HW2

March 7, 2025

[7]:
```python
from ucimlrepo import fetch_ucirepo
import warnings
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.kernel_ridge import KernelRidge
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error

warnings.filterwarnings("ignore")

communities_and_crime = fetch_ucirepo(id=183)
X = communities_and_crime.data.features
y = communities_and_crime.data.targets.iloc[:, 0]

X.replace("?", np.nan, inplace=True)
X.dropna(axis=1, inplace=True)
y.dropna(inplace=True)
X = X.loc[y.index]

drop_cols = ['state', 'county', 'community', 'communityname', 'fold']
X = X.drop(columns=drop_cols, errors='ignore')

X = X.apply(pd.to_numeric, errors='coerce')
y = y.apply(pd.to_numeric, errors='coerce')

scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
    random_state=99)
```

1 Q2

2 a

[14]:
```
from sklearn.model_selection import KFold

def k_fold_cv(X, y, n_splits=5):
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=99)

    alphas = np.logspace(-4, 2, 10)
    degrees = [2, 3, 4, 5]
    gammas = np.logspace(-2, 2, 8)
    best_score = float('inf')
    best_params = {}

    for alpha in alphas:
        for degree in degrees:
            # Polynomial Kernel
            poly_errors = []
            for train_idx, val_idx in kf.split(X):
                X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
                y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

                model = KernelRidge(alpha=alpha, kernel='poly', degree=degree)
                model.fit(X_train, y_train)
                y_pred = model.predict(X_val)
                poly_errors.append(mean_squared_error(y_val, y_pred))

            avg_poly_error = np.mean(poly_errors)
            if avg_poly_error < best_score:
                best_score = avg_poly_error
                best_params = {'kernel': 'poly', 'alpha': alpha, 'degree':␣
↪degree}

        for gamma in gammas:
            # RBF Kernel
            rbf_errors = []
            for train_idx, val_idx in kf.split(X):
                X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
                y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

                model = KernelRidge(alpha=alpha, kernel='rbf', gamma=gamma)
                model.fit(X_train, y_train)
                y_pred = model.predict(X_val)
                rbf_errors.append(mean_squared_error(y_val, y_pred))

            avg_rbf_error = np.mean(rbf_errors)
```

```
            if avg_rbf_error < best_score:
                best_score = avg_rbf_error
                best_params = {'kernel': 'rbf', 'alpha': alpha, 'gamma': gamma}

    print("Best Parameters:", best_params)
    print("Best MSE:", best_score)
    return best_params, best_score

best_params, best_score = k_fold_cv(X_train, y_train)
```

```
Best Parameters: {'kernel': 'poly', 'alpha': 4.641588833612772, 'degree': 2}
Best MSE: 0.01833117790926982
```

[13]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'alpha': np.logspace(-4, 2, 10),
    'kernel': ['poly', 'rbf'],
    'degree': [2, 3, 4, 5]
}

model = KernelRidge()
grid_search = GridSearchCV(model, param_grid, cv=5,␣
 ↪scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

y_pred = grid_search.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Final MSE with Best Parameters: {mse}")
```

```
Best Parameters: {'alpha': 4.641588833612772, 'degree': 2, 'kernel': 'poly'}
Final MSE with Best Parameters: 0.016660790628545773
```

3 Q3

4 a)

[36]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

mnist = fetch_openml("mnist_784", version=1, parser='auto')
```

