Performance comparison of three SVM packages based on R

Gu,Yilin
Business School, Macau University of Science and Technology

Dec 15, 2019

Abstract

SVM (Support Vector Machine) is a commonly used classification algorithm in machine learning. As a commonly used machine learning program implementation language, R provides several packages that implement SVM algorithms, including SVMlight, libSVM, LineaR, etc. Due to differences in the underlying algorithms of different packages, there are differences in their performances. These differences are reflected in the classification accuracy, computing time and so on. This article tests the performance of three SVM packages on the Adult Data Set dataset to evaluate their differences and explain the reasons for the differences.

Keywords: SVM, SVMlight, libsvm, lineaR, kernel functions

Introduction of SVM

SVM is a classification algorithm in machine learning.A Support Vector Machine models the situation by creating a feature space, which is a finite-dimensional vector space, each dimension of which represents a "feature" of a particular object. To put it simply, SVM is a two-class classification model. Its basic model is defined as a linear classifier with the largest interval in the feature space. The learning strategy of SVM is to maximize the interval.

SVM includes two methods: linear SVM and nonlinear SVM. According to whether the data set is linearly separable, the corresponding method is selected for classification. However, the essence of the two methods is to use the n-1 dimensional hyperplane to classify the n-dimensional data. For all hyperplanes that can separate the data set, SVM selects the hyperplane with the largest distance from the hyperplane to the data point closest to the hyperplane. This means that SVM can tolerate some noise in the test data and improve the generalization ability of the model.

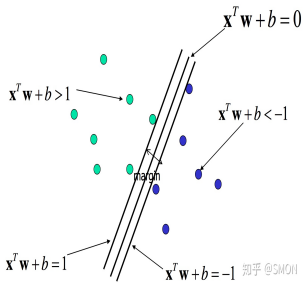

Figure 1. linear SVM

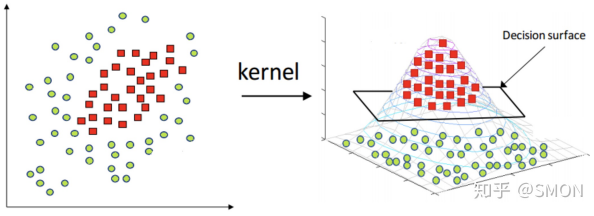

Figure 2. nonlinear SVM

Next, we introduce the important parameter C in the SVM algorithm.Assuming that the normal vector used to distinguish the hyperplane of the data set is $\boldsymbol{W}$, and the penalty function $\boldsymbol{l(x_i, y_i)}$ for the sample classification error, the optimization goal of the SVM algorithm is as follows.

$$\min_{W,l} \quad \frac{1}{2} \parallel W \parallel^2 + C \sum_{i=1}^{n} l(x_i, y_i) \tag{1}$$

$C > 0$ is called the penalty parameter. The smaller the C, the smaller the penalty for misclassification, and the larger the penalty for misclassification is. The smaller C is, the easier it is to under-fit, and the larger C is, the easier it is to over-fit.

Data preprocessing

We use R to complete the process of data preprocessing, which includes:

- Import data set
- Deleting samples with missing values
- One-hot encoding of categorical variables
- Standardization of numerical variables
- Dividing the training set and testing set

Step1:Import data set

```
data1 = read.csv("adult.data",sep=",",header=F)
data2 = read.csv("adult.test",sep=",",header=F)
data = rbind(data1,data2)
names(data) = c("age","workclass","fnlwgt","education",
"education-num","marital-status","occupation","relationship",
"race","sex","capital-gain","capital-loss","hours-per-week",
"native-country","Label")
```

The data set has fifteen variables, divided into two parts: training set and test set, with 32561 and 16281 samples respectively. In order to facilitate processing, the preprocessing process merges and disrupts the two data sets. And re-divide them after the processing is completed.

Step2:Deleting samples with missing values

The missing value of this data set is marked as "?". For easy processing, we mark all "?" as "NA" and delete all samples containing "NA".

```
for(i in 1:15){
  for(j in 1:48842){
    if(data[j,i] == " ?"){
      data[j,i] = NA
    }else{
      data[j,i] = data[j,i]
    }
  }
}
data_clean = na.omit(data)
```

In this process, a total of 3620 samples with missing values were deleted. There are 45222 valid samples remaining.

