**Report**

**Introduction:**

There are four classes in my submitted code,

(1) Instance (this class could be used to initialize examples that we train or test).

(2) SVM\_Preprocessor (this class implements some methods about preprocessing the data)

(3) SVM ( this class could be used to initialize a SVM training model. I mainly implement the SMO algorithm in this class)

(4) Machine ( The main() method is in this class).

In the header of SVM\_Preprocessor and SVM, there are comments to demonstrate the architecture of these two classes respectively.

**Answers to the questions:**

2.

**（1）Abandon samples with missing terms.**

This function is implemented in the constructor of SVM\_Preprocessor. The comments would assist you to find it.

**(2) Dealing with discrete(categorical) features.**

(In this step, just regard the label as a feature).

**First**: I write a program to get an overview of this features. Because I need all the information about possible values that each feature could have.

**Second:** For each feature that is not numerical, I manually look through all the possible values and order them in some way. These results are stored in an CSV file called “attributes.csv” and the values are ordered from the first column to the last one. (Some way means some intuitive way. For example, for the “occupation” features, I order them according to my impression about how much that kind of job’s salary is.

**Third:** I used the “ attributes.csv” file to build a translator that would be used to transfer the data from discrete string to continuous numerical. This is implemented in the Machine’s preprocess method. ( How do I build? Basically, the first-row value of each feature will be assigned to 1, and the second-row value will be assigned to 2, and so on… until the last row of that feature.)

**Fourth,** I use the translator to transfer the data in the original “adult.csv” file. ( This is implemented in the constructor of SVM\_Preprocessor.

**Fifth,** I normalize the data and delete one “education” feature that used to be discrete string. Because we already have a numerical “education” feature.

After the five steps, all the discrete string values are transferred to numerical values. I just regard these numerical values as continuous.

**(3) Split the dataset for stratified 10-fold-cross validation.**

First, the whole dataset are divided into two different datasets with respect to their labels. For each dataset, shuffle it first and then divide it uniformly into 10 datasets. Then I combine 1 dataset that has first label and 1 dataset that has another label, and form a new dataset. In this way, I construct the 10-fold-cross validation datasets.

**(4) Analyze the features and make a scatter plot with the two features that have the highest information gain.**

I implement this function in the method “information\_Gain()” of class “SVM\_Preprocessor”. For each feature, I use 100 split points to get which split point could result in the most information gain, and I assigned that information gain to this feature. The results are as follows:

Information gain:

0.0728174706909055

0.0116373352344417

4.3334080818979714E-4

0.07034532353548906

0.1535271769370607

0.06176714006077311

0.06930540458653278

0.008195237934386923

0.03740640712976859

0.08736532015781706

0.023115735687622204

0.04031766260526448

0.005702804648251776

And we can see that the attribute 4 and attribute 9 has the most information gain. And they corresponding to “marital.status” and “capital.gain” respectively.

Here are the figures plotted with the “marital.status” as horizontal axis and the “capital.gain” as vertical axis:

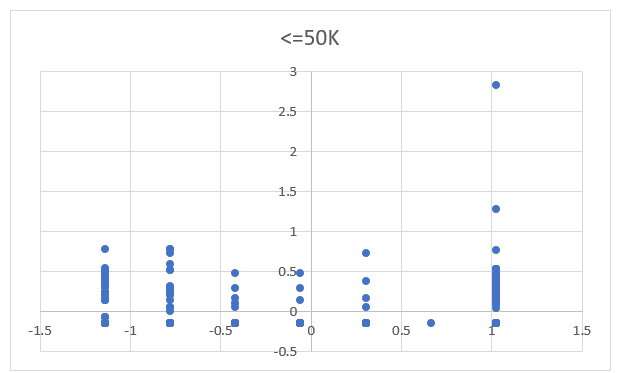


Figure 1

The figure 1 depicts the distribution of the adults whose revenue is less than or equal to 50K. We can see that most of the points are located in the region: horizontal axis ( -1.5,1.5), vertical axis ( -0.3, 0.5)

