M6 Lab: Model Selection with K-Fold Cross Validation (a1a dataset)

This lab demonstrates K-Fold Cross Validation using scikit-learn with Support Vector Machine (SVM) classifiers.

Using a1a dataset - smaller and faster for demonstration purposes

Part I: Understanding the Model Parameters

SVM Classifier Parameters Explanation:

- **C**: Regularization parameter. Controls the trade-off between achieving a low training error and a low testing error (generalization). Smaller values create a wider margin but may misclassify more points. Default is 1.0.
- **kernel**: Specifies the kernel type to be used in the algorithm. Options include 'linear', 'poly', 'rbf', 'sigmoid'. The kernel function transforms the data into a higher dimensional space.
- **degree**: Degree of the polynomial kernel function ('poly'). Ignored by all other kernels. Default is 3.
- **gamma**: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Defines how far the influence of a single training example reaches. Low values mean 'far' and high values mean 'close'. Default is 'scale'.
- **coef0**: Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'. Default is 0.0.
- **tol**: Tolerance for stopping criterion. The algorithm stops when the optimization improvement is below this threshold. Default is 1e-3.
- **verbose**: Enable verbose output during training. Controls the amount of detail printed during fitting. Default is False.
- **max_iter**: Hard limit on iterations within solver. -1 means no limit. Default is -1.
- **decision_function_shape**: Determines the shape of the decision function. 'ovo' (one-vs-one) or 'ovr' (one-vs-rest). Default is 'ovr'.
- **random_state**: Controls the random number generation for shuffling the data. Pass an int for reproducible output across multiple function calls.

Part II: Simple Cross Validation

In this section, we'll perform basic cross-validation with different kernel configurations.

In [1]:
```python
from sklearn.datasets import load_svmlight_file
from sklearn import svm
from sklearn.model_selection import cross_val_score
import numpy as np
```

Load the Dataset

In [2]:
```python
print("Loading Dataset...")
X, y = load_svmlight_file("a1a.txt")
print(f"Dataset loaded: {X.shape[0]} samples, {X.shape[1]} features")
```

```
Loading Dataset...
Dataset loaded: 1605 samples, 119 features
```

Test 1: Linear Kernel with C=1

In [3]:
```
print("Creating classifier object...")
clf = svm.SVC(kernel='linear', C=1, random_state=42)

print("Training classifier with cross validation, k=5")
scores = cross_val_score(clf, X, y, cv=5)

print("Training Complete!")
acc = scores.mean()
stdiv = scores.std()

print(f"Cross Validation Mean Accuracy = {acc:.2f}")
print(f"Standard Deviation of the Mean Accuracy across all runs = {stdiv:.2f}")
```

```
Creating classifier object...
Training classifier with cross validation, k=5
Training Complete!
Cross Validation Mean Accuracy = 0.83
Standard Deviation of the Mean Accuracy across all runs = 0.02
```

Test 2: RBF Kernel with gamma=0.1

In [4]:
```
print("Creating classifier with RBF kernel, gamma=0.1...")
clf = svm.SVC(kernel='rbf', gamma=0.1, random_state=42)

print("Training classifier with cross validation, k=5")
scores = cross_val_score(clf, X, y, cv=5)

print("Training Complete!")
acc = scores.mean()
stdiv = scores.std()

print(f"Cross Validation Mean Accuracy = {acc:.2f}")
print(f"Standard Deviation of the Mean Accuracy across all runs = {stdiv:.2f}")
```

```
Creating classifier with RBF kernel, gamma=0.1...
Training classifier with cross validation, k=5
Training Complete!
Cross Validation Mean Accuracy = 0.83
Standard Deviation of the Mean Accuracy across all runs = 0.03
```

Test 3: RBF Kernel with gamma=0.01

In [5]:
```
print("Creating classifier with RBF kernel, gamma=0.01...")
clf = svm.SVC(kernel='rbf', gamma=0.01, random_state=42)

print("Training classifier with cross validation, k=5")
scores = cross_val_score(clf, X, y, cv=5)

print("Training Complete!")
acc = scores.mean()
stdiv = scores.std()

print(f"Cross Validation Mean Accuracy = {acc:.2f}")
print(f"Standard Deviation of the Mean Accuracy across all runs = {stdiv:.2f}")
```

```
Creating classifier with RBF kernel, gamma=0.01...
Training classifier with cross validation, k=5
Training Complete!
Cross Validation Mean Accuracy = 0.82
Standard Deviation of the Mean Accuracy across all runs = 0.02
```

