Machine Learning Homework 5 Report

0756079 資科工碩一 陳冠聞

Linear kernel is defined as (u'*v)

Polynomial kernel is defined as ((gamma*u'*v + coef0)^degree)

RBF kernel is defined as (exp(-gamma*|u-v|^2))

Linear + RBF is defined as lam_1*(u'*v) + lam_2*(exp(-gamma*|u-v|^2))

1. Use different kernel functions

	Linear	Polynomial	RBF
Accuracy	95.08%	34.68%	95.32%

All use default parameters, i.e.

For Linear kernel, we use C=1.

For Polynomial kernel, we use C = 1, gamma = 1/784, degree = 3, and coef0 = 0

For RBF kernel, we use C = 1, gamma = 1/784 for RBF kernel

We can see that both Linear Kernel & RBF kernel perform good, but polynomial perform bad, it's because the default degree 3, gamma about 1e-3 and coef=0 is not suit for this dataset, if we lower the degree, or set coef=1, or set gamma=1e-2 will all make polynomial kernel reach 90% accuracy.

Code as follow

```
In [1]: from svmutil import *
        import pandas as pd
        import numpy as np

In [2]: # Read Preprocessed data
        y_train, x_train = svm_read_problem(r'train.data')
        y_test, x_test = svm_read_problem(r'test.data')

In [3]: # Use different kernel functions  (linear, polynomial, and RBF kernels)
        print('Linear SVM:')
        linear_model =  svm_train(y_train, x_train, '-t 0' )
        p_label, p_acc, p_val = svm_predict(y_test,  x_test, linear_model)

        print('\nPolynoimal SVM:')
        poly_model =  svm_train(y_train, x_train, '-t 1' )
        p_label, p_acc, p_val = svm_predict(y_test,  x_test, poly_model)

        print('\nRBF SVM:')
        RBF_model =  svm_train(y_train, x_train, '-t 2' )
        p_label, p_acc, p_val = svm_predict(y_test,  x_test, RBF_model)

        Linear SVM:
        Accuracy = 95.08% (2377/2500) (classification)

        Polynoimal SVM:
        Accuracy = 34.68% (867/2500) (classification)

        RBF SVM:
        Accuracy = 95.32% (2383/2500) (classification)
```

In [1], import necessary package

In [2], read data which have been preprocessed to fit libsvm format.

In [3], Train SVM model with different kernels and test them.

2. Grid search

	Linear	Polynomial	RBF
Accuracy	95%(#1)	97.68% (#2)	98.08%

All use best parameters found in grid search for 5-fold cross validation

For Linear kernel, we use C = 2.0, which perform best for c in [1, 2, 3, 4, 5]

For Polynomial kernel, we use C =2.0, gamma=0.1, degree = 2, and coef = 0 , which perform best for c in [1, 2, 3], gamma g in [1e-3, 1e-2, 1e-1], degree d in [3,2,1] Coef r in [0,1]

For RBF kernel, we use C = 3.0, gamma = 0.01 which perform best for c in [1, 2, 3, 4, 5], gamma g in [1e-1, 1e-2, 1e-3, 1e-4, 1e-5]

(#1) We can see that the accuracy of linear kernel is lower than default setting, it's because we use parameters that perform best in 5-fold but it's possible it performs worse in real test data.

(#2) The accuracy of Polynomial Kernel improves quite a lot, but almost all setting can reach 90% above accuracy except default setting.

The accuracy of RBF kernel is improved by 2.76%.

Code as follow

```
In [4]: def grid_search(y_train, X_train, args='-t 2 -v 5 -q', para_range= {'c':[1,2,3], 'g':[None], 'd':[None], 'r':[None]} ):
            best_acc = 0.0
            best_para = {'c':0, 'g':0, 'd':0, 'r':0}
            for c in para_range['c']:
                for g in para_range['g']:
                    for d in para_range['d']:
                        for r in para_range['r']:
                            arg_c = '' if c is None else ' -c %s '%str(c)
                            arg_g = '' if g is None else ' -g %s '%str(g)
                            arg_d = '' if d is None else ' -d %s '%str(d)
                            arg_r = '' if r is None else ' -r %s '%str(r)
                            args_ = args + arg_c + arg_g + arg_d + arg_r
                            acc = svm_train(y_train, X_train, args_)
                            if acc > best_acc:
                                best_acc = acc
                                best_para = {'c':c, 'g':g, 'd':d, 'r':r}
            print("Best Parameters ", best_para , 'Accuracy :', best_acc)

In [5]: # Linear
        grid_search(y_train, X_train, args='-t 0 -v 5 ', para_range= {'c':[1,2,3], 'g':[None], 'd':[None], 'r':[None]})
        # Poly
        grid_search(y_train, X_train, args='-t 1 -v 5', para_range= {'c':[1,2,3], 'g':[1e-3, 1e-2, 1e-1], 'd':[3,2,1], 'r':[0,1]} )
        # RBF
        grid_search(y_train, X_train, args='-t 2 -v 5', para_range= {'c':[1,2,3,4,5], 'g':[1e-1, 1e-2, 1e-3, 1e-4], 'd':[None], 'r':[Non

        Best Parameters : {'c': 2, 'g': 0, 'd': 0, 'r': 0} Accuracy : 96.64
        Best Parameters  {'c': 2, 'g': 0.1, 'd': 2, 'r': 0} Accuracy : 98.14
        Best Parameters  {'c': 3, 'g': 0.01, 'd': 0, 'r': 0} Accuracy : 98.57
```

