Assignment 1- yumin cao a1754926

Basic of SVM and the primal&dual form of hard margin
Giving the training dataset D = {(X1, y1),...,(Xn, yn)}, in which Xi is the vector representing the features of each instance and yi is the value of +1 or -1 for binary classification. The basic idea of support vector classification is to find a **hyperplane** that can seperate the samples of different categories.
Generalisation is important, because we want the model performs stable and good not only on training data but also on test data. For a better **generalisation**, the location of that hyperplane should be as far as possible to both of the two classes' samples to get 'tolerance' ability for the test data, because the training data we using may has limitation in data distribution and noise.
The hyperplane can be described as the following equation:

$$W^T X + b = 0$$

In this equation, W = (W1,W2,...,Wd) and is the normal vector of the hyperplane which determines the direction of it. And b is the displacement determining the distance between the hyperplane and the origin.
Then we can describe the distance between an arbitrary point in the sample space and the hyperplane with the following equation:

$$r = \frac{W^T X + b}{||W||}$$

||W|| is the l2 form. Supporting the hyperplane can seperate the samples correctly, then if y = +1, we have

$$W^T X_i + b > 0$$

If y = -1 , we have $W^T X_i + b < 0$
Then we have $y_i \left(W^T x_i + b\right) \geq 1$

Support vectors are the points which are **nearest** to the hyperplane to make the previous equation true. Which means that equation equals to 1 given the constraint min|W.T Xi +b|=1.
The sum of the two different-class support vectors' distance (vertical l2 distance after **projection**) to the hyperplane (usually named **max margin**) is the equation:

$$r = \frac{2}{||W||}$$

Then our purpose is to maximum that r, which equals to:

$$min \frac{1}{2} ||W||^2 \ \ s.t. \ y_i \left(W^T x_i + b\right) \geq 1$$

This is the basic form of SVM, also called the **primal form** of hard margin. This function is a **convex quadratic programming** problem, while we can use the **Lagrange multipliers** to optimise this problem. Lagrange multipliers is a method that seeking the extremum of multi function under constraints. It can convert the d-variables & k-constraints problems to d+k unconstraint problem by seeking the minimal point on the d-1 dimension surface which determined by the constraint.

Adding the lagrange multiple alpha to the previous equation we get:

$$L(W, b, \alpha) = \frac{1}{2}||W||^2 + \sum \alpha_i \left(1 - y_i \left(W^T X_i + b\right)\right)$$

After partial derivative calculation we get the **dual form** of hard margin problem.

$$min\frac{1}{2}\sum\sum \alpha_i \alpha_j y_i y_j X_i^T X_j - \sum \alpha_i$$

Kernel function and Linear inseparable
In some cases, the training data is not linear separable, then we need to map the data to a higher dimensional space to make those points separable. We use kernel function to make this possible.

$$K(X_i, X_j) = < \Phi(X_i), \Phi(X_j) >$$

However, it is hard to determine the suitable kernel function and even it becomes separable, we cannot know whether it is overfitting. So we use the soft margin method to relief that problem. In detail, we allow some samples not satisfying the constraint $y_i\left(W^T x_i + b\right) \geq 1$

We use the hinge loss to describe that: $l_{hinge}(z) = max(0, 1 - z)$

This equation means that if a sample in the space is strictly follow the constraint, then it contributes 0 to our **loss function**, else we should record that loss. Introduce **slack variables** representing this hinge loss, as well as C to **punish** this loss, finding a balance to get a better result. Finally we can get our goal function of **primal form**.

$$min\frac{1}{2}||W||^2 + C\sum \xi_i \quad s.t.\ y_i(W^T X_i + b) >= 1 - \xi_i, \xi_i >= 0$$

Similarly, we use the lagrange multipliers to optimise the problem, and we can get the **dual form**:

$$min\frac{1}{2}\sum\sum \alpha_i \alpha_j y_i y_j X_i^T X_j - \sum \alpha_i \qquad s.t. \sum_{i=1}^{m} \alpha_i y_i = 0,\ 0 \leq \alpha_i \leq C$$

The influence to SVM between using kernel and slack variables is different. Although adding the slack variable and punishment C can make the model handle the non-linear separable problem, it is still a linear classification model, but the kernel can transform the hyperplane into curved surface and making the SVM becoming a non-linear classification model.