Test 4: Polynomial Kernel with degree=2

In [6]:
```
print("Creating classifier with polynomial kernel, degree=2...")
clf = svm.SVC(kernel='poly', degree=2, random_state=42)

print("Training classifier with cross validation, k=5")
scores = cross_val_score(clf, X, y, cv=5)

print("Training Complete!")
acc = scores.mean()
stdiv = scores.std()

print(f"Cross Validation Mean Accuracy = {acc:.2f}")
print(f"Standard Deviation of the Mean Accuracy across all runs = {stdiv:.2f}")
```

```
Creating classifier with polynomial kernel, degree=2...
Training classifier with cross validation, k=5
Training Complete!
Cross Validation Mean Accuracy = 0.83
Standard Deviation of the Mean Accuracy across all runs = 0.02
```

Part III: Parameter Fine Tuning

Now we'll use GridSearchCV to systematically test multiple parameter combinations.

In [7]:
```
import pandas as pd
from sklearn.model_selection import GridSearchCV
```

Basic Grid Search (Original Example)

In [8]:
```
print("Loading Dataset...")
X, y = load_svmlight_file("a1a.txt")

print("Creating Parameter Grid...")
param_grid = [
    {'C': [1, 10], 'kernel': ['linear']},
    {'C': [1, 10], 'gamma': [0.001, 0.01], 'kernel': ['rbf']},
]

print("Creating classifier object...")
svc = svm.SVC()

print("Creating a grid search cross validator object...")
clf = GridSearchCV(svc, param_grid)

print("Fitting the models with different parameters...")
clf.fit(X, y)

print("Writing all fitting results...")
df = pd.DataFrame(clf.cv_results_)
df.to_csv("Parameter_Tuning_Results_a1a.csv")

print("Results saved to Parameter_Tuning_Results_a1a.csv")
```

```
Loading Dataset...
Creating Parameter Grid...
Creating classifier object...
Creating a grid search cross validator object...
Fitting the models with different parameters...
Writing all fitting results...
Results saved to Parameter_Tuning_Results_a1a.csv
```

Display Results Summary

In [9]:
```
# Display basic info about the results
print("\nColumns in the results dataframe:")
print(df.columns.tolist())

print("\nNumber of parameter combinations tested:", len(df))

print("\nBest parameters found:")
print(clf.best_params_)

print("\nBest cross-validation score:")
print(f"{clf.best_score_:.4f}")
```

```
Columns in the results dataframe:
['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_C', 'param_kernel', 'param_gam
ma', 'params', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_test_score', 'split4_t
est_score', 'mean_test_score', 'std_test_score', 'rank_test_score']

Number of parameter combinations tested: 6

Best parameters found:
{'C': 10, 'kernel': 'linear'}

Best cross-validation score:
0.8287
```

Extended Grid Search with Polynomial Kernel

In [10]:
```
print("Creating Extended Parameter Grid with Polynomial Kernel...")
param_grid_extended = [
    {'C': [0.01, 0.1, 1, 10], 'kernel': ['linear']},
    {'C': [0.01, 0.1, 1, 10], 'gamma': [0.001, 0.01], 'kernel': ['rbf']},
    {'C': [0.01, 0.1, 1, 10], 'degree': [2, 3], 'kernel': ['poly']},
]

print("Creating classifier object...")
svc = svm.SVC()

print("Creating a grid search cross validator object...")
clf_extended = GridSearchCV(svc, param_grid_extended)

print("Fitting the models with different parameters...")
clf_extended.fit(X, y)

print("Writing all fitting results...")
df_extended = pd.DataFrame(clf_extended.cv_results_)
df_extended.to_csv("Parameter_Tuning_Results_Extended_a1a.csv")

print("Results saved to Parameter_Tuning_Results_Extended_a1a.csv")
```

```
Creating Extended Parameter Grid with Polynomial Kernel...
Creating classifier object...
Creating a grid search cross validator object...
Fitting the models with different parameters...
Writing all fitting results...
Results saved to Parameter_Tuning_Results_Extended_a1a.csv
```

