SVM

The Goal is to minimize the cost function for SVM

$$\min_{w,b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$s.t: \ \forall i \in [N] : y_n(\mathbf{w}^T x_n + b) \geqslant 1$$

Part 1 will use Breast Cancer to illustrate and Part 2 will use iris dataset to illustrate

we will use the following libraries:

- pandas --- only to load in data
- numpy
- cvxopt
- sklearn --- only to compare the outcome of my SVM with the library SVM in the end

```
import pandas as pd
import numpy as np
from cvxopt import matrix, solvers
```

1.Using cvxopt.solvers.qp(P, q, G, h, A, b)

A standard form of Quadratic Programming(QP) at cvxopt document is:

$$\begin{aligned} \min \ & \frac{1}{2}x^T Px + q^T x \\ & Gx \preceq h \\ & Ax = b \end{aligned}$$

Data loading and preprocessing

- I used Pandas to load in the data from the breast cancer dataset.
- Also, for the y(label), I have changed every 0 to -1 in order to simplify later work without loss of generosity

```
cancer_x_train = pd.read_csv('dataset_files/cancer_X_train.csv')
cancer_y_train = pd.read_csv('dataset_files/cancer_Y_train.csv')
cancer_x_train = cancer_x_train.iloc[0:, ].values
cancer_y_train = cancer_y_train.iloc[0:, 0].values

cancer_x_test = pd.read_csv('dataset_files/cancer_X_test.csv')
cancer_y_test = pd.read_csv('dataset_files/cancer_Y_test.csv')
cancer_x_test = cancer_x_test.iloc[0:, ].values
cancer_y_test = cancer_y_test.iloc[0:, 0].values

cancer_y_train=cancer_y_train.astype(float)
cancer_y_train= (cancer_y_train-0.5)*2

cancer_y_test=cancer_y_test.astype(float)
cancer_y_test= (cancer_y_test-0.5)*2
```

Soft-margin SVM with QP by cvxopt

under this case:

```
class SVM_soft():

    def __init__(self):
        self.w = self.b = None

    def fit(self, X, y, C=1.0):
        n = len(X)
        d = len(X[0])
        #initialize p q G h
        self.P = matrix(np.identity(d + 1 + n, dtype=np.float))
        self.q = matrix(np.zeros((d + 1 + n), dtype=np.float))
        self.G = matrix(np.zeros((n + n, d + 1 + n), dtype=np.float))
        self.h = -matrix(np.ones((n+n,), dtype=np.float))

        #put in values for p q G h
        self.q[-n:,0] = C

        self.h[-n:,0] = 0

        self.P[0, 0] = 0
        self.P[1+d:, :] = 0

        for i in range(n):
            self.G[i, 0] = -y[i]
            self.G[i, 1: 1+d] = -X[i, :] * y[i]
            self.G[i, 1 + d + i] = -1
            self.G[i + n, i + d + 1] = -1
```

```
        # QP
        sol = solvers.qp(self.P,self.q,self.G,self.h)

        self.w = np.zeros(d,)
        self.b = sol["x"][0]
        for i in range(1, d + 1):
            self.w[i - 1] = sol["x"][i]

        return self.w, self.b

    def predict(self, X):
        return np.sign(np.dot(self.w, X.T) + self.b)
```

Results under different C

In this section, we will pick different C values to see the effects on the accuracy.

We choose 0, 1, 10, 100, 0.01 and 0.001

1. C = 0 (i.e no slack, hard margin)

```
svm_soft = SVM_soft()
svm_soft.fit(cancer_x_train, cancer_y_train, 0)

print("Training set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_train) == cancer_y_train).mean()
* 100))
print("Testing set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_test) == cancer_y_test).mean() *
100))
```

```
Training set accuracy:  62.6761 %
Testing set accuracy:  62.9371 %
```

2. C = 1.0 (default)

```
svm_soft.fit(cancer_x_train, cancer_y_train, 1)

print("Training set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_train) == cancer_y_train).mean()
* 100))
print("Testing set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_test) == cancer_y_test).mean() *
100))
```

```
Training set accuracy:  96.4789 %
Testing set accuracy:  95.1049 %
```

3. C = 10

```
svm_soft.fit(cancer_x_train, cancer_y_train, 10)

print("Training set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_train) == cancer_y_train).mean()
* 100))
print("Testing set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_test) == cancer_y_test).mean() *
100))
```

```
Training set accuracy:  97.6526 %
Testing set accuracy:  95.8042 %
```