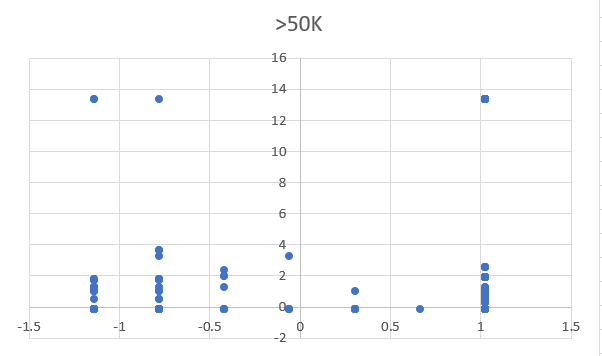


Figure 2

The figure 2 depicts the distribution of the adults whose revenue is bigger than 50K. We can see the distribution in the vertical axis is broader than that of figure 1. Most of the points are located in the region: horizontal axis ( -1.5,1.5), vertical axis ( -0.3, 4). (12,14).

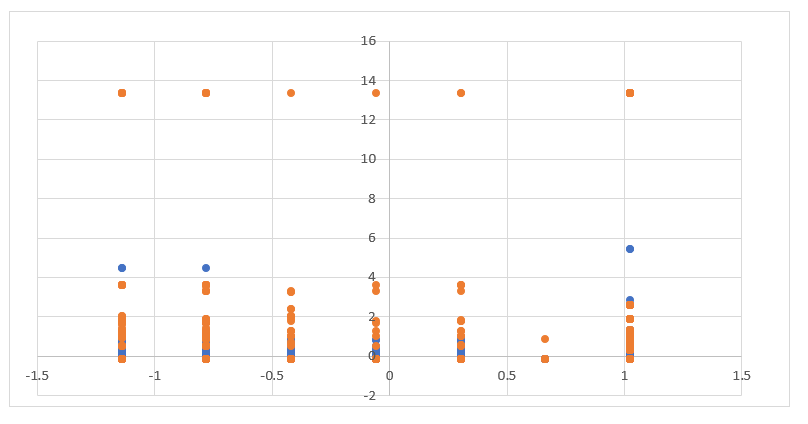


Figure 3

Figure 3 depicts the distribution of all the data. The orange color points represent the adults whose revenues are bigger than 50K. While the blue color points represent the other part of the data. Although these points are overlapped with each other, which means these data are not linearly separable with respect to the two features. We can combine the observations from Figure 1, Figure 2 and Figure 3 to roughly separate these two classes.

3.

(1)

I trained my SVM with just two features, “marital.status” and “capital.gain”. And I set the coefficient of slack variables as C = 0.1.

The results are showed in figure 4.

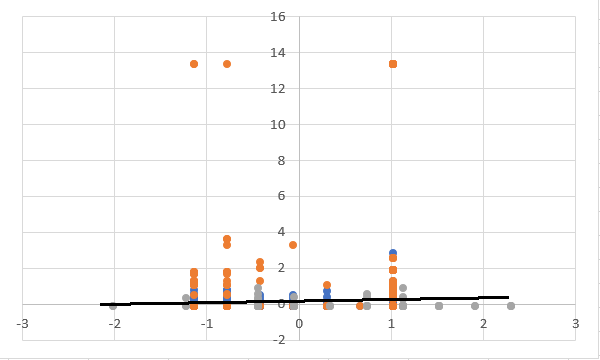


Figure 4

In Figure 4,

The orange color points represent the >50K class

The blue color points represent the <50K class.

The gray color points represent the support vectors.

The black line is the divided boundry that the SVM output.

Consider Figure 1, and Figure 2, the results seem to be right. Actually, the parameters that the SVM output is :

w1 = -2.42861286636753E-17

w2 = 2.011742883792258

b = 0.7033793307226492

These parameters corresponding to a line that is around “y = 0.35”, I think it make sense.