Print Ranked Results

In [11]:
```
# Sort by rank and display results
results_sorted = df_extended.sort_values('rank_test_score')

print("\nRanked Parameter Sets:\n")
for idx, row in results_sorted.iterrows():
    rank = row['rank_test_score']
    params = row['params']
    mean_acc = row['mean_test_score']
    std_dev = row['std_test_score']
    print(f"Rank {rank}: {params}, Mean Test Accuracy={mean_acc}, Mean StdDev={std_dev}")
```

```
Ranked Parameter Sets:

Rank 1: {'C': 0.1, 'kernel': 'linear'}, Mean Test Accuracy=0.8317757009345794, Mean StdDev=0.020944219705883
63
Rank 1: {'C': 1, 'degree': 2, 'kernel': 'poly'}, Mean Test Accuracy=0.8317757009345794, Mean StdDev=0.024922
118380062308
Rank 3: {'C': 1, 'degree': 3, 'kernel': 'poly'}, Mean Test Accuracy=0.8305295950155763, Mean StdDev=0.025809
738538576925
Rank 4: {'C': 10, 'kernel': 'linear'}, Mean Test Accuracy=0.8286604361370715, Mean StdDev=0.0200928542013671
07
Rank 5: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}, Mean Test Accuracy=0.8274143302180686, Mean StdDev=0.0234
45342956045834
Rank 6: {'C': 1, 'kernel': 'linear'}, Mean Test Accuracy=0.8274143302180684, Mean StdDev=0.02005417687156521
2
Rank 7: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}, Mean Test Accuracy=0.8249221183800624, Mean StdDev=0.023
593889381634255
Rank 8: {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}, Mean Test Accuracy=0.8230529595015575, Mean StdDev=0.02464
014944957656
Rank 9: {'C': 0.1, 'degree': 2, 'kernel': 'poly'}, Mean Test Accuracy=0.8137071651090343, Mean StdDev=0.0179
9318265195471
Rank 9: {'C': 0.01, 'kernel': 'linear'}, Mean Test Accuracy=0.8137071651090343, Mean StdDev=0.01412558759969
425
Rank 11: {'C': 10, 'degree': 2, 'kernel': 'poly'}, Mean Test Accuracy=0.811214953271028, Mean StdDev=0.01701
7445836445686
Rank 12: {'C': 0.1, 'degree': 3, 'kernel': 'poly'}, Mean Test Accuracy=0.8080996884735201, Mean StdDev=0.021
72659909552984
Rank 13: {'C': 10, 'degree': 3, 'kernel': 'poly'}, Mean Test Accuracy=0.7968847352024921, Mean StdDev=0.0142
6233413365682
Rank 14: {'C': 0.1, 'gamma': 0.001, 'kernel': 'rbf'}, Mean Test Accuracy=0.7538940809968847, Mean StdDev=0.0
Rank 14: {'C': 0.01, 'gamma': 0.01, 'kernel': 'rbf'}, Mean Test Accuracy=0.7538940809968847, Mean StdDev=0.0
Rank 14: {'C': 0.01, 'gamma': 0.001, 'kernel': 'rbf'}, Mean Test Accuracy=0.7538940809968847, Mean StdDev=0.
0
Rank 14: {'C': 0.01, 'degree': 2, 'kernel': 'poly'}, Mean Test Accuracy=0.7538940809968847, Mean StdDev=0.0
Rank 14: {'C': 0.01, 'degree': 3, 'kernel': 'poly'}, Mean Test Accuracy=0.7538940809968847, Mean StdDev=0.0
Rank 14: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}, Mean Test Accuracy=0.7538940809968847, Mean StdDev=0.0
Rank 14: {'C': 0.1, 'gamma': 0.01, 'kernel': 'rbf'}, Mean Test Accuracy=0.7538940809968847, Mean StdDev=0.0
```

