Assignment #2

Programming Assignment

Lecturer: Prof. Jaesik Park
Teaching Assistants: Cheolhong Min, Milo Deroussi, Youngjoong Kim

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Due date: 11:59 PM April 18, 2025

Evaluation policy:

- Late submission penalty
  - Late submission penalty of 30% per day will be applied to the total score.
    - 30% penalty until 11:59 PM April 19, 60% on April 20, 90% on April 21
  - After 11:59 PM April 22
    - 100% penalty is applied for the submission.
- Your code will be automatically tested using an evaluation program.
  - Each problem has the maximum score.
  - A score will be assigned based on the behavior of the program.
- We won't accept any submission via email - it will be ignored.
- Do not modify auxiliary files.
  - Such as: "test.py", and so on.
- Run your Python script and check your program before the submission.
- Please do not use any libraries other than primitive libraries and numpy (2.2.3)
  - Such as: "Scipy", "Scikit-learn"
  - Any submission using other libraries will be disregarded.
- Never try to copy your classmates' source code. Both of you and your classmates will get severe penalties, and the case will be reported to the University's office.

File you need to submit for programming assignment 2:

- pa2.py (Do not change the filename!)

Any questions? Please use the **Classum** community (using a tag, "Assignment #2").

0. Prerequisite

Environment

- OS: Windows, Mac OS, or Linux
- Language: **Python 3.12**
- Required package: **numpy 2.2.3**

Install Python and PIP. If you already have them, you can skip this step.

- For Windows and Mac Users
  : Download and install Python from https://www.python.org/downloads/.

  If installing on Windows, make sure to check **'Add python.exe to PATH'**.

```
$ python -c “import pip; print(‘installed’)”
installed
```

- For Linux Users (e.g., Ubuntu 22.04)

```
$ apt-get update
$ apt-get install -y python3 python3-pip
$ ln -s python3 /usr/bin/python
$ python -c “import pip; print(‘installed’)”
installed
```

Basic instruction

- Install the dependencies and run the script to test its functionality

```
$ pip install numpy==2.2.3
$ python test.py
[*] Nothing returned. Please check the return
value of `prepare_dataset`.
[*] Response;
TestResult(dataset=None, models=None, metric=None,
state=FAILED_STEP1, _traceback=)
```

Step-by-step guidelines

Our goal in this programming assignment is to implement **a handwritten digit image classifier using a Support Vector Machine**. We will provide step-by-step guidelines. Follow them carefully and submit your assignment.
The assignment will be evaluated automatically using different datasets than the one provided. Do not hardcode any values, and make sure to correctly fill in the required sections of the skeleton code, `pa2.py`.

1. Prepare the digit dataset *(Code is provided)*

    a. Use a function to load a CSV-formatted handwritten digit image dataset. The function should take the file path as the path argument and return a tuple containing the loaded images and their corresponding labels.

    b. Input & Output
    Input: a path to a CSV-formatted handwritten digit image dataset.

    Output: The function should return a tuple of two arrays
    - **Images**: A NumPy array of shape [#images, 16, 16] with np.float32 data type, where the values are scaled to the range in [0.0, 1.0].
    - **Labels**: A NumPy array of shape [#images] with np.long data type, where values range from 0 to 9, indicating the digit represented by each image.

    c. Example input & output

Input	Output
`Path("trainset.csv")`	`(array([[[0., 0., …, 0.], …], …], shape=(60, 16, 16), dtype=float32), array([0, 0, …, 9, 9]))`
`Path("evalset.csv")`	`(array([[[0., 0., …], …], …], shape=(60, 16, 16), dtype=float32), array([0, 0, …, 9, 9]))`

d. Example execution

In this assignment, **we provide the code of 'prepare_dataset'**, so you can see the following result after you run `test.py`.

```
$ python test.py
[*] Successfully load the dataset.
[*] Invalid value returned. Please check the return value of
`compute_kernel_matrix`
[*] Response;
TestResult(dataset=..., models=None, metric=None,
state=FAILED_STEP2, _traceback=)
```

2. Compute the kernel matrix

a. Implement a function to compute the kernel matrix between two sets of flattened images. The function should take two image sets as input and return the computed kernel matrix. For efficiency, you need to compute this for all pairs of vectors in X and Y.

b. Input & Output

Input: two sets of flattened images and a bandwidth parameter, sigma.