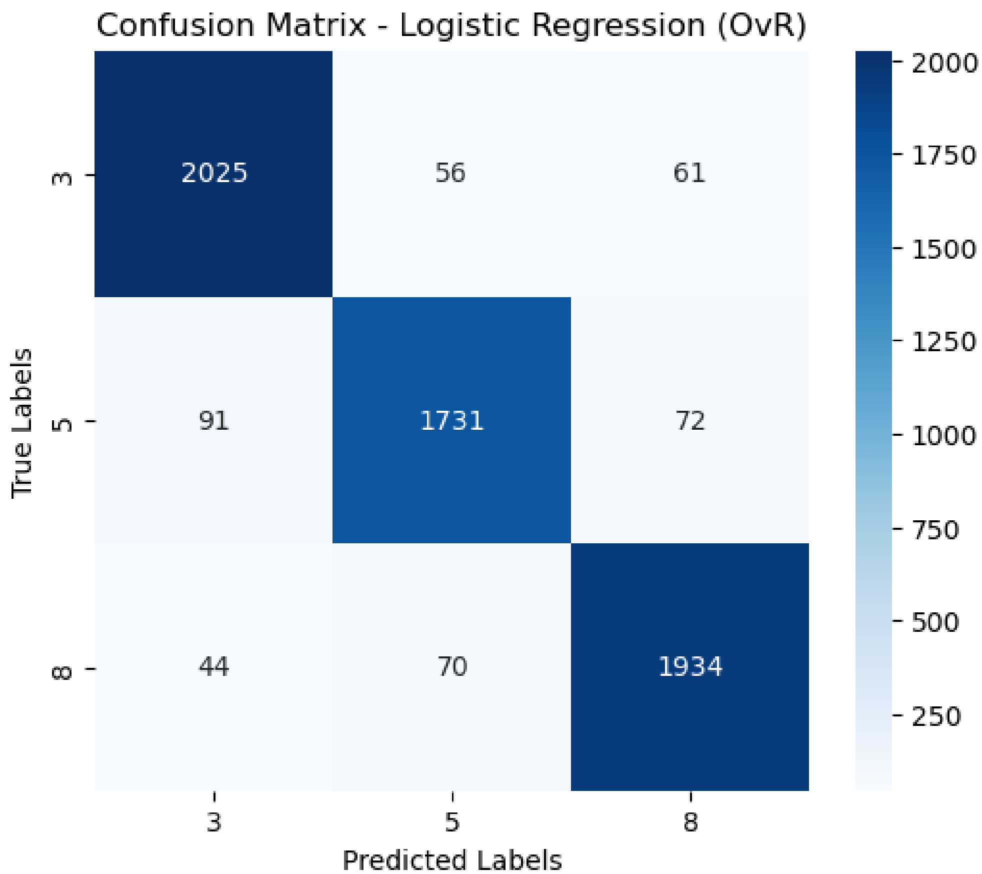
```
X = mnist['data']
y = mnist['target'].astype(int)

selected_digits = [3, 5, 8]
mask = y.isin(selected_digits)
X_filtered = X[mask]
y_filtered = y[mask]

X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered,
                                                    test_size=0.3,
                                                    random_state=99,
                                                    stratify=y_filtered)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

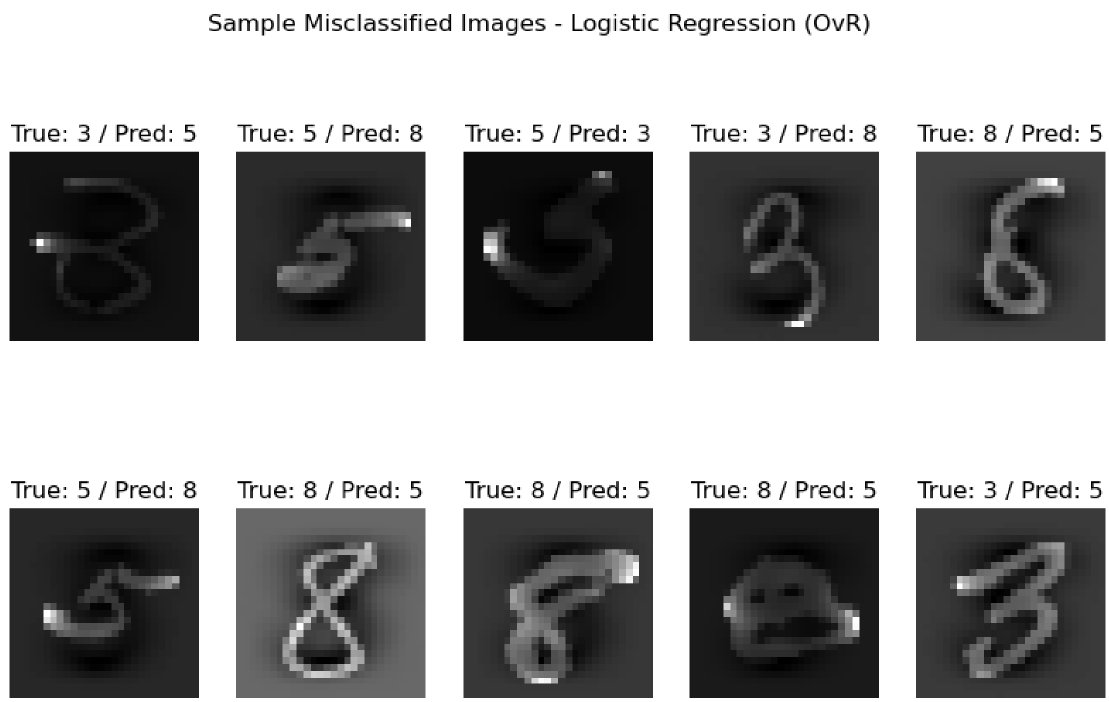
5 Logistic Regression: One-vs-Rest

[24]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

log_reg = LogisticRegression(multi_class='ovr', max_iter=1000, random_state=99)
log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Logistic Regression (OvR) Accuracy: {accuracy:.4f}")

plt.figure(figsize=(6, 5))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[3, 5, 8],␣
 ↪yticklabels=[3, 5, 8])
plt.title("Confusion Matrix - Logistic Regression (OvR)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

misclassified_idx = np.where(y_test != y_pred)[0]
plt.figure(figsize=(10, 6))
for i, idx in enumerate(misclassified_idx[:10]):
```

```
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
    plt.title(f"True: {y_test.iloc[idx]} / Pred: {y_pred[idx]}")
    plt.axis('off')

plt.suptitle("Sample Misclassified Images - Logistic Regression (OvR)")
plt.show()
```

```
Logistic Regression (OvR) Accuracy: 0.9352
```



