ml2022 lab2

PB21000302 Zhang Cheng

October 28, 2022

1 Introduction

Lab2 focus on the realization of SVM in two ways. In short I used SMO to solve this and a method that is similar to gradient descent also known as support vector regression.

Now I'm gonna talk about the principle of these methods briefly. To realize SVM, we need to turn the problem into its dual, as this problem is convex, the answer to its dual is exactly its answer.SMO solves the dual problem by searching for the two parameters that violate the KKT conditions and update them to minimize the error. Because this problem is only quadratic, this is easy. However, in many problems(like this one), the dataset is not linear separable, therefore here comes soft-margin SVM the only difference is that it's solution(parameter $\boldsymbol{\alpha}$) is smaller than a given C ,KKT is changed together.

Another way is support vector regression, it is a regression with a linear kernel and a ϵ tolerent region.

2 Result

In general the two methods mentioned above did differently in this task, we can see from the picture the test I select can generally illustrate that the svr is the fastest in all algorithms and svm provided in sklearn is the most accurate one,its error rate is close to the average mislabel rate while my svr is about 3% higher and smo is 8% higher

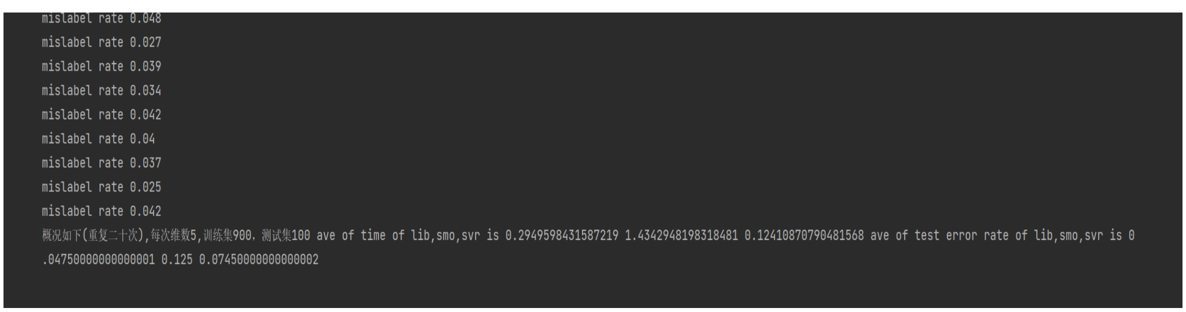

```
mislabel rate 0.048
mislabel rate 0.027
mislabel rate 0.039
mislabel rate 0.034
mislabel rate 0.042
mislabel rate 0.04
mislabel rate 0.037
mislabel rate 0.025
mislabel rate 0.042
概况如下(重复二十次),每次维数5,训练集900，测试集100 ave of time of lib,smo,svr is 0.2949598431587219 1.4342948198318481 0.12410870790481568 ave of test error rate of lib,smo,svr is 0
.04750000000000001 0.125 0.07450000000000002
```

Figure 1: baseline result num=900 dim=5

now I will analysis the problems of my svr and smo. First I will talk about the success of my svr algorithm in time. First is the vectorization

```
J=0
grad=np.zeros(self.dim)
```

```
gradb=0
for i in range(self.num):
    t=np.dot(self.theta,X[i,:])+self.b-y[i]
    if t*y[i]<-ep and y[i]>0:
        J=J+abs(t)
        grad=grad-lr*X[i,:]*y[i]
        gradb=gradb+lr*y[i]
```

this is my function for computing loss and gradient with only one cycle, which accelerates e computation a lot. what's more I found this is a convex problem in a linear restriction which has only one solution, in order to fit this problem fast (with relatively big learning rate) and accurately, I let the program to stop if the loss is going up when the optimization has gone for a while.

```
if (iter>100) and (J>J_his[-1]):
        break
```

Because the dataset in this problem is not linear separable and the gradient always exist, this method can prevent our parameter to deviate due to the gradients from noise data.

now that we have seen the performance of svr in this task, I will now look into my smo. In short, I modified the choosing of index in often used SMO(choosing the largest error one in data violate KKT as the first and choose another that is most distant to it) to update every data that violates KKT and randomly choose the second index

```
for i in range(self.num):
    if (self.meet_kkt(i,y[i],X[i,:])==0):
    ......

def find_j(self, best_i,X):
    l = list(range(len(self.a)))
    seq = l[: best_i] + l[best_i + 1:]
    best_j = random.choice(seq)
return best_j
```

