Homework 2 Part 2

This is an individual assignment.

Write your own code. You may repurpose any functions built during lecture.

In [7]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

Problem 1 (20 points)

In this problem you will be working with the handwritten digits 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 [8]:
```
from sklearn.datasets import load_digits

digits = load_digits(return_X_y=False)

print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

In [5]:

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

X.shape, t.shape
```

Out[5]:

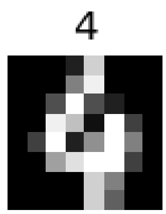
```
((1797, 64), (1797,))
```

Each image can be reshaped as a 8×8 grayscale image and plotted:

In [5]:

```
img_no = 100
```

```python
plt.figure(figsize=(1,1))
plt.imshow(X[img_no,:].reshape(8,8), cmap='gray')
plt.title(t[img_no]) # insert title with the label
plt.axis('off');
```



Here's some image examples from all 10 classes:

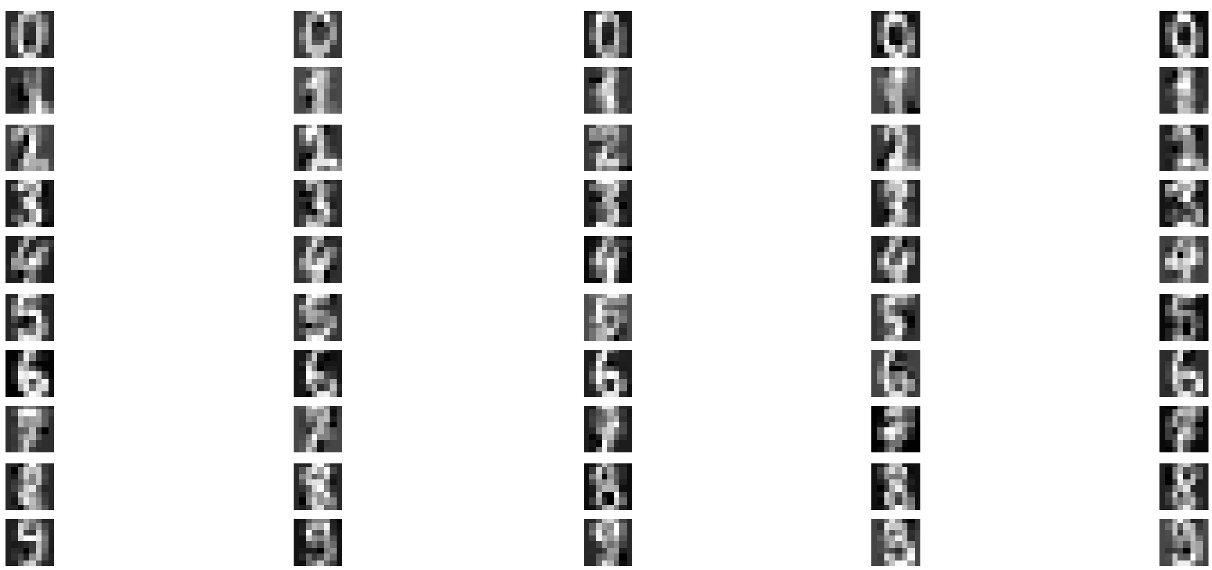
In [6]:
```python
plt.figure(figsize=(5,3))
grid=1
for j in range(10):
    loc = np.where(t==j)[0]
    idx_rd = np.random.choice(loc,15,replace=False)
    for i in range(15):
        plt.subplot(10,15,grid)
        plt.imshow(X[idx_rd[i],:].reshape(8,8), cmap='gray')
        plt.axis('off')
        grid+=1
```

1. (10 points) **Assume that each class is modeled according to a multivariate Gaussian distribution, $P(\mathbf{x}|C_i) \sim G(\mu_i, \Sigma_i)$. Use MLE approach to estimate the parameters of the data likelihoods. Use your estimated density functions to generate new samples (5 samples) and plot them.**

- **Do the new samples *look* as expected?**
- **What can you do to improve the results?**