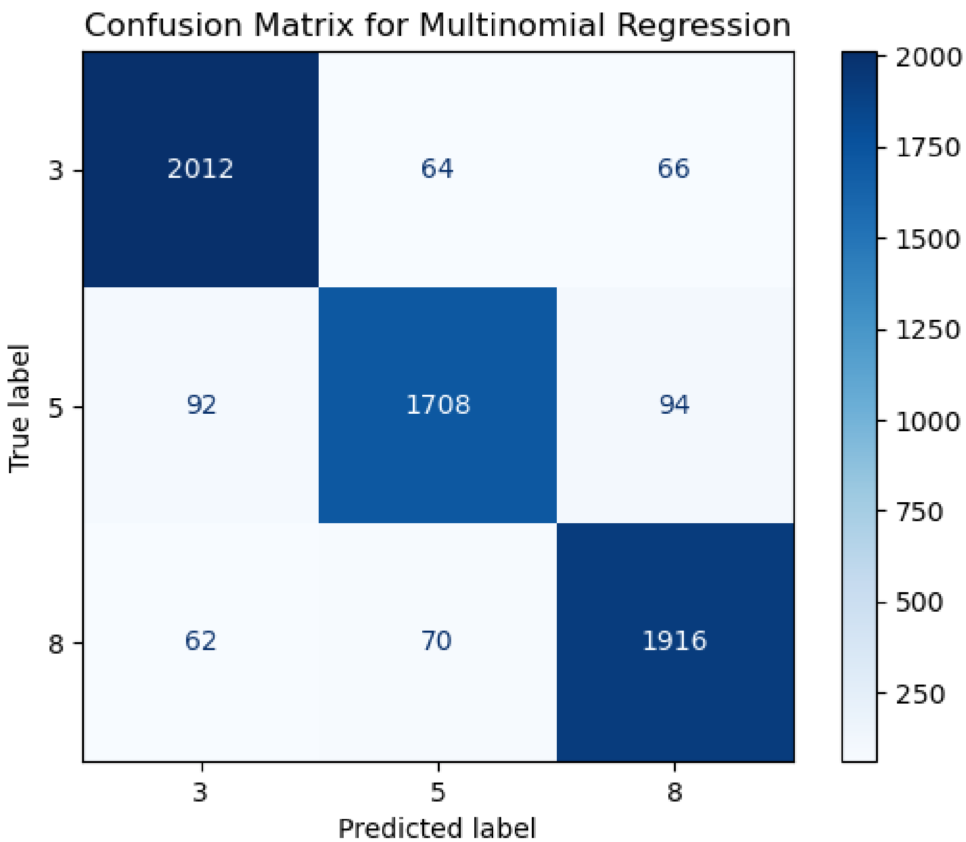


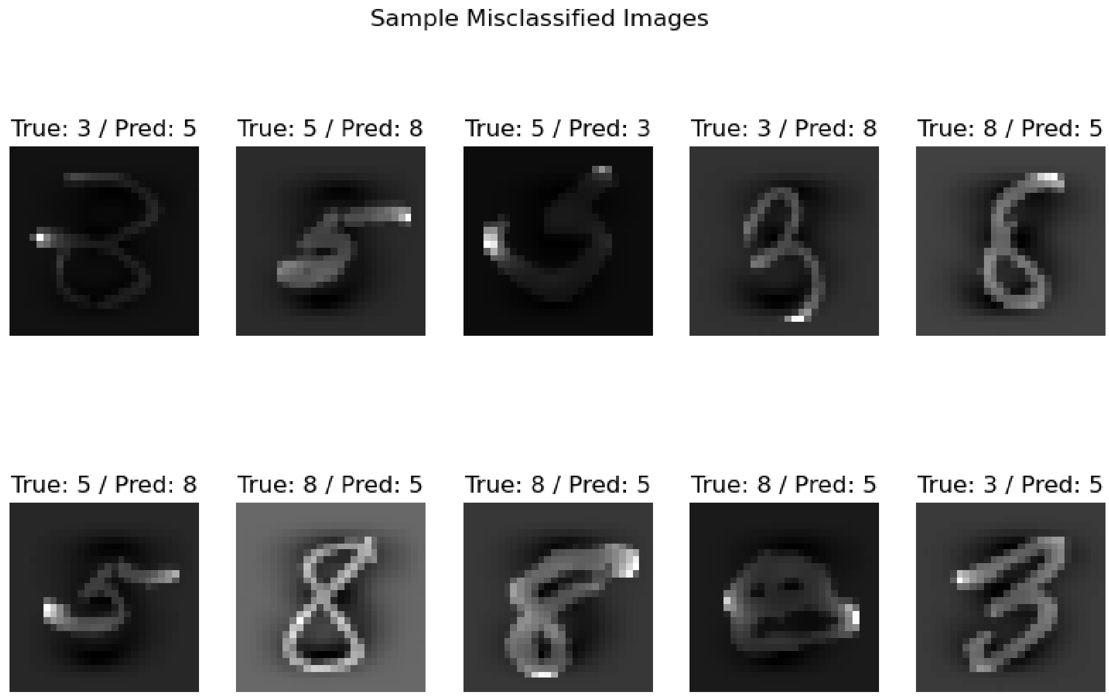
6 Multinomial Regression

[25]:
```
model = LogisticRegression(multi_class='multinomial', solver='lbfgs',␣
    ↪max_iter=1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Multinomial Regression Accuracy: {accuracy:.4f}")

conf_matrix = confusion_matrix(y_test, y_pred, labels=selected_digits)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=selected_digits)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix for Multinomial Regression')
plt.show()

misclassified_idx = np.where(y_test != y_pred)[0]
plt.figure(figsize=(10, 6))
for i, idx in enumerate(misclassified_idx[:10]):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
    plt.title(f"True: {y_test.iloc[idx]} / Pred: {y_pred[idx]}")
```

```
    plt.axis('off')

plt.suptitle("Sample Misclassified Images")
plt.show()
```

```
Multinomial Regression Accuracy: 0.9264
```



