P1

1. The original objective maximizes the margin by minimizing $|\beta|^{-1}$, which is equivalent to maximizing $|\beta|$. To normalize $|\beta| = 1$, the scaling factor is absorbed, and the margin constraint becomes:

$$y_i(\beta^T x_i + \beta_0) \geq M,$$

where $M = \min_i y_i(\beta^T x_i + \beta_0)$. This reformulation maintains equivalence.

2. From the dual form of the SVM, the weight vector β is expressed as:

$$\beta = \sum_{i=1}^{n} a_i y_i x_i,$$

where $a_i > 0$ only for support vectors. Substituting β into the decision function:

$$f(x) = \beta^T x + \beta_0 = \left(\sum_{i=1}^{n} a_i y_i x_i \right)^T x + \beta_0,$$

simplifies to:

$$f(x) = \beta_0 + \sum_{i=1}^{n} a_i \langle x_i, x \rangle.$$

3. For the RBF (Radial Basis Function) kernel:

$$K(x, x') = \exp(-\gamma |x - x'|^2),$$

the parameter γ determines the smoothness of the decision boundary:

- High γ: The decision boundary becomes very flexible, fitting tightly to the training data, leading to high variance and low bias (overfitting).
- Low γ: The decision boundary becomes smoother and less sensitive to individual data points, resulting in low variance and high bias (underfitting).

4. Handling missing data in PCA:

   PCA requires complete data for covariance or SVD computation. Missing data can distort results.

   Best preprocessing practices:

   A. Imputation: Replace missing values using: - Mean, median, or mode of the feature. - KNN-based imputation. - Regression-based methods.
   B. Standardization: Normalize features to have zero mean and unit variance.
   C. Removing incomplete rows/columns: If the amount of missing data is small.

5. The K-means++ algorithm initializes cluster centers as follows:

A. Select the first center randomly.
B. For each subsequent center, select a point with probability proportional to its squared distance from the nearest existing center.

Advantages over standard K-means:

A. Reduces the chance of poor initialization.
B. Speeds up convergence.
C. Improves final clustering performance by ensuring diverse initial centers.

P2

1. Best Parameters: {'n_estimators': 250, 'max_features': 'log2'}, Best CV Error: 0.1868

   Validation Confusion Matrix:

   4041 153 229 182

   273 2817 153 276

   564 201 1537 355

   237 196 215 4813

   Top 10 Important Keywords:

   'space' 'religion' 'graphics' 'jews' 'team' 'government' 'god' 'car' 'christian' 'windows'

2. Best Parameters: {'n_estimators': 250, 'learning_rate': 0.2}, Best CV Error: 0.1765

   Validation Confusion Matrix:

   4128 77 203 197

   285 2747 155 332

   565 120 1621 351

   225 122 235 4879

3. Boosting trees are better.

4. Cross-Validation Misclassification Error: 0.2221

   Validation Confusion Matrix:

   3942 78 308 277

   279 2404 235 601

531 123 1542 461

204 147 363 4747

5. Cross-Validation Misclassification Error: 0.2385

   Validation Confusion Matrix:

   3884 617 55 49

   159 3234 25 101

   569 873 987 228

   225 864 108 4264

6. XGBoost is the best method.

```
   Model  CV Error
```

Random Forest 0.186800

```
     XGBoost  0.176518
         LDA  0.222079
         QDA  0.238456
```

P3

All the following times **include the time for cross validation**.

1. Best C: 0.01

   Misclassification Error: 0.0061

   Confusion Matrix:

   1251 11

   ```
   4 1196
   ```
   Training Time: 1.82 seconds

2. Best Parameters: {'C': 1000, 'gamma': 0.001}

   Misclassification Error: 0.0049

   Confusion Matrix:

   1255 7

   ```
   5 1195
   ```
   Training Time: 6.41 seconds

3. Binary Model Comparison (3 vs 6)

Linear Kernel: Best C=0.01, Error=0.0061, Time=1.82s

RBF Kernel: Best C=1000, Gamma=0.001, Error=0.0049, Time=6.41s

So Linear Kernel is fast, but RBF Kernel has less Error.