SMO for calculating the dual problem
The idea of SMO(sequential minimal optimisation) is to fixed the parameters except alpha, and select alpha each time which most violate the KKT (Karush-Kuhn-Tucker) constraint, keeping update the alpha to make the function convergent.
This is a QP problem that has the optimised result. After finding the alpha, we can use the following equation to calculate W and b.

$$w = \sum \alpha_i y_i X_i \qquad b = \frac{1}{|S|}\sum\left(\frac{1}{y_s} - \sum \alpha_i y_i X_i^T X_s\right)$$

Experiments

Processing the data

This code using the csv file, but the csv file doesn't have name of each feature, so I add names of each column. Then, the value in the label is 0 and +1, which is not standard SVM label value, so I replace them with -1 and +1.
Then I use KFold function from sklearn to get the index of our data for cross validation.
At first, I used all the training data as cross validation data, and I trained my model for 3 hours but still can't stop, so I changed the amount of the cross validation dataset.
I only use half of the training data as my cross validation (CV) data, then I split the CV data into 3 pieces. Every time I will use 2/3 of the CV data as training data and the rest 1/3 as testing data to get the accuracy with each parameter. My purpose is, finding the best parameters and avoid my model overfitting.

Class primal and dual

In primal class, I define the train function, predict function and accuracy function. In dual class, I define the train function, predict function, accuracy function and p function (for calculating the P of standard cvxopt QP form).
My derivation is:

Primal :

$$\min \frac{1}{2}w^2 + \frac{c}{N}\sum_i \xi_i \quad \text{s.t. } \xi_i \geq 0,\ y_i(wx_i+b) \geq 1-\xi_i$$

* suppose $x = [b\ w\ \xi]^T$

$$\Rightarrow \min \frac{1}{2}[b\,w\,\xi] \begin{bmatrix} 0 & 0_d^T & 0_N^T \\ 0_d & I_d & \cancel{\ } 0_N^T \\ 0_N & 0_N & \cancel{0}\, I_N \end{bmatrix} [b\,w\,\xi]^T + \frac{c}{N}[0, 0_d^T, 1_N^T][b\,w\,\xi]^T$$

$$\left[\begin{array}{cc|cccc} -y_1 & -y_1x_1^T & 1 & 0 & 0 & \cdots \\ -y_2 & -y_2x_2^T & 0 & 1 & & \vdots \\ \vdots & \vdots & \vdots & & 1 & \vdots \\ -y_N & -y_nx_n^T & \cdots & \cdots & \cdots & 1 \end{array}\right] [b\ w\ \xi]^T \leq \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{bmatrix}$$

then we can get P, q, G, h

dual : $\min \frac{1}{2}\sum\sum \alpha_i\alpha_j y_iy_j x_i^Tx_j - \sum \alpha_i$ suppose $x = \alpha$.

$\Rightarrow \min \frac{1}{2}\alpha^T K \alpha - 1_N^T \alpha.$, where K can be calculated.

