performing OCR with SVMs

Nithya Sarabudla

03-24-2024

Step 2 - exploring and preparng the data

```r
# load the data
letters <- read.csv("/Users/nithyasarabudla/Downloads/letterdata.csv", stringsAsFactors = TRUE)
# Display the structure of the letters data
str(letters)
```

```
## 'data.frame':    20000 obs. of  17 variables:
##  $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...
##  $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...
##  $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...
##  $ width : int  3 3 6 6 3 5 5 3 4 13 ...
##  $ height: int  5 7 8 6 1 8 4 2 4 9 ...
##  $ onpix : int  1 2 6 3 1 3 4 1 2 7 ...
##  $ xbar  : int  8 10 10 5 8 8 8 8 10 13 ...
##  $ ybar  : int  13 5 6 9 6 8 7 2 6 2 ...
##  $ x2bar : int  0 5 2 4 6 6 6 2 2 6 ...
##  $ y2bar : int  6 4 6 6 6 9 6 2 6 2 ...
##  $ xybar : int  6 13 10 4 6 5 7 8 12 12 ...
##  $ x2ybar: int  10 3 3 4 5 6 6 2 4 1 ...
##  $ xy2bar: int  8 9 7 10 9 6 6 8 8 9 ...
##  $ xedge : int  0 2 3 6 1 0 2 1 1 8 ...
##  $ xedgey: int  8 8 7 10 7 8 8 6 6 1 ...
##  $ yedge : int  0 4 3 2 5 9 7 2 1 1 ...
##  $ yedgex: int  8 10 9 8 10 7 10 7 7 8 ...
```

The dataset comprises 20,000 examples of 26 capital English letters, represented through 16 statistical features derived from images.These attributes include measures such as the horizontal and vertical dimensions of the glyph, the proportion of black versus white pixels, and the average position of pixels both horizontally and vertically.

```r
# Split the letter data into training and testing sets
letters_train <- letters[1:16000, ]
letters_test  <- letters[16001:20000, ]
```

The dataset is split into training and testing subsets. The first 16,000 records are assigned to letters_train for training the model, and the next 4,000 records are allocated to letters_test for evaluating the model's performance.

Step 3 - training a model on the data

```
# Load the kernlab library for support vector machine modeling
library(kernlab)
# Train a support vector machine classifier using the vanilladot kernel
letter_classifier <- ksvm(letter ~ ., data = letters_train,
                            kernel = "vanilladot")
```

```
##  Setting default kernel parameters
```

```
letter_classifier
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 7037
##
## Objective Function Value : -14.1746 -20.0072 -23.5628 -6.2009 -7.5524 -32.7694 -49.9786 -18.1824 -62
## Training error : 0.130062
```

The kernlab library is loaded to provide functions for support vector machine (SVM) modeling. An SVM classifier is trained on the training dataset using a linear kernel (vanilladot). The model aims to classify letters based on the given attributes. After training, the model's summary is displayed, which includes information about the type of SVM, the kernel used, the number of support vectors, and the training error.

step 4 - evaluating model performance

```
# Make predictions on the testing data using the trained classifier
letter_predictions <- predict(letter_classifier, letters_test)
head(letter_predictions)
```

```
## [1] U N V X N H
## Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

The previously trained SVM classifier to make predictions on the testing set. The predict() function applies the model to new data and returns the predicted letter classifications. The head() function then displays the first few predictions to provide a quick look at the results.we can see that the first six predicted letters were U, N, V, X, N, and H

```
# Create a contingency table to compare predicted and actual letters
table(letter_predictions, letters_test$letter)
```

