Preprocessing of categorical predictors in SVM, KNN and KDC (contributed by Xi Cheng)

Classification Methods

Preprocessing of categorical predictors in SVM, KNN and KDC (contributed by Xi Cheng)

Non-numerical data such as categorical data are common in practice. Some classification methods are adaptive to categorical predictor variables in nature, but some methods can be only applied to continuous numerical data. Among the three classification methods, only Kernel Density Classification can handle the categorical variables in theory, while kNN and SVM are unable to be applied directly since they are based on the Euclidean distances. In order to define the distance metrics for categorical variables, the first step of preprocessing of the dataset is to use dummy variables to represent the categorical variables.

Secondly, due to the distinct natures of categorical and numerical data, we usually need to standardize the numerical variables, such as the contributions to the euclidean distances from a numerical variable and a categorical variable are basically on the same level.

Finally, the introduction of dummy variables usually increase the dimension significantly. By various experiments, we find that dimension reduction techniques such as PCA usually improve the performance of these three classifiers significantly.

Car Evaluation Data Set

This dataset is from UCI machine learning repository, which was derived from a simple hierarchical decision model. The model evaluates cars according to the following six categorical features:
• V1: the buying price (v-high, high, med, low),
• V2: the price of maintenance (v-high, high, med, low),
• V3: the number of doors (2, 3, 4, 5-more),
• V4: the capacity in terms of persons to carry (2, 4, more),
• V5: the size of luggage boot (small, med, big),
• V6: and the estimated safety of the car (low, med, high).

For kernel density classification, I use NaiveBayes function with the argument usekernel = T in klaR package, which is used to fit Naive Bayes model in which predictors are assumed to be independent within each class label, and kernel density estimation can be used to estimate their class-conditional distributions. Although it can be applied works to categorical variables directly, the misclassification rates are quite high. See the tables as follows:

```r
### import data ###
library(readr)
car <- read.csv("~/Desktop/RTG/dataset/car.data.txt", header = F)
# V7: unacc, acc, good, vgood
roww <- nrow(car)
coll <- ncol(car)
numTrain <- floor((2/3) * roww)
numTest <- roww - numTrain
training <- car[sample(roww, numTrain), ]
test <- car[sample(roww, numTest), ]

### KDC ###
library(MASS)
library(klaR)
nb1 <- NaiveBayes(V7 ~., data=training, usekernel=T)
p1 <- predict(nb1, test[,1:6])
table(true = test$V7, predict = p1$class)
p2 <- predict(nb1, training[,1:6])
table(true = training$V7, predict = p2$class)
1 - mean(p1$class != test$V7)

## Confusion matrix of the training data ##
predict
true  acc good unacc vgood
acc 208 10 53 0
good 34 11 0 0
unacc 46 1 745 0
vgood 17 0 0 27

## Confusion matrix of the testing data ##
predict
true  acc good unacc vgood
```

https://stats.libretexts.org/Bookshelves/Advanced_Statistics_Computing/RTG%3A_Classification_Methods/4%3A_Numerical_...
For SVM classification, we can set dummy variables to represent the categorical variables. For each variable, we create dummy variables of the number of the level. For example, for V1, which has four levels, we then replace it with four variables, V1.high, V1.low, V1.med, and V1.vhigh. If $V1 = \text{vhigh}$ for a particular row, then $V1.\text{vhigh} = 1$ with $V1.\text{low} = 0$ and $V1.\text{med} = 0$. Since there is no numeric predictor variables in the dataset, we don’t need to consider the issue of standardization of numerical variables. Then I use `svm` function from `e1071` package with both radial and linear kernel. The two important parameters `cost` and `gamma` are obtained by `tune.svm` function. The classification results are shown below.

```r
### encode to dummy variables ###
library(lattice)
library(ggplot2)
library(caret)
dummies <- dummyVars(~ ., data=training[, -7])
c2 <- predict(dummies, training[, -7])
d_training <- as.data.frame(cbind(training$V7, c2))

dummies <- dummyVars(~ ., data=test[, -7])
c2 <- predict(dummies, test[, -7])
d_test <- as.data.frame(cbind(test$V7, c2))

### SVM ###
library(e1071)
gammalist <- c(0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.04, 0.045, 0.05)
tune.out <- tune.svm(as.factor(V1) ~., data=d_training,
                     kernel="radial", cost=2^(-1:5), gamma = gammalist)
summary(tune.out)
svm1 <- predict(tune.out$best.model, d_test[, -1])
confusionMatrix(svm1, as.factor(d_test$V1))

tune.out2 <- tune.svm(as.factor(V1) ~., data=d_training,
                     kernel="linear", cost=2^(-1:5), gamma = gammalist)
summary(tune.out2)
svm2 <- predict(tune.out2$best.model, d_test[, -1])
confusionMatrix(svm2, as.factor(d_test$V1))

## Test on Training Set ##
predict
true 1 2 3 4
  1 271 0 0 0
  2 0 45 0 0
  3 0 0 792 0
```
For kNN classification, I use `knn` function from `class` package after all categorical variables are encoded to dummy variables. The parameter $k$ is obtained by `tune.knn` function by 10-fold cross validation. The classification result is shown below.

```
>> predict
true 1 2 3 4
1 119 0 5 0
2 4 23 0 0
3 4 0 395 0
4 3 0 0 23
```

The classification success rate for testing on the test set of these three methods are shown below.