$$[y_1, y_2 \cdots y_N]\alpha = 0$$

$$\left.\begin{array}{l} -\alpha_i \leq 0 \\ \alpha_i \leq \frac{c}{N} \end{array}\right\} \Rightarrow \left.\begin{array}{l} -I\cdot\alpha \leq 0 \\ I\alpha \leq \frac{c}{N}. \end{array}\right\} \Rightarrow \begin{bmatrix} -I \\ I \end{bmatrix} \alpha \leq \begin{bmatrix} 0_N \\ (\frac{c}{N})_N \end{bmatrix}$$

then we can get $P, q, G, h, A, b.$

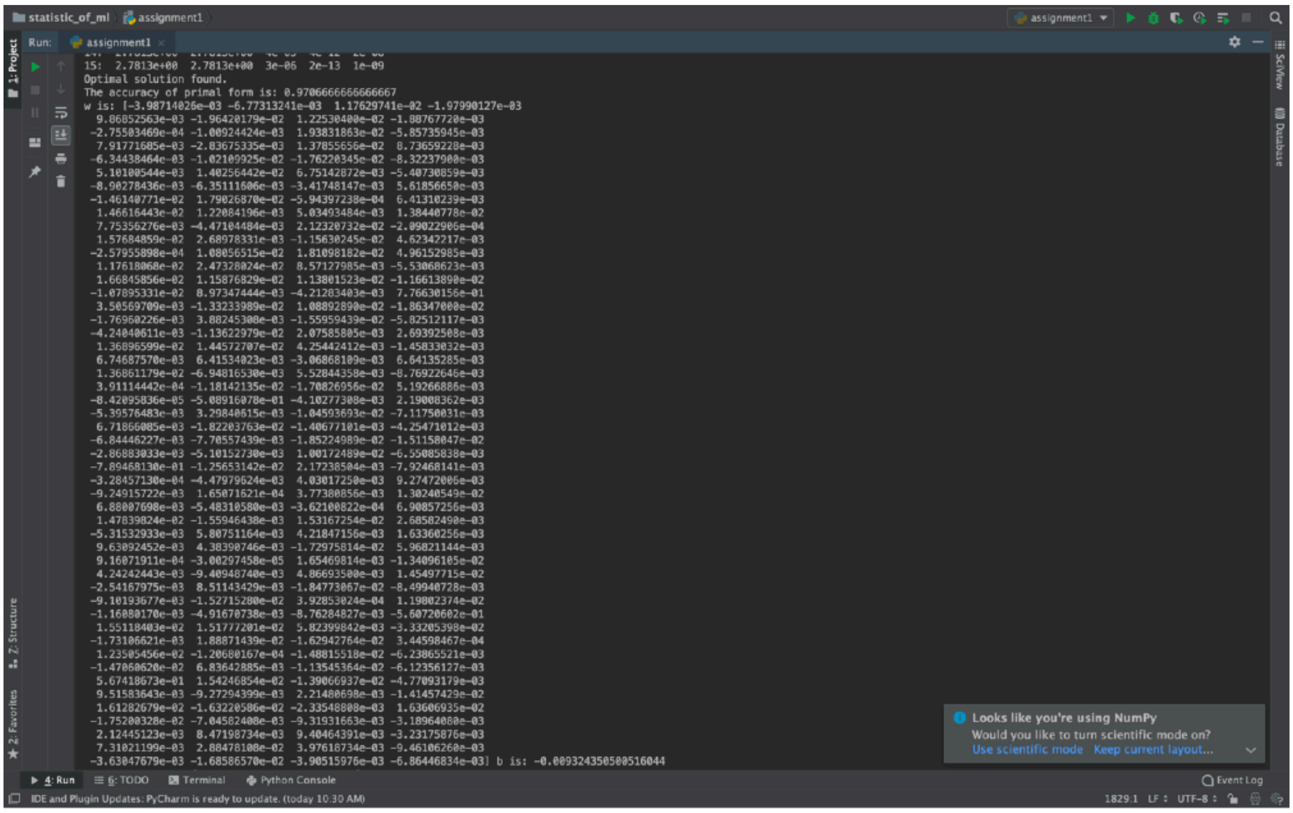
Turning parameters

The parameters we need to turn is C, because it is impossible to try every value between 1 and 100, so at first I try the C value in 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

After finding the best C in these values, then I searched a better C around that C value. Such as, after first choosing, the C is 10, and then I search the value between 2 to 19.