At first I select the point that violate KKT most(α_1) and the point that is most distant to itα_2 to update $\boldsymbol{\alpha}$, but this may be stuct by the situation that after one change α_1 is still the one that violate KKT most and the update is ended. The result of the first algorithm is that it cost a lot of time and get a very bad result. So I chose to traverse the dataset and update every α_1 that violate KKT, as for α_2, I experimented many methods on it, such as choose the most distant one or choose the one whose error derives most from α_1 and random choosing, the accuracy is approximately the same, so I chose randomly picking α_2 to save time.

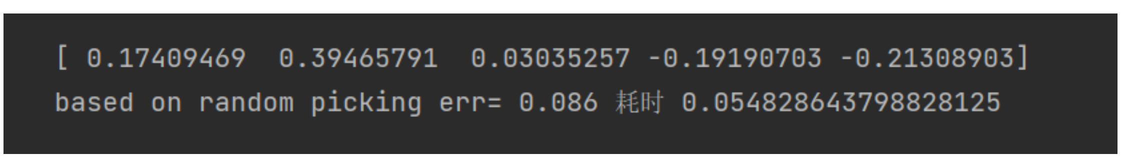
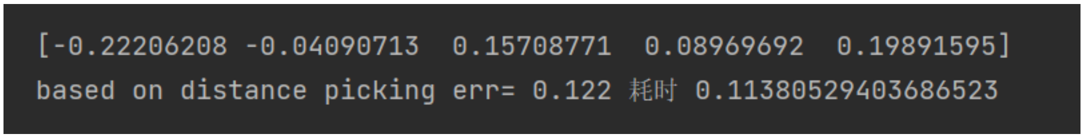


Figure 2: random



```
[-0.22206208 -0.04090713  0.15708771  0.08969692  0.19891595]
based on distance picking err= 0.122 耗时 0.11380529403686523
```

Figure 3: distance

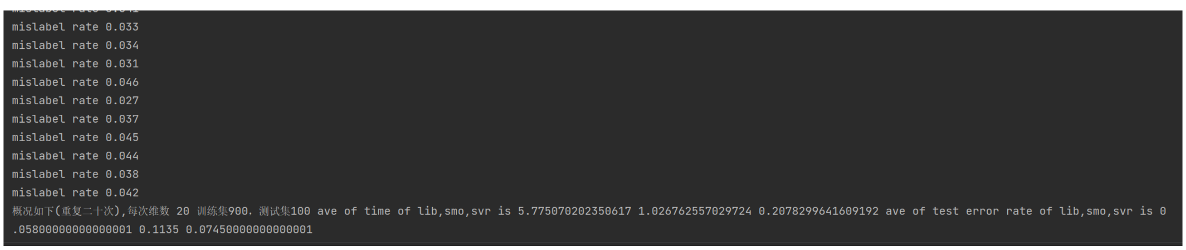
```
[ 0.1913572  -0.12054824 -0.29774492 -0.05352127  0.15156149]
based on error picking err= 0.132 耗时 0.10028958320617676
```

Figure 4: error

You may worry about whether this algorithm will converge very slowly because every update it traverse the whole dataset. Actually because this algorithm update every α_1 violates KKT we need less epochs to optimize our vector,let n=num of training sample, d=dim of vector we can have approximately n updates every epoch, while if we choose to update α_1 that violates KKT most, we only get ONE update every epoch, as loop structure costs a lot of time in python, this will reduce training time to $\frac{1}{n}$ and it is stable with d.