4. Best C: 0.1

   Misclassification Error: 0.0460

   Confusion Matrix:

   1346 9 1 7

```
 7 1134   18   26

18   14 1061   33

26   17   45 1044
```
   Training Time: 15.83 seconds

5. Best Parameters: {'C': 10, 'gamma': 0.01}

   Misclassification Error: 0.0357

   Confusion Matrix:

   1123 0 6 1 0 3 6 0 1 0

```
 0 1342   12    1    1    0    1    2    3    1

 4    1 1155    6    4    1    0    8    5    1

 0    2   37 1195    0    9    1    7    5    6

 2    2   16    0 1134    2    4    4    0   11

 4    2    9   18    1 1067   11    2   10    2

 4    0   16    0    3    7 1166    0    4    0

 1    2   18    2    6    1    0 1217    0   17

 5    3   10   12    6    7    1    5 1074    9

 4    2   11    3   16    2    0   13    3 1099
```
   Training Time: 1656.18 seconds

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, Qua
from sklearn.model_selection import cross_val_score, StratifiedKFold, cro
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
from xgboost import XGBClassifier  # 使用 XGBoost

# 数据加载和预处理
def load_data():
    # 加载数据
    wordlist = pd.read_csv("20newsgroup/wordlist.txt", header=None, names
    documents = pd.read_csv("20newsgroup/documents.txt", header=None, sep
    newsgroups = pd.read_csv("20newsgroup/newsgroups.txt", header=None, n
    groupnames = pd.read_csv("20newsgroup/groupnames.txt", header=None, n

    # 构建稠密矩阵
    num_posts = newsgroups.shape[0]  # 行数
    num_keywords = wordlist.shape[0]  # 列数
    X = np.zeros((num_posts, num_keywords))
    for _, row in documents.iterrows():
        X[int(row["row"]) - 1, int(row["col"]) - 1] = int(row["value"])

    # 修正类别标签为从 0 开始
    y = newsgroups["group"].values
    y = y - y.min()  # 将最小值变为 0（例如 [1, 2, 3, 4] -> [0, 1, 2, 3]）

    return X, y, wordlist, groupnames

# 加载数据
X, y, wordlist, groupnames = load_data()
print("Unique classes in y after mapping:", np.unique(y))  # 确保类别正确

# 随机森林分类器（含调参）
def random_forest_classifier(X, y):
    print("\n--- Random Forest ---")
    best_score = 1.0
    best_params = {}
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    for n_estimators in [200, 250, 300]:
        for max_features in ['sqrt', 'log2']:
            rf = RandomForestClassifier(n_estimators=n_estimators, max_fe
            cv_scores = cross_val_score(rf, X, y, cv=skf, scoring='accura
            cv_error = 1 - np.mean(cv_scores)
            print(f"Parameters: n_estimators={n_estimators}, max_features
            if cv_error < best_score:
                best_score = cv_error
                best_params = {"n_estimators": n_estimators, "max_feature
                best_model = rf

    print(f"Best Parameters: {best_params}, Best CV Error: {best_score:.4

    # 交叉验证阶段混淆矩阵
    y_pred_cv = cross_val_predict(best_model, X, y, cv=skf)
    conf_matrix_cv = confusion_matrix(y, y_pred_cv)
    print("Validation Confusion Matrix:\n", conf_matrix_cv)

    # 使用最佳参数训练最终模型并在训练集上计算混淆矩阵
    best_model.fit(X, y)
    y_pred_train = best_model.predict(X)
    conf_matrix_train = confusion_matrix(y, y_pred_train)
    print("Training Confusion Matrix:\n", conf_matrix_train)

    # 输出特征重要性
```

```
    feature_importances = best_model.feature_importances_
    important_features = np.argsort(feature_importances)[-10:]
    print("Top 10 Important Keywords:")
    print(wordlist.iloc[important_features]["word"].values)

    return best_score

# XGBoost 分类器（含调参）
def xgboost_classifier(X, y):
    print("\n--- XGBoost ---")
    best_score = 1.0
    best_params = {}
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    for n_estimators in [200, 250, 300]:
        for learning_rate in [0.1, 0.2, 0.3]:
            xgb = XGBClassifier(n_estimators=n_estimators, learning_rate=
            cv_scores = cross_val_score(xgb, X, y, cv=skf, scoring='accur
            cv_error = 1 - np.mean(cv_scores)
            print(f"Parameters: n_estimators={n_estimators}, learning_rat
            if cv_error < best_score:
                best_score = cv_error
                best_params = {"n_estimators": n_estimators, "learning_ra
                best_model = xgb

    print(f"Best Parameters: {best_params}, Best CV Error: {best_score:.4

    # 交叉验证阶段混淆矩阵
    y_pred_cv = cross_val_predict(best_model, X, y, cv=skf)
    conf_matrix_cv = confusion_matrix(y, y_pred_cv)
    print("Validation Confusion Matrix:\n", conf_matrix_cv)