Step3:One-hot encoding of categorical variables

Onehot encoding is a method of converting categorical variables into numerical variables. Assuming that the values of a categorical variable are A, B, and C, the method will construct three new features, which are respectively denoted as kind-A, kind-B and kind-C. If the sample value is A in the categorical variable, it corresponds to the new feature kind-A The value is 1, and other features are valued as 0. The rest of the options are handled in the same way.

id	height	nationality
1	178	AUS
2	187	UK
3	160	NZ
4	158	NZ
5	174	AUS
6	201	UK
7	179	NZ
8	172	UK
9	172	NZ
10	164	NZ

Figure 3. before ont-hot encoding

id	height	nationality	nationality.AUS	nationality.UK	nationality.NZ
1	171	AUS	1	0	0
2	176	UK	0	1	0
3	185	NZ	0	0	1
4	157	NZ	0	0	1
5	172	AUS	1	0	0
6	182	UK	0	1	0
7	170	NZ	0	0	1
8	164	UK	0	1	0
9	182	NZ	0	0	1
10	183	NZ	0	0	1

Figure 4. after one-hot encoding

In statistics, categorical variables are also called dummy variables. Therefore, the "dummy" package in R can be used to encode categorical variables.

```
library("dummy")
data_dummy = dummy(data_clean)
```

Because some specific details of the data set are messy, such as the inconsistent string format, the lack of dimensions in the test set, etc., we performed some cleaning of the data set itself before data encoding. In order to save space, the complete code is not shown, only the one-hot encoding of the key categorical variable is shown.

Step4:Standardization of numerical variables

Since the measurement units of different features are the same, the values correspond-

ing to different features may not be on the same order of magnitude. If the original data is used directly, the data with a larger value will make a greater contribution to the model. This kind of error is something we don't want to see. Therefore, we need to scale the data to a small specific interval. We convert the data into dimensionless pure values, which helps to compare and weight indicators of different units or magnitudes.

The scale function in R can help us complete this process.

```
data_scale = cbind(scale(data_clean[,1]),
        scale(data_clean[,3]),scale(data_clean[,5]),
        scale(data_clean[,11:13]))
```

Step5:Dividing the training set and testing set

This experiment uses the first 35,000 samples as the training set, and the remaining samples as the test set. The last feature (income level) is used as the label of the two categories, where ">50k" is 1, and anyway, it is 0.

```
X = variable
y = factor(label)
train = sample(1:dim(data_final)[1],35000)
X_train = X[train,]
X_test = X[-train,]
y_train = y[train]
y_test = y[-train]
train_set = cbind(X_train,y_train)
test_set = cbind(X_test,y_test)
```

In the next part, we will use three different packages to implement the SVM algorithm based on the above processed data, and compare the differences between different packages.In order to control the variables, $C = 1$ in all SVM algorithms in the following experiments.

SVMlight

SVMlight is an SVM program written in C. The klaR package in R implements the functions in machine learning, and the SVM method is based on the SVMlight algorithm provided in C. Therefore, we can use the SVM algorithm in the klaR package to simulate SVMlight.

- linear SVM

```
library(klaR)
model_light = svmlight(formula = y_train ~ .,
                       data = train_set,
                       type = 'C',
                       svm.options = "-c 1 -t 0")
light_train_pred = predict(model_light,X_train)
light_train_mtx = table(light_train_pred$class,
                        y_train)
light_acc_train = sum(diag(light_train_mtx))/sum(light_train_mtx)

print(light_train_mtx)
print(light_acc_train)

light_test_pred = predict(model_light,X_test)
light_test_mtx = table(light_test_pred$class,
                       y_test)
light_acc_test = sum(diag(light_test_mtx))/sum(light_test_mtx)

print(light_test_mtx)
print(light_acc_test)
```

Training the model took 294 seconds.

We use the confusion matrix to show the fit effect of the model on the training set

and the test set.

	0	1
0	5077	1676
1	3589	24658

Table 1: The confusion matrix on the training set

	0	1
0	1469	531
1	1073	7149

Table 2: The confusion matrix on the test set

According to the above table, it can be seen that svmlight has an accuracy of 0.8496 on the training set of the data set under the linear classifier and 0.8431 on the test set. This shows that the model built by SVMlight performs well on this data set and has strong generalization ability.

