CSE 5526 Programming Assignment 3

Summary Report

In this project, I used a popular open-source SVM toolbox LIBSVM with Python to identify whether a genomic sequence is an ncRNA.

I first trained a set of linear SVMs with different values of the parameter C using the organized LIBSVM format. I trained SVMs for each c in sequence C=(2^{-4}, 2^{-3}, 2^{-2} …, 2^{7}, 2^{8}) and conduct classification based on the trained SVMs on the test dataset to figure out the changing trend of accuracy with respect to different C.

The results can be seen in the following table.

Table 1. Experiment results for different value of C

Value of C	Classification Accuracy
2^{-4}	66.4336%
2^{-3}	66.4336%
2^{-2}	66.4336%
2^{-1}	77.8222%
2^{0}	92.5075%
2^{1}	94.006%
2^{2}	93.7063%
2^{3}	93.8062%
2^{4}	93.8062%
2^{5}	93.8062%
2^{6}	93.8062%
2^{7}	93.8062%
2^{8}	93.8062%

Classification Accuracy can be plotted in the following figure.

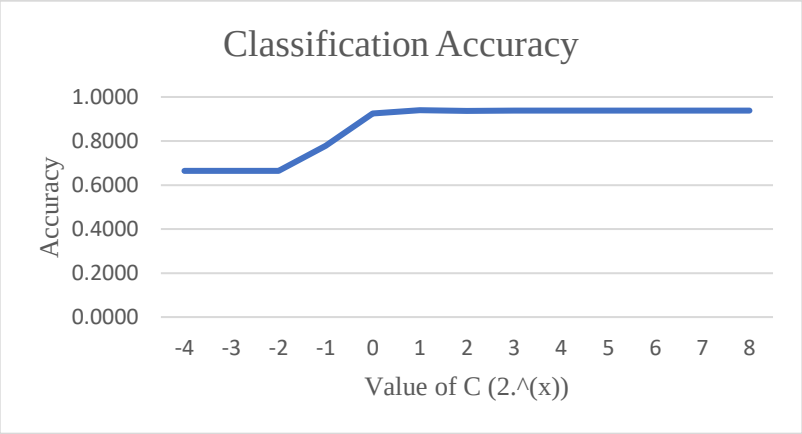

Fig.1 Classification accuracy for different value of C

We can see from Table 1 and Fig 1 that the classification accuracy increases during the small value of C. When value of C equals 2^1, the accuracy reaches the largest. And after this, classification decreases a little and then keep the same. Parameter C controls the tradeoff between minimizing the classification error and maximizing the margin of separation. So, in our case, 2^1 is the best value of C.

Secondly, I also trained RBF kernel SVMs using 5-fold cross validation to choose the best C and α as described in our instruction. I tried the values for both C and α in the following exponential steps: (2^{-4}, 2^{-3}, 2^{-2} …, 2^7, 2^8). For each of the combination of C and α, I got a cross validation result. The results can form a cross validation matrix.

The matrix of cross validation results is shown below:

[0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.7050.705
0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.706 0.706 0.705 0.705
0.705 0.705 0.705 0.705 0.705 0.705 0.705 0.72 0.758 0.759 0.741 0.708 0.705
0.705 0.705 0.705 0.705 0.705 0.709 0.777 0.81 0.808 0.795 0.779 0.753 0.711
0.705 0.705 0.705 0.705 0.755 0.877 0.892 0.885 0.868 0.835 0.806 0.772 0.744
0.705 0.705 0.707 0.814 0.916 0.934 0.926 0.92 0.896 0.854 0.816 0.791 0.761
0.705 0.708 0.846 0.932 0.944 0.939 0.933 0.923 0.9 0.855 0.815 0.791 0.761
0.709 0.86 0.934 0.945 0.942 0.939 0.931 0.922 0.897 0.865 0.819 0.788 0.761
0.865 0.938 0.946 0.945 0.944 0.94 0.931 0.919 0.899 0.858 0.818 0.788 0.761
0.94 0.947 0.946 0.944 0.941 0.94 0.933 0.915 0.896 0.851 0.82 0.788 0.761
0.947 0.948 0.946 0.945 0.941 0.933 0.929 0.912 0.895 0.852 0.82 0.789 0.761
0.949 0.946 0.946 0.945 0.939 0.929 0.922 0.903 0.886 0.849 0.82 0.789 0.761
0.945 0.948 0.942 0.94 0.936 0.933 0.911 0.904 0.877 0.849 0.82 0.79 0.761]

The highest classification accuracy I got is the red number in the matrix, which corresponds to **C=128** and **α=2^{-4}**. Then we used the best values of C and α to train the RBF kernel SVM and used the trained SVM to classify the test set. The final classification accuracy on test set is 94.2058% (943/1001).