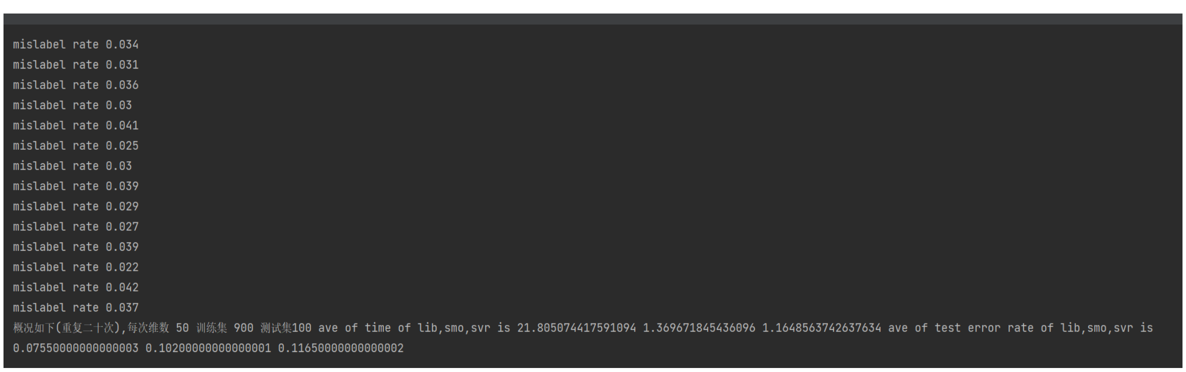
3 comparison

In this section I will use different combination of size of training set and dimension of data to compare three algorithms I will compare the base line given before ((dim,num)= (5,900)) and more dims(dim=20 and 50) and larger training set (num=9000, 90000)



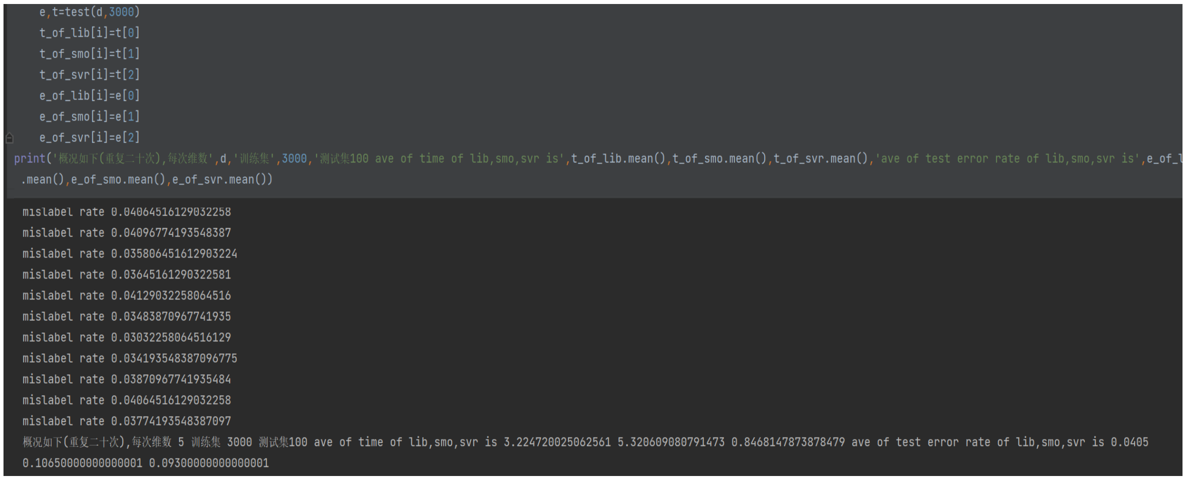
```
mislabel rate 0.033
mislabel rate 0.034
mislabel rate 0.031
mislabel rate 0.046
mislabel rate 0.027
mislabel rate 0.037
mislabel rate 0.045
mislabel rate 0.044
mislabel rate 0.038
mislabel rate 0.042
概况如下(重复二十次),每次维数 20 训练集900. 测试集100 ave of time of lib,smo,svr is 5.775070202350617 1.026762557029724 0.2078299641609192 ave of test error rate of lib,smo,svr is 0
.05800000000000001 0.1135 0.07450000000000001
```

Figure 5: d=20



```
mislabel rate 0.034
mislabel rate 0.031
mislabel rate 0.036
mislabel rate 0.03
mislabel rate 0.041
mislabel rate 0.025
mislabel rate 0.03
mislabel rate 0.039
mislabel rate 0.029
mislabel rate 0.027
mislabel rate 0.039
mislabel rate 0.022
mislabel rate 0.042
mislabel rate 0.037
概况如下(重复二十次),每次维数 50 训练集 900 测试集100 ave of time of lib,smo,svr is 21.805074417591094 1.369671845436096 1.1648563742637634 ave of test error rate of lib,smo,svr is
0.07550000000000003 0.10200000000000001 0.11650000000000002
```