4. C = 100

```
svm_soft.fit(cancer_x_train, cancer_y_train, 100)

print("Training set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_train) == cancer_y_train).mean()
* 100))
print("Testing set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_test) == cancer_y_test).mean() *
100))
```

```
Training set accuracy:  98.8263 %
Testing set accuracy:  96.5035 %
```

5. C = 0.01

```
svm_soft.fit(cancer_x_train, cancer_y_train, 1e-2)

print("Training set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_train) == cancer_y_train).mean()
* 100))
print("Testing set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_test) == cancer_y_test).mean() *
100))
```

```
Training set accuracy:  95.7746 %
Testing set accuracy:   93.007 %
```

6. C = 0.0001

```
svm_soft.fit(cancer_x_train, cancer_y_train, 1e-4)

print("Training set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_train) == cancer_y_train).mean()
* 100))
print("Testing set accuracy: {:8.6}
%".format((svm_soft.predict(cancer_x_test) == cancer_y_test).mean() *
100))
```

```
Training set accuracy:  93.4272 %
Testing set accuracy:  94.4056 %
```

c value	Training set accuracy	Testing set accuracy
0	62.6761 %	62.9371 %
0.0001	93.4272 %	94.4056 %
0.01	95.7746 %	93.007 %
1	96.4789 %	95.1049 %
10	97.6526 %	95.8042 %
100	98.8263 %	96.5035 %

Results under sklearn

In this section, we used Sklearn to show how good the library SVM will be and make a comparison with our model.

SVM under sklearn does not support C=0, the no various case, thus we choose a smallest enough number C = 1e-10 to represent this case.

```
from sklearn.svm import SVC
```

```
svm_1 = SVC(kernel = 'linear', C=1e-10, random_state=1)
```

```
svm_2 = SVC(kernel = 'linear', C=1, random_state=1)
svm_3 = SVC(kernel = 'linear', C=10, random_state=1)
svm_4 = SVC(kernel = 'linear', C=100, random_state=1)
svm_5 = SVC(kernel = 'linear', C=1e-2, random_state=1)
svm_6 = SVC(kernel = 'linear', C=1e-4, random_state=1)
svm_1.fit(cancer_x_train, cancer_y_train)
svm_2.fit(cancer_x_train, cancer_y_train)
svm_3.fit(cancer_x_train, cancer_y_train)
svm_4.fit(cancer_x_train, cancer_y_train)
svm_5.fit(cancer_x_train, cancer_y_train)
svm_6.fit(cancer_x_train, cancer_y_train)

print("C = 1e-10: {:8.6} %".format((svm_1.predict(cancer_x_test) ==
cancer_y_test).mean() * 100))
print("C = 1: {:8.6} %".format((svm_2.predict(cancer_x_test) ==
cancer_y_test).mean() * 100))
print("C = 10: {:8.6} %".format((svm_3.predict(cancer_x_test) ==
cancer_y_test).mean() * 100))
print("C = 100: {:8.6} %".format((svm_4.predict(cancer_x_test) ==
cancer_y_test).mean() * 100))
print("C = 1e-2: {:8.6} %".format((svm_5.predict(cancer_x_test) ==
cancer_y_test).mean() * 100))
print("C = 1e-4: {:8.6} %".format((svm_6.predict(cancer_x_test) ==
cancer_y_test).mean() * 100))
```

```
C = 1e-10:  62.9371 %
C = 1:  95.8042 %
C = 10:  96.5035 %
C = 100:  95.8042 %
C = 1e-2:   93.007 %
C = 1e-4:  94.4056 %
```

c value	sklearn	Our SVM
0	62.9371 %	62.9371 %
0.0001	94.4056 %	94.4056 %
0.01	93.007 %	93.007 %
1	95.8042 %	95.1049 %
10	96.5035 %	95.8042 %
100	95.8042 %	96.5035 %

From the above statistics, it is very obvious that my SVM has a similar performance with the SVM in the sklearn library for the testing data set.