In [8]:
```python
from scipy.stats import multivariate_normal

class_means = []
class_covariances = []

regularization_term = 1e-5
```

```python
for i in range(10):
    class_samples = X[t == i]
    class_mean = np.mean(class_samples, axis=0)
    class_covariance = np.cov(class_samples, rowvar=False) + regularization_term * np.

    class_means.append(class_mean)
    class_covariances.append(class_covariance)

num_samples = 5
generated_samples = []

for i in range(10):
    distribution = multivariate_normal(mean=class_means[i], cov=class_covariances[i])
    samples = distribution.rvs(num_samples)
    generated_samples.append(samples)

plt.figure(figsize=(10, 4))
grid = 1

for i in range(10):
    for j in range(num_samples):
        plt.subplot(10, num_samples, grid)
        plt.imshow(generated_samples[i][j].reshape(8, 8), cmap='gray')
        plt.axis('off')
        grid += 1

plt.show()
```



Not really, they look fuzzy. In my opinion, we can generate more samples for each class to provide better distribution or increase the regularization term to stabilize the covariance matrix.

In [ ]:

```python

```

In [ ]:

```python

```

1. (10 points) **Repeat the same exercise for the Olivetti faces data:**

In [4]:
```python
from sklearn.datasets import fetch_olivetti_faces

faces = fetch_olivetti_faces(return_X_y=False)

print(faces.DESCR)
```

```
.. _olivetti_faces_dataset:

The Olivetti faces dataset
--------------------------

`This dataset contains a set of face images`_ taken between April 1992 and
April 1994 at AT&T Laboratories Cambridge. The
:func:`sklearn.datasets.fetch_olivetti_faces` function is the data
fetching / caching function that downloads the data
archive from AT&T.

.. _This dataset contains a set of face images: https://cam-orl.co.uk/facedatabase.html

As described on the original website:

    There are ten different images of each of 40 distinct subjects. For some
    subjects, the images were taken at different times, varying the lighting,
    facial expressions (open / closed eyes, smiling / not smiling) and facial
    details (glasses / no glasses). All the images were taken against a dark
    homogeneous background with the subjects in an upright, frontal position
    (with tolerance for some side movement).

**Data Set Characteristics:**

    =================   =====================
    Classes                                40
    Samples total                         400
    Dimensionality                       4096
    Features            real, between 0 and 1
    =================   =====================

The image is quantized to 256 grey levels and stored as unsigned 8-bit
integers; the loader will convert these to floating point values on the
interval [0, 1], which are easier to work with for many algorithms.

The "target" for this database is an integer from 0 to 39 indicating the
identity of the person pictured; however, with only 10 examples per class, this
relatively small dataset is more interesting from an unsupervised or
semi-supervised perspective.

The original dataset consisted of 92 x 112, while the version available here
consists of 64x64 images.

When using these images, please give credit to AT&T Laboratories Cambridge.
```

