ML Homework 05-2: SVM

- Student ID: 309553002
- Name: 林育愷

Code with Detailed Explanations

Prerequisites

I use Python 3.6 for this implementation based on platform Ubuntu 18.04, with the following packages

- NumPy,
- SciPy,
- Matplotlib,
- seaboan, and
- libsvm (https://github.com/cjlin1/libsvm, embedded in this project).

Input Data

We use MNIST as our dataset. As it is of csv format, we can simply import them with NumPy.

```
import numpy as np


# image_filename = 'X_train.csv' or 'X_test.csv'
# label_filename = 'Y_train.csv' or 'Y_test.csv'
def load_data(image_filename, label_filename):
    images = np.genfromtxt(image_filename, delimiter=',')
    labels = np.genfromtxt(label_filename, delimiter=',').flatten()
    return images, labels
```

Part 1

According to the documentation [1], LIBSVM supports the following types of kernel functions:

- linear: $u^T v$,
- polynomial: $(\gamma u^T v + c_0)^d$,
- radial basis function (RBF): $\exp(-\gamma \|u - v\|^2)$, and
- sigmoid: $\tanh(\gamma u^T v + c_0)$.

In the first part we are going to perform linear, polynomial, and RBF, so we define ones corresponding arguments.

```python
import enum


class KernelType(enum.Enum):
    LINEAR = '-t 0'
    POLYNOMIAL = '-t 1'
    RADIAL_BASIS_FUNCTION = '-t 2'
    PRECOMPUTED_KERNEL = '-t 4'
```

Later we feed them into solver and find out the corresponding results.

```python
import time
import libsvm.python.svmutil as svmutil


for ktype in KernelType:
    # skip custom mode
    if ktype == KernelType.PRECOMPUTED_KERNEL:
        continue

    param = ktype.value + ' -q'
    start = time.perf_counter()
    model = svmutil.svm_train(train_label, train_image, param)
    end = time.perf_counter()
    duration = end - start

    _, accuracy, _ = svmutil.svm_predict(test_label, test_image, model)
    print('[%s] accuracy: %.2f%%, mse: %.2f, time: %.2f sec' %
          (ktype.name, accuracy[0], accuracy[1], duration))
```

Part 2

In the second part we are going to fine-tune the parameters of C-SVC with RBF kernel function. It means there are C and γ that we are going to figure out.

Note that each run takes time and only a processor is used, so I use multiprogramming to boost the running time.

```python
import multiprocessing as mp
import numpy as np


NUMBER_OF_CPUS = mp.cpu_count()
```

```python

ktype = KernelType.RADIAL_BASIS_FUNCTION
cs = np.exp(np.arange(-5, 3))
gammas = np.exp(np.arange(-9, 0))
accuracy_matrix = np.zeros(cs.shape + gammas.shape)

for i, c in enumerate(cs):
    args_list = []
    for j, gamma in enumerate(gammas):
        param = '-q %s -v 3 -s 0 -c %f -g %f' % (ktype.value, c, gamma)
        svm_args = (train_label, train_image, param)
        args_list.append(svm_args)

    with mp.Pool(NUMBER_OF_CPUS) as pool:
        results = pool.starmap(svmutil.svm_train, args_list)

    accuracy_matrix[i] = results
```

Part 3

In the final part we are going to build a custom kernel function. In this assignment the kernel function is set to be

$$k(x, x') = k_{\text{linear}}(x, x') + k_{\text{RBF}}(x, x').$$

For efficiency we separate distance function and kernel function in RBF so as to improve the running time. Also by documentation the format for each row should be `[i, k(x_i, x_1), k(x_i, x_2), ..., k(x_i, x_n)]`.

```python
import numpy as np
from scipy.spatial.distance import cdist


def calc_feature_distance(x1, x2):
    return cdist(x1, x2, 'sqeuclidean')


def linear_plus_rbf_kernel(x1, x2, feature_distance, gamma):
    linear = x1 @ x2.T
    # https://www.csie.ntu.edu.tw/~cjlin/libsvm/
    rbf = np.exp(-gamma * feature_distance)
    param = np.hstack(
        (np.arange(x1.shape[0])[:, np.newaxis] + 1, linear + rbf))
    return param
```