Conclusion

1. When C is small, the training set accuracy is small or near zero(when C = 0 i.e no slack). And when C gets bigger, the training set accuracy will also be bigger.
2. When C gets bigger, the time used to find the optimal solution is longer(shown by the number of lines each fit).
3. When C gets bigger, the testing set accuracy has a tendency to be bigger. However, bigger C can also give a lower testing set accuracy than a smaller C considering individual cases. Thus, we will pick C = 1 to be the default value considering its accepted performance and accuracy for the later stage

2.Dealing with several classes

Basic Idea

When we have to deal with multiple case, we will reduce the problem into dealing with 2 classes and use the above SVM:

1. One verses one SVM(OVO): we will create a SVM between each 2 classes. Thus it will be k(k-1)/2 in total(k is the number of classes). Then we will use a voting strategy called 'Max win' to decide its final result. During this process, each sample will be classified into one class each SVM and we will add one credit to this class. In the end, each sample will have different credits on each class labels and we will label it with the max credit class. If the credit is equal. Then we choose the one that come last for simplicity.
2. One verses rest SVM(OVR): we label one class to be the positive label, and the others to be the negative label. Thus we still have 2 labels in total for one SVM. And we will need to create k SVMs in total(k is the number of classes) and will get k results. If several SVM label one same class to be positive, then we just choose the one that come last for simplicity.If a sample has no positive cases in each SVM, then we label it to be the last one for simplicity.

Data loading

- I use Pandas to load in the data from the iris dataset.
- For the y(label), when training the model, I will use 1 to represent the positive class and -1 to represent the negative class.

```
iris_x_train = pd.read_csv('dataset_files/iris_X_train.csv')
iris_y_train = pd.read_csv('dataset_files/iris_Y_train.csv')
iris_x_train = iris_x_train.iloc[0:, ].values
iris_y_train = iris_y_train.iloc[0:, 0].values

iris_x_test = pd.read_csv('dataset_files/iris_X_test.csv')
iris_y_test = pd.read_csv('dataset_files/iris_Y_test.csv')
iris_x_test = iris_x_test.iloc[0:, ].values
iris_y_test = iris_y_test.iloc[0:, 0].values.astype(float)
```

OVO(One vs one)

Data Preprocessing

Here, we need to train 3 SVMs.

For the first one, it labels class 0 as the positive class and others to be the negative class.

For the second one, it labels class 1 as the positive class and others to be the negative class.

For the third one, it labels class 2 as the positive class and others to be the negative class.

```
iris_y_train_0 = np.where(iris_y_train == 0, 1, -1).astype(float)
iris_y_train_1 = np.where(iris_y_train == 1, 1, -1).astype(float)
iris_y_train_2 = np.where(iris_y_train == 2, 1, -1).astype(float)
```

Train models

```
svm_soft_0 = SVM_soft()
svm_soft_1 = SVM_soft()
svm_soft_2 = SVM_soft()

svm_soft_0.fit(iris_x_train, iris_y_train_0, 1)
svm_soft_1.fit(iris_x_train, iris_y_train_1, 1)
svm_soft_2.fit(iris_x_train, iris_y_train_2, 1)
```

In the following step, after predicting, we will label each result with the SVM outcome. For example, if svm_soft_0 regards one sample to be positive, then we label it as class 0. For the ones with none or multiple positive outcomes, we label it by one of the positive label randomly.

Predict

```
predict1 = svm_soft_0.predict(iris_x_test)
predict2 = svm_soft_1.predict(iris_x_test)
predict3 = svm_soft_2.predict(iris_x_test)
```

```
list0 = []
list1 = []
list2 = []

for i in range(len(predict1)):
    if predict1[i] == 1:
        list0.append(i)
    if predict2[i] == 1:
        list1.append(i)
    if predict3[i] == 1:
        list2.append(i)


result = [2] * len(predict1)

for k in list0:
    result[k] = 0
for k in list1:
    result[k] = 1
for k in list2:
    result[k] = 2

result0 = [0] * len(predict1)

for k in list0:
    result0[k] = 0
for k in list1:
    result0[k] = 1
for k in list2:
    result0[k] = 2
```

Testing

```
print("if the non-positive are all set to label 0 Accuracy: {:8.6}
%".format((result0 == iris_y_test).mean() * 100))
print("if the non-positive are all set to label 1/2 Accuracy: {:8.6}
%".format((result == iris_y_test).mean() * 100))
```

```
if the non-positive are all set to label 0 Accuracy:    100.0 %
if the non-positive are all set to label 1/2 Accuracy:     82.0 %
```

OVR(One vs Rest)

Here, we need to train 3 SVMs.

The first one is to label the whole dataset to be 0 or 1.

The second one is to label the whole dataset to be 1 or 2.

The third one is to label the whole dataset to be 0 or 2.

