Homework 3 Part 2

This is an individual assignment.

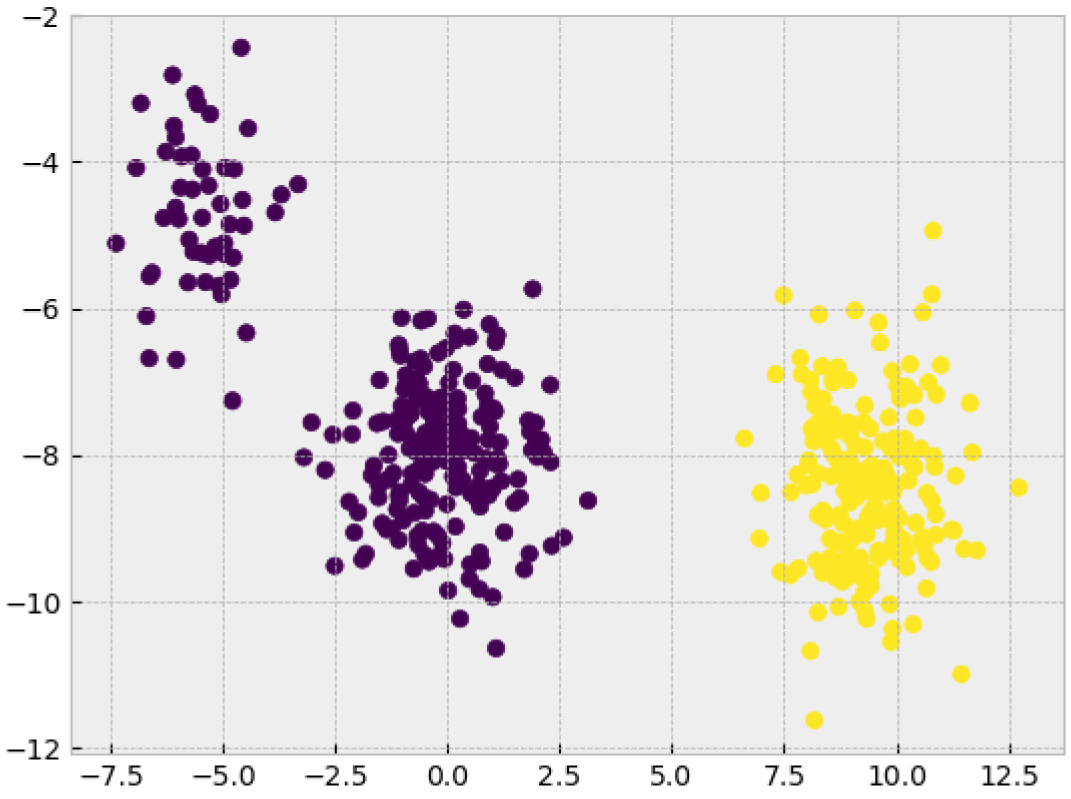
Write your own code and justify all your answers. You may repurpose any functions built during lecture. You may use `scikit-learn` functions.

In [2]:
```
# Import libraries and magics

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')

from sklearn.model_selection import train_test_split
```

Exercise 1 (10 points)

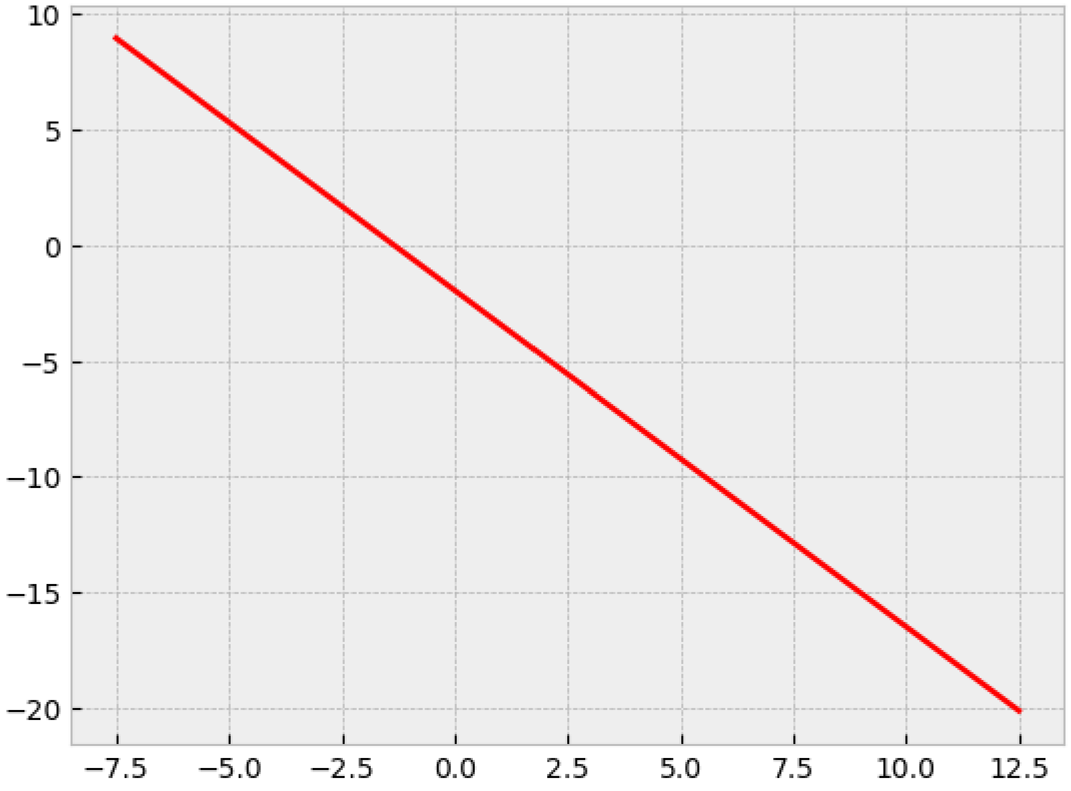
Consider the dataset we worked with in class:

In [4]:
```
X = np.load('data.npy')
t = np.load('labels.npy')

plt.scatter(X[:,0], X[:,1], c=t);
```



1. (8 points) **Solve for the parameters ($\mathbf{w}$ and w_0) of the discriminant function you solved for in HW3-Part1-Q1. Plot the corresponding discriminant function.**

```
In [6]: mean_0 = np.mean(X[t == 0], axis=0)
        mean_1 = np.mean(X[t == 1], axis=0)

        cov = np.cov(X.T)

        cov_inv = np.linalg.inv(cov)

        w = cov_inv.dot(mean_1 - mean_0)
        w0 = -0.5 * (mean_1.dot(cov_inv).dot(mean_1) - mean_0.dot(cov_inv).dot(mean_0))

        x = np.linspace(-7.5, 12.5, 100)
        y = (-w[0] / w[1]) * x - w0 / w[1]

        plt.plot(x, y, 'r')
        plt.show()
```



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

2. (2 points) **Compare your solution with the standard LDA discriminant function.**

They are very similar. Both solutions involve computing the mean of each class, the covariance matrix of the data, and the inverse of the covariance matrix. However, the LDA discriminant function also involves computing the between-class scatter matrix, which is not required in my solution..

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Exercise 2 (35 points)

In this problem you will be working with the [handwritten digits (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html)](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html) from `scikit-learn` . The dataset contains 1797 samples. Each sample is a 64-dimensional vector representing all pixels of a 8×8 grayscale image of a handwritten digit. There are a total of 10 digits (10 targets) and about 180 images per digit. Let's load the data:

In [7]:

```
from sklearn.datasets import load_digits

digits = load_digits(return_X_y=False)

# print(digits.DESCR)
```

In [8]:

```
X = digits.data # training data
t = digits.target # target values

X.shape, t.shape
```

Out[8]:

```
((1797, 64), (1797,))
```

```
In [5]: plt.figure(figsize=(5,3))
        grid=1
        for j in range(10):
            loc = np.where(t==j)[0]
            idx_rd = np.random.choice(loc,15,replace=False)
            for i in range(15):
                plt.subplot(10,15,grid)
                plt.imshow(X[idx_rd[i],:].reshape(8,8), cmap='gray')
                plt.axis('off')
                grid+=1
```

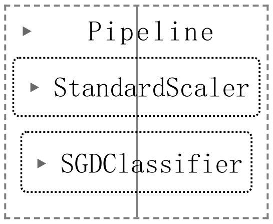
Consider the following training-test split and use it for the next set of questions:

```
In [9]: X_train, X_test, t_train, t_test = train_test_split(X, t,
                                                            stratify=t,
                                                            test_size=0.2,
                                                            random_state=0)
```

1. (2.5 points) **Create a** `Pipeline` **object to train a Logistic Regression classifier (** `SGDClassifier` **). Make sure to use the proper objective function (parameter** `loss` **) and include a scaling step in the pipeline.**

```
In [11]: from sklearn.pipeline import Pipeline
         from sklearn.preprocessing import StandardScaler
         from sklearn.linear_model import SGDClassifier

         pipeline = Pipeline([
             ('scaler', StandardScaler()),
             ('classifier', SGDClassifier(loss='log_loss'))
         ])

         pipeline.fit(X_train, t_train)
```

