As a Materials Analysis in forensic lab, the following is one of the projects. The purpose of this project is to determine the type of glass, which has 7 levels. The glass data set contains 214 instances, and 11 features. The analyses in this project include using KNN classification and SVM to examine the type of glass, and comparing the effectiveness of these tow algorithms.

**Step 1 – collecting data**

The glass data set contains 214 instances, 7 type of glass, which are:

1. Building windows float processed
2. Building windows non-float processed
3. Vehicle windows float processed
4. Vehicle windows non float processed (none in this dataset)
5. Containers
6. Tableware
7. Headlamps

**Step 2- exploring and preparing the data**

There are no missing values in the glass data. First, use str() function to examine the structure of the glass data, the variables are all numeric. The first column is ID, and the last column is our predictor variable-type. We need to drop the ID feature, since it does not provide useful information for classification. The summary () function give us the summary statistics.

Summary Statistics:

Attribute: Min Max Mean SD Correlation with class

2. RI: 1.5112 1.5339 1.5184 0.0030 -0.1642

3. Na: 10.73 17.38 13.4079 0.8166 0.5030

4. Mg: 0 4.49 2.6845 1.4424 -0.7447

5. Al: 0.29 3.5 1.4449 0.4993 0.5988

6. Si: 69.81 75.41 72.6509 0.7745 0.1515

7. K: 0 6.21 0.4971 0.6522 -0.0100

8. Ca: 5.43 16.19 8.9570 1.4232 0.0007

9. Ba: 0 3.15 0.1750 0.4972 0.5751

10. Fe: 0 0.51 0.0570 0.0974 -0.1879

The variable Mg has a strong correlation with class type, Na, Al, Ba have medium correlation with class type, and Ca almost has no correlation with type. But KNN algorithm treat every feature “equal” in learning process, this might decrease the prediction accuracy.

Since the explanatory variables have different scale, before using KNN algorithm, we apply normalization to rescale the features to a standard range of values while SVM can do this automatically. We create a normalize() function to solve this problem in KNN algorithm.

normalize <- function(x){

return((x-min(x))/(max(x)-min(x)))

}

And then, take a look at the target variable –type. The table() output indicates the containment of the glass type.

> table(glass$type)

building\_windows\_float\_processed building\_windows\_non\_float\_processed

70 76

vehicle\_windows\_float\_processed containers

17 13

tableware headlamps

9 29

Thus, the class distribution is as follws:

Class Distribution: (out of 214 total instances)

-- 163 Window glass (building windows and vehicle windows)

-- 87 float processed

-- 70 building windows

-- 17 vehicle windows

-- 76 non-float processed

-- 76 building windows

-- 0 vehicle windows

-- 51 Non-window glass

-- 13 containers

-- 9 tableware

-- 29 headlamps

**Step 3 – training a model on the data**

We use cross-validation to train the model. For KNN, we use knn.cv() function, leave-one-out method, which performs k-fold CV using a fold for each one of the data’s examples. For SVM, we use 10-fold cross validation to train the model. It randomly divide the data into 10 completely separate random partitions which is folds. In the package “kernlab”, the parameter “cross” can help us to apply cross validation in SVM.

**Step 4 – evaluating model performance**

Through assign different values to the model function parameters to improve model performance and then compare the two algorithms’ accuracy and analyze the data. The accuracy is computed by correct prediction / total prediction.

* **KNN algorithm**

We change different k to get the best accuracy.

|  |  |  |  |
| --- | --- | --- | --- |
| **K value** | **Correct prediciton** | **Total prediction** | **Accuracy percent** |
| 3 | 149 | 214 | 69.6 |
| 5 | 147 | 214 | 68.7 |
| 7 | 139 | 214 | 65.0 |
| 11 | 133 | 214 | 62.1 |

When k=3, we get 69.6 percent accuracy using KNN algorithm leave-one-out cross validation.

* **SVM algorithm**

We use ksvm() function in “kernlab” package to train the model. To get a better parameter, we try 30 parameter C values from in linear kernel. And 30 C value, 30 sigmas from in RBF kernel.

And we find that C=0.004175 has the least error rate for linear SVM, which is 34.6 percent, so the accuracy is 65.4 percent. For RBF kernel SVM, C= 0.010826, sigma=0.303920 get the least error rate, which is 28.1 percent, so the accuracy is 71.9 percent.

|  |  |  |  |
| --- | --- | --- | --- |
| **SVM** | **C value** | **Sigma** | **Accuracy percent** |
| **Linear kernel** | 0.004175 | -- | 65.4 |
| **RBF kernel** | 0.010826 | 0.303920 | 71.9 |

* **Conclusion**

RBF kernel SVM is the best leaner to the glass data set than KNN algorithm and linear kernel. It reaches 71.9 percent accurate in classifying the glass type between 7 classes, which is fairly good considering the class number.

KNN algorithm’s accuracy changed little through change different values to K. But different parameter values in SVM, the accuracy changes a lot.

*R-Code:*

**#Glass identification data set**

**#read data**

glass <- read.csv("D:\\statistics\\6620 Statistical learning\\project 2\\glass.csv",header=FALSE,col.names=c("ID","RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","type"))

#drop ID feature

glass <- glass[-1]

str(glass)

summary(glass)

glass$type <- factor(glass$type,levels=c(1,2,3,5,6,7),labels=c("building\_windows\_float\_processed","building\_windows\_non\_float\_processed",

"vehicle\_windows\_float\_processed",

"containers","tableware","headlamps"))

table(glass$type)

round(prop.table(table(glass$type))\*100,digits=1)

##############################################################################

**##Applying KNN algorithm to glass dataset using cross validation**

##############################################################################

library(class)

k <- 11

glass\_pred <- knn.cv(glass\_n, glass$type, k)

#print(paste("knn k=", k, " leave-one-out cross validation accuracy is: ", sum(glass\_pred==glass$type)/length(glass\_pred)))

print(paste("knn k=", k, "# correct prediction", sum(glass\_pred==glass$type),

"total prediction", length(glass\_pred),

" leave-one-out cross validation accuracy is: ", sum(glass\_pred==glass$type)/length(glass\_pred)))

##############################################################################

**##Applying SVM to glass data set using 10-fold cross validation**

##############################################################################

library(kernlab)

Cs <- 10^seq(-3, 3, length.out=30)

Sigmas <- 10^seq(-3, 3, length.out=30)

kfold <- 10

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Linear SVM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

minErr <- 10

for (C in Cs)

{

err <- cross(ksvm(type ~., data=glass, kernel="vanilladot", cross=kfold, kpar=list()))

if (err < minErr)

{

print(sprintf("better parameter set found:err=%f, C=%f", err, C))

minErr <- err

}

}

minErr <- 10

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RBF Kernel SVM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for (C in Cs)

{

for (sigma in Sigmas)

{

err <- cross(ksvm(type ~., data=glass, kernel="rbfdot", cross=kfold, kpar=list(sigma)))

if (err < minErr)

{

print(sprintf("better parameter set found:err=%f, C=%f, sigma=%f", err, C, sigma))

minErr <- err

}

}

}