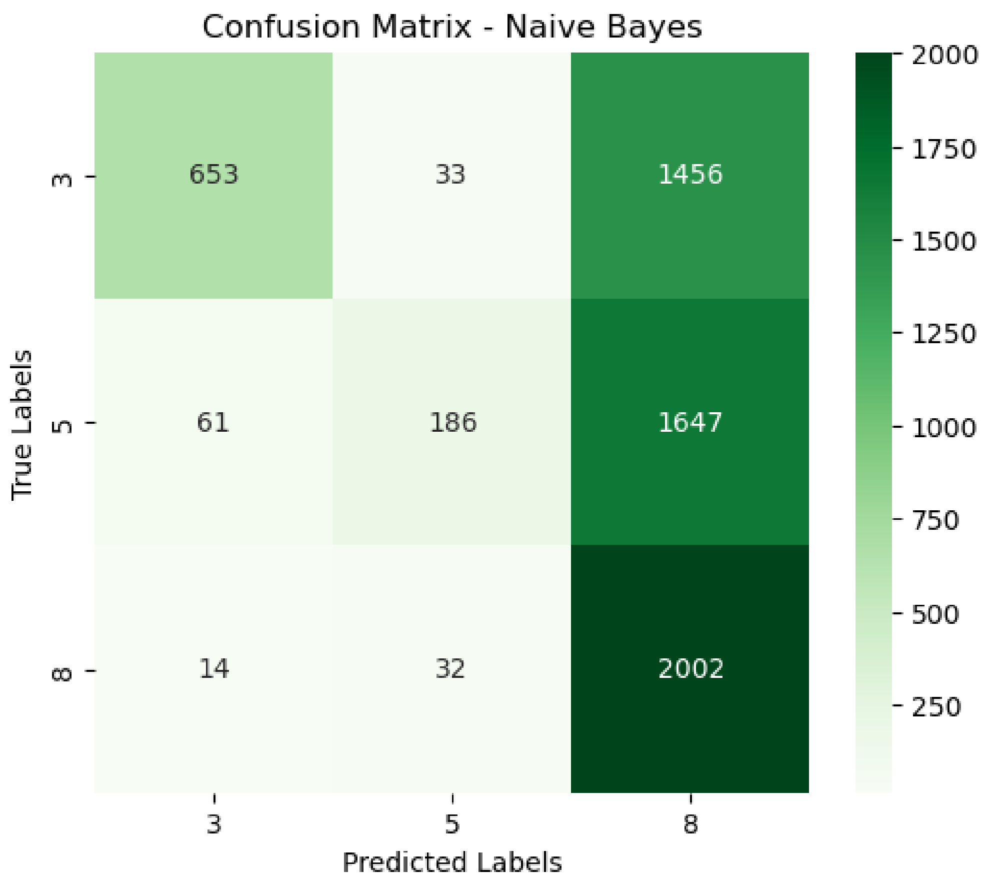


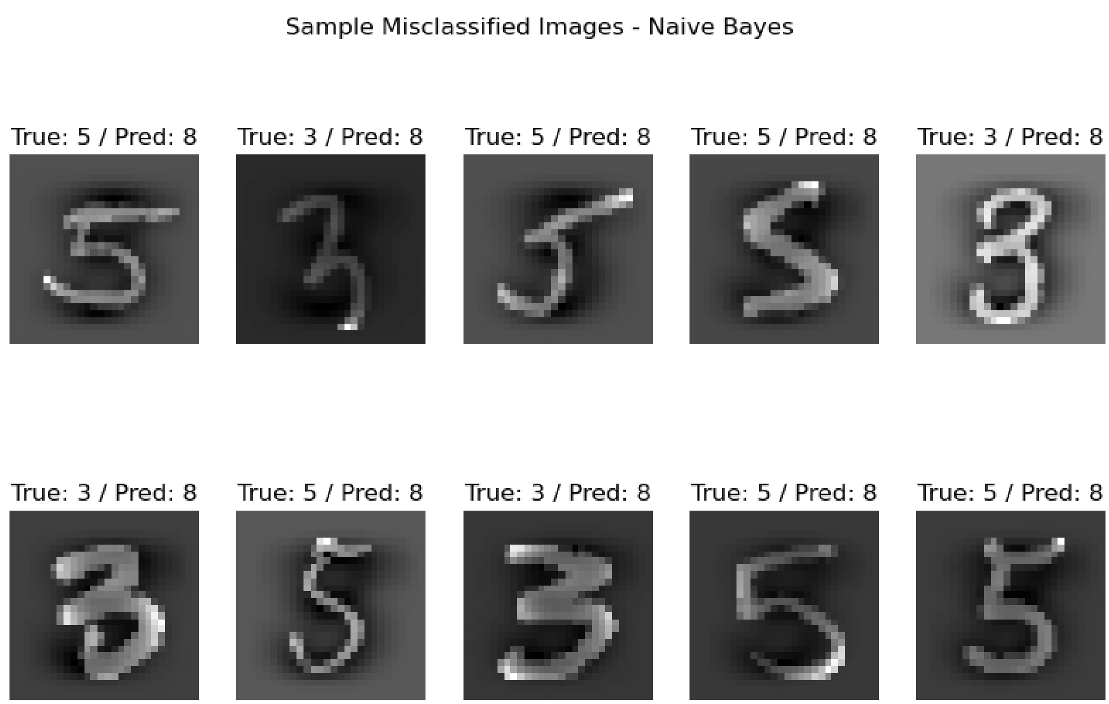
7 Naive Bayes

[21]:
```
from sklearn.naive_bayes import GaussianNB

nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

y_pred = nb_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Naive Bayes Accuracy: {accuracy:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(6, 5))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=[3, 5, 8],␣
↪yticklabels=[3, 5, 8])
plt.title("Confusion Matrix - Naive Bayes")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```python
misclassified_idx = np.where(y_test != y_pred)[0]
plt.figure(figsize=(10, 6))
for i, idx in enumerate(misclassified_idx[:10]):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
    plt.title(f"True: {y_test.iloc[idx]} / Pred: {y_pred[idx]}")
    plt.axis('off')

plt.suptitle("Sample Misclassified Images - Naive Bayes")