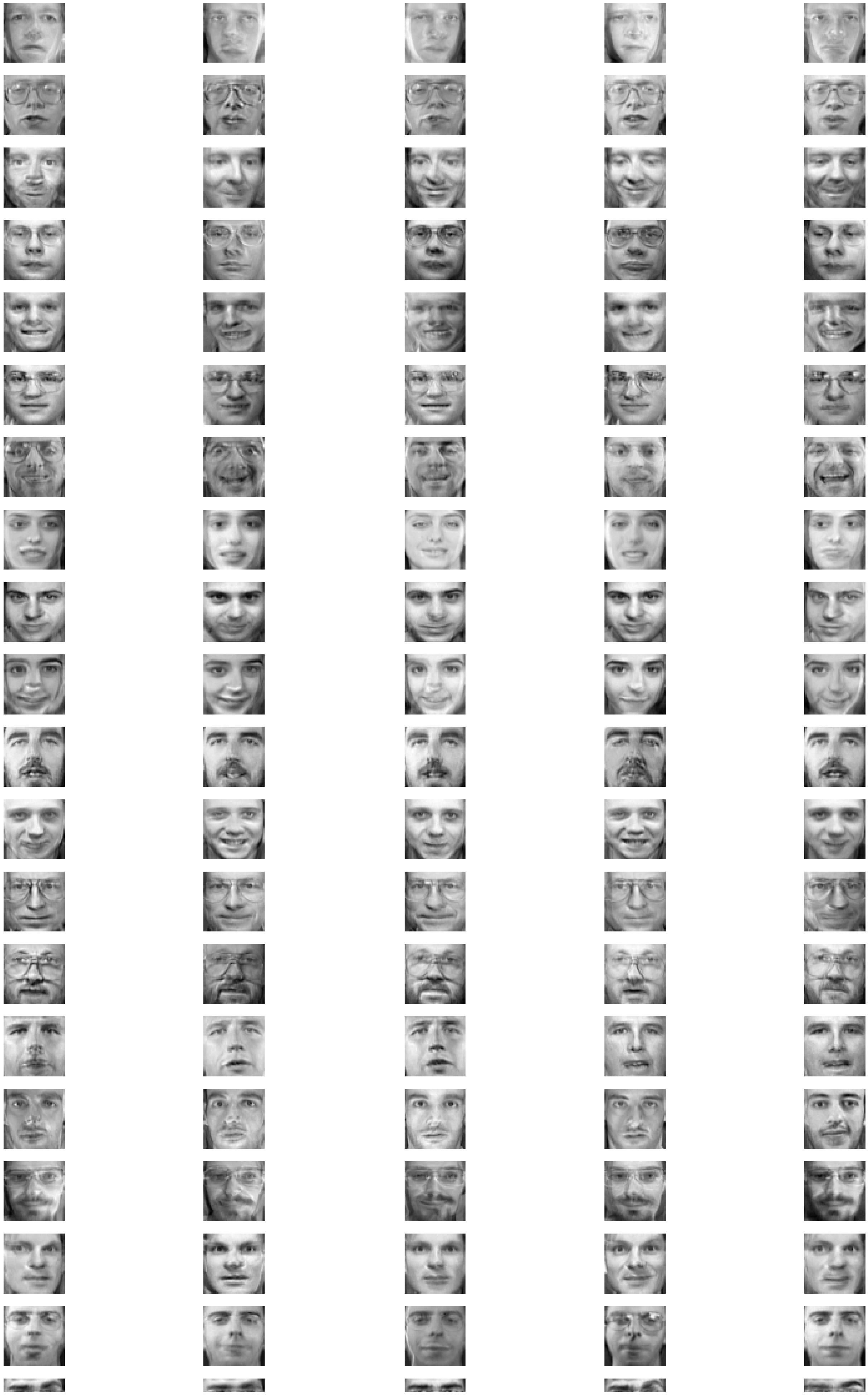
In [5]:
```python
data   = faces.data # training data
target = faces.target # target values

data.shape, target.shape
```

Out[5]:
```
((400, 4096), (400,))
```

In [12]:

```
plt.figure(figsize=(4,25))
grid=1
for j in range(40):
    loc = np.where(target==j)[0]
    idx_rd = np.random.choice(loc,5,replace=False)
    for i in range(5):
        plt.subplot(40,5,grid)
        plt.imshow(data[idx_rd[i],:].reshape(64,64), cmap='gray')
        plt.axis('off')
        grid+=1
```

In [9]:
```
from scipy.stats import multivariate_normal

class_means_faces = []
class_covariances_faces = []

regularization_term_faces = 1e-4

for i in range(40):
    class_samples_faces = data[target == i]
    class_mean_faces = np.mean(class_samples_faces, axis=0)
    class_covariance_faces = np.cov(class_samples_faces, rowvar=False) + regularizatio

    class_means_faces.append(class_mean_faces)
    class_covariances_faces.append(class_covariance_faces)

num_samples_faces = 5
generated_samples_faces = []

for i in range(40):
    distribution_faces = multivariate_normal(mean=class_means_faces[i], cov=class_covar
    samples_faces = distribution_faces.rvs(num_samples_faces)
    generated_samples_faces.append(samples_faces)

plt.figure(figsize=(10, 30))
grid_faces = 1

for i in range(40):
    for j in range(num_samples_faces):
        plt.subplot(40, num_samples_faces, grid_faces)
        plt.imshow(generated_samples_faces[i][j].reshape(64, 64), cmap='gray')
        plt.axis('off')
        grid_faces += 1

plt.show()
```



In [ ]:

In [ ]:

In [ ]:

Problem 2 (25 points)

In this problem, you will be working with a crab dataset. The dataset contains 200 samples. Each sample is a 7-dimensional vector representing crab attributes (front lip width, rear width, length, width, depth, male and female), namely 5 morphological measurements on 50 crabs each of two color forms and both sexes, of the species *Leptograpsus* variegatus collected at Fremantle, W. Australia.

