ML HW5 Support Vector Machine

資科工碩 0756166 楊牧樺

Please execute step by step.

Part 1- SVM on MNIST dataset

0. Preparation

0.1 Import librarys

In this project, we use LIBSVM (https://www.csie.ntu.edu.tw/~cjlin/libsvm/) library.
piaip's Using (lib)SVM Tutorial (https://www.csie.ntu.edu.tw/~piaip/svm/svm_tutorial.html#format)

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from svmutil import *
```

0.2 Read training and testing data

In [ ]:

```
x_train = np.genfromtxt('X_train.csv', delimiter=',')
y_train = np.genfromtxt('Y_train.csv', delimiter=',')
x_test = np.genfromtxt('X_test.csv', delimiter=',')
y_test = np.genfromtxt('Y_test.csv', delimiter=',')
```

0.3 Transform data to specified format in LIBSVM

LIBSVM use *sparse matrix* to store data.

[label] [index1]:[value1] [index2]:[value2] ...
[label] [index1]:[value1] [index2]:[value2] ...

ex:
label = [1,2]
data = [{1:2,3:1},{3:2,10:1}]

In [ ]:

```
def sparse_matrix(x):
    row = x.shape[0]
    col = x.shape[1]
    idx_offset = 1

    x = [{idx+idx_offset:x[i][idx] \
        for _,idx in np.ndenumerate(np.argwhere(x[i]!=0))} \
        for i in range(x.shape[0])]
    return x
```

In [ ]:

```
X_train=sparse_matrix(x_train)
X_test=sparse_matrix(x_test)
Y_train=list(y_train)
Y_test=list(y_test)
```

0.4 Construct problem according to training data

In [ ]:

```
problem = svm_problem(Y_train, X_train)
```

1. Compare different kernel functions

Use default settings and quiet mode

-t kernel_type : set type of kernel function (default 2)

- 0 - linear
- 1 - polynomial
- 2 - radial basis function

-q : quiet mode (no outputs)

1.1 Train and predict with linear kernel function

K(u,v) = u^T v

In [ ]:

```
model_linear = svm_train(problem,'-t 0 -q')
pred_linear = svm_predict(Y_test,X_test, model_linear)
```

1.2 Train and predict with polynomial kernel function

K(u,v,γ,coef0,d) = $(\gamma * u^T v + coef0)^d$

-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)

In [ ]:

```
model_poly = svm_train(problem,'-t 1 -q')
pred_poly = svm_predict(Y_test,X_test, model_poly)
```

1.3 Train and predict with RBF kernel function

K(u,v,γ) = $exp(-\gamma|u-v|^2)$

-g gamma : set gamma in kernel function (default 1/num_features)

In [ ]:

```
model_RBF = svm_train(problem,'-t 2 -q')
pred_RBF = svm_predict(Y_test,X_test, model_RBF)
```

Compare:
In default settings, the performance is worst when using polynomial kernel function.

2. Use C-SVC

Use [grid search with cross-validation (https://www.jianshu.com/p/55b9f2ea283b)](https://www.jianshu.com/p/55b9f2ea283b)
[LIBSVM學習（六）代碼結構及c-SVC過程 (https://blog.csdn.net/u014772862/article/details/51835192)](https://blog.csdn.net/u014772862/article/details/51835192)

-v n : n-fold cross validation
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)

Discuss:

When C is large, it means that slack has a big influence.
When C is small, it means that slack has little effect.

Practice:

1. Prepare lots of pre-classified (correct) data
2. Split them into several training sets randomly.
3. Train with some arguments and predict other sets of data to calculate the accuracy.
4. Change the arguments and repeat until we get good accuracy.

2.1 Search best parameter for linear kernel

In [ ]:

```
best_param={}
log_c_range = np.arange(-5, 5, dtype=float)
c_range = 10 ** log_c_range
accuracy=[]
parameter=[]
param_str=[]
for c in c_range:
    param = svm_parameter(f'-c {c} -t 0 -v 5 -q')
    param_str.append(param)
    acc = svm_train(problem, param)
    accuracy.append(acc)

print(f'Best cost is 10^{log_c_range[np.argmax(accuracy)]}')
print(f'Best cross validation accuracy is {np.max(accuracy)}%')
best_param['linear']=param_str[np.argmax(accuracy)]
```

Print each step

In [ ]:

```
for i in range(len(c_range)):
    print(f'Cost = 10^{log_c_range[i]}, Cross Validation Accuracy = {accuracy
[i]}%')
```

2.2 Search best parameter for polynomial kernel

Observed:

Searching best parameter for polynomial kernel takes longest execution time because it has to adjust the most parameters.

In [ ]:

```
log_c_range = np.arange(-2, 2, dtype=float)
c_range = 10 ** log_c_range
log_g_range = np.arange(-3, 1, dtype=float)
g_range = 10 ** log_g_range
d_range = np.arange(2, 11, 2)
r_range = np.arange(0, 1)
accuracy=[]
parameter=[]
param_str=[]
best_acc = 0
for c in c_range:
    for g in g_range:
        for d in d_range:
            for r in r_range:
                param = svm_parameter(f'-t 1 -c {c} -g {g} -d {d} -r {r} -v 5 -
q')
                param_str.append(param)
                acc = svm_train(problem, param)
                param_dic={}
                param_dic['cost'] = c
                param_dic['gamma'] = g
                param_dic['degree'] = d
                param_dic['coef0'] = r
                parameter.append(param_dic)
                accuracy.append(acc)

print(f'Best parameter is {parameter[np.argmax(accuracy)]}')
print(f'Best cross validation accuracy is {np.max(accuracy)}%')
best_param['polynomial']=param_str[np.argmax(accuracy)]
```

Print each step

In [ ]:

```
for i in range(len(parameter)):
    print(f'Parameter = {parameter[i]}, Cross Validation Accuracy = {accuracy
[i]}%')
```

2.3 Search best parameter for RBF kernel

In [ ]:

```
log_c_range = np.arange(-5, 5, dtype=float)
c_range = 10 ** log_c_range
log_g_range = np.arange(-3, 1, dtype=float)
g_range = 10 ** log_g_range
accuracy=[]
parameter=[]
param_str=[]
for c in c_range:
    for g in g_range:
        param = svm_parameter(f'-t 1 -c {c} -g {g} -v 5 -q')
        param_str.append(param)
        acc = svm_train(problem, param)
        param_dic={}
        param_dic['cost'] = c
        param_dic['gamma'] = g
        parameter.append(param_dic)
        accuracy.append(acc)

print(f'Best parameter is {parameter[np.argmax(accuracy)]}')
print(f'Best cross validation accuracy is {np.max(accuracy)}%')
best_param['RBF']=param_str[np.argmax(accuracy)]
```

Print each step

In [ ]:

```
for i in range(len(parameter)):
    print(f'Parameter = {parameter[i]}, Cross Validation Accuracy = {accuracy
[i]}%')
```

3. Use linear+RBF kernel

Use the precomputed kernel feature provided by libsvm.

3.1 Define linear + RBF kernel function

Creat a linear_design_x and a rbf_design_x using formulas of 1.1 and 1.3 and combine them.

In [ ]:

```
def linear_RBF_kernel(u,v):
    gamma=0.01
    linear_design_x = np.dot(u,v.T)
    rbf_design_x = np.sum(u**2, axis=1)[:,None] \
                   + np.sum(v**2, axis=1)[None,:] - 2*linear_design_x
    rbf_design_x= np.abs(rbf_design_x) * (-gamma)
    rbf_design_x = np.exp(rbf_design_x)
    design_x = linear_design_x + rbf_design_x
    return design_x
```

3.2 Prepare training data

Assume there are L training instances x1, ..., xL.
Let K(x, y) be the kernel.
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)

That is, in the training file the first column must be the "ID" of xi. In testing, ? can be any value.

In [ ]:

```
def precomputed_sparse_matrix(x):
    row = x.shape[0]
    col = x.shape[1]
    idx_offset = 0
    x = np.append(np.linspace(1,row,row), x)
    x = x.reshape(col+1,row).T
    x = [{idx+idx_offset:x[i][idx] \
        for _,idx in np.ndenumerate(np.argwhere(x[i]!=0))} \
        for i in range(x.shape[0])]
    return x
```

Convert to precomputed data.

In [ ]:

```
X_train_precomputed = linear_RBF_kernel(x_train,x_train)
X_train_precomputed = precomputed_sparse_matrix(X_train_precomputed)
X_test_precomputed = linear_RBF_kernel(x_test,x_train)
X_test_precomputed = precomputed_sparse_matrix(X_test_precomputed)
```

3.3 Construct problem according to training data

problem option : isKernel = True

In [ ]:

```
problem_precomputed = svm_problem(Y_train, X_train_precomputed, isKernel=True)
```

3.4 Search best parameter for linear + RBF kernel

train option : -t 4 precomputed kernel (kernel values in training_set_file)

In [ ]:

```
log_c_range = np.arange(-6, 6, dtype=float)
c_range = 10 ** log_c_range
accuracy=[]
param_str=[]
for c in c_range:
    param = svm_parameter(f'-c {c} -t 4 -v 5 -q')
    param_str.append(param)
    acc = svm_train(problem_precomputed, param)
    accuracy.append(acc)

print(f'Best cost is 10^{log_c_range[np.argmax(accuracy)]}')
print(f'Best cross validation accuracy is {np.max(accuracy)}%')
best_param['linear + RBF']=param_str[np.argmax(accuracy)]
```

In [ ]:

```
for i in range(len(c_range)):
    print(f'Cost = 10^{log_c_range[i]}, Cross Validation Accuracy = {accuracy
[i]}%')
```

4. compare four kernels

Show the performance with the best parameters obtained earlier of four kernels.

In [ ]:

```
for kernel,param in best_param.items():
    if kernel == 'linear + RBF':
        acc = svm_train(problem_precomputed, param)
        print(f'kernel type : {kernel} , accuraccy : {acc}%')

    else:
        acc = svm_train(problem, param)
        print(f'kernel type : {kernel} , accuraccy : {acc}%')
```

Part 2- Find out support vectors

0. Preparation

0.1 Read training data

In [ ]:

```
x_train = np.genfromtxt('Plot_X.csv', delimiter=',')
y_train = np.genfromtxt('Plot_Y.csv', delimiter=',')
```

In [ ]:

```
x=x_train[:,0]
y=x_train[:,1]
```

0.2 Transform data to specified format in LIBSVM

In [ ]:

```
X_train=sparse_matrix(x_train)
Y_train=list(y_train)
```

In [ ]:

```
X_train_precomputed = linear_RBF_kernel(x_train,x_train)
X_train_precomputed = precomputed_sparse_matrix(X_train_precomputed)
```

0.3 Construct problem according to training data

In [ ]:

```
problem = svm_problem(Y_train, X_train)
```

In [ ]:

```
problem_precomputed = svm_problem(Y_train, X_train_precomputed, isKernel=True)
```

1. SVM model with linear kernel function

1.1 Train with linear kernel function

In [ ]:

```
model_linear = svm_train(problem,'-t 0 -q')
```

1.2 Get support vectors

In [ ]:

```
SV=model_linear.get_SV()
```

In [ ]:

```
SV_x = [dic[1] for dic in SV]
SV_y = [dic[2] for dic in SV]
```

1.3 Visualization

Convert label to the corresponding color in advance.

In [ ]:

```
label=np.array(list(map(str,Y_train)))
label[label=='0.0']='b'
label[label=='1.0']='g'
label[label=='2.0']='y'
```

In [ ]:

```
plt.figure()
plt.scatter(x, y,c=list(label),alpha=0.5,s=10)
plt.scatter(SV_x, SV_y,c='r',alpha=0.5,s=15,marker='X')
plt.show()
```

2. SVM model with polynomial kernel function

2.1 Train with polynomial kernel function

In [ ]:

```
model_poly = svm_train(problem,'-t 1 -q')
```

2.2 Get support vectors

In [ ]:

```
SV=model_poly.get_SV()
```

In [ ]:

```
SV_x = [dic[1] for dic in SV]
SV_y = [dic[2] for dic in SV]
```

2.3 Visualization

In [ ]:

```
plt.figure()
plt.scatter(x, y,c=list(label),alpha=0.5,s=10)
plt.scatter(SV_x, SV_y,c='r',alpha=0.5,s=15,marker='X')
plt.show()
```

3. SVM model with RBF kernel function

3.1 Train with RBF kernel function

Observed:

The effect of this method is closely related to γ.

In [ ]:

```
model_RBF = svm_train(problem,'-t 2 -g 0.01 -q')
```

3.2 Get support vectors

In [ ]:

```
SV=model_RBF.get_SV()
```

In [ ]:

```
SV_x = [dic[1] for dic in SV]
SV_y = [dic[2] for dic in SV]
```

3.3 Visualization

In [ ]:

```
plt.figure()
plt.scatter(x, y,c=list(label),alpha=0.5,s=10)
plt.scatter(SV_x, SV_y,c='r',alpha=0.5,s=15,marker='X')
plt.show()
```

4. SVM model with linear+RBF kernel function

4.1 Train and predic with linear+RBF kernel function

In [ ]:

```
model_linear_RBF = svm_train(problem_precomputed,'-t 4 -q')
```

4.2 Get support vectors

In [ ]:

```
SV_index=np.array(model_linear_RBF.get_sv_indices()) #get index
```

In [ ]:

```
SV_x = x_train[SV_index-1,0]
SV_y = x_train[SV_index-1,1]
```

4.3 Visualization

In [ ]:

```
plt.figure()
plt.scatter(x, y,c=list(label),alpha=0.5,s=10)
plt.scatter(SV_x, SV_y,c='r',alpha=0.5,s=15,marker='X')
plt.show()
```