- Gaussian kernel

```
model_light = svmlight(formula = y_train ~ .,
                       data = train_set,
                       type = 'C',
                       svm.options = "-c 1 -t 2")
```

Training the model took 11093 seconds.

Similarly, we show the confusion matrix on the training set and the test set.

	0	1
0	6951	751
1	1713	25585

Table 3: The confusion matrix on the training set

	0	1
0	1051	405
1	1493	7273

Table 4: The confusion matrix on the test set

The accuracy of 0.9296 on the training set of the data set under the linear classifier and 0.8143 on the test set.

It is not difficult to see that although the SVMlight with Gaussian kernel as the kernel function has higher accuracy in train set, it has less accuracy in test set,which means the ability to generalize is weaker than line kernel.This phenomenon is called overfitting. It also takes more computing time.

libsvm

The libSVM package is an SVM package developed by Professor Lin Chih-Jen of National Taiwan University. It can run in Python, R, MATLAB and other language environments, and it is a SVM toolkit that everyone uses most at present. The e1071 package in R is a package that contains various classification and clustering algorithms in machine learning. The SVM algorithm is implemented using libSVM. Therefore, the SVM function of the e1071 package can be used to simulate libsvm.

- linear SVM

```
1 library(e1071)
```

```
model_lib = svm(formula = y_train ~.,
                data = train_set,
                skernel = 'linear',
                type = 'C',
                cost = 1)

lib_train_pred = predict(model_lib,X_train)
lib_train_mtx = table(pred = lib_train_pred,
                      true = y_train)
lib_acc_train = sum(diag(lib_train_mtx))/sum(lib_train_mtx)

print(lib_train_mtx)
print(lib_acc_train)

lib_test_pred = predict(model_lib,X_test)
lib_test_mtx = table(pred = lib_test_pred,
                     true = y_test)
lib_acc_test = sum(diag(lib_test_mtx))/sum(lib_test_mtx)

print(lib_test_mtx)
print(lib_acc_test)
```

The modeling process of libSVM took 395 seconds.The model established by this method is applied to the data set, and the confusion matrix obtained is as follows.

	0	1
0	4983	1590
1	3681	24746

Table 5: The confusion matrix on the training set

According to the above table, it can be seen that libsvm has an accuracy of 0.8494

	0	1
0	1469	458
1	1075	7220

Table 6: The confusion matrix on the test set

on the training set of the data set under the linear classifier and 0.8500 on the test set. This shows that the model built by libSVM performs well on this data set and has strong generalization ability.

- Gaussian kernel

```
model_lib = svm(formula = y_train ~.,
                data = train_set,
                skernel = 'radial',
                type = 'C',
                cost = 1)
```

Training the model took 446 seconds.

Similarly, we show the confusion matrix on the training set and the test set.

	0	1
0	4983	1590
1	3681	24746

Table 7: The confusion matrix on the training set

The accuracy of 0.8494 on the training set of the data set under the linear classifier and 0.8500 on the test set.

	0	1
0	1469	458
1	1075	7220

Table 8: The confusion matrix on the test set

In the svm classification algorithm provided by libsvm, the results of linear classifier and Gaussian kernel function classification are completely consistent. But the Gaussian kernel function classification took nearly twice as long.

LiblineaR

The LiblineaR package provides a linear prediction model based on C and C++, which can provide the linear classifier function of SVM. Therefore, we use it as the third SVM model for performance comparison.

```
library(LiblineaR)

model_Linear = LiblineaR(data = X_train,
                         target = y_train,
                         bias=TRUE,
                         verbose=FALSE)

Line_train_pred = predict(model_Linear,X_train)
Line_train_mtx = table(Line_train_pred$predictions,
                       y_train)
Line_acc_train = sum(diag(Line_train_mtx))/sum(Line_train_mtx)

 print(Line_train_mtx)
 print(Line_acc_train)

Line_test_pred = predict(model_Linear,X_test)
Line_test_mtx = table(Line_test_pred$predictions,
```

```
                                y_test)
Line_acc_test = sum(diag(Line_test_mtx))/sum(Line_test_mtx)

print(Line_test_mtx)
print(Line_acc_test)
```

Training the model took 3 seconds.

The confusion matrix is as follows.