```
##
## letter_predictions   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
##                  A 144   0   0   0   0   0   0   0   0   1   0   0   1   2   2
##                  B   0 121   0   5   2   0   1   2   0   0   1   0   1   0   0
##                  C   0   0 120   0   4   0  10   2   2   0   1   3   0   0   2
##                  D   2   2   0 156   0   1   3  10   4   3   4   3   0   5   5
##                  E   0   0   5   0 127   3   1   1   0   0   3   4   0   0   0
##                  F   0   0   0   0   0 138   2   2   6   0   0   0   0   0   0
##                  G   1   1   2   1   9   2 123   2   0   0   1   2   1   0   1
##                  H   0   0   0   1   0   1   0 102   0   2   3   2   3   4  20
##                  I   0   1   0   0   0   1   0   0 141   8   0   0   0   0   0
##                  J   0   1   0   0   0   1   0   2   5 128   0   0   0   0   1
##                  K   1   1   9   0   0   0   2   5   0   0 118   0   0   2   0
##                  L   0   0   0   0   2   0   1   1   0   0   0 133   0   0   0
##                  M   0   0   1   1   0   0   1   1   0   0   0   0 135   4   0
##                  N   0   0   0   0   0   1   0   1   0   0   0   0   0 145   0
##                  O   1   0   2   1   0   0   1   2   0   1   0   0   0   1  99
##                  P   0   0   0   1   0   2   1   0   0   0   0   0   0   0   2
##                  Q   0   0   0   0   0   0   8   2   0   0   0   3   0   0   3
##                  R   0   7   0   0   1   0   3   8   0   0  13   0   0   1   1
##                  S   1   1   0   0   1   0   3   0   1   1   0   1   0   0   0
##                  T   0   0   0   0   3   2   0   0   0   0   1   0   0   0   0
##                  U   1   0   3   1   0   0   0   2   0   0   0   0   0   0   1
##                  V   0   0   0   0   0   1   3   4   0   0   0   0   1   2   1
##                  W   0   0   0   0   0   0   1   0   0   0   0   0   2   0   0
##                  X   0   1   0   0   2   0   0   1   3   0   1   6   0   0   1
##                  Y   3   0   0   0   0   0   0   1   0   0   0   0   0   0   0
##                  Z   2   0   0   0   1   0   0   0   3   4   0   0   0   0   0
##
## letter_predictions   P   Q   R   S   T   U   V   W   X   Y   Z
##                  A   0   5   0   1   1   1   0   1   0   0   1
##                  B   2   2   3   5   0   0   2   0   1   0   0
##                  C   0   0   0   0   0   0   0   0   0   0   0
##                  D   3   1   4   0   0   0   0   0   3   3   1
##                  E   0   2   0  10   0   0   0   0   2   0   3
##                  F  16   0   0   3   0   0   1   0   1   2   0
##                  G   2   8   2   4   3   0   0   0   1   0   0
##                  H   0   2   3   0   3   0   2   0   0   1   0
##                  I   1   0   0   3   0   0   0   0   5   1   1
##                  J   1   3   0   2   0   0   0   0   1   0   6
##                  K   1   0   7   0   1   3   0   0   5   0   0
##                  L   0   1   0   5   0   0   0   0   0   0   1
##                  M   0   0   0   0   0   3   0   8   0   0   0
##                  N   0   0   3   0   0   1   0   2   0   0   0
##                  O   3   3   0   0   0   3   0   0   0   0   0
##                  P 130   0   0   0   0   0   0   0   0   1   0
##                  Q   1 124   0   5   0   0   0   0   0   2   0
##                  R   1   0 138   0   1   0   1   0   0   0   0
##                  S   0  14   0 101   3   0   0   0   2   0  10
##                  T   0   0   0   3 133   1   0   0   0   2   2
##                  U   0   0   0   0   0 152   0   0   1   1   0
##                  V   0   3   1   0   0   0 126   1   0   4   0
##                  W   0   0   0   0   0   4   4 127   0   0   0
##                  X   0   0   0   1   0   0   0   0 137   1   1
##                  Y   7   0   0   0   3   0   0   0   0 127   0
##                  Z   0   0   0  18   3   0   0   0   0   0 132
```

A contingency table is created to compare the predicted letters against the actual letters from the testing set. This table helps in visually assessing which letters are correctly identified and where misclassifications occur.

```r
# Check the agreement between predicted and actual lettersa
agreement <- letter_predictions == letters_test$letter
table(agreement)
```

```
## agreement
## FALSE  TRUE
##   643  3357
```

calculates whether each prediction matches the actual letter and summarizes the results in a table, showing the number of correct and incorrect predictions. Using the table() function, we see that the classifier correctly identified the letter in 3,357 out of the 4,000 test records

```r
# Calculate the proportion of agreement
prop.table(table(agreement))
```

```
## agreement
##   FALSE    TRUE
## 0.16075 0.83925
```

The proportion of correct predictions (agreement between the predicted and actual letters) is calculated to quantify the model's accuracy on the testing set.The accuracy is about 80 percent.

step 5 - improving model performance

Changing the SVM kernel function

```r
set.seed(12345)
letter_classifier_rbf <- ksvm(letter ~ ., data = letters_train,
                              kernel = "rbfdot")
```

To potentially improve model performance, this chunk trains a new SVM classifier using the Gaussian RBF (Radial Basis Function) kernel instead of the linear kernel. Setting a seed ensures that results are reproducible.

```r
# Make predictions using the SVM classifier with RBF kernel
letter_predictions_rbf <- predict(letter_classifier_rbf,
                                  letters_test)
```

Predictions are made on the testing set using the RBF kernel-based SVM model to evaluate its performance.