<table>
<thead>
<tr>
<th></th>
<th>KDC</th>
<th>SVM</th>
<th>kNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success Rate</td>
<td>0.8524</td>
<td>0.9965</td>
<td>0.9722</td>
</tr>
</tbody>
</table>

We can see that handling categorical variables using dummy variables works for SVM and kNN and they perform even better than KDC. Here, I try to perform the PCA dimension reduction method to this small dataset, to see if dimension reduction improves classification for categorical variables in this simple case. Here I choose the first 15 principal components. The classification results are shown below.

Naive Bayes (KDE): 0.9861111

```
>> predict
true 1 2 3 4
1 141 2 0 0
2 2 18 1 0
3 0 0 390 3
4 0 0 0 19
```

Naive Bayes (Normal): 0.984375

```
>> predict
true 1 2 3 4
1 140 3 0 0
2 4 16 1 0
3 0 0 393 0
4 0 0 1 18
```
SVM (radial, gamma = 0.02, cost = 8, Number of Support Vectors: 308): 1

<table>
<thead>
<tr>
<th>Reference</th>
<th>Prediction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>143</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>393</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

SVM (linear, gamma = 0.005, cost = 1, Number of Support Vectors: 201): 1

<table>
<thead>
<tr>
<th>Reference</th>
<th>Prediction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>143</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>393</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

kNN
- sampling method: 10-fold cross validation -> k = 6 (0.984375)

According to the results, we can see that performing PCA improves the classification, especially for KDE. Here, we may be interested in the change in parameters and the number of support vectors in SVM method after using `tune.svm` function according to different number of principal components we use.

Mushroom Database

This dataset is obtained from UCI Machine Learning Repository, which was derived from Audobon Society Field Guide. Mushrooms are described in terms of physical characteristics, and we want to classify a mushroom to be poisonous or edible. There are 22 predictor variables, such as cap-shape (bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s) and habitat (grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d), which are all categorical variables. Since the dimension of the dataset would be even higher after encoding all categorical variables into dummy variables, I used Principal Component Analysis (PCA) to perform dimension reduction.

From the plot above, we can see that 40 components results in variance close to 80%. Therefore, in this case, we’ll select number of components as 40 [PC1 to PC40] and proceed to the modeling stage. The methods of applying the three classifications are similar to the ones used in last section.

### Kernel Density Classification ###

#### Test on Training Set ####
The success rates of testing on the test set using these three classification methods are shown below.

<table>
<thead>
<tr>
<th></th>
<th>KDC</th>
<th>SVM</th>
<th>kNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9918759</td>
<td>1</td>
<td>0.9985229</td>
<td></td>
</tr>
</tbody>
</table>

We can see that all methods perform well on this mushroom dataset when we choose to use number of components as 40. The following plot shows the classification success rate when selecting different number of components.

**Connect-4 Data Set**

This data set is derived from UCI Machine Learning Repository, which contains 67557 instances of data and all legal 42 positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced. The outcome class is the game theoretical value for the first player (win, loss, draw). The attributes are the 42 positions, each of which is a categorical variable and has three levels, x=first player has taken, o=second player has taken, b=blank. I first tried using Kernel Density Classification directly on the dataset. The classification result is shown below.
The classification success rate is only 72.19%, which implies that we need to preprocess the data later. Here, I encoded the variables to dummy variables to firstly try SVM and kNN methods. The classification results are shown below.

### SVM ###
#### Test on Test Set ####
```
predict
true  draw  loss  win
draw  164  269  1770
loss  232  2273  3051
win  247  693  13820
```

The classification success rate is only 72.19%, which implies that we need to preprocess the data later. Here, I encoded the variables to dummy variables to firstly try SVM and kNN methods. The classification results are shown below.

### SVM ###
#### Test on Training Set ####
```
predict
true  1   2   3
  1  4093  106  152
  2   33 10972   57
  3   51  50  29524
```
#### Test on Test Set ####
```
predict
true  1   2   3
  1  1654  247  302
  2   148 5162  246
  3   199  263 14298
```

### kNN ###
#### Test on Training Set ####
```
predict
true  1   2   3
  1   267  368  1568
  2    89 3785  1682
  3    27  248 14485
```
#### Test on Test Set ####
```
predict
true  1   2   3
  1   0   0  4351
  2   0   0 11062
  3   0   0 29625
```

The classification success rate is 93.76% for SVM, and 82.32% for kNN. After encoding all variables to dummy variables, we can also try KDC in this case. The classification is shown below.

### Test on Training Set ###
```
predict
true  1   2   3
  1   0   0   0
  2   0   0  1062
  3   0   0 29625
```
### Test on Test Set ###
```
predict
true  1   2   3
  1   0   0  2203
  2   0   0  5556
  3   0   0 14760
```
In this case, KDC doesn’t work and can’t classify data to different classes. Then, we want to perform PCA to reduce dimension.

Here I firstly choose the first 80 principal components. The KDC method have better performances with classification success rate 87.35% as shown below.

```r
## Test on Training Set ##
## predict
## true 1 2 3
## 1 3061 1217 73
## 2 692 8788 1582
## 3 145 1603 27877

## Test on Test Set ##
## predict
## true 1 2 3
## 1 1454 712 37
## 2 352 4348 856
## 3 81 811 13868
```

Then I perform SVM and kNN after this dimension reduction.

```r
#### SVM ####
## Test on Training Set ##
## predict
## true 1 2 3
## 1 4351 0 0
## 2 0 11062 0
## 3 0 0 29625

## Test on Test Set ##
## predict
## true 1 2 3
## 1 2194 8 1
## 2 1 5550 5
## 3 0 0 14759

#### kNN ####
## predict
## true 1 2 3
## 1 1125 736 342
## 2 190 4510 856
## 3 20 406 14334
```