HW1

September 30, 2018

```
In [1]: # numpy is the very basic package in python
        # sklearn contain the original SVM
        import numpy as np
        from sklearn import svm
        import time
```

```
In [2]: # fix the seed
        np.random.seed(1)
        ## generate the dataset
        # the dimension and size of the data
        p = 10
        n_train = 2000
        n_test = 200


        # two distribution we sample data from
        mu1 = np.repeat(0.5,p)
        sigma1 = np.eye(p)
        mu2 = np.repeat(-0.5,p)
        sigma2 = np.eye(p)

        # construct the training dataset
        train_x1 = np.random.multivariate_normal(mu1,sigma1,n_train)
        train_y1 = np.repeat(1,n_train)

        train_x2 = np.random.multivariate_normal(mu2,sigma2,n_train)
        train_y2 = np.repeat(-1,n_train)

        train_x = np.vstack((train_x1,train_x2))
        train_y = np.hstack((train_y1,train_y2))

        print('the size of the training dataset is:',train_x.shape)
        print(train_y.shape)

        # constructing the testing dataset
        test_x1 = np.random.multivariate_normal(mu1,sigma1,n_test)
        test_y1 = np.repeat(1,n_test)
```

```
test_x2 = np.random.multivariate_normal(mu2,sigma2,n_test)
test_y2 = np.repeat(-1,n_test)

test_x = np.vstack((test_x1,test_x2))
test_y = np.hstack((test_y1,test_y2))

print('the size of the testing dataset is:',test_x.shape)
print(test_y.shape)
```

```
the size of the training dataset is: (4000, 10)
(4000,)
the size of the testing dataset is: (400, 10)
(400,)
```

1 T1

```
In [3]: ## T1
        # construct SVM from sklearn package
        time_start = time.time()
        model = svm.SVC(kernel="rbf")
        print(model.fit(train_x,train_y))
        pred = model.predict(test_x)
        # compute the accuracy of our SVM model
        accu = sum(pred == test_y)/len(test_y)
        print("Accuracy of SVM:",accu)
        # compute the time epoch
        print("Time for performance SVM:",time.time()-time_start,"s")
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Accuracy of SVM: 0.935
Time for performance SVM: 0.13187074661254883 s
```

2 T2

```
In [17]: ## T2
         # adding column one at the front of the matrix
         add_train = np.repeat(1,2*n_train).reshape(-1,1)
         arg_train_x = np.hstack((add_train,train_x))

         add_test = np.repeat(1,2*n_test).reshape(-1,1)
         arg_test_x = np.hstack((add_test,test_x))
```

```python
print("argumented dimension after adding bias column:\n"
      ,arg_train_x.shape,arg_test_x.shape)
# hyper-parameter settings
C = 1
beta = np.repeat(0.1,p)
MAX_L = 10000
# step size
mu = 0.001

# mapping function and gradient function
def mapping(x):
    mapping = x
    return mapping
def gradient(x,y,beta):
    indicator = y*x@beta - 1
    if indicator<0:
        g = -y*x

    else:
        g = 0
    return g
# start counting time for SGD
time_start_sgd = time.time()
# one SGD sample approach. Training stage of the model
for i in range(MAX_L):
    index = np.random.randint(4000)
    x = train_x[index,]
    y = train_y[index]
    z = beta - mu * gradient(x,y,beta)
    # introduce the stopping criteria,
    # meanwhile gaurantee the loop goes for a while
    # because we may sample a point which not change the beta
    if i > 1000:
        if max(z-beta) < 0.0001:
            print(i,'iteration: Optimal find')
            break
    beta = z
print("beta:\n",beta)

# prediction stage of the model,
# assess the performance in terms of accuracy and time consumption
pred_SGD = list()
for i in range(2*n_test):
    if test_x[i,]@beta > 0:
        pred_SGD.append(1)
    else:
        pred_SGD.append(-1)
# print(np.array(pred_SGD))
```

```
    # print(test_y)
    accu = sum(np.array(pred_SGD) == test_y)/len(test_y)
    print("Accuracy of SGD:",accu)
    # compute the time epoch
    print("Time for performance SGD:",time.time()-time_start_sgd,"s")
```

```
argumented dimension after adding bias column:
 (4000, 11) (400, 11)
1002 iteration: Optimal find
beta:
 [0.25755508 0.26298576 0.23128091 0.25647379 0.25701372 0.21696959
 0.27952904 0.26042558 0.25392188 0.22442451]
Accuracy of SGD: 0.9425
Time for performance SGD: 0.017499923706054688 s
```

3 compare & comment

3.1 compare:

3.1.1 the time taken by SGD case is much shorter than the original SVM model. simply because we take in only one data point at a time, the time complexity is $O(iteration)$ but for the SVM case the time complexity is $O(n)$ so there is a time boost in SGD case.

3.1.2 Meanwhile, we can see that the performance of SGD is about the same or a little higher than the SVM package in sklearn under the same hyperparameter C.

3.2 comment:

3.2.1 SGD method is a good way to implement SVM, because of its time complexity and accuracy performance.

3.2.2 we can find out that in the SGD case. To simplify the problem and reduce the computational complexity, I did not use the mapping h(x). I directly use the x. Maybe a proper h(x) can boost the performance of SGD again.

3.2.3 Higher performance maybe due to the reason that SGD is actually not finding the optimal of the original problem rather a ball around the optimal. So, this means when the case happens that the original SVM with the hyperparameter which is overfitting. The SGD is like a regulation to the optimal problem. Thus having a higher performance in the testing set.