```r
# Check the agreement between predicted and actual letters using the RBF kernel
agreement_rbf <- letter_predictions_rbf == letters_test$letter
table(agreement_rbf)
```

```
## agreement_rbf
## FALSE  TRUE
##   278  3722
```

Assesses the accuracy of the RBF kernel-based model by comparing the predicted letters against the actual ones and summarizing the results in a table.

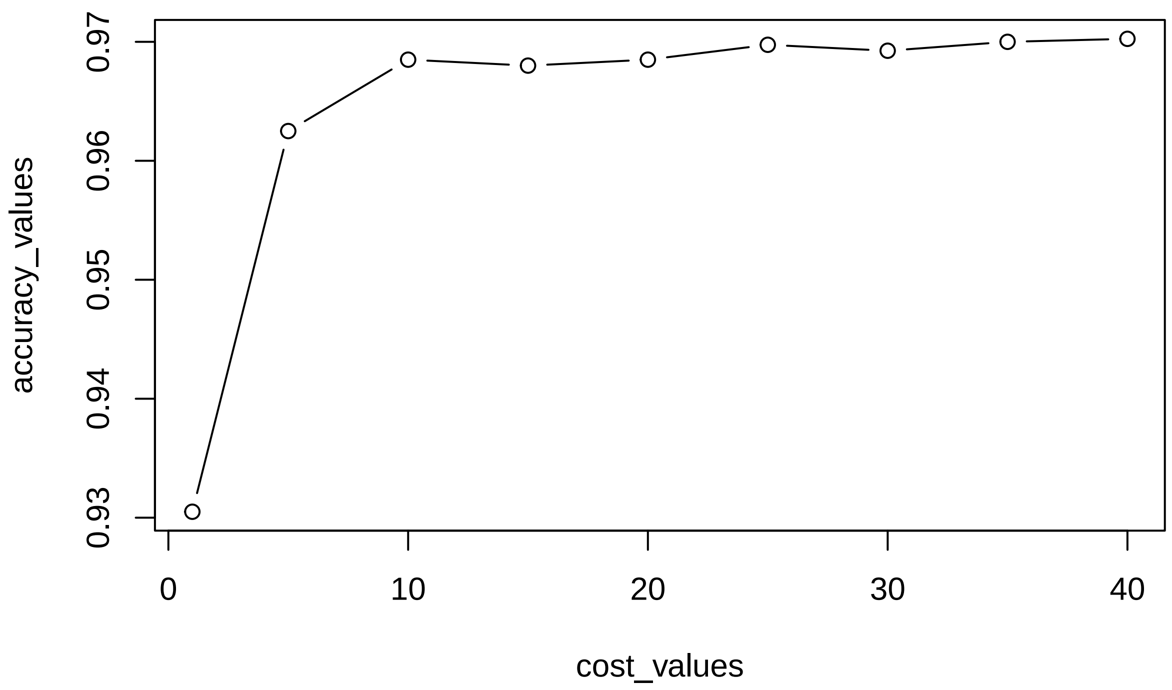
```
# Calculate the proportion of agreement using the RBF kernel
prop.table(table(agreement_rbf))
```

```
## agreement_rbf
##  FALSE   TRUE
## 0.0695 0.9305
```

Calculates the proportion of correct predictions made by the RBF kernel-based SVM model, providing a clear measure of its accuracy. Able to increase the accuracy of our character recognition model from 84 percent to 93 percent.

Identifying the best SVM cost parameter

```
# Identifying the best SVM cost parameter
cost_values <- c(1, seq(from = 5, to = 40, by = 5))
accuracy_values <- sapply(cost_values, function(x) {
    set.seed(12345)
    m <- ksvm(letter ~ ., data = letters_train,
              kernel = "rbfdot", C = x)
    pred <- predict(m, letters_test)
    agree <- ifelse(pred == letters_test$letter, 1, 0)
    accuracy <- sum(agree) / nrow(letters_test)
    return (accuracy)
  })
plot(cost_values, accuracy_values, type = "b")
```



In the quest to optimize the performance of an SVM model for optical character recognition (OCR), the exploration of the cost parameter, C, plays a crucial role. The cost parameter influences the trade-off between model complexity and the degree to which deviations from a perfect classification are tolerated. By experimenting with a range of cost values, from 1 to 40, the model's sensitivity to this parameter is assessed. This process is facilitated through a systematic approach using the sapply() function to evaluate model accuracy across different C values. The outcome reveals that increasing C from the default value of 1 to 10 or beyond significantly enhances accuracy, reaching up to 97%.