    # 使用最佳参数训练最终模型并在训练集上计算混淆矩阵
    best_model.fit(X, y)
    y_pred_train = best_model.predict(X)
    conf_matrix_train = confusion_matrix(y, y_pred_train)
    print("Training Confusion Matrix:\n", conf_matrix_train)

    return best_score

# 线性判别分析
def lda_classifier(X, y):
    print("\n--- Linear Discriminant Analysis ---")
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    lda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
    cv_scores = cross_val_score(lda, X, y, cv=skf, scoring='accuracy')
    cv_error = 1 - np.mean(cv_scores)
    print(f"Cross-Validation Misclassification Error: {cv_error:.4f}")  #

    # 交叉验证阶段混淆矩阵
    y_pred_cv = cross_val_predict(lda, X, y, cv=skf)
    conf_matrix_cv = confusion_matrix(y, y_pred_cv)
    print("Validation Confusion Matrix:\n", conf_matrix_cv)

    # 模型训练与训练集评估
    lda.fit(X, y)
    y_pred_train = lda.predict(X)
    conf_matrix_train = confusion_matrix(y, y_pred_train)
    print("Training Confusion Matrix:\n", conf_matrix_train)
```

```python
    return cv_error

# 二次判别分析
def qda_classifier(X, y):
    print("\n--- Quadratic Discriminant Analysis ---")
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    qda = QuadraticDiscriminantAnalysis(reg_param=0.1)
    cv_scores = cross_val_score(qda, X, y, cv=skf, scoring='accuracy')
    cv_error = 1 - np.mean(cv_scores)
    print(f"Cross-Validation Misclassification Error: {cv_error:.4f}")  #

    # 交叉验证阶段混淆矩阵
    y_pred_cv = cross_val_predict(qda, X, y, cv=skf)
    conf_matrix_cv = confusion_matrix(y, y_pred_cv)
    print("Validation Confusion Matrix:\n", conf_matrix_cv)

    # 模型训练与训练集评估
    qda.fit(X, y)
    y_pred_train = qda.predict(X)
    conf_matrix_train = confusion_matrix(y, y_pred_train)
    print("Training Confusion Matrix:\n", conf_matrix_train)

    return cv_error

# 比较所有模型
def compare_models(X, y):
    print("\n--- Model Comparison ---")
    rf_error = random_forest_classifier(X, y)
    xgb_error = xgboost_classifier(X, y)
    lda_error = lda_classifier(X, y)
    qda_error = qda_classifier(X, y)

    models = ["Random Forest", "XGBoost", "LDA", "QDA"]
    cv_errors = [rf_error, xgb_error, lda_error, qda_error]

    print("\n--- Summary of Model Performances ---")
    results = pd.DataFrame({
        "Model": models,
        "CV Error": cv_errors
    })
    print(results)

# 执行所有模型
compare_models(X, y)
```

```
Unique classes in y after mapping: [0 1 2 3]

--- Model Comparison ---

--- Random Forest ---
Parameters: n_estimators=200, max_features=sqrt, CV Error=0.1899
Parameters: n_estimators=200, max_features=log2, CV Error=0.1877
Parameters: n_estimators=250, max_features=sqrt, CV Error=0.1904
Parameters: n_estimators=250, max_features=log2, CV Error=0.1868
Parameters: n_estimators=300, max_features=sqrt, CV Error=0.1906
Parameters: n_estimators=300, max_features=log2, CV Error=0.1875
Best Parameters: {'n_estimators': 250, 'max_features': 'log2'}, Best CV Er
ror: 0.1868
Validation Confusion Matrix:
 [[4041  153  229  182]
 [ 273 2817  153  276]
 [ 564  201 1537  355]
 [ 237  196  215 4813]]
Training Confusion Matrix:
 [[4370   95   69   71]
 [ 200 3115   73  131]
 [ 270  112 2113  162]
 [ 148  134   93 5086]]
Top 10 Important Keywords:
['space' 'religion' 'graphics' 'jews' 'team' 'government' 'god' 'car'
 'christian' 'windows']

