Machine Learning HW5-2

309551064 張凱翔

1. Code With Detailed Explanations

The figure below is my main function.

```
if __name__ == '__main__':
    X_train, Y_train, X_test, Y_test = read_file()
    print('Part 1:')
    part1(X_train, Y_train, X_test, Y_test)
    print('==============================================================')
    print('Part 2:')
    part2(X_train, Y_train, X_test, Y_test)
    print('==============================================================')
    print('Part 3:')
    part3(X_train, Y_train, X_test, Y_test)
```

First, I read data from "X_train.csv", "Y_train.csv", "X_test.csv" and "Y_test.csv with numpy function "loadtxt" and parameters setting with dtype=np.float and delimeter=',' respectively.

```
def read_file():
    X_train = np.loadtxt('./X_train.csv', dtype=np.float, delimiter=',')
    Y_train = np.loadtxt('./Y_train.csv', dtype=np.float, delimiter=',')
    X_test = np.loadtxt('./X_test.csv', dtype=np.float, delimiter=',')
    Y_test = np.loadtxt('./Y_test.csv', dtype=np.float, delimiter=',')
    return X_train, Y_train, X_test, Y_test
```

Part 1.

After reading files, I call the function named part1.

```
def part1(X_train, Y_train, X_test, Y_test):
    kernel = ['linear', 'polynomial', 'RBF']
    for i in range(3):
        print('Kernel Function: {}'.format(kernel[i]))
        parameter = '-q -t ' + str(i)
        model = svm_train(Y_train , X_train, parameter)
        svm_predict(Y_test, X_test, model)
```

I initialize a list with different kernel name and follow the order of the parameters in [1] and add parameters **'-q'** to disable screen output of 'svm_train' found in [2].

```
-t kernel_type : set type of kernel function (default 2)
        0 -- linear: u'*v
        1 -- polynomial: (gamma*u'*v + coef0)^degree
        2 -- radial basis function: exp(-gamma*|u-v|^2)
```

At last, call 'svm_train' and 'svm_predict' to train the model with different kernels and evaluate them.

Part 2.

After finishing the function part1, I call the function named part2.

```
def part2(X_train, Y_train, X_test, Y_test):
    cost = ['1', '2', '3']
    gamma = ['0.25', '0.5']
    degree = ['2', '3', '4']
    coef0 = ['0', '1', '2']
    best_parameter = ''
    best_accuracy = 0
    for i in range(3):
        parameter = '-v 10 -q -t 0 -c ' + cost[i]
        accuracy = svm_train(Y_train, X_train, parameter)
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_parameter = parameter

    for i in range(3):
        for j in range(2):
            for k in range(3):
                for l in range(3):
                    parameter = '-v 10 -q -t 1 -c ' + cost[i] + ' -g ' + gamma[j] + ' -d ' + degree[k] + ' -r ' + coef0[l]
                    accuracy = svm_train(Y_train, X_train, parameter)
                    if accuracy > best_accuracy:
                        best_accuracy = accuracy
                        best_parameter = parameter

    for i in range(3):
        for j in range(2):
            parameter = '-v 10 -q -t 2 -c ' + cost[i] + ' -g ' + gamma[j]
            accuracy = svm_train(Y_train, X_train, parameter)
            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_parameter = parameter

    print('Best Accuracy: {}'.format(best_accuracy))
    print('Corresponding Parameter: {}'.format(best_parameter))
```

First, I initialize four lists of different parameters for grid search and two variables to store best parameters and accuracy.

```
cost = ['1', '2', '3']
gamma = ['0.25', '0.5']
degree = ['2', '3', '4']
coef0 = ['0', '1', '2']
best_parameter = ''
best_accuracy = 0
```

The first one is linear kernel and it has no specific parameters needed to set, so I only set parameters of cross-validation **'-v 10'** for 10-folds, **'-q'** to disable screen output of 'svm_train', **'-t 0'** for linear kernel and **'-c *'** for different costs to C-SVC. After calculating one accuracy, it will compare with the best one and update if needed.

```
for i in range(3):
    parameter = '-v 10 -q -t 0 -c ' + cost[i]
    accuracy = svm_train(Y_train, X_train, parameter)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_parameter = parameter
```