In the end, for each sample, we will condct a voting to decide what class it is.

Data Preprocessing

For the data, we will first pick out the training data which does not belong to the SVM and then set the y numbers to be -1 and 1 for training SVMs.

```
iris_y_train_0_1 = list(iris_y_train)
iris_y_train_1_2 = list(iris_y_train)
iris_y_train_0_2 = list(iris_y_train)

iris_x_train_0_1 = list(iris_x_train)
iris_x_train_1_2 = list(iris_x_train)
iris_x_train_0_2 = list(iris_x_train)

ylist0 = []
ylist1 = []
ylist2 = []

for i in range(len(iris_y_train)):
    if iris_y_train[i] == 2:
        ylist2.append(i)

    if iris_y_train[i] == 0:
        ylist0.append(i)

    if iris_y_train[i] == 1:
        ylist1.append(i)


ylist0.reverse()
ylist1.reverse()
ylist2.reverse()


for k in ylist2:
    del iris_y_train_0_1[k]
    del iris_x_train_0_1[k]

for k in ylist0:
    del iris_y_train_1_2[k]
    del iris_x_train_1_2[k]
```

```
for k in ylist1:
    del iris_y_train_0_2[k]
    del iris_x_train_0_2[k]

# 1 is positive and 0 is negative
iris_x_train_0_1 = np.array(iris_x_train_0_1)
iris_y_train_0_1 = np.array(iris_y_train_0_1)
iris_y_train_0_1 = np.where(iris_y_train_0_1 == 0, -1,
1).astype(float)

# 2 is positive and 1 is negative
iris_x_train_1_2 = np.array(iris_x_train_1_2)
iris_y_train_1_2 = np.array(iris_y_train_1_2)
iris_y_train_1_2 = np.where(iris_y_train_1_2 == 1, -1,
1).astype(float)

# 2 is positive and 0 is negative
iris_x_train_0_2 = np.array(iris_x_train_0_2)
iris_y_train_0_2 = np.array(iris_y_train_0_2)
iris_y_train_0_2 = np.where(iris_y_train_0_2 == 0, -1,
1).astype(float)
```

Train models

The first SVM is trained to distinguish class 0 and 1.

The second SVM is trained to distinguish class 1 and 2.

The third SVM is trained to distinguish class 0 and 2.

```
svm_soft_0_1 = SVM_soft()
svm_soft_1_2 = SVM_soft()
svm_soft_0_2 = SVM_soft()


svm_soft_0_1.fit(iris_x_train_0_1, iris_y_train_0_1, 1)
svm_soft_1_2.fit(iris_x_train_1_2, iris_y_train_1_2, 1)
svm_soft_0_2.fit(iris_x_train_0_2, iris_y_train_0_2, 1)
```

Predict

In the following step, after predicting, we will decide the class based on the outcome of the previous 3 SVMs. For the ones with equal votes for some types or no votes, we label it by one of the positive label randomly.

```
predict0_1 = svm_soft_0.predict(iris_x_test).astype(int)
predict1_2 = svm_soft_1.predict(iris_x_test).astype(int)
predict0_2 = svm_soft_2.predict(iris_x_test).astype(int)
```

```
final = [-1] *  len(predict0_1)
for i in range(len(final)):
    if predict0_1[i] == -1 and predict0_2[i] == -1:
        final[i] = 0
        continue
    if predict0_1[i] == 1 and predict1_2[i] == -1:
        final[i] = 1
        continue
    if predict1_2[i] == 1 and predict1_2[i] == 1:
        final[i] = 2
        continue
    final[i] = 2
```

```
print("Testing set Accuracy: {:8.6} %".format((final ==
iris_y_test).mean() * 100))
```

```
Testing set Accuracy:     56.0 %
```

Conclusion

1. OVR
    - Advantage: For problems with large number of labels, it requires to train less SVM. It is also relatively easier to implement in the preprocessing period. For small size problem(like this 3 label), it has a good performance
    - Disadvantage: Each SVM are treated equally while they may have different credibility. For problems with large number of labels, it may fail to give a satisfying result.
2. OVO
    - Advantage: For problems with large number of labels, it may give a satisfying result as the voting strategy will be very useful in large scale.
    - Disadvantage: For problems with small number of labels, it may fail to give a satisfying result as the voting strategy will still cause a lot of uncertainty. For problems with large number of labels, it requires to train more SVMs and it is more computationally expensive.