- **X:** A NumPy array of shape [N, H*W] with np.float32 data type, where each row is a flattened image with values scaled to the range [0.0, 1.0].
- **Y**: A NumPy array of shape [M, H*W] with np.float32 data type. When Y is None, compute the kernel matrix between X and itself.
- **sigma**: a bandwidth parameter for the Gaussian kernel

Output: The kernel matrix (a numpy array) with shape [N, M], containing Gaussian kernel values between all pairs of images from X and Y.

c. Example input & output

Input	Output
`compute_kernel_matrix(np.ones((3, 256)), np.ones((3, 256)), 1.0)`	`array([[1., 1., 1.], [1., 1., 1.], [1., 1., 1.]])`
`compute_kernel_matrix(np.ones((3, 256)), np.zeros((2, 256)), 10.0)`	`array([[0.2780373, 0.2780373], [0.2780373, 0.2780373], [0.2780373, 0.2780373]])`

For various reasons, the output value may differ from the example.
Do not worry about minor differences.

d. Example execution

```
>> from numpy as np
>> from pa2 import compute_kernel_matrix
>> X = np.ones((3, 256))
>> Y = np.zeros((3, 256))
>> compute_kernel_matrix(X, Y, 10.0)
array([[0.2780373, 0.2780373], [0.2780373, 0.2780373],
[0.2780373, 0.2780373]])
```

After you correctly implementing the `compute_kernel_matrix`;

```
$ python test.py
[*] Successfully load the dataset.
[*] Kernel matrix computation passed the test.
[*] Invalid return value. Please check the return value of
`train_binary_svm`
[*] Response: TestResult(dataset=..., models=None, metric=None,
state=FAILED_STEP3, _traceback=)
```

TIP: Use matrix operations for efficient computation. The squared Euclidean distance can be expressed as: $||x - y||^2 = ||x||^2 + ||y||^2 - 2x \cdot y$

3. Train a binary SVM classifier

a. Implement a function to train a binary SVM classifier for one-vs-rest classification using a simplified Sequential Minimal Optimization (SMO) algorithm. This function should optimize the alpha parameters and bias term for a given target class.

b. Input & Output

Input: a loaded dataset with training hyperparameters.

- **dataset**: a loaded dataset, the return values of `prepare_dataset`.
- **target_class:** the target class for binary classification (one-vs-rest).
- **num_iterations:** maximum number of iterations for the optimization.
- **C:** regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error**.**
- **sigma:** a bandwidth parameter of the Gaussian kernel.
- **tol:** tolerance for KKT (Karush-Kuhn-Tucker) conditions.
- **eps:** numerical stability constant.

Output: a tuple containing:

- **alphas:** A NumPy array of shape [B] with np.float32 data type, where B is the number of training examples.
- **bias:** A float value representing the bias term.

c. Example execution

```
>> from pathlib import Path
>> import numpy as np
>> from pa2 import prepare_dataset, train_binary_svm
>> images, labels = prepare_dataset(Path("./trainset.csv"))
>> alphas, bias = train_binary_svm((images, labels), 0, 100,
1.0, 1.0)
>> print(alphas.shape)
(300,)
>> print(bias)
-0.8430294
```

For various reasons, the output value may differ from the example.
Do not worry about minor differences.

After you correctly implementing the `train_binary_svm`;

```
$ python test.py
[*] Successfully load the dataset.
[*] Kernel matrix computation passed the test.
[*] Binary SVM training passed the test
[*] Invalid value returned. Please check the return value of
`train_multi_class_svm`
[*] Response;
TestResult(dataset=..., models=None, metric=None,
state=FAILED_STEP4, _traceback=)
```

4. Train a multi-class SVM classifier

a. Implement a function to train a multi-class SVM classifier using the one-vs-rest approach. This function should use the binary SVM classifier you implemented in the previous step to **train separate models for each digit class**.

b. Input & Output

Input: a loaded dataset with training hyperparameters.