Again we use multiprogramming to run multiple cases of γ so as to find a good configuration.

```python
import multiprocessing as mp
import numpy as np


ktype = KernelType.PRECOMPUTED_KERNEL
gammas = np.exp(np.arange(-9, 0))
args_list = []

train_feature_distance = calc_feature_distance(train_image,
```

```python
                                                  train_image)
test_feature_distance = calc_feature_distance(test_image, test_image)

for j, gamma in enumerate(gammas):
    param = '-q %s' % (ktype.value)
    train_kernel = linear_plus_rbf_kernel(train_image, train_image,
                                          train_feature_distance,
                                          gamma)
    test_kernel = linear_plus_rbf_kernel(test_image, test_image,
                                         test_feature_distance, gamma)

    svm_args = (train_label, train_kernel, test_label, test_kernel,
                param)
    args_list.append(svm_args)

with mp.Pool(NUMBER_OF_CPUS) as pool:
    accuracy_matrix = pool.starmap(train_and_predict, args_list)

accuracy_matrix = np.array([accuracy_matrix])
```

Experiments Settings and Results

For visualization we use `seaborn` to colorize the accuracy matrix.

```python
import seaborn as sns


sns.heatmap(accuracy_matrix, xticklabels=np.log(gammas), annot=True)
```

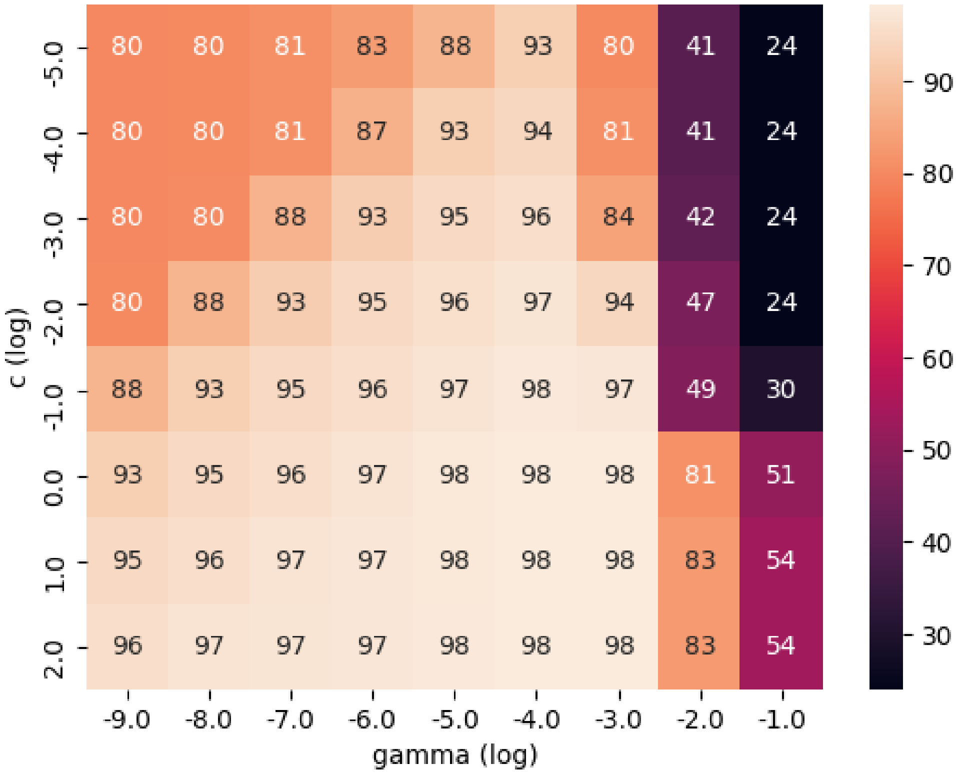
Part 1

We see that RBF and linear have similar accuracy of classification problem but polynomial kernel function get poor result and takes lots of time.

```
PART I
------------------------------------------------------------------
[LINEAR] accuracy: 95.08%, mse: 0.14, time: 1.80 sec
[POLYNOMIAL] accuracy: 34.68%, mse: 2.62, time: 21.00 sec
[RADIAL_BASIS_FUNCTION] accuracy: 95.32%, mse: 0.15, time: 3.92 sec
```