- **Dataset Source: Campbell, N.A. and Mahon, R.J. (1974) A multivariate study of variation in two species of rock crab of genus *Leptograpsus*. *Australian Journal of Zoology* 22, 417–425.**

Let's load the data:

In [22]:
```
data = pd.read_csv("crab.txt", delimiter="\t")

data
```

Out[22]:

	Species	FrontalLip	RearWidth	Length	Width	Depth	Male	Female
0	0	20.6	14.4	42.8	46.5	19.6	1	0
1	1	13.3	11.1	27.8	32.3	11.3	1	0
2	0	16.7	14.3	32.3	37.0	14.7	0	1
3	1	9.8	8.9	20.4	23.9	8.8	0	1
4	0	15.6	14.1	31.0	34.5	13.8	0	1
...	...	...	...	...	...	...	...	...
195	1	12.3	11.0	26.8	31.5	11.4	1	0
196	1	12.0	11.1	25.4	29.2	11.0	0	1
197	1	8.8	7.7	18.1	20.8	7.4	1	0
198	1	16.2	15.2	34.5	40.1	13.9	0	1
199	0	15.6	14.0	31.6	35.3	13.8	0	1

200 rows × 8 columns

The first column corresponds to the class label (crab species) and the other 7 columns correspond to the features. Use the first 140 samples as your training set and the last 60 samples as your test set.

In [23]:
```
# Partitioning the data into training and test sets

X_train = data.iloc[:140,1:].to_numpy()
t_train = data.iloc[:140,0].to_numpy()

X_test = data.iloc[140:,1:].to_numpy()
t_test = data.iloc[140:,0].to_numpy()

X_train.shape, X_test.shape, t_train.shape, t_test.shape
```

Out[23]:
```
((140, 7), (60, 7), (140,), (60,))
```

Answer the following questions:

1. (15 points) **Implement the Naive Bayes classifier, under the assumption that your data likelihood model $p(x|C_j)$ is a multivariate Gaussian and the prior probabilities $p(C_j)$ are dictated by the number of samples $n_j \in \mathbb{R}$ that you have for each class. Build your own code to implement the classifier.**

In [27]:
```
class NaiveBayesClassifier:
    def __init__(self, regularization=1e-4):
        self.class_priors = None
        self.class_likelihoods = None
        self.regularization = regularization

    def fit(self, X_train, t_train):
        unique_classes, class_counts = np.unique(t_train, return_counts=True)
        self.class_priors = class_counts / len(t_train)

        self.class_likelihoods = []
        for class_label in unique_classes:
            class_data = X_train[t_train == class_label]
            class_mean = np.mean(class_data, axis=0)
            class_covariance = np.cov(class_data, rowvar=False) + np.eye(X_train.shape
            self.class_likelihoods.append((class_mean, class_covariance))

    def predict(self, X_test):
        predictions = []
        for sample in X_test:
            likelihoods = []
            for class_mean, class_covariance in self.class_likelihoods:
                mvn = multivariate_normal(mean=class_mean, cov=class_covariance)
                likelihood = mvn.pdf(sample)
                likelihoods.append(likelihood)
            posterior_probs = likelihoods * self.class_priors
            predicted_class = np.argmax(posterior_probs)
            predictions.append(predicted_class)
        return np.array(predictions)

naive_bayes_classifier = NaiveBayesClassifier()
naive_bayes_classifier.fit(X_train, t_train)
predictions = naive_bayes_classifier.predict(X_test)
```

In [ ]:

In [ ]:

1. (5 points) **Did you encounter any problems when implementing the probabilistic generative model? What is your solution for the problem? Explain why your solution works. (Note: There is more than one solution.)**

Yes, I met the "LinAlgError" because I didn't handle the covarience matrix well at the first place. To solve it, I added a small regularization term to the covariance matrix, ensuring that the covariance matrix becomes positive definite.