- **dataset**: a loaded dataset, the return values of prepare_dataset.
- **num_iterations**: maximum number of iterations for the optimization.
- **C**: regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error.
- **sigma**: a bandwidth parameter of the Gaussian kernel.

Output: a **list of tuples**, where each tuple contains:

- **alphas**: A NumPy array of shape [B] with np.float32 data type, the trained alpha parameters for a specific class.
- **bias**: A float value representing the bias term for that class.

c. Example execution

```
>> from pathlib import Path
>> import numpy as np
>> from pa2 import prepare_dataset, train_multi_class_svm
>> images, labels = prepare_dataset(Path("./trainset.csv"))
>> models = train_multi_class_svm((images, labels), 100, 1.0,
1.0)
>> print(len(models))
10
>> print(models[0][0].shape, models[0][1])
(300,) -0.8496258
```

For various reasons, the output value may differ from the example.
Do not worry about minor differences.

After you correctly implementing the `train_multi_class_svm`;

```
$ python test.py
```

```
[*] Successfully loaded the dataset.
[*] Kernel matrix computation passed the test.
[*] Binary SVM training passed the test.
Training SVM for class 0...
Training SVM for class 1...
Training SVM for class 2...
Training SVM for class 3...
Training SVM for class 4...
Training SVM for class 5...
Training SVM for class 6...
Training SVM for class 7...
Training SVM for class 8...
Training SVM for class 9...
[*] Multi-class SVM training done.
[*] Invalid return value. Please check the return value of
`classify`
[*] Response;
TestResult(dataset=..., models=..., metric=None,
state=FAILED_STEP5, _traceback=)
```

5. Classify the given image

a. Implement a function to classify a given image using the trained multi-class SVM classifier. The function should take the trained models, training data, and a test image as input, then return the predicted digit label.

b. Input & Output

Input: a trained multi-class SVM classifier and a target image to classify.

- **models**: list of tuples containing trained alphas and bias for each class.
- **train_images**: the training images used during model training.
- **train_labels**: the training labels corresponding to the training images.
- **test_image**: the target image to classify.
- **sigma**: a bandwidth parameter of the Gaussian kernel.

Output: an integer representing the estimated digit label (0-9).

c. Example execution

```
>> from pathlib import Path
>> import numpy as np
>> from pa1 import prepare_dataset, train_multi_class_svm,
classify
>> images, labels = prepare_dataset(Path(“./trainset.csv”))
>> models = train_multi_class_svm((images, labels), 100, 1.0,
1.0)
>> predict = classify(models, images, labels, images[10], 1.0)
>> print(f"Predicted: {predict}, Actual: {labels[10]}")
Predicted: 0, Actual: 0
```

After you correctly implementing the `classify`;

```
$ python test.py
[*] Successfully loaded the dataset.
[*] Kernel matrix computation passed the test.
[*] Binary SVM training passed the test.
Training SVM for class 0...
Training SVM for class 1...
Training SVM for class 2...
Training SVM for class 3...
```

```
Training SVM for class 4...
Training SVM for class 5...
Training SVM for class 6...
Training SVM for class 7...
Training SVM for class 8...
Training SVM for class 9...
[*] Multi-class SVM training done.
[*] Training accuracy: 1.00
[*] Evaluation accuracy: 0.31
[*] Testing completed successfully.
[*] Response;
TestResult(dataset=..., models=..., metric={'training-
accuracy': 1.0, 'evaluation-accuracy': 0.30666666666666664},
state=SUCCESS, _traceback=)
```

If you use the default hyperparameters, you will likely see lower evaluation accuracy. To avoid penalties, you should achieve the evaluation accuracy **at least 0.7** on the test dataset.

TIP: Consider the balance between overfitting and underfitting. In the example above, there's a significant gap between the high training accuracy and lower evaluation accuracy (is it overfitting or underfitting?). Adjusting the values of C (regularization parameter) and sigma (kernel bandwidth) can help mitigate this issue.

6. Submit the assignment

After verifying that **`state=SUCCESS`**, submit your `pa2.py` to ETL.
We expect you to achieve more than **70% evaluation accuracy**, which will allow you to obtain a high score on the test set without much difficulty.

Don't forget there is also a **handwritten problem**!