My final result is **C=14**. Although this method may only find the local minimus value, but this method saves a lot of **time and CPU**. And also, I used primal form when finding the best C, because the C are same both in primal form and dual form, so I directly use this C in the final dual form part.

The **w and b (-0.009) as well as accuracy (0.971) of primal form** is:



```
15:  2.7813e+00  2.7813e+00  3e-06  2e-13  1e-09
Optimal solution found.
The accuracy of primal form is: 0.9706666666666667
w is: [-3.98714026e-03 -6.77313241e-03  1.17629741e-02 -1.97990127e-03
  9.86852563e-03 -1.96420179e-02  1.22530400e-02 -1.88767720e-03
 -2.75503469e-04 -1.00924424e-03  1.93831863e-02 -5.85735945e-03
  7.91771685e-03 -2.83675335e-03  1.37855656e-02  8.73659228e-03
 -6.34438464e-03 -1.02109925e-02 -1.76220345e-02 -8.32237900e-03
  5.10100544e-03  1.40256442e-02  6.75142872e-03 -5.40730859e-03
 -8.90278436e-03 -6.35111606e-03 -3.41748147e-03  5.61856650e-03
 -1.46140771e-02  1.79026870e-02 -5.94397238e-04  6.41310239e-03
  1.46616443e-02  1.22084196e-03  5.03493484e-03  1.38440778e-02
  7.75356276e-03 -4.47104484e-03  2.12320732e-02 -2.09022906e-04
  1.57684859e-02  2.68978331e-03 -1.15630245e-02  4.62342217e-03
 -2.57955898e-04  1.08056515e-02  1.81098182e-02  4.96152985e-03
  1.17618068e-02  2.47328024e-02  8.57127985e-03 -5.53068623e-03
  1.66845856e-02  1.15876829e-02  1.13801523e-02 -1.16613890e-02
 -1.07895331e-02  8.97347444e-03 -4.21283403e-03  7.76630156e-01
  3.50569709e-03 -1.33233989e-02  1.08892890e-02 -1.86347000e-02
 -1.76960226e-03  3.88245308e-03 -1.55959439e-02 -5.82512117e-03
 -4.24040611e-03 -1.13622979e-02  2.07585805e-03  2.69392508e-03
  1.36896599e-02  1.44572707e-02  4.25442412e-03 -1.45833032e-03
  6.74687570e-03  6.41534023e-03 -3.06868109e-03  6.64135285e-03
  1.36861179e-02 -6.94816530e-03  5.52844358e-03 -8.76922646e-03
  3.91114442e-04 -1.18142135e-02 -1.70826956e-02  5.19266886e-03
 -8.42095836e-05 -5.08916078e-01 -4.10277308e-03  2.19008362e-03
 -5.39576483e-03  3.29840615e-03 -1.04593693e-02 -7.11750031e-03
  6.71866085e-03 -1.82203763e-02 -1.40677101e-03 -4.25471012e-03
 -6.84446227e-03 -7.70557439e-03 -1.85224989e-02 -1.51158047e-02
 -2.86883033e-03 -5.10152730e-03  1.00172489e-02 -6.55085838e-03
 -7.89468130e-01 -1.25653142e-02  2.17238504e-03 -7.92468141e-03
 -3.28457130e-04 -4.47979624e-03  4.03017250e-03  9.27472006e-03
 -9.24915722e-03  1.65071621e-04  3.77380856e-03  1.30240549e-02
  6.88007698e-03 -5.48310580e-03 -3.62100822e-04  6.90857256e-03
  1.47839824e-02 -1.55946438e-03  1.53167254e-02  2.68582490e-03
 -5.31532933e-03  5.80751164e-03  4.21847156e-03  1.63360256e-03
  9.63092452e-03  4.38390746e-03 -1.72975814e-02  5.96821144e-03
  9.16071911e-04 -3.00297458e-05  1.65469814e-03 -1.34096105e-02
  4.24242443e-03 -9.40948740e-03  4.86693500e-03  1.45497715e-02
 -2.54167975e-03  8.51143429e-03 -1.84773067e-02 -8.49940728e-03
 -9.10193677e-03 -1.52715280e-02  3.92853024e-04  1.19802374e-02
 -1.16080170e-03 -4.91670738e-03 -8.76284827e-03 -5.60720602e-01
  1.55118403e-02  1.51777201e-02  5.82399842e-03 -3.33205398e-02
 -1.73106621e-03  1.88871439e-02 -1.62942764e-02  3.44598467e-04
  1.23505456e-02 -1.20680167e-04 -1.48815518e-02 -6.23865521e-03
 -1.47060620e-02  6.83642885e-03 -1.13545364e-02 -6.12356127e-03
  5.67418673e-01  1.54246854e-02 -1.39066937e-02 -4.77093179e-03
  9.51583643e-03 -9.27294399e-03  2.21480698e-03 -1.41457429e-02
  1.61282679e-02 -1.63220586e-02 -2.33548808e-03  1.63606935e-02
 -1.75200328e-02 -7.04582408e-03 -9.31931663e-03 -3.18964080e-03
  2.12445123e-03  8.47198734e-03  9.40464391e-03 -3.23175876e-03
  7.31021199e-03  2.88478108e-02  3.97618734e-03 -9.46106260e-03
 -3.63047679e-03 -1.68586570e-02 -3.90515976e-03 -6.86446834e-03] b is: -0.009324350500516044
```