**(2). Change the C parameters.**

I trained the SVM with different C values. And the results are as follows:

|  |  |  |
| --- | --- | --- |
| The value of C | Accuracy | Speed |
| 0.0001 | 0.25 | Very slow |
| 0.001 | 0.783 | Ok |
| 0.01 | 0.791 | Ok |
| 0.1 | 0.790 | Ok |
| 0.5 | 0.789 | Ok |
| 1 | 0.789 | Ok |
| 10 | 0.78 | OK |
| 50 | -- | may not converge |
| 100 | -- | May not converge |
|  |  |  |
|  |  |  |

Table 1

From the results, we can find:

When C is too small, the SVM training accuracy are very low. And the training speed are also slow.

When C is too large, the SVM training speed is too slow to make sure whether it is converged or not.

With a specific range of C, the accuracy and speed are all good. And there are small difference among these models corresponding to those C values.

Here are the boundaries (shown in figures) corresponding to different C:

C = 0.0001

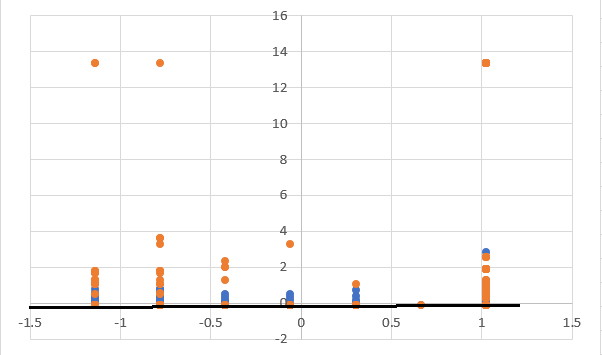


Figure 5

In Figure 5, the boundary is “y=0”. Obviously, this boundary didn’t separate the points well. Because C is so small that the SVM model underfit the training data. That is why its performance is so poor. And the accuracy is just 25%!

C = 0.01

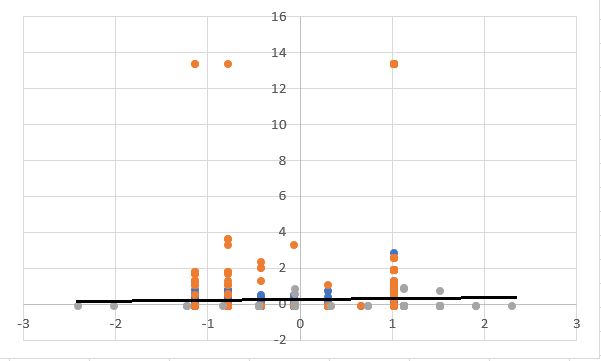


Figure 6

C=1

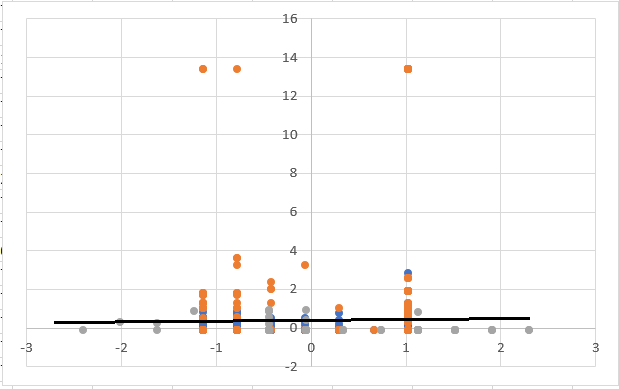


Figure 7

Figure 6, Figure 7 are similar to Figure 4, even though there is some slight difference among the support vectors. The three boundaries in these three figures are all approximate to “ y = 0.35”. From table 1 we know that their training accuracies are approximate to each other.

And the accuracy could be decreased as C is increased from the optimal value.

When C is very large, the SVM may not converge.