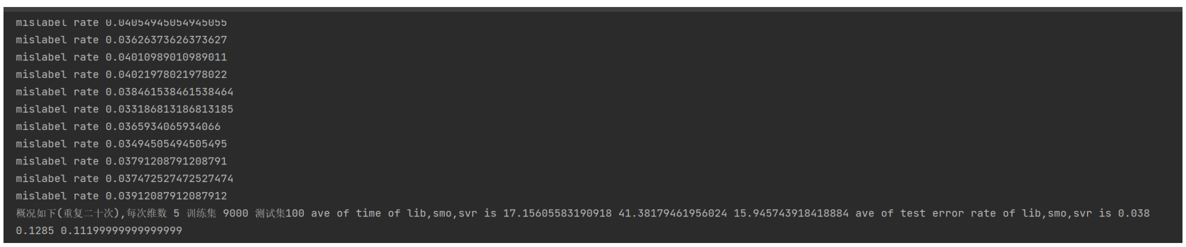
Figure 6: d=50



```
    e,t=test(d,3000)
    t_of_lib[i]=t[0]
    t_of_smo[i]=t[1]
    t_of_svr[i]=t[2]
    e_of_lib[i]=e[0]
    e_of_smo[i]=e[1]
    e_of_svr[i]=e[2]
print('概况如下(重复二十次),每次维数',d,'训练集',3000,'测试集100 ave of time of lib,smo,svr is',t_of_lib.mean(),t_of_smo.mean(),t_of_svr.mean(),'ave of test error rate of lib,smo,svr is',e_of_l
 .mean(),e_of_smo.mean(),e_of_svr.mean())
```

```
mislabel rate 0.04064516129032258
mislabel rate 0.04096774193548387
mislabel rate 0.035806451612903224
mislabel rate 0.03645161290322581
mislabel rate 0.04129032258064516
mislabel rate 0.03483870967741935
mislabel rate 0.03032258064516129
mislabel rate 0.034193548387096775
mislabel rate 0.03870967741935484
mislabel rate 0.04064516129032258
mislabel rate 0.03774193548387097
概况如下(重复二十次),每次维数 5 训练集 3000 测试集100 ave of time of lib,smo,svr is 3.224720025062561 5.320609080791473 0.8468147873878479 ave of test error rate of lib,smo,svr is 0.0405
0.10650000000000001 0.09300000000000001
```

Figure 7: n=3000



```
mislabel rate 0.04054945054945055
mislabel rate 0.03626373626373627
mislabel rate 0.04010989010989011
mislabel rate 0.04021978021978022
mislabel rate 0.038461538461538464
mislabel rate 0.033186813186813185
mislabel rate 0.0365934065934066
mislabel rate 0.03494505494505495
mislabel rate 0.03791208791208791
mislabel rate 0.037472527472527474
mislabel rate 0.03912087912087912
概况如下(重复二十次),每次维数 5 训练集 9000 测试集100 ave of time of lib,smo,svr is 17.15605583190918 41.38179461956024 15.945743918418884 ave of test error rate of lib,smo,svr is 0.038
0.1285 0.11199999999999999
```

Figure 8: n=9000

Table 1: Sum Up of Performance.

sample	time of sklearn	time of SMO	time of SVR	error of sklearn	error of SMO	error of SVR
n=900,dim=5	0.295	1.434	0.124	0.0475	0.125	0.0745
n=3000,dim=5	3.224	5.321	0.847	0.0405	0.107	0.093
n=9000,dim=5	17.16	41.381	15.945	0.038	0.1285	0.112
n=900,dim=20	5.775	1.027	0.208	0.058	0.1135	0.0745
n=900,dim=50	21.805	1.370	1.164	0.0755	0.102	0.117

The result shows that dimension has little to do with SMO 's error rate and didn't change but the error rate of sklearn and SVR increased slightly. Time consumption of svr and smo increase in proportion with size of training set, but time consumption of sklearn didn't change.

When it comes to more dimension, time consumption of SKlearn increased a lot while time consumption of svr and smo increase in proportion.

the result shows that SVM from sklearn is not so suitable for dataset with more dimension as its performance decreases with the number of dimensions.

Also, SVR is not suitable for large dataset because it is not only slower but also less accurate.

SMO has a stable but mediocre performance in every task.