ML HW5-2 Report

309552007 袁鈺勛

A. Code

a. Modes

There are three modes in the program. Mode 0 is part 1. Mode 1 is part 2. And mode 2 is part 3. It can be controlled by "–m" parameter in the command.

```
if mode == 0:
    info_log('=== Comparison of linear, polynomial and RBF kernels ===')
    linear_poly_rbf_comparison(tr_image, tr_label, te_image, te_label)
elif mode == 1:
    info_log('=== Grid search ===')
    grid_search(tr_image, tr_label)
else:
    info_log('=== Combination of linear and RBF kernels ===')
    linear_rbf_combination(tr_image, tr_label, te_image, te_label)
```

b. Part 1

In part 1, I utilize the "–t" parameter to choose the kernel to be used for training and "-q" parameter to prevent "svm_train" from printing redundant information. And I record the training time and use "svm_predict" function to get the prediction accuracy.

```
# Kernel names
kernels = ['Linear', 'Polynomial', 'RBF']

# Get performance of each kernel
for idx, name in enumerate(kernels):
    param = svm_parameter(f"-t {idx} -q")
    prob = svm_problem(training_label, training_image)

    print(f'# {name}')

    start = time.time()
    model = svm_train(prob, param)
    svm_predict(testing_label, testing_image, model)
    end = time.time()

    print(f'Elapsed time = {end - start:.2f}s\n')
```

c. Part 2

In part 2, there are parameter for grid search, e.g. constant, degree etc.

```
# Parameters
cost = [np.power(10.0, i) for i in range(-1, 2)]
degree = [i for i in range(0, 3)]
gamma = [1.0 / 784] + [np.power(10.0, i) for i in range(-1, 1)]
constant = [i for i in range(-1, 2)]
```

The is the function “grid_search_cv” which is used for each kernel for the cross validation of current parameters. Parameter “-v” is used to specify the number of folds for cross-validation. And I use 3 folds for cross-validation.

```
def grid_search_cv(training_image: np.ndarray, training_label: np.ndarray, parameters: str,
                   is_kernel: bool = False) -> float:
    """
    Cross validation for the given kernel and parameters
    :param training_image: training images
    :param training_label: training labels
    :param parameters: given parameters
    :param is_kernel: whether training_image is actually a precomputed kernel
    :return: accuracy
    """
    param = svm_parameter(parameters + ' -v 3 -q')
    prob = svm_problem(training_label, training_image, isKernel=is_kernel)
    return svm_train(prob, param)
```

During the grid search of linear kernel, I use different cost parameter “C” of C-SVC to get the highest accuracy. The parameter producing best accuracy is the best parameter for linear kernel.

```
if name == 'Linear':
    info_log('# Linear')
    for c in cost:
        parameters = f'-t {idx} -c {c}'
        acc = grid_search_cv(training_image, training_label, parameters)

        if acc > max_acc:
            max_acc = acc
            best_para = parameters
    best_parameter.append(best_para)
    max_accuracy.append(max_acc)
```

During the grid search of polynomial kernel, I use different cost, degree, gamma and constant parameters to get the highest accuracy. The parameters producing the best accuracy are the best parameters of polynomial kernel. Parameters "-d", "-g" and "-r" are used to specify the degree, gamma and constant of the kernel respectively.

```
elif name == 'Polynomial':
    info_log('# Polynomial')
    for c in cost:
        for d in degree:
            for g in gamma:
                for const in constant:
                    parameters = f'-t {idx} -c {c} -d {d} -g {g} -r {const}'
                    acc = grid_search_cv(training_image, training_label, parameters)

                    if acc > max_acc:
                        max_acc = acc
                        best_para = parameters
    best_parameter.append(best_para)
    max_accuracy.append(max_acc)
```