	0	1
0	5267	1877
1	3397	24459

Table 9: The confusion matrix on the training set

	0	1
0	1544	559
1	1000	7119

Table 10: The confusion matrix on the test set

The accuracy of 0.8493 on the training set of the data set under the linear classifier and 0.8574 on the test set.

The results of the SVM algorithm provided in the liblineaR package on this data set are not significantly different from the results of the other programs mentioned above. Its performance on the training set and test set is basically the same as the linear classification SVM algorithm in other packages. However, the time it takes for liblineaR to train the model is greatly reduced.

Comparison of results

- **Linear kernel function:** svmlight vs libsvm vs liblineaR

	SVMlight	libSVM	liblineaR
acc of train set	0.8496	0.8494	0.8493
acc of test set	0.8431	0.8500	0.8574
operation time	294	395	3

Table 11: Comparison of results with linear kernel function

The three linear SVM classifiers have little difference between the training set and the test set, and the error is only about 0.5 percent. SVMlight has the highest accuracy on the training set, but SVMlight has the worst performance on the test set. This shows that SVMlight has overfitting. The best performance on the test set is liblineaR, but it performs the worst on the training set.

According to the documentation of the three SVM packages, we found that the three methods use three different loss functions. Therefore, the result is the subtle difference in classification accuracy. The following table shows the loss functions of the three SVM packages.

package	Loss function
SVMlight	$min_\alpha \quad \frac{1}{2}\alpha^T Q\alpha - 1^T\alpha$
libSVM	$min_\alpha \quad \frac{1}{2}\alpha^T Q\alpha - e^T\alpha$
liblineaR	Not open source

Table 12: Loss functions of three SVM packages

In terms of running time, liblineaR has a huge advantage over the other two linear SVM classifiers.By consulting the documentation of the three packages, we learned that the difference in speed of SVMlight, libSVM and liblineaR mainly comes from two aspects:

- Data storage method
- Computational complexity

SVMlight and libSVM adopt a complete data storage method. That is, the values of all features are stored in a matrix. However, liblineaR adopts a sparse matrix storage method, which only stores the non-zero value characteristics of the sample, thus simplifying the storage of the sample.

At the level of algorithm design, because liblineaR defaults the data set to be linearly separable, it can be assumed that the classification problem of SVM is a convex optimization problem, and the problem can be solved by solving the dual problem. The complexity is $O(N)$. Since SVMlight and libSVM include the parameter kernel function, although the kernel is set to be linear, the algorithm is still solving the classification problem of the kernel function, and the complexity is $O(N^2)$. The difference in complexity also leads to a huge difference in the running speed of algorithms on large data sets.

package	data storage	Complexity
SVMlight	Complete matrix	$O(N^2)$
libSVM	Complete matrix	$O(N^2)$
liblineaR	sparse matrix	$O(N)$

Table 13: Factors affecting computing time

- **Gaussian kernel function:** svmlight vs libsvm

It can be seen that when SVMlight uses the Gaussian kernel function, there is a serious overfitting phenomenon and the running time is greatly increased. The performance of libsvm is no different from linear classification. Since both SVMlight and libSVM need to calculate the Hessian matrix, the requirements for computing power and storage space are huge. And libSVM has designed the SMO method in the algorithm to simplify the calculation, which in turn saves the running time of the algorithm.

	SVMlight	libSVM
acc of train set	0.9296	0.8494
acc of test set	0.8143	0.8500
operation time	11093	395

Table 14: Comparison of results with Gaussian kernel function

Conclusion

According to the above experiments, we have obtained the applicable scenarios of the three SVM packages. For large-scale linearly separable problems, liblineaR has a huge advantage in running speed, and the accuracy is not significantly different from the other two. For complex classification problems (linear classifiers perform poorly), libsvm and svmlight can be better solved. But efficiency is also sacrificed to a certain extent. Especially svmlight, although its model is more accurate, its running time on large-scale data sets is amazing. Finally, we use a table to summarize the results of this experiment.

package	advantage	disadvantage
SVMlight	Performs well on no linearly separable data sets.	It runs slowly on large-scale data sets.
libSVM	It runs faster on large-scale data sets and is suitable for data sets that are not linearly separable.	The running speed on large-scale linearly separable data sets is not dominant.
liblineaR	The speed is extremely fast and the classification effect is good on linear separable data sets.	Only applicable to linearly separable data sets.

Table 15: Conclusion of the lab