**(3) Train the SVM using all the features . (I still delete one education feature, because there are two “education” features and they represent the same thing.)**

|  |  |
| --- | --- |
| C | Accuracy |
| 1.00E-04 | 0.248756 |
| 2.00E-04 | 0.798563 |
| 4.00E-04 | 0.824655 |
| 8.00E-04 | 0.835158 |
| 0.0016 | 0.840243 |
| 0.0032 | 0.842676 |
| 0.0064 | 0.843118 |
| 0.0128 | 0.844555 |
| 0.0256 | 0.843228 |
| 0.0512 | 0.844002 |
| 0.1024 | 0.84356 |
| 0.2048 | 0.843118 |
| 0.4096 | 0.844223 |
| 0.8192 | 0.84356 |
| 1.6384 | 0.84356 |
| 3.2768 | 0.843781 |
| 6.5536 | 0.843671 |

Table 2

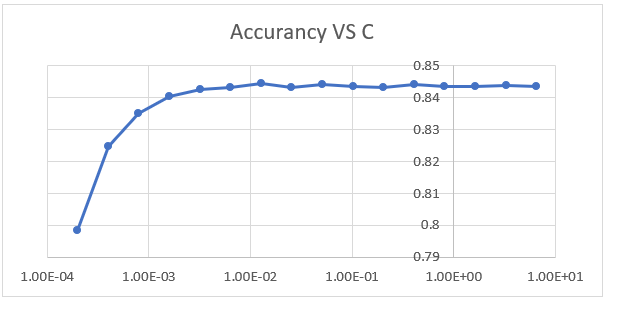
I use these results to plot Figure 8 as follows:

Figure 8

We can see that the accuracy become bigger as C increases from 0.0001 to 0.01. Then the accuracy fluctuates around 0.844 as C increase from 0.01. But the training time is longer and longer as C increase from 0.01. Concerning the Table 2, I think 0.02 is an optimal value for C.

4.

**(1) compare the performance of different kernels (Linear, RBF, Polynomial)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Recall | precision | accuracy | F1\_score | Variance |
| Linear kernel  (C = 0.02) | 0.5631 | 0.7413 | 0.8424 | 0.6400 | 4.3583 |
| RBF kernel  (, C = 1) | 0.5533 | 0.7563 | 0.8445 | 0.6391 | 0.9011 |
| Polynomial kernel (C = 0.02) | 0.5604 | 0.7628 | 0.8473 | 0.6461 | 17.9351 |

From the recall, precision, accuracy, and F1\_score perspectives, there are not much difference among the performances of these three kernels.

As for the variance, obviously the RBF kernel’s performance is the best.

**(2) Try to get higher performance with different models.**

First I try to use adaboosting to get a higher performance.

Adaboost (T = 20, Weak classifier(stumps)):

Adaboost, recall: 0.4951111111111111

Adaboost, precision: 0.7640603566529492

Adaboost, accuracy: 0.8363736871199557

Adaboost, F1\_score: 0.6008629989212514

Adaboost, variance: 1.8485806018951798

The following Figure 9 depicts how the accuracy is improved as the T increases.

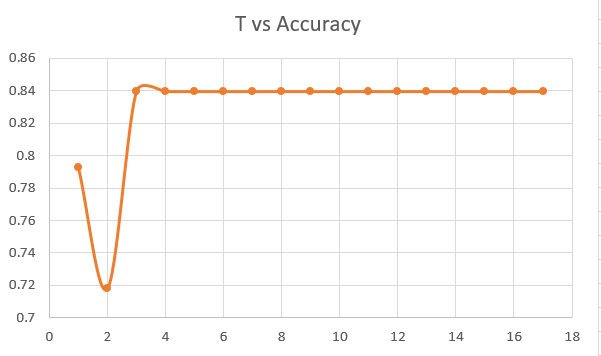


Figure 9

We can see that when the T is bigger than 4, the accuracy doesn’t change anymore. It is because adaboost is robust against overfitting. That means T = 4 is a good choice. Although the performance of adaboost is similar to that of other kernels that I tried above, its training speed is much faster than other kernel algorithm.

Then I try to use bagging to get a higher performance.( SVM linear classifier)

Here is one result:

Bagging, recall: 0.5709549071618037

Bagging, precision: 0.7519650655021834

Bagging, accuracy: 0.8459119496855346

Bagging, F1\_score: 0.6490765171503958

Bagging, variance: 4.207831257412943

I just noticed that no mater how many samples I use, the performance remains the same. Maybe the variance of the SVM model is not high comparing to its bias. Even though the bagging method can reduce the variance of training models, it can not make an improvement with respect to bias.

This bagging is very time consuming. It is not a good choice.