--- XGBoost ---
Parameters: n_estimators=200, learning_rate=0.1, CV Error=0.1798
Parameters: n_estimators=200, learning_rate=0.2, CV Error=0.1766
Parameters: n_estimators=200, learning_rate=0.3, CV Error=0.1778
Parameters: n_estimators=250, learning_rate=0.1, CV Error=0.1781
Parameters: n_estimators=250, learning_rate=0.2, CV Error=0.1765
Parameters: n_estimators=250, learning_rate=0.3, CV Error=0.1792
Parameters: n_estimators=300, learning_rate=0.1, CV Error=0.1771
Parameters: n_estimators=300, learning_rate=0.2, CV Error=0.1766
Parameters: n_estimators=300, learning_rate=0.3, CV Error=0.1794
Best Parameters: {'n_estimators': 250, 'learning_rate': 0.2}, Best CV Erro
r: 0.1765
Validation Confusion Matrix:
 [[4128   77  203  197]
 [ 285 2747  155  332]
 [ 565  120 1621  351]
 [ 225  122  235 4879]]
Training Confusion Matrix:
 [[4259   61  134  151]
 [ 254 2850  123  292]
 [ 464   80 1838  275]
 [ 192   80  176 5013]]

--- Linear Discriminant Analysis ---
Cross-Validation Misclassification Error: 0.2221
Validation Confusion Matrix:
 [[3942   78  308  277]
 [ 279 2404  235  601]
 [ 531  123 1542  461]
 [ 204  147  363 4747]]
Training Confusion Matrix:
 [[4018   76  263  248]
 [ 309 2417  215  578]
```

```
 [ 578  122 1519  438]
 [ 229  145  346 4741]]

--- Quadratic Discriminant Analysis ---
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Cross-Validation Misclassification Error: 0.2385
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/liuliangjie/Library/Python/3.9/lib/python/site-packages/sklearn/dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Validation Confusion Matrix:
 [[3884  617   55   49]
 [ 159 3234   25  101]
 [ 569  873  987  228]
 [ 225  864  108 4264]]
Training Confusion Matrix:
 [[3908  594   54   49]
 [ 156 3240   24   99]
 [ 571  843 1028  215]
 [ 230  855  108 4268]]

--- Summary of Model Performances ---
           Model  CV Error
0  Random Forest  0.186800
1        XGBoost  0.176518
2            LDA  0.222079
3            QDA  0.238456
```

In [2]:

```python
import pandas as pd
import numpy as np
from sklearn.svm import SVC
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, classificat
import time
from sklearn.preprocessing import StandardScaler

# 数据加载和预处理
def load_data():
    train_data = pd.read_csv("MNIST/train_resized.csv").values
    test_data = pd.read_csv("MNIST/test_resized.csv").values

    X_train, y_train = train_data[:, 1:], train_data[:, 0]
    X_test, y_test = test_data[:, 1:], test_data[:, 0]

    return X_train, y_train, X_test, y_test

X_train, y_train, X_test, y_test = load_data()

# 添加标准化器
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 二分类任务（线性核）
def svm_linear_binary(X_train, y_train, X_test, y_test):
    print("\n--- Binary Classification (3 vs 6, Linear Kernel) ---")
    train_idx = np.isin(y_train, [3, 6])
    test_idx = np.isin(y_test, [3, 6])
    X_train_bin, y_train_bin = X_train[train_idx], y_train[train_idx]
    X_test_bin, y_test_bin = X_test[test_idx], y_test[test_idx]
    y_train_bin = (y_train_bin == 6).astype(int)
    y_test_bin = (y_test_bin == 6).astype(int)

    print(f"Training Samples: {X_train_bin.shape[0]}, Test Samples: {X_te

    param_grid = {'C': [0.001, 0.01, 0.1]}
    clf = GridSearchCV(SVC(kernel='linear'), param_grid, cv=5)
    start_time = time.time()
    clf.fit(X_train_bin, y_train_bin)
    train_time = time.time() - start_time