plt.show()
```

```
Naive Bayes Accuracy: 0.4670

Classification Report:
              precision    recall  f1-score   support

           3       0.90      0.30      0.46      2142
           5       0.74      0.10      0.17      1894
           8       0.39      0.98      0.56      2048

    accuracy                           0.47      6084
   macro avg       0.68      0.46      0.40      6084
weighted avg       0.68      0.47      0.40      6084
```


Confusion Matrix - Naive Bayes
True Labels
Predicted Labels
3
5
8
3
5
8
653
33
1456
61
186
1647
14
32
2002
2000
1750
1500
1250
1000
750
500
250

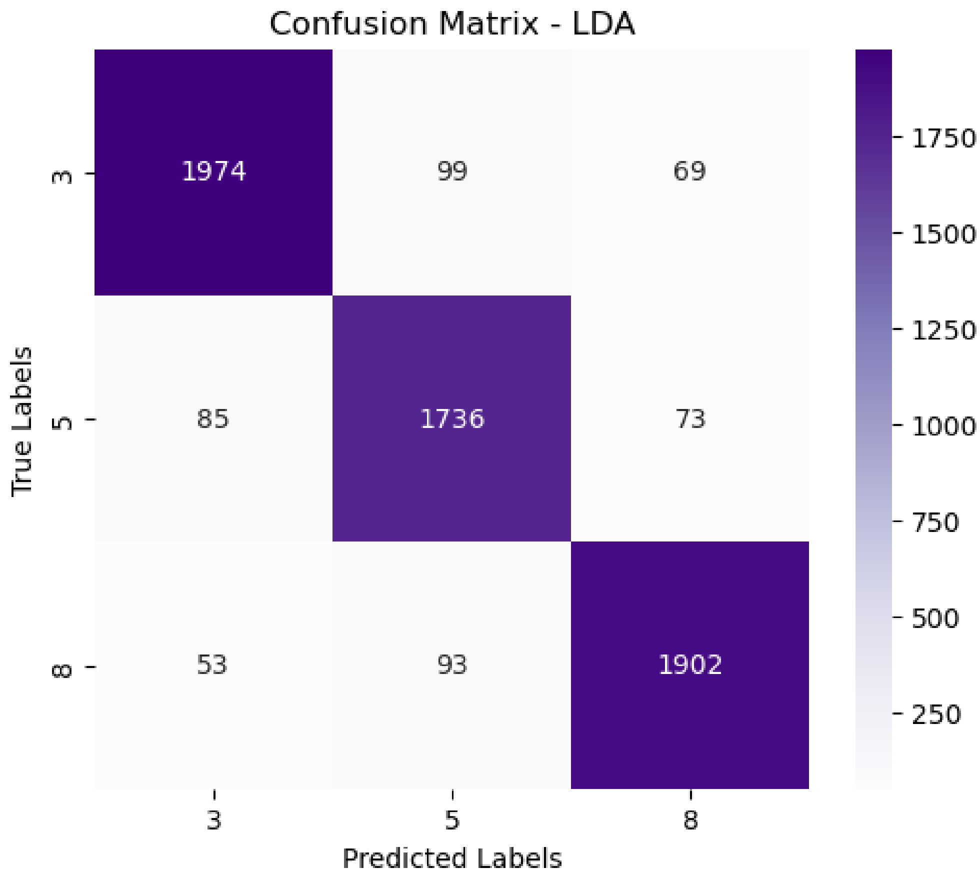

Sample Misclassified Images - Naive Bayes
True: 5 / Pred: 8
True: 3 / Pred: 8
True: 5 / Pred: 8
True: 5 / Pred: 8
True: 3 / Pred: 8
True: 3 / Pred: 8
True: 5 / Pred: 8
True: 3 / Pred: 8
True: 5 / Pred: 8
True: 5 / Pred: 8