The second one is polynomial kernel and it has three specific parameters degree, gamma and coef0 needed to set respectively. I set parameters of cross-validation **'-v 10'** for 10-folds, **'-q'** to disable screen output of 'svm_train', **'-t 1'** for polynomial kernel, **'-c *'** for different costs to C-SVC, **'-g *'** for different gammas, **'-d *'** for different degrees and **'-r *'** for different coef0s. After calculating one accuracy, it will compare with the best one and update if needed.

```
for i in range(3):
    for j in range(2):
        for k in range(3):
            for l in range(3):
                parameter = '-v 10 -q -t 1 -c ' + cost[i] + ' -g ' + gamma[j] + ' -d ' + degree[k] + ' -r ' + coef0[l]
                accuracy = svm_train(Y_train, X_train, parameter)
                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    best_parameter = parameter
```

The last one is RBF kernel and it has one specific parameters gamma needed to set. I set parameters of cross-validation **'-v 10'** for 10-folds, **'-q'** to disable screen output of 'svm_train', **'-t 2'** for RBF kernel, **'-c *'** for different costs to C-SVC and **'-g *'** for different gammas. After calculating one accuracy, it will compare with the best one and update if needed.

```
for i in range(3):
    for j in range(2):
        parameter = '-v 10 -q -t 2 -c ' + cost[i] + ' -g ' + gamma[j]
        accuracy = svm_train(Y_train, X_train, parameter)
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_parameter = parameter
```

Finally, I output the best parameters and accuracy.

```
print('Best Accuracy: {}'.format(best_accuracy))
print('Corresponding Parameter: {}'.format(best_parameter))
```

Part 3.

After finishing the function part2, I call the function named part3.

```
def part3(X_train, Y_train, X_test, Y_test):
    negative_gamma = -1 / 784
    train_linear_kernel = X_train.dot(X_train.transpose())
    train_rbf_kernel = np.exp(negative_gamma * cdist(X_train, X_train, 'sqeuclidean'))
    X_train_kernel = np.concatenate((np.arange(1, 5001).reshape((5000, 1)), train_linear_kernel + train_rbf_kernel), axis=1)

    test_linear_kernel = X_test.dot(X_train.transpose())
    test_rbf_kernel = np.exp(negative_gamma * cdist(X_test, X_train, 'sqeuclidean'))
    X_test_kernel = np.concatenate((np.arange(1, 2501).reshape((2500, 1)), test_linear_kernel + test_rbf_kernel), axis=1)


    prob  = svm_problem(Y_train, X_train_kernel, isKernel=True)
    param = svm_parameter('-q -t 4')
    model = svm_train(prob, param)
    svm_predict(Y_test, X_test_kernel, model)
```

To use a user-defined kernel, I should calculate training data and testing data according to the kernel needed to use and original data.

Linear Kernel:

$$k(x, y) = x^T y + c$$

RBF Kernel:

$$k(x, y) = \exp\left(-\gamma \|x - y\|^2\right)$$

```
Assume there are L training instances x1, ..., xL and.
Let K(x, y) be the kernel
value of two instances x and y. The input formats
are:

New training instance for xi:

<label> 0:i 1:K(xi,x1) ... L:K(xi,xL)

New testing instance for any x:

<label> 0:? 1:K(x,x1) ... L:K(x,xL)
```

I calculate the data with kernel functions according to the input of the figure above, add the two kernel data and add a label in front of each data just like the format asked to.

At last, call 'svm_problem' and set **'isKernel=True'** for precomputed kernel, set parameters **'-q'** to disable screen output of 'svm_train' and **'-t 4'** for precomputed kernel to predict the result.