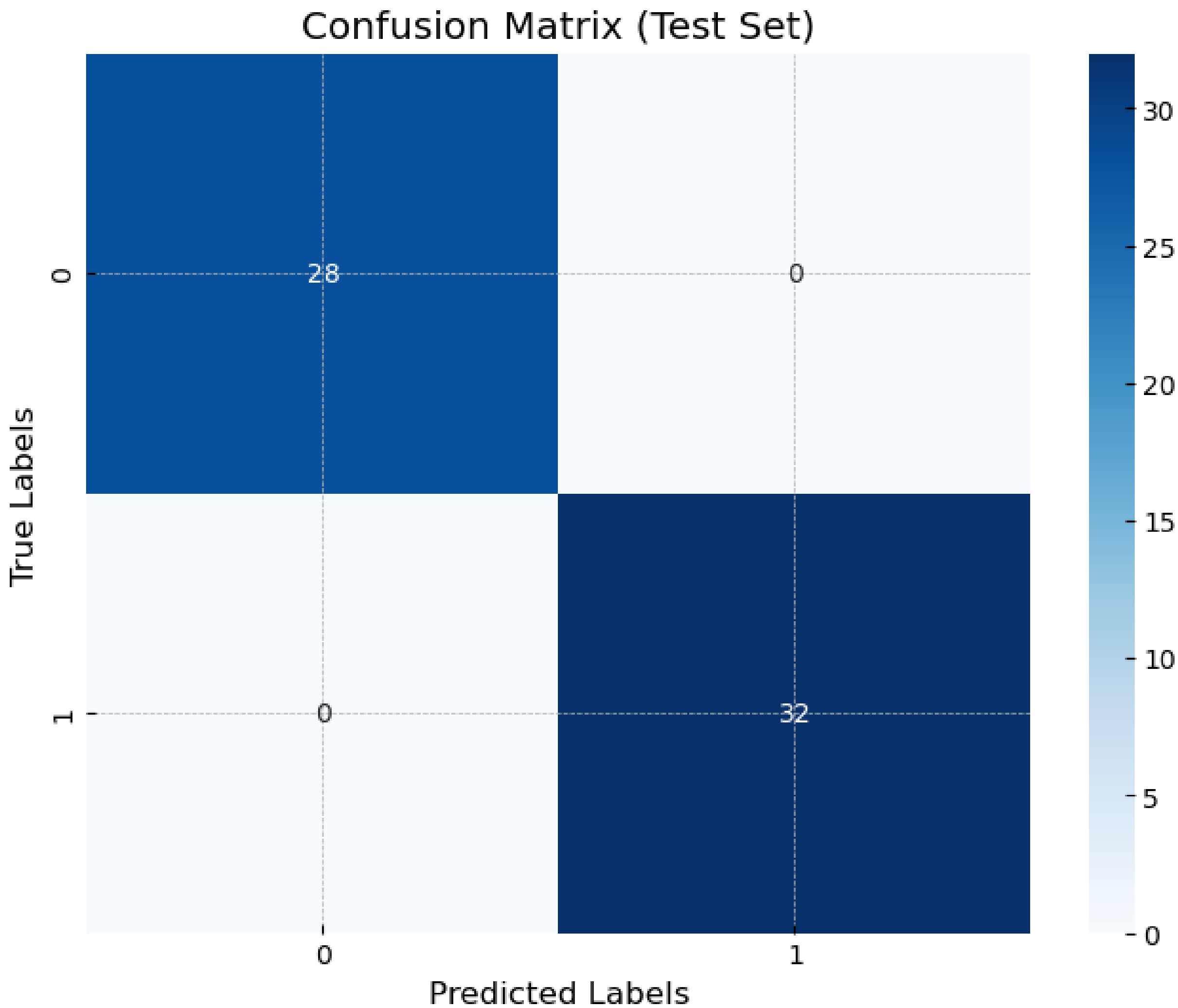
In [ ]:

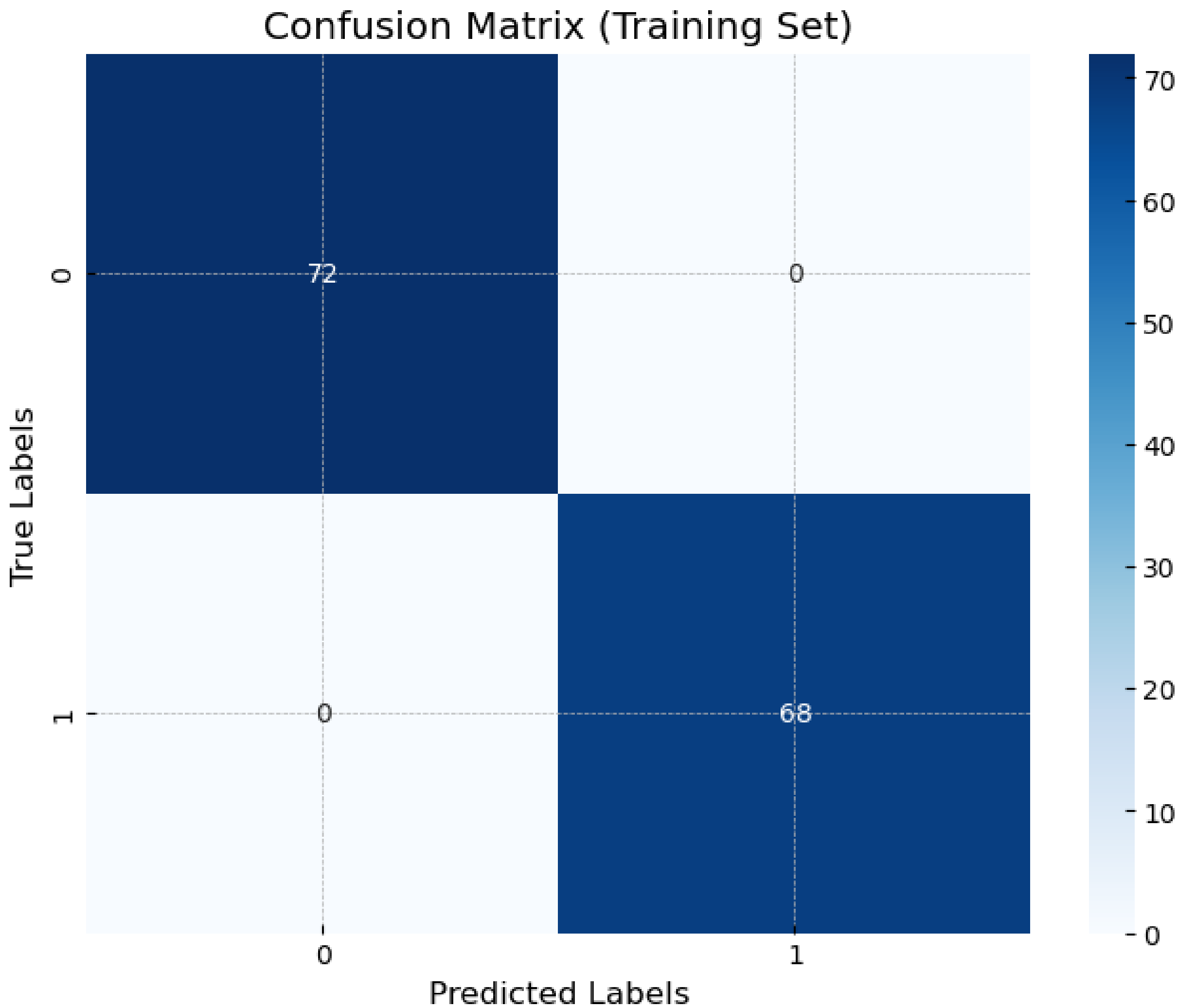
In [ ]:

1. (5 points) **Report your classification results in terms of a confusion matrix in both training and test set. (You can use the function `confusion_matrix` from the module `sklearn.metrics`.)**

In [29]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_matrix_test = confusion_matrix(t_test, predictions)

predictions_train = naive_bayes_classifier.predict(X_train)
conf_matrix_train = confusion_matrix(t_train, predictions_train)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_test, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique
plt.title("Confusion Matrix (Test Set)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_train, annot=True, fmt="d", cmap="Blues", xticklabels=np.uniqu
plt.title("Confusion Matrix (Training Set)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```


Confusion Matrix (Test Set)
28
0
0
32
True Labels
Predicted Labels
0
1
30
25
20
15
10
5
0



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.
