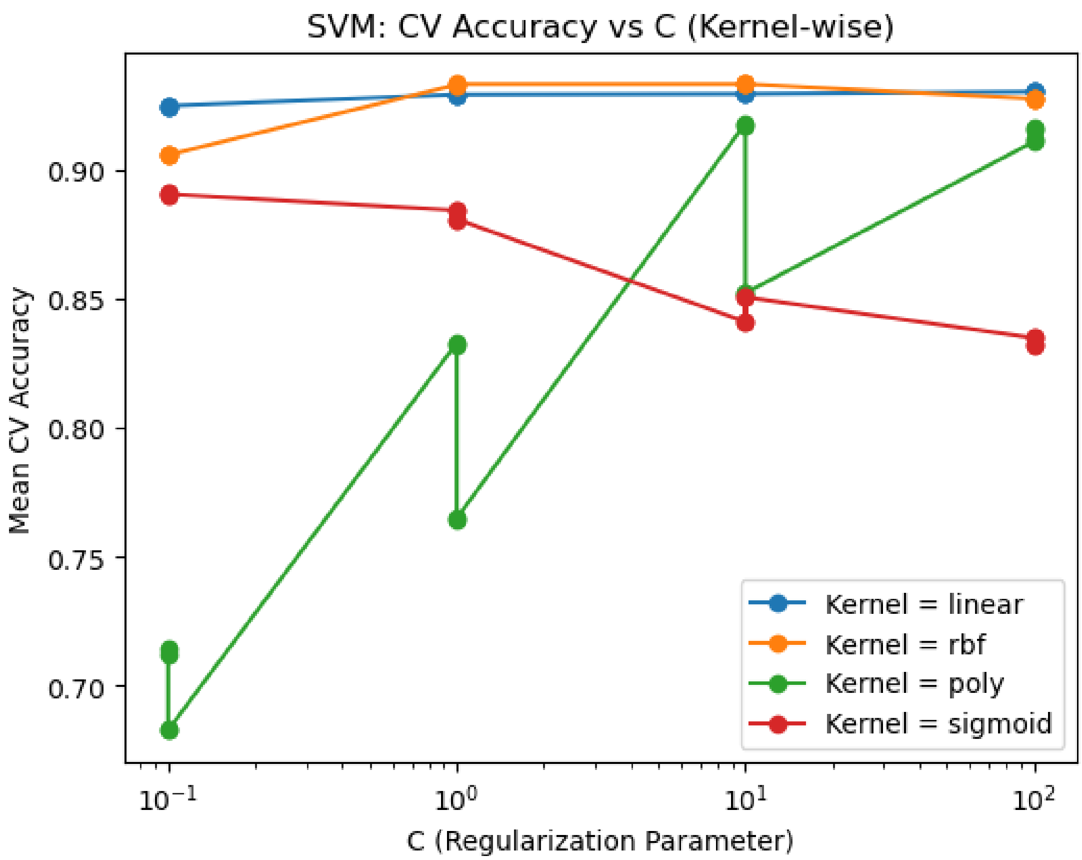
[43]:
```python
grid_search_svm.best_score_
```

[46]:
```python
svm_results = pd.DataFrame(grid_search_svm.cv_results_)

plt.figure()

for kernel in svm_results['param_kernel'].unique():
    subset = svm_results[svm_results['param_kernel'] == kernel]
    plt.plot(
        subset['param_C'],
        subset['mean_test_score'],
        marker='o',
        label=f'Kernel = {kernel}'
    )

plt.xscale('log')
plt.xlabel('C (Regularization Parameter)')
plt.ylabel('Mean CV Accuracy')
plt.title('SVM: CV Accuracy vs C (Kernel-wise)')
plt.legend()
plt.show()
```



[32]:

```
# Cross-Validation for Logistic Regression

from sklearn.model_selection import cross_val_score

cv_lr = cross_val_score(
    best_lr,
    X_train_scaled,
    y_train,
    cv=5,
    scoring='accuracy'
)

cv_lr
```

[33]:

```
cv_lr.mean()
```

[35]:

```
# Cross-Validation for SVM

cv_svm = cross_val_score(
```

```
    best_svm,
    X_train_scaled,
    y_train,
    cv=5,
    scoring='accuracy'
)

cv_svm
```

[36]:
```
cv_svm.mean()
```

Hyperparameter Tuning Results

Model	Search Method	Best Parameters	Best CV Accuracy
Logistic Regression	Grid / Random	c = 10	0.9305
SVM	Grid / Random	c = 10	0.9207

Logistic Regression Performance

Metric	Value
Accuracy	0.9305
Precision	0.9211
Recall	0.9008
F1 Score	0.9108
Training Time (s)	0.1202

SVM Kernel-wise Performance

Kernel	Accuracy	F1 Score	Training Time (s)
Linear	0.9294	0.9093	0.6027
Polynomial	0.7795	0.6219	0.6716
RBF	0.9272	0.9055	0.3623
Sigmoid	0.8849	0.8527	0.3351

Fold	Logistic Regression	SVM
Fold 1	0.9402	0.9429
Fold 2	0.9157	0.9320
Fold 3	0.9225	0.9334
Fold 4	0.9171	0.9211
Fold 5	0.9252	0.9361
Average	0.9241	0.9331

Criterion	Logistic Regression	SVM
Accuracy	0.9305	0.9294
Model Complexity	Low	High
Training Time	Low	High
Interpretability	High	Low

K-Fold Cross-Validation Results (K = 5)

Comparative Analysis

Observations

- The Support Vector Machine (SVM) with RBF kernel was identified as the best-performing classifier, achieving higher accuracy and F1-score than Logistic Regression.
- Hyperparameter tuning showed that a regularization strength of $C = 10$ produced optimal performance for both Logistic Regression and SVM, effectively controlling overfitting.
- Kernel-wise analysis of SVM indicated that the RBF kernel captured non-linear patterns in the data better than linear, polynomial, and sigmoid kernels.
- The tuned SVM demonstrated a better bias–variance trade-off, offering strong generalization on unseen test data.

Learning Outcomes

- Understood probabilistic and margin-based classifiers.
- Applied hyperparameter tuning.
- Evaluated classification models.
- Interpreted experimental results.

References

- Scikit-learn: Logistic Regression

- Scikit-learn: Support Vector Machines
- Scikit-learn: Hyperparameter Optimization
- Spambase Dataset – Kaggle
- UCI ML Repository – Spambase