8 Linear Discriminant Analysis

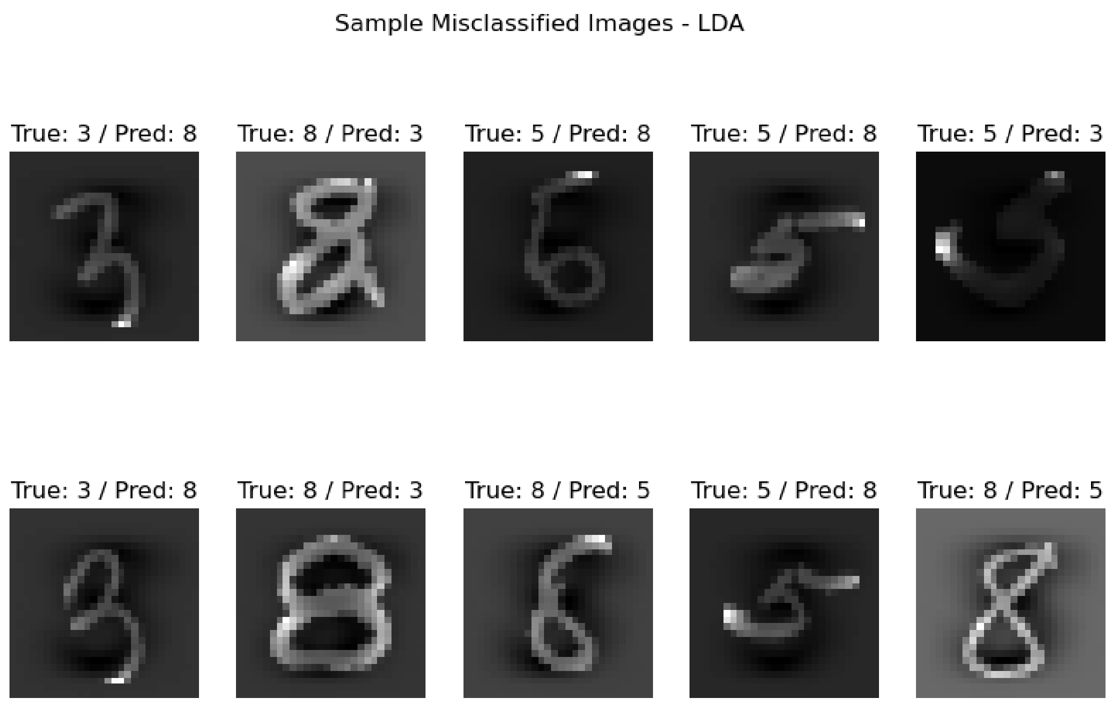
[22]:
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda_model = LinearDiscriminantAnalysis()
lda_model.fit(X_train, y_train)

y_pred = lda_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Linear Discriminant Analysis (LDA) Accuracy: {accuracy:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(6, 5))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=[3, 5, 8],␣
 ↪yticklabels=[3, 5, 8])
plt.title("Confusion Matrix - LDA")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

misclassified_idx = np.where(y_test != y_pred)[0]
plt.figure(figsize=(10, 6))
for i, idx in enumerate(misclassified_idx[:10]):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
    plt.title(f"True: {y_test.iloc[idx]} / Pred: {y_pred[idx]}")
    plt.axis('off')