In [4], [5] we use grid search to find best parameters for each model

```
In [6]: # use params found in grid search to train & test model
        print('Linear SVM:')
        linear_model =  svm_train(y_train, X_train, '-t 0 -c 2.0' )
        p_label, p_acc, p_val = svm_predict(y_test,  X_test, linear_model)

        print('\nPolynoimal SVM:')
        poly_model =  svm_train(y_train, X_train, '-t 1 -c 2  -g 0.1  -d 2  -r 0' )
        p_label, p_acc, p_val = svm_predict(y_test,  X_test, poly_model)

        print('\nRBF SVM:')
        RBF_model =  svm_train(y_train, X_train, '-t 2 -c 3.0  -g 0.01' )
        p_label, p_acc, p_val = svm_predict(y_test,  X_test, RBF_model)

        Linear SVM:
        Accuracy = 95% (2375/2500) (classification)

        Polynoimal SVM:
        Accuracy = 97.68% (2442/2500) (classification)

        RBF SVM:
        Accuracy = 98.08% (2452/2500) (classification)
```

In [6] , we apply parameters found in grid search to train model and test

3. linear kernel + RBF kernel

	RBF	Linear + RBF
Accuracy	98.08%	96%

We do experiment on Linear + RBF kernel by fix c= 3.0 g=0.02 to compare with RBF kernel in question 2 and only adjust lam_1 and lam_2 for both lam_1 and lam_2 are in [0.2, 0.4, 0.6, 0.8, 1.0].

Result show that best accuracy for Linear + RBF is 96% for lam_1=0.2 and lam_2=1.0 in our setting is outperformed by pure RBF kernel, indicate that RBF kernel might be the most appropriate kernel for this dataset.

Codes are in next page

```
In [7]: from sklearn.metrics.pairwise import rbf_kernel

        def create_kernel(x1, x2, lam_1=1, lam_2=1): # train/train or train/test
            # create dense data for x1 (train)
            max_key=np.max([np.max(v.keys()) for v in x1])
            arr=np.zeros( (len(x1), len(max_key) ))
            for row, vec in enumerate(x1):
                for k, v in vec.items():
                    arr[row][k-1]=v
            x1 = np.copy(arr)

            #create dense data for x2
            max_key=np.max([np.max(v.keys()) for v in x2])
            arr=np.zeros( (len(x2), len(max_key) ))
            for row, vec in enumerate(x2):
                for k, v in vec.items():
                    arr[row][k-1]=v
            x2 = np.copy(arr)

            #create a linear kernel matrix
            k_linear = np.zeros( (len(x2), len(x1)) )
            k_linear = np.dot(x2, x1.T)

            #create a RBF kernel matrix
            k_RBF = np.zeros( (len(x2), len(x1)) )
            k_RBF = rbf_kernel(x2, x1, gamma=0.01)

            #create kernel matrix
            k = np.zeros( (len(x2), len(x1)+1) )
            k[:,1:] = lam_1*k_linear + lam_2*k_RBF
            k[:,:1] = np.arange(len(x2))[:,np.newaxis]+1

            kernel = [list(row) for row in k]
            return kernel
```

In [7] , we define the function to compute the kernel

```
In [*]: lam_1_range = [0.2, 0.4, 0.6, 0.8, 1.0]
        lam_2_range = [0.2, 0.4, 0.6, 0.8, 1.0]
        for lam_1 in lam_1_range:
            for lam_2 in lam_2_range:
                if lam_1 == lam_2 and lam_1!= 1: continue
                print("lam_1:%.1f, lam_2:%.1f"%(lam_1, lam_2), end = ' | ')
                kernel = create_kernel(X_train, X_train, lam_1, lam_2)
                model = svm_train(y_train, kernel, '-t 4 -c 3 -g 0.01')
                kernel = create_kernel(X_train, X_test, lam_1, lam_2)
                p_label, p_acc, p_val = svm_predict(y_test, kernel, model)

        lam_1:0.2, lam_2:0.4 | Accuracy = 95.72% (2393/2500) (classification)
        lam_1:0.2, lam_2:0.6 | Accuracy = 95.76% (2394/2500) (classification)
        lam_1:0.2, lam_2:0.8 | Accuracy = 95.92% (2398/2500) (classification)
        lam_1:0.2, lam_2:1.0 | Accuracy = 96% (2400/2500) (classification)
        lam_1:0.4, lam_2:0.2 | Accuracy = 95.12% (2378/2500) (classification)
```

In [8] , we use different weight for linear & RBF to compute kernel, and train & test model according to the computed kernel.

For more code detail, you can visit https://github.com/aa10402tw/Machine-Learning-Implementation/tree/master/HW5/libsvm-3.23/python