Out[11]:



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

2. (5 points) **Carry experimental design to tune 2 hyperparameters, the learning rate scheduler and the Ridge regularizer parameter. In your grid search object, consider a 10-fold CV scheme.**

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

         pipeline = Pipeline([
             ('scaler', StandardScaler()),
             ('classifier', SGDClassifier(loss='log_loss', eta0=0.1))
         ])

         param_grid = {
             'classifier__alpha': [0.001, 0.01, 0.1, 1, 10],
             'classifier__learning_rate': ['constant', 'optimal', 'invscaling', 'adaptive']
         }

         grid_search = GridSearchCV(pipeline, param_grid, cv=10)

         grid_search.fit(X_train, t_train)

         print('Best hyperparameters:', grid_search.best_params_)
```

```
Best hyperparameters: {'classifier__alpha': 0.001, 'classifier__learning_rate': 'ada
ptive'}
```

In [ ]:

In [ ]:

In [ ]:

3. (5 points) **Report performance in test set including accuracy, f1-score and confusion matrices. Compare performances between all classifiers.**

In [16]:
```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

t_pred = grid_search.predict(X_test)

accuracy = accuracy_score(t_test, t_pred)
f1 = f1_score(t_test, t_pred, average='weighted')
confusion = confusion_matrix(t_test, t_pred)

print('Accuracy:', accuracy)
print('F1-score:', f1)
print('Confusion matrix:\n', confusion)
```

```
Accuracy: 0.9611111111111111
F1-score: 0.9613175138381873
Confusion matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 35  0  0  0  0  0  0  1  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 35  0  0  0  0  2  0]
 [ 0  2  0  0 33  0  0  0  1  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  1  0  0  0  0 35  0  0  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  4  0  0  0  0  1  0 29  1]
 [ 0  0  0  0  0  1  0  0  0 35]]
```

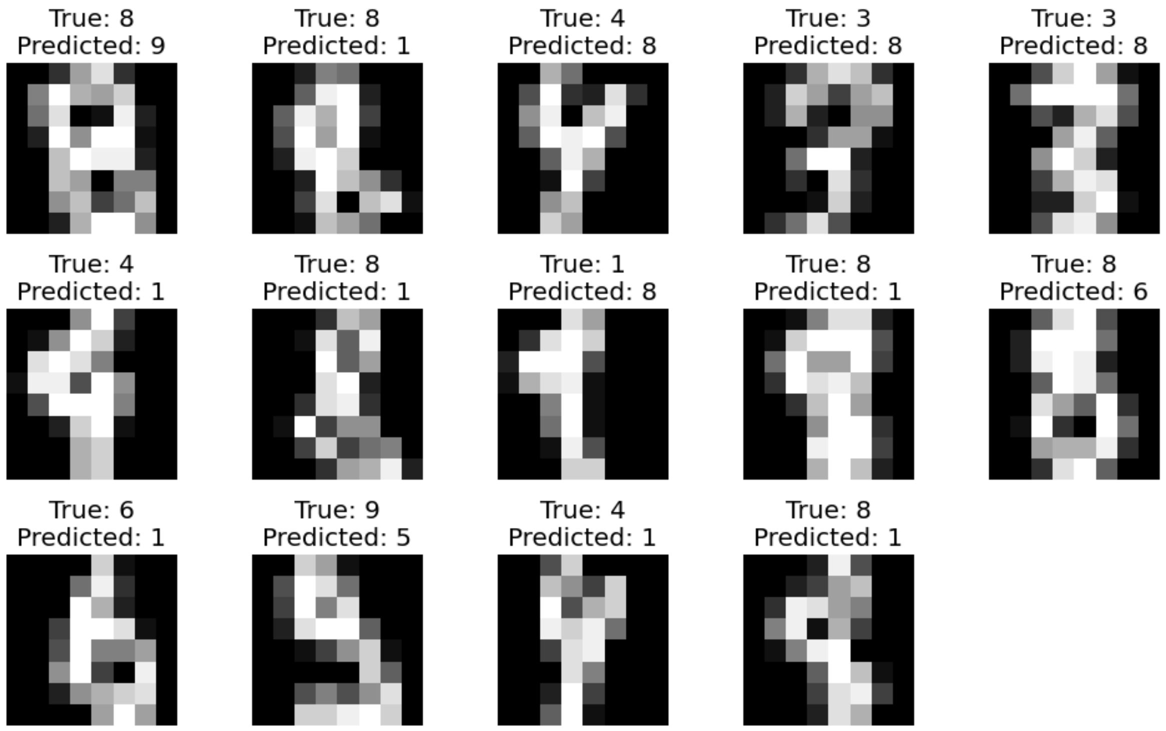
In [ ]:

In [ ]:

In [ ]:

4. (5 points) **Visualize images that were misclassified in test. Provide a discussion on your observations and what procedures could you take to improve results on the bets performing classifier.**

In [19]:
```python
t_pred = grid_search.predict(X_test)

