Data Mining HW5

LIBSVM

Name: 李友岐 Student ID: r06921048

5.1 Iris dataset : Testing label is provided.

a. Comparison of performance with and without scaling. [5%]

A:

I compare with or without scaling under different kernel functions.
(other parameters remain default)
(lower and upper bound for scaling: -1, 1)
(the percentage is the testing accuracy)

	Without scaling	with scaling
0 -- linear:	100.00%	100.00%
1 -- polynomial:	98.67%	69.33%
2 -- radial basis function:	97.33%	97.33%
3 -- sigmoid:	33.33%	97.33%

For linear function and radial basis function, the performance of with or without scaling are the same. For polynomial function, scaling results in a worse performance. For sigmoid function, scaling results in a way better performance.

b. Comparison of different kernel functions. [5%]

A:

I compare the performance of different kernel functions under the condition with or without scaling.

(other parameters remain default)
(lower and upper bound for scaling: -1, 1)
(the percentage is the testing accuracy)

	Without scaling	with scaling
0 -- linear:	100.00%	100.00%
1 -- polynomial:	98.67%	69.33%
2 -- radial basis function:	97.33%	97.33%
3 -- sigmoid:	33.33%	97.33%

Without scaling, linear function results in the best performance and polynomial function results in the second-best performance. With scaling, linear function still results in the best performance but the second-best performance are under radial basis function and sigmoid function.

c. Parameter set and performance of your best model. (Report training accuracy and testing accuracy) [5%]

A:

Parameter set (without scaling):

```
./libsvm-3.23/svm-train -t 0 iris/iris.tr model1
```

Kernel function is linear, the other parameters remain default.

```
svm_type c_svc
kernel_type linear
```

Performance:

training accuracy : 98.67%

testing accuracy : 100%

d. More discussions is welcome. [Bonus 1%]

A:

From the result in (a) and (b), we observe that scaling is not always helpful to the performance. Sometimes, it worsen the performance.

5.2 News dataset : Testing label is provided.

a. Comparison of performance with and without scaling. [5%]

A:

I compare with or without scaling under different kernel functions.
(other parameters remain default)
(lower and upper bound for scaling: 0, 1)
(the percentage is the testing accuracy)

	Without scaling	with scaling
0 -- linear:	84.20%	80.49%
1 -- polynomial:	27.76%	27.76%
2 -- radial basis function:	27.76%	27.76%
3 -- sigmoid:	27.76%	27.76%

For linear function, scaling actually worsen the performance. For the other three kernel functions, with or without scaling result in same performance.

b. Comparison of 5-1-a and 5-2-a. [5%]

A:

In 5-1-a, for polynomial function and sigmoid function, scaling results in different performance. While in 5-2-a, for polynomial function and sigmoid function, scaling results in same performance. Besides, in 5-1-a, for linear function, scaling results in same performance. While in 5-2-a, for linear function, scaling results in a worse performance.

c. Comparison of different kernel functions. [5%]

A:

I compare the performance of different kernel functions under the condition with or without scaling.
(other parameters remain default)
(lower and upper bound for scaling: 0, 1)
(the percentage is the testing accuracy)

	Without scaling	with scaling
0 -- linear:	84.20%	80.49%
1 -- polynomial:	27.76%	27.76%
2 -- radial basis function:	27.76%	27.76%
3 -- sigmoid:	27.76%	27.76%

No matter with or without scaling, linear function always provides the best Performance while the other three kernel functions result in the same low performance.

d. Parameter set and performance of your best model. (Report training accuracy and testing accuracy) [5%, Surpass baseline 5%]

A:

Although linear function provides high test accuracy, grid.py from LibSVM only supports radial basis function. Thus, I still use radial basis function to tune the parameters.

Parameter set (without scaling):

```
./libsvm-3.23/svm-train -c 2.0 -g 1.0 news/news.tr model2
```

```
svm_type c_svc
kernel_type rbf
gamma 1
```

Performance:

training accuracy : 97.72%

testing accuracy : 84.62%

e. We know that the curse of dimensionality causes overfitting. How does it influence Naive Bayesian, Decision Tree and SVM separately? [5%]

A:

Testing accuracy of News dataset of my best model:

Naive Bayesian: 89.44%

Decision Tree: 65.45%

SVM: 84.62%

News dataset has lots of attributes and thereby leads to the curse of dimensionality. Based on my result, it influences Decision Tree heavily, whose test accuracy only has 65.45%. SVM still has 84.62% test accuracy, which means the curse of dimensionality doesn't influence it too much. Naive Bayesian has 89.44% test accuracy, which is very high. The curse of dimensionality only influence it a little. Thus, my results shows that Naive Bayesian is most appropriate for the datasets containing lots of attributes.

f. More discussions is welcome. [Bonus 1%]

A:

From the result in (a), (b) and (c), we once again observe that scaling is not always helpful to the performance. Sometimes, it worsen the performance. Maybe the lower and upper bound of scaling we set are the key to the resulted performance. Different scaling range may lead to different result.