Display Top 5 Results in a Clean Format

In [12]:
```
# Show top 5 results in a more readable format
print("\nTop 5 Parameter Combinations:\n")
top_5 = results_sorted.head(5)
display_cols = ['rank_test_score', 'params', 'mean_test_score', 'std_test_score']
print(top_5[display_cols].to_string(index=False))
```

```
Top 5 Parameter Combinations:

 rank_test_score                                    params  mean_test_score  std_test_score
               1            {'C': 0.1, 'kernel': 'linear'}         0.831776        0.020944
               1   {'C': 1, 'degree': 2, 'kernel': 'poly'}         0.831776        0.024922
               3   {'C': 1, 'degree': 3, 'kernel': 'poly'}         0.830530        0.025810
               4             {'C': 10, 'kernel': 'linear'}         0.828660        0.020093
               5 {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}         0.827414        0.023445
```

Extra Credit: K-Fold Cross Validation with Different Testing Folds

Implementing standard K-Fold CV where each fold serves as the test set once.

In [13]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
```

In [14]:
```
print("Loading Dataset...")
X, y = load_svmlight_file("a1a.txt")
```

```python
# Setup K-Fold with K=5
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Model configurations to test
models = [
    ('Linear', svm.SVC(kernel='linear', random_state=42)),
    ('RBF gamma=0.01', svm.SVC(kernel='rbf', gamma=0.01, random_state=42)),
    ('RBF gamma=0.001', svm.SVC(kernel='rbf', gamma=0.001, random_state=42))
]

print("\nRunning K-Fold Cross Validation with K=5\n")

# Test each model
for model_name, model in models:
    print(f"Testing {model_name}...")
    fold_accuracies = []

    fold_num = 1
    for train_index, test_index in kfold.split(X):
        # Split data
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Train model
        model.fit(X_train, y_train)

        # Predict and calculate accuracy
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        fold_accuracies.append(accuracy)

        print(f"  Fold {fold_num}: Accuracy = {accuracy:.4f}")
        fold_num += 1

    # Calculate average
    avg_accuracy = np.mean(fold_accuracies)
    std_accuracy = np.std(fold_accuracies)

    print(f"  Average Accuracy: {avg_accuracy:.4f}")
    print(f"  Standard Deviation: {std_accuracy:.4f}")
    print()
```

```
Loading Dataset...

Running K-Fold Cross Validation with K=5

Testing Linear...
  Fold 1: Accuracy = 0.8380
  Fold 2: Accuracy = 0.8193
  Fold 3: Accuracy = 0.8411
  Fold 4: Accuracy = 0.8318
  Fold 5: Accuracy = 0.8505
  Average Accuracy: 0.8361
  Standard Deviation: 0.0104

Testing RBF gamma=0.01...
  Fold 1: Accuracy = 0.8193
  Fold 2: Accuracy = 0.8349
  Fold 3: Accuracy = 0.8505
  Fold 4: Accuracy = 0.8131
  Fold 5: Accuracy = 0.8318
  Average Accuracy: 0.8299
  Standard Deviation: 0.0130

Testing RBF gamma=0.001...
  Fold 1: Accuracy = 0.7414
  Fold 2: Accuracy = 0.7539
  Fold 3: Accuracy = 0.7850
  Fold 4: Accuracy = 0.7227
  Fold 5: Accuracy = 0.7664
  Average Accuracy: 0.7539
  Standard Deviation: 0.0212
```

Compare All Models

In [15]:
```python
# Summary comparison
print("\n" + "="*60)
print("SUMMARY: Which model gives the highest accuracy?")
print("="*60)

results_summary = []
for model_name, model in models:
    fold_accuracies = []
    for train_index, test_index in kfold.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        fold_accuracies.append(accuracy_score(y_test, y_pred))

    avg_acc = np.mean(fold_accuracies)
    results_summary.append((model_name, avg_acc))

# Sort by accuracy
results_summary.sort(key=lambda x: x[1], reverse=True)

for i, (name, acc) in enumerate(results_summary, 1):
    print(f"{i}. {name}: {acc:.4f}")

print(f"\nBest Model: {results_summary[0][0]} with accuracy {results_summary[0][1]:.4f}")
```

```
============================================================
SUMMARY: Which model gives the highest accuracy?
============================================================
1. Linear: 0.8361
2. RBF gamma=0.01: 0.8299
3. RBF gamma=0.001: 0.7539

Best Model: Linear with accuracy 0.8361
```