Comparing the two parts of this projects, we can found that the classification accuracy we got using cross validation to choose best C and α values in RBF kernel SVM is higher than the highest accuracy in linear SVM. Cross validation is an effective method to select parameters in SVM.

From this project, I learned how to use the popular open-source SVM toolbox in Python to perform SVM classification with both linear and RBF kernel SVM algorithms and I think now I have a better understanding of SVM.

Source Code:

```
# CSE 5526 Programming Assignment 3 SVM #
# There are two parts in this project
# When you execute part 1, you can comment part 2 and vice versa
from libsvm.python.svm import *
from libsvm.python.svmutil import *
import random
import numpy as np

y_train, x_train = svm_read_problem('ncrna_s.train.txt')
num_train=len(y_train)
y_test, x_test = svm_read_problem('ncrna_s.test.txt')
num_test=len(y_test)
print('training set:',num_train)
print('testing set:',num_test)
c_list=[2**(i-4) for i in range(0,13)]
alpha_list=[2**(i-4) for i in range(0,13)]

# Part 1: Classification using linear SVMs
prob    = svm_problem(y_train, x_train)
for c in c_list:
    print('value of c is: ',c)
    param = svm_parameter('-t 0 -h 0 -c '+str(c))
    m = svm_train(prob, param)
    p_label, p_acc, p_val = svm_predict(y_test, x_test, m)

# Part 2: Classification using RBF kernel SVM

def calculate_acc(cv_train,cv_test,param,num_subset):

    y_cv_train=[cv_train[i][0] for i in range(num_subset*4)]
    x_cv_train=[cv_train[i][1] for i in range(num_subset*4)]

    y_cv_test=[cv_test[i][0] for i in range(num_subset)]
    x_cv_test=[cv_test[i][1] for i in range(num_subset)]
    prob = svm_problem(y_cv_train,x_cv_train)
    m = svm_train(prob, param)
    p_label, p_acc, p_val = svm_predict(y_cv_test, x_cv_test, m)
    return p_acc[0]/100

data_pair=[(y_train[i],x_train[i]) for i in range(num_train)]
random.shuffle(data_pair)
num_cv=int(num_train/2)
CV_set=data_pair[:num_cv]
```

```
num_subset=int(num_cv/5)
CV_set_y=CV_set
CV_set_1=CV_set[:num_subset]
CV_set_2=CV_set[num_subset:num_subset*2]
CV_set_3=CV_set[num_subset*2:num_subset*3]
CV_set_4=CV_set[num_subset*3:num_subset*4]
CV_set_5=CV_set[num_subset*4:num_subset*5]
ave_acc_mat=[]
for c in c_list:
    ave_acc_list=[]
    for alpha in alpha_list:
        param = svm_parameter('-t 2 -h 0 -g '+str(alpha)+' -c '+str(c))
        acc_list=[]
        #set 1
        cv_train=CV_set_2+CV_set_3+ CV_set_4 + CV_set_5
        cv_test=CV_set_1
        acc1=calculate_acc(cv_train,cv_test,param,num_subset)
        acc_list.append(acc1)
        #set 2
        cv_train=CV_set_1+CV_set_3+ CV_set_4 + CV_set_5
        cv_test=CV_set_2
        acc1=calculate_acc(cv_train,cv_test,param,num_subset)
        acc_list.append(acc1)
        #set 3
        cv_train=CV_set_2+CV_set_1+ CV_set_4 + CV_set_5
        cv_test=CV_set_3
        acc1=calculate_acc(cv_train,cv_test,param,num_subset)
        acc_list.append(acc1)
        #set 4
        cv_train=CV_set_2+CV_set_3+ CV_set_1 + CV_set_5
        cv_test=CV_set_4
        acc1=calculate_acc(cv_train,cv_test,param,num_subset)
        acc_list.append(acc1)
        #set 5
        cv_train=CV_set_2+CV_set_3+ CV_set_4 + CV_set_1
        cv_test=CV_set_5
        acc1=calculate_acc(cv_train,cv_test,param,num_subset)
        acc_list.append(acc1)
        ave_acc=sum(acc_list)/len(acc_list)
        ave_acc_list.append(ave_acc)
    ave_acc_mat.append(ave_acc_list)

print('\n got the index\n')
acc_mat=np.array(ave_acc_mat).reshape(13,13)
```

```
print(acc_mat)
index=np.argmax(acc_mat)
row=index//13
column=index%13
C=c_list[row]
alpha=alpha_list[column]
print('\n predict with the whole training set\n')
prob    = svm_problem(y_train, x_train)
param = svm_parameter('-t 2 -h 0 -g '+str(alpha)+' -c '+str(c))
m = svm_train(prob, param)
p_label, p_acc, p_val = svm_predict(y_test, x_test, m)
```