Looks like you're using NumPy
Would you like to turn scientific mode on?
Use scientific mode   Keep current layout...

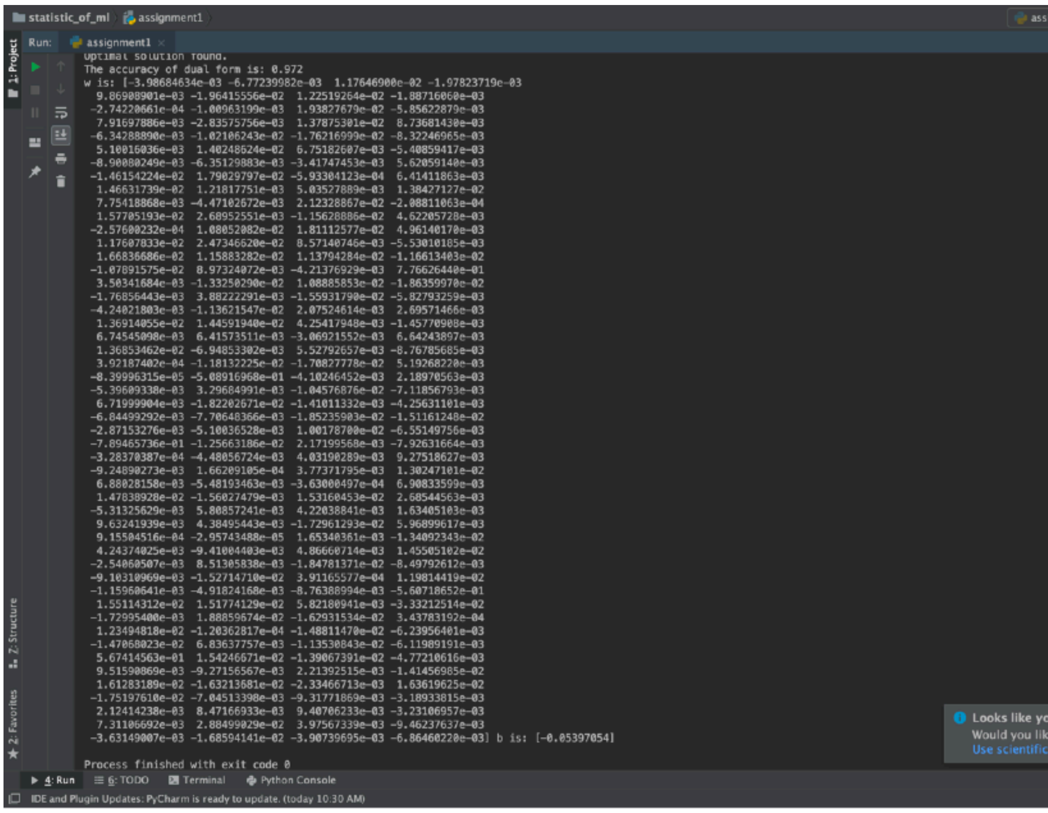
IDE and Plugin Updates: PyCharm is ready to update. (today 10:30 AM)