plt.suptitle("Sample Misclassified Images - LDA")
plt
```

```
Linear Discriminant Analysis (LDA) Accuracy: 0.9224

Classification Report:
              precision    recall  f1-score   support

           3       0.93      0.92      0.93      2142
           5       0.90      0.92      0.91      1894
           8       0.93      0.93      0.93      2048
```

accuracy			0.92	6084
macro avg	0.92	0.92	0.92	6084
weighted avg	0.92	0.92	0.92	6084



[22]: <module 'matplotlib.pyplot' from 'C:\\Users\\Halvin\\anaconda3\\Lib\\site-packages\\matplotlib\\pyplot.py'>

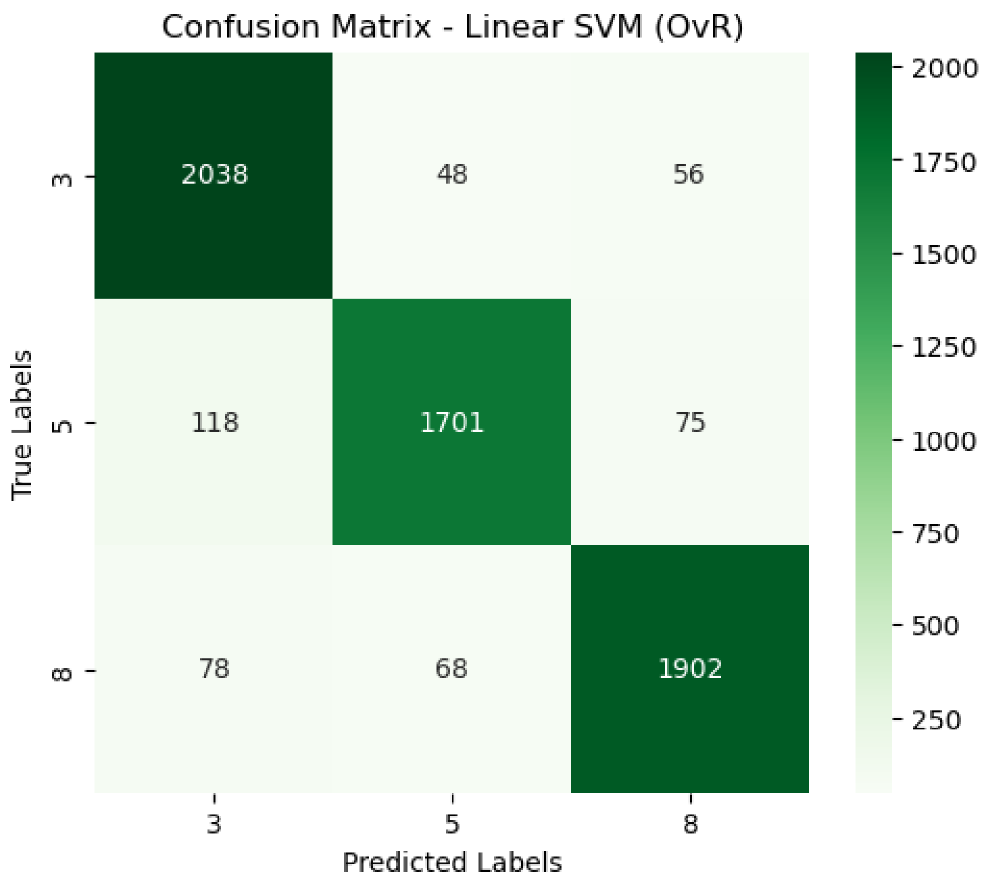


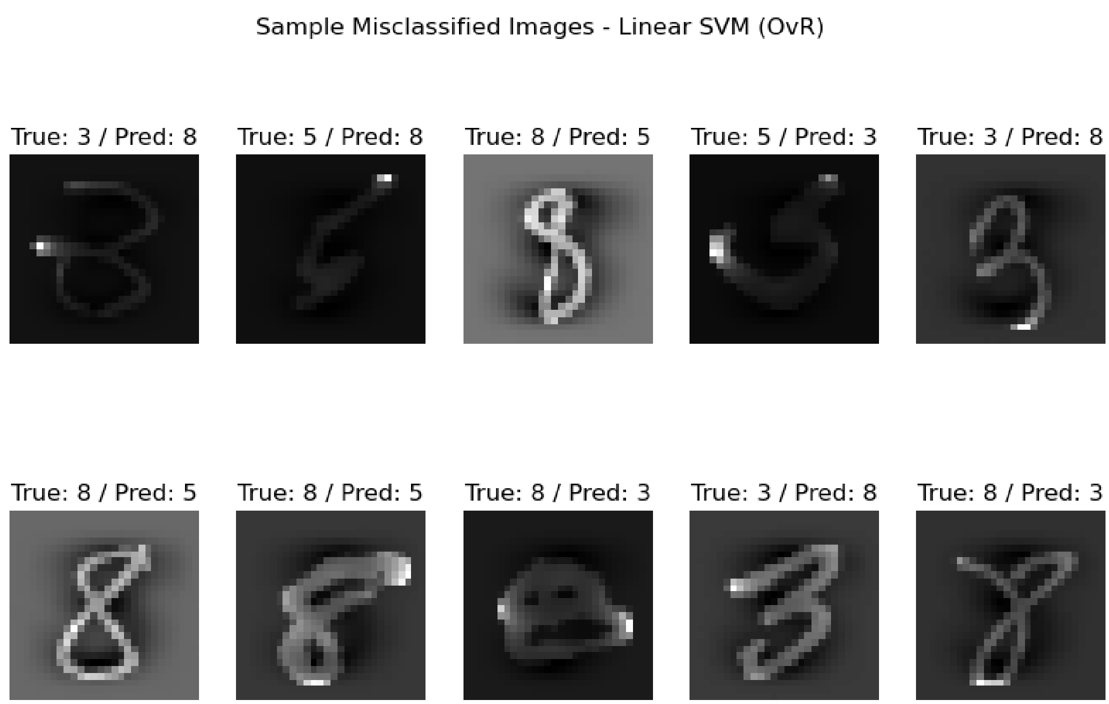
9 Linear SVM: One-vs-Rest

[23]:
```python
from sklearn.svm import SVC

svm_model = SVC(kernel='linear', decision_function_shape='ovr', random_state=99)
svm_model.fit(X_train, y_train)

y_pred = svm_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Linear SVM (OvR) Accuracy: {accuracy:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(6, 5))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=[3, 5, 8],␣
↪yticklabels=[3, 5, 8])
plt.title("Confusion Matrix - Linear SVM (OvR)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```python
misclassified_idx = np.where(y_test != y_pred)[0]
plt.figure(figsize=(10, 6))
for i, idx in enumerate(misclassified_idx[:10]):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
    plt.title(f"True: {y_test.iloc[idx]} / Pred: {y_pred[idx]}")
    plt.axis('off')

plt.suptitle("Sample Misclassified Images - Linear SVM (OvR)")
plt.show()
```

```
Linear SVM (OvR) Accuracy: 0.9272

Classification Report:
              precision    recall  f1-score   support

           3       0.91      0.95      0.93      2142
           5       0.94      0.90      0.92      1894
           8       0.94      0.93      0.93      2048