5.3 Abalone dataset : Testing label is provided.

a. Your data preprocessing and scaling range. Please state clearly. [10%]

A:

First read in two files abalone_train.csv and abalone_test.csv, then split the last column of them as the target label for training and testing, respectively. The remaining part of them are the data for training and testing, respectively. Most of the data are already a float number, except for the first column. It contains 'F', 'M' and 'I', which are char. Thus, I use ord() from Python to convert these char to their corresponding decimal value on ASCII (e.g. 'F' -> 70). Therefore, all the data become numbers. Then I output the data and target label

of abalone_train.csv and abalone_test.csv to the format LibSVM require. The resulted file of them are named as abalone.tr and abalone.te , respectively.

Finally, I use svm-scale to scale the two resulted files, where the scaling lower and upper bound are -1 and 1, respectively. The output file after scaling are named as abalone.tr.scale and abalone.te.scale . All the data become a float number between -1 and 1 consequently.

b. Comparison of different kernel functions. [5%]

A:

I compare the performance of different kernel functions under the condition with or without scaling. The percentage below is the testing accuracy.
(other parameters remain default)
(lower and upper bound for scaling: -1, 1)

	Without scaling	with scaling
0 -- linear:	64.62%	65.29%
1 -- polynomial:	66.54%	59.16%
2 -- radial basis function:	59.16%	63.76%
3 -- sigmoid:	35.47%	61.75%

Without scaling, polynomial function results in the best performance, and linear function is the second-best. With scaling, linear function provides the best performance while radial basis function provides the second-best performance.

c. Parameter set and performance of your best model. (Report training accuracy and testing accuracy) [5%, Surpass baseline 5%]

A:

Although other kernel functions may also provide high test accuracy, grid.py from LibSVM only supports radial basis function. Thus, I still use radial basis function to tune the parameters.

Parameter set (with scaling):

```
./libsvm-3.23/svm-train -c 256 -g 0.25 abalone/abalone.tr.scale model3
```

```
svm_type c_svc
kernel_type rbf
gamma 0.25
```

Performance:

training accuracy : 68.70%

testing accuracy : 67.02%

d. More discussion is welcome. [Bonus 1%]

A:

In (b), polynomial function provides best performance among all kernel functions if we don't do scaling. However, polynomial function results in the worst performance among all kernel functions if we do scaling. I think this phenomenon may be related to the characteristic of polynomial function.

5.4 Income dataset

a. Your data preprocessing / data cleaning. Please state clearly. [10%]

A:

First read in two files income_train.csv and income_test.csv, then split the last column of income_train.csv as the target label for training. The remaining part of income_train.csv and the whole income_test.csv are the data for training and testing, respectively.

Second, because most of the data are string, I transform all string values to integer by summing the ASCII decimal value of each character of the string. (e.g. 'Male' -> 77+97+108+101 -> 383).

To handle missing value, I change each missing value to be the mean of the column it belongs to by SimpleImputer from sklearn. Thus, there is no missing value in data afterwards. To further preprocess/clean the data, I transform the data into One Hot encoding by get_dummies from Python pandas. As a result, the data only contains 0 and 1, and the number of columns increase.

Finally, I output the data and target label of income_train.csv and income_test.csv to the format LibSVM require. Note that the target label of income_test.csv are all set to 0. Since income_test.csv doesn't contain target label, we label each row to 0 manually. The resulted file of them are named as income.tr and income.te , respectively. Since the data only contains 0 and 1 already, we don't need to do scaling.

b. How do you choose parameters set and kernel function ? [5%]

A:

LibSVM's grid.py is a parameter selection tool for C-SVM classification using the RBF kernel. I use it to try parameter settings automatically in a specific range and thereby find the most suitable parameters. Since grid.py from LibSVM only supports radial basis function, I choose radial basis function as kernel function to tune the parameters.

```
grid.py -log2c -10,10,1 -log2g -5,5,1 -v 5 income/income.tr
```

c. Report cross validation accuracy, and training accuracy. [5%]

A:

cross validation accuracy: 85.91%

training accuracy: 89.19%

d. Parameter set of your best model. [Surpass baseline 5%, Top 20% in class: 5%]

A:

```
./libsvm-3.23/svm-train -c 4 -g 0.0625 income/income.tr model4
```

```
svm_type c_svc
kernel_type rbf
gamma 0.0625
```

e. More discussion or observation are welcome. [Bonus 1%]

A:

In data preprocessing, we use one hot encoding to make data only contain 0 and 1. Consequently, the cross validation accuracy of the best model is 85.86%. If we don't use One Hot encoding, then the result become as follows.

Without Scaling:

cross validation accuracy: 78.14%

With Scaling (lower and upper bound= 0, 1):

cross validation accuracy: 83.29%

Therefore, we observe that the performance comparison under different data preprocessing approaches is One Hot encoding > Scaling > No scaling.