The w and b(-0.05)

as well as

accuracy (0.972) of

dual form is:



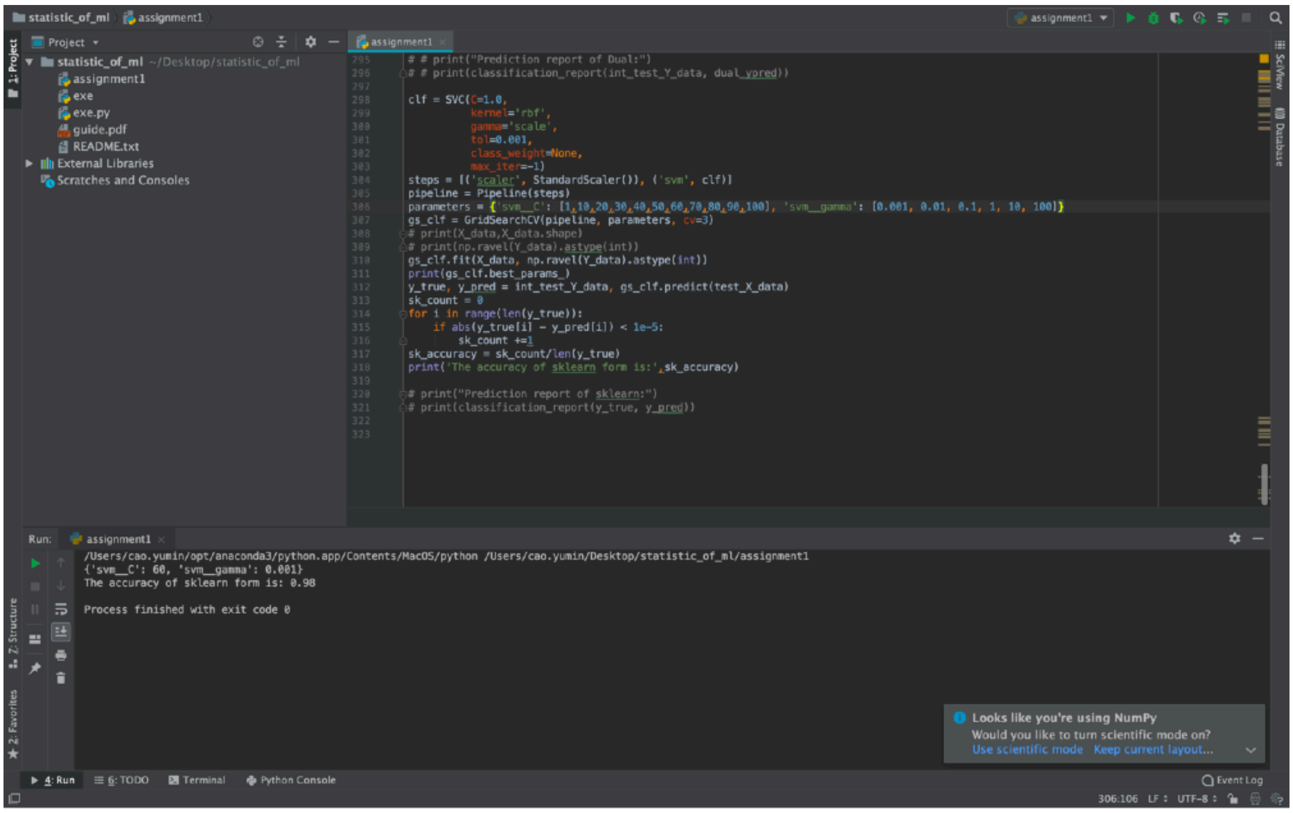
```
Optimal solution found.
The accuracy of dual form is: 0.972
w is: [-3.98684634e-03 -6.77239982e-03  1.17646900e-02 -1.97823719e-03
  9.86908901e-03 -1.96415556e-02  1.22519264e-02 -1.88716060e-03
 -2.74220661e-04 -1.00963199e-03  1.93827679e-02 -5.85622879e-03
  7.91697886e-03 -2.83575756e-03  1.37875301e-02  8.73681430e-03
 -6.34288890e-03 -1.02106243e-02 -1.76216999e-02 -8.32246965e-03
  5.10016036e-03  1.40248624e-02  6.75182607e-03 -5.40859417e-03
 -8.90080249e-03 -6.35129883e-03 -3.41747453e-03  5.62059140e-03
 -1.46154224e-02  1.79029797e-02 -5.93304123e-04  6.41411863e-03
  1.46631739e-02  1.21817751e-03  5.03527889e-03  1.38427127e-02
  7.75418868e-03 -4.47102672e-03  2.12328867e-02 -2.08811063e-04
  1.57705193e-02  2.68952551e-03 -1.15628886e-02  4.62205728e-03
 -2.57600232e-04  1.08052082e-02  1.81112577e-02  4.96140170e-03
  1.17607833e-02  2.47346620e-02  8.57140746e-03 -5.53010185e-03
  1.66836686e-02  1.15883282e-02  1.13794284e-02 -1.16613403e-02
 -1.07891575e-02  8.97324072e-03 -4.21376929e-03  7.76626440e-01
  3.50341684e-03 -1.33250290e-02  1.08885853e-02 -1.86359970e-02
 -1.76856443e-03  3.88222291e-03 -1.55931790e-02 -5.82793259e-03
 -4.24021803e-03 -1.13621547e-02  2.07524614e-03  2.69571466e-03
  1.36914055e-02  1.44591940e-02  4.25417948e-03 -1.45770908e-03
  6.74545098e-03  6.41573511e-03 -3.06921552e-03  6.64243897e-03
  1.36853462e-02 -6.94853302e-03  5.52792657e-03 -8.76785685e-03
  3.92187402e-04 -1.18132225e-02 -1.70827778e-02  5.19268220e-03
 -8.39996315e-05 -5.08916968e-01 -4.10246452e-03  2.18970563e-03
 -5.39609338e-03  3.29684991e-03 -1.04576876e-02 -7.11856793e-03