    accuracy                           0.93      6084
   macro avg       0.93      0.93      0.93      6084
weighted avg       0.93      0.93      0.93      6084
```


Confusion Matrix - Linear SVM (OvR)
True Labels
Predicted Labels
3
5
8
2038
48
56
118
1701
75
78
68
1902
2000
1750
1500
1250
1000
750
500
250

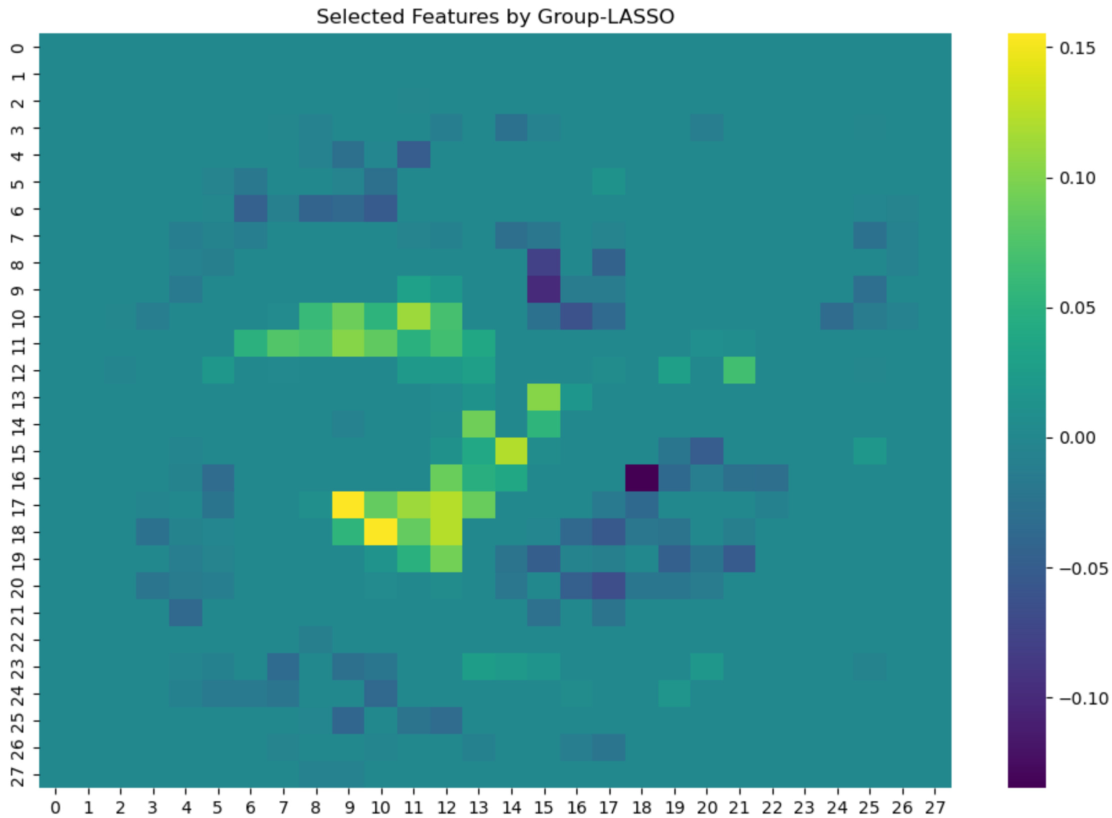

Sample Misclassified Images - Linear SVM (OvR)
True: 3 / Pred: 8
True: 5 / Pred: 8
True: 8 / Pred: 5
True: 5 / Pred: 3
True: 3 / Pred: 8
True: 8 / Pred: 5
True: 8 / Pred: 5
True: 8 / Pred: 3
True: 3 / Pred: 8
True: 8 / Pred: 3

10 Group-lasso Regularization

[35]:
```python
from group_lasso import GroupLasso

group_lasso = GroupLasso(
    groups=np.arange(X_train.shape[1]),
    group_reg=0.01,
    l1_reg=0.001,
    scale_reg="group_size",
    n_iter=1000
)

group_lasso.fit(X_train, y_train)

selected_features = np.where(group_lasso.coef_ != 0)[0]
X_train_selected = X_train[:, selected_features]
X_test_selected = X_test[:, selected_features]

log_reg = LogisticRegression(multi_class='multinomial', max_iter=1000)
log_reg.fit(X_train_selected, y_train)

y_pred = log_reg.predict(X_test_selected)

accuracy = accuracy_score(y_test, y_pred)
print(f"Group-LASSO Logistic Regression Accuracy: {accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(12, 8))
sns.heatmap(group_lasso.coef_.reshape(28, 28), cmap='viridis')
plt.title("Selected Features by Group-LASSO")
plt.show()
```

```
Group-LASSO Logistic Regression Accuracy: 0.9206

Classification Report:
              precision    recall  f1-score   support

           3       0.93      0.93      0.93      2142
           5       0.90      0.90      0.90      1894
           8       0.93      0.93      0.93      2048

    accuracy                           0.92      6084
   macro avg       0.92      0.92      0.92      6084
```

```
weighted avg       0.92      0.92      0.92      6084
```




```
[ ]:
```