During the grid search of RBF kernel, I use cost and gamma parameters to get the highest accuracy. The parameters producing the best accuracy are the best parameters of RBF kernel.

```
else:
    info_log('# RBF')
    for c in cost:
        for g in gamma:
            parameters = f'-t {idx} -c {c} -g {g}'
            acc = grid_search_cv(training_image, training_label, parameters)

            if acc > max_acc:
                max_acc = acc
                best_para = parameters
    best_parameter.append(best_para)
    max_accuracy.append(max_acc)
```

d. Part 3

- linear: $\langle x, x' \rangle$. • rbf: $\exp(-\gamma \|x - x'\|^2)$,

According to the kernel functions in the above images, I construct the kernel functions of the two.

```
def linear_kernel(x: np.ndarray, y: np.ndarray) → np.ndarray:
    """
    Linear kernel, <x, y>
    :param x: point x
    :param y: point y
    :return: linear distance between them
    """
    return x.dot(y.T)


def rbf_kernel(x: np.ndarray, y: np.ndarray, gamma: float) → np.ndarray:
    """
    RBF kernel exp^(γ * ||x - y||^2)
    :param x: point x
    :param y: point y
    :param gamma: gamma coefficient
    :return: polynomial distance between them
    """
    return np.exp(-gamma * cdist(x, y, 'sqeuclidean'))
```

The parameters of in the following image are used for grid search. Parameter “rows” is used to produce the serial number in the first column of the kernel result.

```
# Parameters
cost = [np.power(10.0, i) for i in range(-2, 3)]
gamma = [1.0 / 784] + [np.power(10.0, i) for i in range(-3, 2)]
rows, _ = training_image.shape
```

Then I use grid search to find the best parameters with highest accuracy.

```
# Use grid search to find best parameters
linear = linear_kernel(training_image, training_image)
best_parameter = ''
best_gamma = 1.0
max_accuracy = 0.0
for c in cost:
    for g in gamma:
        rbf = rbf_kernel(training_image, training_image, g)

        # The combination is linear + RBF, but np.arange is the required serial number from the library
        combination = np.hstack((np.arange(1, rows + 1).reshape(-1, 1), linear + rbf))

        parameters = f'-t 4 -c {c}'
        acc = grid_search_cv(combination, training_label, parameters, True)
        if acc > max_accuracy:
            max_accuracy = acc
            best_parameter = parameters
            best_gamma = g
```

After obtaining the best parameters, I use them to construct the model. And I use the model to make prediction on the testing data.

```python
# Train the model using best parameters
rbf = rbf_kernel(training_image, training_image, best_gamma)
combination = np.hstack((np.arange(1, rows + 1).reshape(-1, 1), linear + rbf))
model = svm_train(svm_problem(training_label, combination, isKernel=True), svm_parameter(best_parameter + ' -q'))

# Make prediction using best parameters
rows, _ = testing_image.shape
linear = linear_kernel(testing_image, testing_image)
rbf = rbf_kernel(testing_image, testing_image, best_gamma)
combination = np.hstack((np.arange(1, rows + 1).reshape(-1, 1), linear + rbf))
svm_predict(testing_label, combination, model)
```

B. Results

a. Part 1

The image below contains the accuracy and elapsed time of each default kernel function.

```
# Linear
Accuracy = 95.08% (2377/2500) (classification)
Elapsed time = 4.40s

# Polynomial
Accuracy = 34.68% (867/2500) (classification)
Elapsed time = 39.82s

# RBF
Accuracy = 95.32% (2383/2500) (classification)
Elapsed time = 9.89s
```

b. Part 2

The image below contains the best parameters with highest accuracy of each kernel function after cross-validation.

```
# Linear
        Max accuracy: 96.66%
        Best parameters: -t 0 -c 0.1
Accuracy = 95.8% (2395/2500) (classification)

# Polynomial
        Max accuracy: 98.04%
        Best parameters: -t 1 -c 0.1 -d 2 -g 0.1 -r 0
Accuracy = 97.76% (2444/2500) (classification)

# RBF
        Max accuracy: 97.0%
        Best parameters: -t 2 -c 10.0 -g 0.0012755102040816326
Accuracy = 96.28% (2407/2500) (classification)
```

c. Part 3

The image below contains the best parameters with highest accuracy of

linear + RBF kernel after cross validation and the accuracy of testing result.

```
# Linear + RBF
        Max accuracy: 97.0%
        Best parameters: -t 4 -c 0.01 -g 1.0

Accuracy = 23.76% (594/2500) (classification)
```

C. Observations

a. Polynomial kernel is time-consuming.

b. From the results of part 1 and part 2, RBF is slightly better than the others. Because RBF maps data into infinite-D feature space, it can separate data better.

c. Since we use grid search to find the best parameters in part 2, the results of all kernels are better than those in part 1.

d. Accuracy of testing may be worse than that of cross-validation because the best parameters for training data may not be the best parameters for testing data.

e. Parameter “C” in C-SVC means the tolerance of outliers. The higher it is, the more prone it is to overfitting.