  6.71999904e-03 -1.82202671e-02 -1.41011332e-03 -4.25631101e-03
 -6.84499292e-03 -7.70648366e-03 -1.85235903e-02 -1.51161248e-02
 -2.87153276e-03 -5.10036528e-03  1.00178700e-02 -6.55149756e-03
 -7.89465736e-01 -1.25663186e-02  2.17199568e-03 -7.92631664e-03
 -3.28370387e-04 -4.48056724e-03  4.03190289e-03  9.27518627e-03
 -9.24890273e-03  1.66209105e-04  3.77371795e-03  1.30247101e-02
  6.88028158e-03 -5.48193463e-03 -3.63000497e-04  6.90833599e-03
  1.47838928e-02 -1.56027479e-03  1.53160453e-02  2.68544563e-03
 -5.31325629e-03  5.80857241e-03  4.22038841e-03  1.63405103e-03
  9.63241939e-03  4.38495443e-03 -1.72961293e-02  5.96899617e-03
  9.15504516e-04 -2.95743488e-05  1.65340361e-03 -1.34092343e-02
  4.24374025e-03 -9.41004403e-03  4.86660714e-03  1.45505102e-02
 -2.54060507e-03  8.51305838e-03 -1.84781371e-02 -8.49792612e-03
 -9.10310969e-03 -1.52714710e-02  3.91165577e-04  1.19814419e-02
 -1.15960641e-03 -4.91824168e-03 -8.76388994e-03 -5.60718652e-01
  1.55114312e-02  1.51774129e-02  5.82180941e-03 -3.33212514e-02
 -1.72995400e-03  1.88859674e-02 -1.62931534e-02  3.43783192e-04
  1.23494818e-02 -1.20362817e-04 -1.48811470e-02 -6.23956401e-03
 -1.47068023e-02  6.83637757e-03 -1.13530843e-02 -6.11989191e-03
  5.67414563e-01  1.54246671e-02 -1.39067391e-02 -4.77210616e-03
  9.51590869e-03 -9.27156567e-03  2.21392515e-03 -1.41456985e-02
  1.61283189e-02 -1.63213681e-02 -2.33466713e-03  1.63619625e-02
 -1.75197610e-02 -7.04513398e-03 -9.31771869e-03 -3.18933815e-03
  2.12414238e-03  8.47166933e-03  9.40706233e-03 -3.23106957e-03
  7.31106692e-03  2.88499029e-02  3.97567339e-03 -9.46237637e-03
 -3.63149007e-03 -1.68594141e-02 -3.90739695e-03 -6.86460220e-03] b is: [-0.05397054]

Process finished with exit code 0
```