    best_C = clf.best_params_['C']
    y_pred = clf.predict(X_test_bin)
    misclassification_error = 1 - accuracy_score(y_test_bin, y_pred)

    print(f"Best C: {best_C}")
    print(f"Misclassification Error: {misclassification_error:.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test_bin, y_pred))
    print(f"Training Time: {train_time:.2f} seconds")

    return {"C": best_C, "error": misclassification_error, "time": train_

# 二分类任务（径向基核）
def svm_rbf_binary(X_train, y_train, X_test, y_test):
    print("\n--- Binary Classification (3 vs 6, RBF Kernel) ---")
    train_idx = np.isin(y_train, [3, 6])
    test_idx = np.isin(y_test, [3, 6])
    X_train_bin, y_train_bin = X_train[train_idx], y_train[train_idx]
    X_test_bin, y_test_bin = X_test[test_idx], y_test[test_idx]
    y_train_bin = (y_train_bin == 6).astype(int)
    y_test_bin = (y_test_bin == 6).astype(int)
```

```
    print(f"Training Samples: {X_train_bin.shape[0]}, Test Samples: {X_te

    param_grid = {'C': [100, 1000, 10000], 'gamma': [0.0001, 0.001, 0.01]
    clf = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=5)
    start_time = time.time()
    clf.fit(X_train_bin, y_train_bin)
    train_time = time.time() - start_time

    best_params = clf.best_params_
    y_pred = clf.predict(X_test_bin)
    misclassification_error = 1 - accuracy_score(y_test_bin, y_pred)

    print(f"Best Parameters: {best_params}")
    print(f"Misclassification Error: {misclassification_error:.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test_bin, y_pred))
    print(f"Training Time: {train_time:.2f} seconds")

    return {"C": best_params["C"], "gamma": best_params["gamma"], "error"

# 多分类任务（1, 2, 5, 8 的分类，线性核）
def svm_linear_multi(X_train, y_train, X_test, y_test):
    print("\n--- Multi-Class Classification (1, 2, 5, 8, Linear Kernel) -
    train_idx = np.isin(y_train, [1, 2, 5, 8])
    test_idx = np.isin(y_test, [1, 2, 5, 8])
    X_train_multi, y_train_multi = X_train[train_idx], y_train[train_idx]
    X_test_multi, y_test_multi = X_test[test_idx], y_test[test_idx]

    print(f"Training Samples: {X_train_multi.shape[0]}, Test Samples: {X_

    param_grid = {'C': [0.01, 0.1, 1]}
    clf = GridSearchCV(SVC(kernel='linear'), param_grid, cv=5)
    start_time = time.time()
    clf.fit(X_train_multi, y_train_multi)
    train_time = time.time() - start_time

    best_C = clf.best_params_['C']
    y_pred = clf.predict(X_test_multi)
    misclassification_error = 1 - accuracy_score(y_test_multi, y_pred)

    print(f"Best C: {best_C}")
    print(f"Misclassification Error: {misclassification_error:.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test_multi, y_pred))
    print(f"Training Time: {train_time:.2f} seconds")

    return {"C": best_C, "error": misclassification_error, "time": train_

# 多分类任务（全10类分类，径向基核）
def svm_full(X_train, y_train, X_test, y_test):
    print("\n--- Multi-Class Classification (All 10 Digits, RBF Kernel) -
    param_grid = {'C': [1, 10, 100], 'gamma': [0.001, 0.01, 0.1]}  # 更新
    clf = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=5)
    start_time = time.time()
    clf.fit(X_train, y_train)
    train_time = time.time() - start_time

    best_params = clf.best_params_
    y_pred = clf.predict(X_test)
    misclassification_error = 1 - accuracy_score(y_test, y_pred)
```

```
    print(f"Best Parameters: {best_params}")
    print(f"Misclassification Error: {misclassification_error:.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print(f"Classification Report:\n", classification_report(y_test, y_pr
    print(f"Training Time: {train_time:.2f} seconds")

    return {"C": best_params["C"], "gamma": best_params["gamma"], "error"

# 模型性能比较
def compare_binary_models(linear_results, rbf_results):
    print("\n--- Binary Model Comparison (3 vs 6) ---")
    print(f"Linear Kernel: Best C={linear_results['C']}, Error={linear_re
    print(f"RBF Kernel: Best C={rbf_results['C']}, Gamma={rbf_results['ga