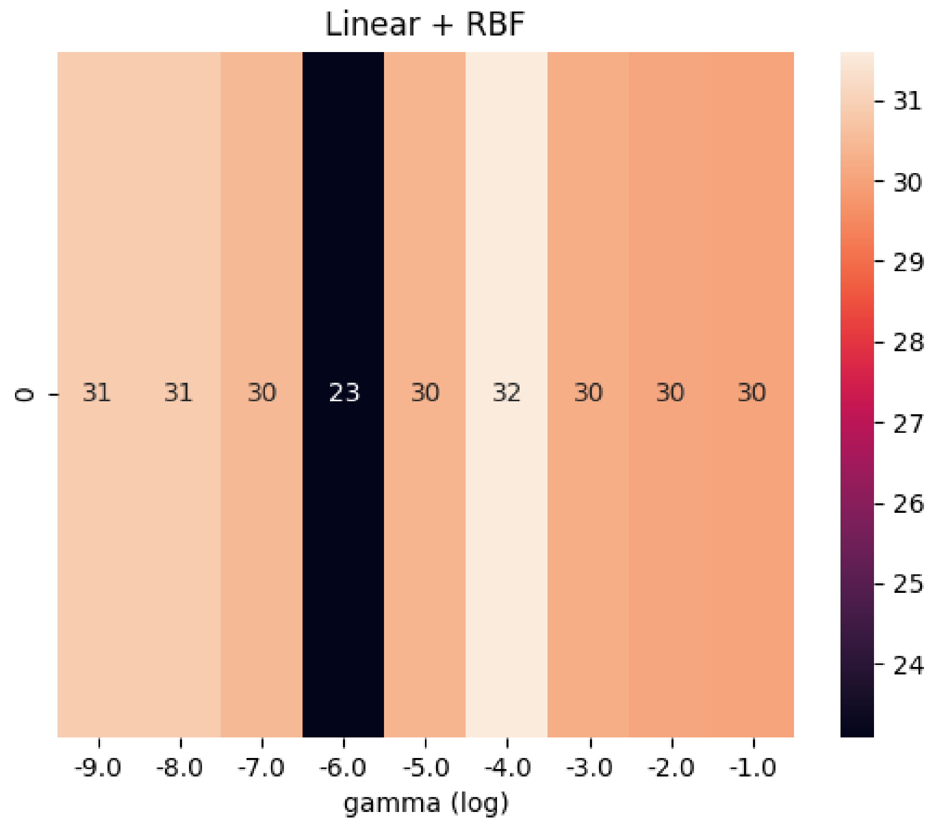
Part 2

In part 2 we find that with $(C, \gamma) = (\exp(2), \exp(-4))$ a better result is obtained (with cross-validation 98.34%).



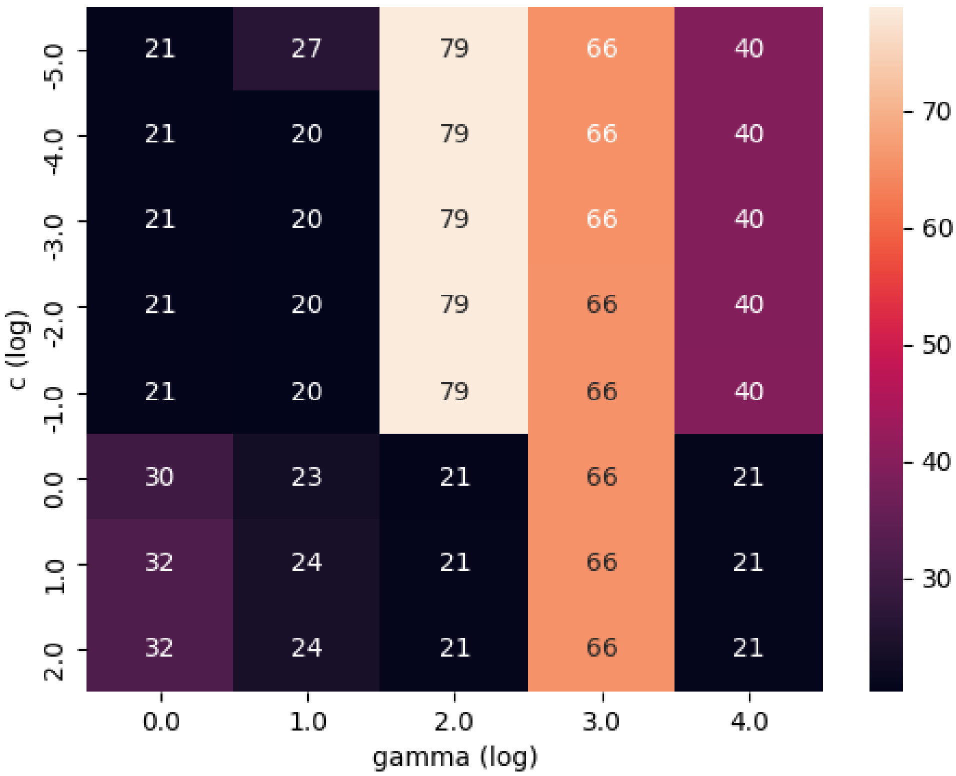
Part 3

We find that the given kernel function for this problem cannot get a better result.



Observations and Discussion

In part 2 we find that with larger C we have better performance. In general the larger γ (greater than 1), the poor result, but there are some exceptions.



References

1. https://github.com/cjlin1/libsvm ↩