IDE and Plugin Updates: PyCharm is ready to update. (today 10:30 AM)

Third party package

I use sklearn-svc to test my accuracy. In this method, my C arrange is also between 1 to 100, but I set a RBF kernel function that the range of sigma value is in 0.001,0.01,0.1,1,10,100. Notice that I didn't use any kernel function at the before.

I used grid search to find the best parameters and at the final, the accuracy is 0.98, with C=60 gamma=0.001.



```python
# # print("Prediction report of Dual:")
# # print(classification_report(int_test_Y_data, dual_ypred))

clf = SVC(C=1.0,
          kernel='rbf',
          gamma='scale',
          tol=0.001,
          class_weight=None,
          max_iter=-1)
steps = [('scaler', StandardScaler()), ('svm', clf)]
pipeline = Pipeline(steps)
parameters = {'svm__C': [1,10,20,30,40,50,60,70,80,90,100], 'svm__gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
gs_clf = GridSearchCV(pipeline, parameters, cv=3)
# print(X_data,X_data.shape)
# print(np.ravel(Y_data).astype(int))
gs_clf.fit(X_data, np.ravel(Y_data).astype(int))
print(gs_clf.best_params_)
y_true, y_pred = int_test_Y_data, gs_clf.predict(test_X_data)
sk_count = 0
for i in range(len(y_true)):
    if abs(y_true[i] - y_pred[i]) < 1e-5:
        sk_count +=1
sk_accuracy = sk_count/len(y_true)
print('The accuracy of sklearn form is:',sk_accuracy)

# print("Prediction report of sklearn:")
# print(classification_report(y_true, y_pred))
```

```
/Users/cao.yumin/opt/anaconda3/python.app/Contents/MacOS/python /Users/cao.yumin/Desktop/statistic_of_ml/assignment1
{'svm__C': 60, 'svm__gamma': 0.001}
The accuracy of sklearn form is: 0.98

Process finished with exit code 0
```

Comparison

My primal form and dual form have almost the same w and b, and their accuracy both on training and testing data are very high(testing 0.97). But the third party SVM method which uses the kernel function has a slightly higher accuracy(0.98).