misclassified = np.where(t_pred != t_test)[0]

plt.figure(figsize=(10, 10))
for i, index in enumerate(misclassified[:25]):
    plt.subplot(5, 5, i + 1)
    plt.imshow(X_test[index].reshape(8, 8), cmap='gray')
    plt.title('True: %s\nPredicted: %s' % (t_test[index], t_pred[index]))
    plt.axis('off')
plt.tight_layout()
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

5. (2.5 points) **Create a** `Pipeline` **object to train a Support Vector Machine classifier (** `SVC` **). Make sure to include a scaling step in the pipeline.**

In [20]:
```python
from sklearn.svm import SVC

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', SVC())
])

pipeline.fit(X_train, t_train)
```

Out[20]:
```
▸ Pipeline
▾ StandardScaler
StandardScaler()
▾ SVC
SVC()
```

In [ ]:

In [ ]:

In [ ]:

6. (5 points) **Carry experimental design to tune 2 hyperparameters, the kernel function and the regularization parameter. In your grid search object, consider a 10-fold CV scheme.**

In [21]:
```python
param_grid = {
    'classifier__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'classifier__C': [0.1, 1, 10, 100]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=10)

grid_search.fit(X_train, t_train)

print('Best hyperparameters:', grid_search.best_params_)
```

```
Best hyperparameters: {'classifier__C': 10, 'classifier__kernel': 'poly'}
```

In [ ]:

In [ ]:

In [ ]:

7. (5 points) **Report performance in test set including accuracy, f1-score and confusion matrices. Compare performances between all classifiers.**

In [22]:

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

t_pred = grid_search.predict(X_test)

accuracy = accuracy_score(t_test, t_pred)
f1 = f1_score(t_test, t_pred, average='weighted')
confusion = confusion_matrix(t_test, t_pred)

print('Accuracy:', accuracy)
print('F1-score:', f1)
print('Confusion matrix:\n', confusion)
```

```
Accuracy: 0.9944444444444445
F1-score: 0.9944830172477448
Confusion matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 36  0  0  0  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 36  0  0  0  0  1  0]
 [ 0  0  0  0 36  0  0  0  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 36  0  0  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  0  0  0  0  0  0  0 35  0]
 [ 0  0  0  0  0  0  0  0  1 35]]
```

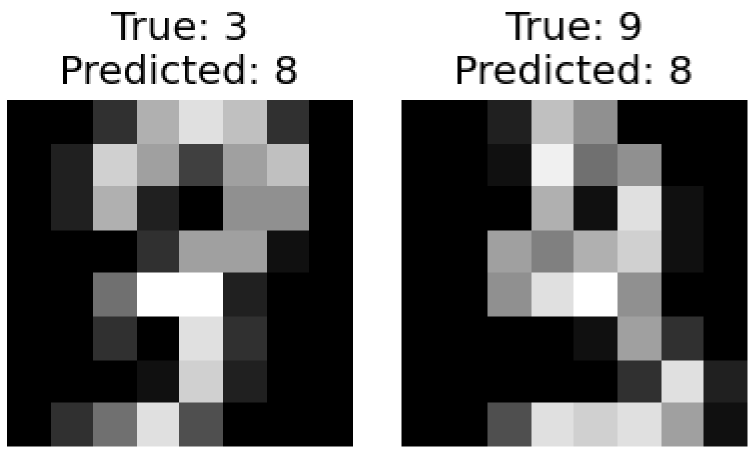
In [ ]:

In [ ]:

In [ ]:

8. (5 points) **Visualize images that were misclassified in test. Provide a discussion on your observations and what procedures could you take to improve results on the bets performing classifier.**

In [23]:
```
t_pred = grid_search.predict(X_test)

misclassified = np.where(t_pred != t_test)[0]

plt.figure(figsize=(10, 10))
for i, index in enumerate(misclassified[:25]):
    plt.subplot(5, 5, i + 1)
    plt.imshow(X_test[index].reshape(8, 8), cmap='gray')
    plt.title('True: %s\nPredicted: %s' % (t_test[index], t_pred[index]))
    plt.axis('off')
plt.tight_layout()
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

On-Time (5 points)

Submit your assignment before the deadline.

Submit Your Solution

Confirm that you've successfully completed the assignment.

Along with the Notebook, include a PDF of the notebook with your solutions.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.