2. Experiments Settings and Results

Part 1.

```
Part 1:
Kernel Function: linear
Accuracy = 95.08% (2377/2500) (classification)
Kernel Function: polynomial
Accuracy = 34.68% (867/2500) (classification)
Kernel Function: RBF
Accuracy = 95.32% (2383/2500) (classification)
==============================================
```

Part 2.

```
Part 2:
Cross Validation Accuracy = 96.18%
Cross Validation Accuracy = 96.3%
Cross Validation Accuracy = 96.34%
Cross Validation Accuracy = 98.24%
Cross Validation Accuracy = 98.2%
Cross Validation Accuracy = 98.1%
Cross Validation Accuracy = 97.74%
Cross Validation Accuracy = 98.02%
Cross Validation Accuracy = 98.14%
Cross Validation Accuracy = 96.76%
Cross Validation Accuracy = 97.38%
Cross Validation Accuracy = 97.78%
Cross Validation Accuracy = 98.24%
Cross Validation Accuracy = 98.2%
Cross Validation Accuracy = 98.2%
Cross Validation Accuracy = 97.74%
Cross Validation Accuracy = 97.94%
Cross Validation Accuracy = 97.96%
Cross Validation Accuracy = 96.64%
Cross Validation Accuracy = 97.02%
Cross Validation Accuracy = 97.4%
Cross Validation Accuracy = 98.16%
Cross Validation Accuracy = 98.12%
Cross Validation Accuracy = 98.14%
Cross Validation Accuracy = 97.8%
Cross Validation Accuracy = 97.98%
Cross Validation Accuracy = 98.04%
Cross Validation Accuracy = 96.76%
Cross Validation Accuracy = 97.24%
Cross Validation Accuracy = 97.66%
Cross Validation Accuracy = 98.18%
Cross Validation Accuracy = 98.14%
Cross Validation Accuracy = 98.08%
Cross Validation Accuracy = 97.8%
Cross Validation Accuracy = 97.84%
Cross Validation Accuracy = 98.04%
Cross Validation Accuracy = 96.82%
Cross Validation Accuracy = 97.04%
Cross Validation Accuracy = 97.2%
Cross Validation Accuracy = 98.22%
Cross Validation Accuracy = 98.18%
Cross Validation Accuracy = 98.14%
Cross Validation Accuracy = 97.66%
Cross Validation Accuracy = 98%
Cross Validation Accuracy = 98.1%
Cross Validation Accuracy = 96.68%
Cross Validation Accuracy = 97.2%
Cross Validation Accuracy = 97.64%
```

```
Cross Validation Accuracy = 98.24%
Cross Validation Accuracy = 98.18%
Cross Validation Accuracy = 98.12%
Cross Validation Accuracy = 97.6%
Cross Validation Accuracy = 97.88%
Cross Validation Accuracy = 97.96%
Cross Validation Accuracy = 96.78%
Cross Validation Accuracy = 97.12%
Cross Validation Accuracy = 97.22%
Cross Validation Accuracy = 63.44%
Cross Validation Accuracy = 44.9%
Cross Validation Accuracy = 66.36%
Cross Validation Accuracy = 46.08%
Cross Validation Accuracy = 66.86%
Cross Validation Accuracy = 45.86%
Best Accuracy: 98.24000000000001
Corresponding Parameter: -v 10 -q -t 1 -c 1 -g 0.25 -d 2 -r 0
=============================================================
```

Part 3.

```
Part 3:
Accuracy = 95.08% (2377/2500) (classification)
```

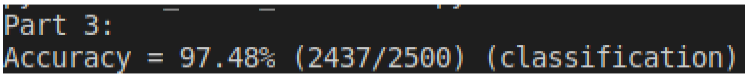
3. Observation and Discussion

I add polynomial kernel with parameters ‘alpha=1’, ‘c=0’ and ‘d=3’ to part3 and predict the result.

Polynomial kernel:

$$k(x, y) = (\alpha x^T y + c)^d$$

The result is much better than linear + RBF kernel but it is still a little bit lower than the result of grid search, so I think grid search is necessary for the best result.



Reference:

[1] https://www.jianshu.com/p/e9cd040de6ce
[2] https://www.csie.ntu.edu.tw/~cjlin/libsvm/
[3] https://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html
[4] http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/
[5] https://github.com/cjlin1/libsvm/blob/master/README
[6] https://blog.csdn.net/Olaking/article/details/43017329
[7] http://blog.sina.com.cn/s/blog_664f17ce0102w5rd.html