# 汇总所有模型的结果
def summarize_results(results):
    print("\n--- Model Performance Summary ---")
    df = pd.DataFrame(results)
    print(df)

# 执行所有任务
results = []
linear_binary = svm_linear_binary(X_train, y_train, X_test, y_test)
results.append({"Task": "Binary (3 vs 6, Linear)", **linear_binary})
rbf_binary = svm_rbf_binary(X_train, y_train, X_test, y_test)
results.append({"Task": "Binary (3 vs 6, RBF)", **rbf_binary})
compare_binary_models(linear_binary, rbf_binary)

linear_multi = svm_linear_multi(X_train, y_train, X_test, y_test)
results.append({"Task": "Multi-Class (1, 2, 5, 8, Linear)", **linear_mult
full_class = svm_full(X_train, y_train, X_test, y_test)
results.append({"Task": "Full 10-Class (RBF)", **full_class})

summarize_results(results)
```

```
--- Binary Classification (3 vs 6, Linear Kernel) ---
Training Samples: 6026, Test Samples: 2462
Best C: 0.01
Misclassification Error: 0.0061
Confusion Matrix:
 [[1251   11]
 [   4 1196]]
Training Time: 1.82 seconds

--- Binary Classification (3 vs 6, RBF Kernel) ---
Training Samples: 6026, Test Samples: 2462
Best Parameters: {'C': 1000, 'gamma': 0.001}
Misclassification Error: 0.0049
Confusion Matrix:
 [[1255    7]
 [   5 1195]]
Training Time: 6.41 seconds

--- Binary Model Comparison (3 vs 6) ---
Linear Kernel: Best C=0.01, Error=0.0061, Time=1.82s
RBF Kernel: Best C=1000, Gamma=0.001, Error=0.0049, Time=6.41s

--- Multi-Class Classification (1, 2, 5, 8, Linear Kernel) ---
Training Samples: 11913, Test Samples: 4806
Best C: 0.1
Misclassification Error: 0.0460
Confusion Matrix:
 [[1346    9    1    7]
 [   7 1134   18   26]
 [  18   14 1061   33]
 [  26   17   45 1044]]
Training Time: 15.83 seconds

--- Multi-Class Classification (All 10 Digits, RBF Kernel) ---
Best Parameters: {'C': 10, 'gamma': 0.01}
Misclassification Error: 0.0357
Confusion Matrix:
 [[1123    0    6    1    0    3    6    0    1    0]
 [   0 1342   12    1    1    0    1    2    3    1]
 [   4    1 1155    6    4    1    0    8    5    1]
 [   0    2   37 1195    0    9    1    7    5    6]
 [   2    2   16    0 1134    2    4    4    0   11]
 [   4    2    9   18    1 1067   11    2   10    2]
 [   4    0   16    0    3    7 1166    0    4    0]
 [   1    2   18    2    6    1    0 1217    0   17]
 [   5    3   10   12    6    7    1    5 1074    9]
 [   4    2   11    3   16    2    0   13    3 1099]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.98      1140
         1.0       0.99      0.98      0.99      1363
         2.0       0.90      0.97      0.93      1185
         3.0       0.97      0.95      0.96      1262
         4.0       0.97      0.97      0.97      1175
         5.0       0.97      0.95      0.96      1126
         6.0       0.98      0.97      0.98      1200
         7.0       0.97      0.96      0.97      1264
         8.0       0.97      0.95      0.96      1132
         9.0       0.96      0.95      0.96      1153
```

```
    accuracy                           0.96     12000
   macro avg       0.96      0.96      0.96     12000
weighted avg       0.96      0.96      0.96     12000

Training Time: 1656.18 seconds

--- Model Performance Summary ---
                              Task        C     error         time  gamma
0          Binary (3 vs 6, Linear)     0.01  0.006093     1.815479    NaN
1             Binary (3 vs 6, RBF)  1000.00  0.004874     6.412924  0.001
2  Multi-Class (1, 2, 5, 8, Linear)     0.10  0.045984    15.833344    NaN
3               Full 10-Class (RBF)    10.00  0.035667  1656.184587  0.010
```

In [ ]: