用支持向量机进行光学字符识别

对许多类型的机器学习算法来说，图像处理都是一项艰巨的任务。将像素模式连接到更高概念的关系是极其复杂的，而且很难定义。例如，让一个人识别一张面孔、一只猫或者字母 A 是很容易的，但用严格的规则来定义这些模式是很困难的。此外，图像数据往往是噪声数据。关于如何捕获图像，有许多细微的变化，这取决于灯光、定位和对象的位置。

支持向量机非常适合处理图像数据带来的挑战，它们能够学习复杂的图案而不需要对噪声过度敏感，它们能够以高的准确度识别光学图案。而且，支持向量机的主要缺点（黑箱模型的代表），对于图像处理并不那么重要。如果一个支持向量机能够区分一只猫和一只狗，那么它是如何做到的并不很重要。

在本节中，我们将研究一个模型，该模型类似于那些往往与桌面文档扫描仪捆绑在一起的光学字符识别（Optical Character Recognition，OCR）软件的核心模型。此类软件的目的是通过将印刷或者手写文本转换成一种电子形式，保存在数据库中来处理纸质文件。当然，由于手写风格和印刷字体有许多变体，所以这是困难的问题。即便如此，软件用户还是期待完美，因为纰漏或者拼写错误可能会导致商业环境中的尴尬或者代价高昂的过失（错误）。让我们来看看支持向量机是否能够胜任这项任务。

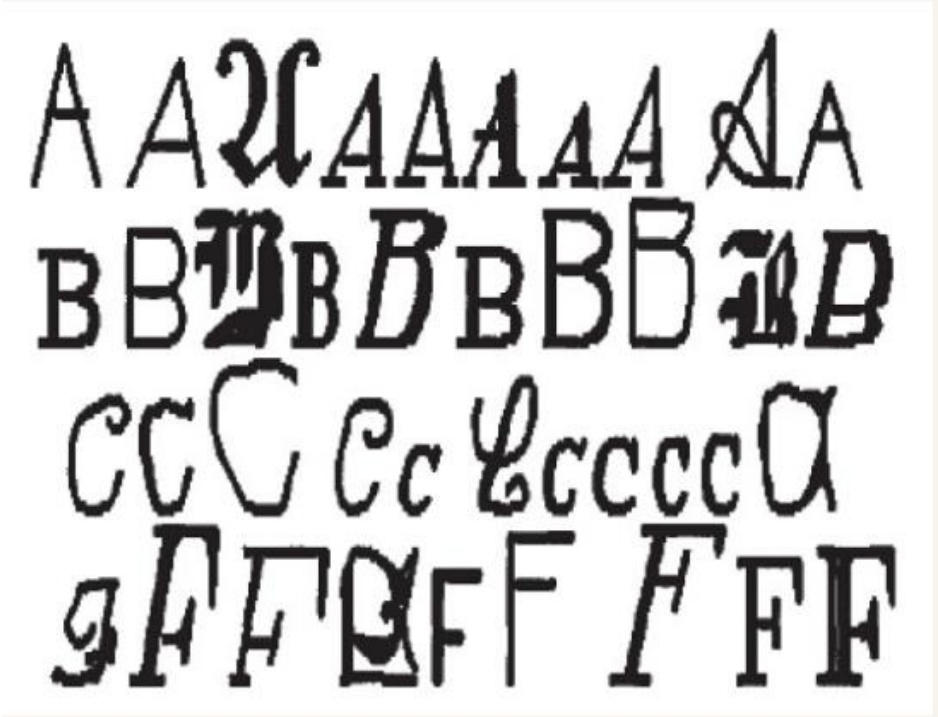
第 1 步——收集数据

数据来源：http://archive.ics.uci.edu/ml 该数据集包含了 26 个大写英文字母的 2000 个案例，使用 20 种不同的随机重塑和扭曲的黑色和白色字体印刷。

当光学字符识别软件第一次处理文件时，它将文件划分成一个矩阵，从而网格中的每一个单元包含一个单一的图像字符（glyph），这是一种适用于字母、符号或者数字的好方式。接着，对于每一个单元，该软件将试图对一组它能识别的所有字符进行图像字符匹配。最后，单个字符将重新在一起组合成词，这可以用文档语言中的字典来有选择地进行拼写检查。

在这个练习中，假设我们已经开发了将文件分割成矩形区域，每一个区域包含一个单一字符的算法；还假设文件中只包含英文字母字符。因此，我们将模拟一个过程，涉及对从 A~Z 的 26 个字母中的一个进行图像字符匹配。

下图由 W.Frey 和 D.J.Slate 发布，提供了一个包含一些印刷图像字符的案例。这种方式的扭曲，用计算机识别字母是具有挑战性的，但这些字母却很容易被人识别：



第 2 步——探索和准备数据

根据 Frey 和 Slate 提供的文件，当图像字符被扫描到计算机中，它们将转换成像素，并且有 16 个统计属性。

这些属性用图像字符的水平和垂直尺寸、黑色（相对于白色）像素的比例、像素的平均水平与垂直位置来测量字符。据推测，字符所构成的盒子的不同区域的黑色像素浓度的差异应该提供了一种区分字母表中 26 个字母的方法。

将数据读到 R 中，确认接收到的数据具有 16 个特征，这些特征定义了每一个字母类的案例。正如预期的那样，letter 有 26 个水平：

```
> letters <- read.csv("letterdata.csv")
> str(letters)
'data.frame':       20000 obs. of  17 variables:
 $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 1
4 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 ...
```

回想一下，支持向量机学习算法需要所有的特征都是数值型的，而且每一个特征需要缩小到一个相当小的区间中。在这种情况下，每一个特征都是一个整数，所以不需要将任意一个因子转换成数字。另一方面，这些整型变量的一些范围显现得相当宽，这似乎暗示需要标准化或者规范化数据。事实上，我们可以跳过这一步，因为用来拟合支持向量机模型的 R 添加包会自动帮助我们对数据进行重新调整。

考虑到大部分的数据准备都已经帮助我们完成，所以我们可以直接跳到机器学习过程的训练和测试阶段。在前面的分析中，需要在训练集和测试集之间随机地划分数据。尽管我们在这里可以做，但是 Frey 和 Slate 已经将数据随机化，并建议使用前 16000 个记录（80%）来建立模型，使用后 4000 条记录（20%）来进行测试。按照他们的建议，我们可以创建训练数据框和测试数据框，如下代码所示：

```
> letters_train <- letters[1:16000, ]
```

```
> letters_test  <- letters[16001:20000, ]
```

既然数据准备好了，那就让我们开始建立分类器吧。

第 3 步——基于数据训练模型

当谈到在 R 中拟合支持向量机模型时，有几个突出的添加包可以选择。e1071 添加包提供了一个屡获殊荣的 LIBSVM 库的 R 接口，即一个用 C++编写的广泛使用的开源支持向量机程序。如果你已经熟悉 LIBSVM，你可能想从这里开始。

同样，如果你已经投入到 SVMlight 算法中， klaR 添加包提供了 R 中该支持向量机的函数实现。

最后，如果你是从头开始，那么用 kernlab 添加包中的支持向量机函数或许是最好的开始。这个添加包的一个有趣优点就是它原本就是在 R 中开发的，而不是在 C 或者 C++中开发的，这使得它可以很容易地设置，没有任何隐藏在幕后的内部结构。或许更重要的是，与其他的选择方案不同，kernlab 添加包可以与 caret 添加包一起使用，这就允许支持向量机模型可以使用各种自动化方法进行训练和评估。

用 kernlab 添加包来训练支持向量机分类器的语法如下所示。如果你碰巧使用其他添加包，那么这些命令在很大程度上也是相似的。默认情况下，ksvm()函数使用高斯 RBF 核函数，但也提供了一些其他的选项。

支持向量机语法

应用 kernlab 添加包中的 ksvm() 函数

建立模型：

```
m <- ksvm(target ~ predictors, data = mydata, kernel = "rbfdot", c = 1)
```

- target：是数据框 mydata 中需要建模的输出变量
- preictors：是给出数据框 mydata 中用于预测的特征的一个 R 公式
- data：给出包含变量 target 和 predictors 的数据框
- kernel：给出隐一个非线性映射 (mapping)，例如 "rbfdot"（径向基函数），"polydot"（多项式函数），"tanhdot"（双曲正切函数），"vanilladot"（线性函数）
- c：用于给出违法约束条件时的惩罚，即对于“软边界”的惩罚的大小。较大的 c 值将导致较窄的边界。

该函数返回一个可以用于预测的 SVM 对象。

进行预测：

```
p <- predict(m, test, type = "response")
```

- m：函数 ksvm() 所训练的模型
- test：包含测试数据的数据框，它具有和用于训练模型的训练数据相同的特征
- type：用于指定预测的类型为 "response"（预测类别），或者 "probabilities"（预测概率，每一列对应一个类水平值）

根据 type 参数的设定，该函数返回一个包含预测类别（或者概率）的向量（或者矩阵）。两元素的列表：$neurons，用于保存神经网络每一层的神经元；$net.result，用于保存模型的预测值。

例子：

```
letter_classifier <- ksvm(letter ~ . , data = letters_train, kernel = "vanilladot")
letter_prediction <- predict(letter_classifier, letters_test)
```

为了提供度量支持向量机性能的基准，我们从训练一个简单的线性支持向量机分类器开始。如果你还没有准备好，使用命令 install.packages（"kernlab"）将 kernlab 添加包安装到系统中。然后，就可以基于训练数据调用 ksvm()函数，并使用 vanilladot 选项指定线性核函数（即 vanilla），如下所示：

```
> install.packages("kernlab")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/kernlab_0.9-25.zip'
Content type 'application/zip' length 2218659 bytes (2.1 MB)
downloaded 2.1 MB

package ‘kernlab’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
    C:\Users\Administrator\AppData\Local\Temp\RtmpuU6kyM\downloaded_packages
```

```
> library(kernlab)
Warning message:
程辑包‘kernlab’是用 R 版本 3.4.1 来建造的
> letter_classifier <- ksvm(letter ~ ., data = letters_train,kernel = "vanilladot")
 Setting default kernel parameters
```

根据计算机的性能，这个运算可能需要一些时间来完成。当它完成后，输入存储模型的名称来看一看关于训练参数和模型拟合度的一些基本信息。

```
> 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 -62.1111 -32.7284 -16.2209 -32.2837 -28.9777 -51.2195 -13.276 -35.6217 -30.8612 -16.5256 -14.6811 -32.7475 -30.3219 -7.7956 -11.8138 -32.3463 -13.1262 -9.2692 -153.1654 -52.9678 -76.7744 -119.2067 -165.4437 -54.6237 -41.9809 -67.2688 -25.1959 -27.6371 -26.4102 -35.5583 -41.2597 -122.164 -187.9178 -222.0856 -21.4765 -10.3752 -56.3684 -12.2277 -49.4899 -9.3372 -19.2092 -11.1776 -100.2186 -29.1397 -238.0516 -77.1985 -8.3339 -4.5308 -139.8534 -80.8854 -20.3642 -13.0245 -82.5151 -14.5032 -26.7509 -18.5713 -23.9511 -27.3034 -53.2731 -11.4773 -5.12 -13.9504 -4.4982 -3.5755 -8.4914 -40.9716 -49.8182 -190.0269 -43.8594 -44.8667 -45.2596 -13.5561 -17.7664 -87.4105 -107.1056 -37.0245 -30.7133 -112.3218 -32.9619 -27.2971 -35.5836 -17.8586 -5.1391 -43.4094 -7.7843 -16.6785 -58.5103 -159.9936 -49.0782 -37.8426 -32.8002 -74.5249 -
```

```
133.3423 -11.1638 -5.3575 -12.438 -30.9907 -141.6924 -54.2
953 -179.0114 -99.... <truncated>
  Training error : 0.130062
```

这些信息几乎没有告诉我们关于模型在现实世界中运行好坏的程度。因此，我们需要根据测试数据集来研究模型的性能，从而判断它是否能够很好地推广到未知的数据。

第 4 步——评估模型的性能

predict()函数允许我们基于测试数据集使用字母分类模型进行预测：

```
  > letter_predictions <- predict(letter_classifier, lett
ers_test)
```

因为我们没有指定 type 参数，所以默认使用了 type="response"，这样就返回了一个向量。该向量包含对应于测试数据中每一行值的一个预测字母，使用 head()函数，我们可以看到前 6 个预测字母是 U、N、V、X、N 和 H：

```
  > head(letter_predictions)
  [1] U N V X N H
  26 Levels: A B C D E F G H I J K L M N O P Q R S T ... Z
```

为了研究分类器的性能，我们需要将测试数据集中的预测值与真实值进行比较。为了这个目的，我们使用 table()函数（这里只显示了全部表格的一部分）：

```
  > table(letter_predictions, letters_test$letter)

  letter_predictions   A   B   C   D   E   F   G   H   I
                 A 144   0   0   0   0   0   0   0   0
                 B   0 121   0   5   2   0   1   2   0
```

C	0	0	120	0	4	0	10	2	2
D	2	2	0	156	0	1	3	10	4
E	0	0	5	0	127	3	1	1	0
F	0	0	0	0	0	138	2	2	6
G	1	1	2	1	9	2	123	2	0
H	0	0	0	1	0	1	0	102	0
I	0	1	0	0	0	1	0	0	141
J	0	1	0	0	0	1	0	2	5
K	1	1	9	0	0	0	2	5	0
L	0	0	0	0	2	0	1	1	0
M	0	0	1	1	0	0	1	1	0
N	0	0	0	0	0	1	0	1	0
O	1	0	2	1	0	0	1	2	0
P	0	0	0	1	0	2	1	0	0
Q	0	0	0	0	0	0	8	2	0
R	0	7	0	0	1	0	3	8	0
S	1	1	0	0	1	0	3	0	1
T	0	0	0	0	3	2	0	0	0
U	1	0	3	1	0	0	0	2	0
V	0	0	0	0	0	1	3	4	0
W	0	0	0	0	0	0	1	0	0
X	0	1	0	0	2	0	0	1	3
Y	3	0	0	0	0	0	0	1	0
Z	2	0	0	0	1	0	0	0	3

letter_predictions	J	K	L	M	N	O	P	Q	R
A	1	0	0	1	2	2	0	5	0
B	0	1	0	1	0	0	2	2	3
C	0	1	3	0	0	2	0	0	0
D	3	4	3	0	5	5	3	1	4
E	0	3	4	0	0	0	0	2	0
F	0	0	0	0	0	0	16	0	0
G	0	1	2	1	0	1	2	8	2
H	2	3	2	3	4	20	0	2	3
I	8	0	0	0	0	0	1	0	0
J	128	0	0	0	0	1	1	3	0
K	0	118	0	0	2	0	1	0	7

L	0	0	133	0	0	0	0	1	0
M	0	0	0	135	4	0	0	0	0
N	0	0	0	0	145	0	0	0	3
O	1	0	0	0	1	99	3	3	0
P	0	0	0	0	0	2	130	0	0
Q	0	0	3	0	0	3	1	124	0
R	0	13	0	0	1	1	1	0	138
S	1	0	1	0	0	0	0	14	0
T	0	1	0	0	0	0	0	0	0
U	0	0	0	0	0	1	0	0	0
V	0	0	0	1	2	1	0	3	1
W	0	0	0	2	0	0	0	0	0
X	0	1	6	0	0	1	0	0	0
Y	0	0	0	0	0	0	7	0	0
Z	4	0	0	0	0	0	0	0	0

letter_predictions	S	T	U	V	W	X	Y	Z
A	1	1	1	0	1	0	0	1
B	5	0	0	2	0	1	0	0
C	0	0	0	0	0	0	0	0
D	0	0	0	0	0	3	3	1
E	10	0	0	0	0	2	0	3
F	3	0	0	1	0	1	2	0
G	4	3	0	0	0	1	0	0
H	0	3	0	2	0	0	1	0
I	3	0	0	0	0	5	1	1
J	2	0	0	0	0	1	0	6
K	0	1	3	0	0	5	0	0
L	5	0	0	0	0	0	0	1
M	0	0	3	0	8	0	0	0
N	0	0	1	0	2	0	0	0
O	0	0	3	0	0	0	0	0
P	0	0	0	0	0	0	1	0
Q	5	0	0	0	0	0	2	0
R	0	1	0	1	0	0	0	0
S	101	3	0	0	0	2	0	10
T	3	133	1	0	0	0	2	2

```
U   0   0 152   0   0   1   1   0
V   0   0   0 126   1   0   4   0
W   0   0   4   4 127   0   0   0
X   1   0   0   0   0 137   1   1
Y   0   3   0   0   0   0 127   0
Z  18   3   0   0   0   0   0 132
```

对角线的值 144、121、120、156 和 127 表示的是预测值与真实值相匹配的总记录数。同样，出错的数目也列出来了。例如，位于行 B 和列 D 的值 5 表示有 5 种情况将字母 D 误认为字母 B。

单个地看每个错误类型，可能会揭示一些有趣的关于模型识别有困难的特定字母类型的模式，但这也是很耗费时间的。因此，我们可以通过计算整体的准确度来简化我们的评估，即只考虑预测的字母是正确的还是不正确的，并忽略错误的类型。

下面的命令返回一个元素为 TRUE 或者 FALSE 值的向量，表示在测试数据集中，模型预测的字母是否与真实的字母相符（即匹配）。

```
> agreement <- letter_predictions == letters_test$lette
r
```

使用 table()函数，我们看到，在 4000 个测试记录中，分类器正确识别的字母有 3357 个：

```
> table(agreement)
agreement
FALSE  TRUE
  643  3357
```

以百分比计算，准确度大约为 84%：

```
> prop.table(table(agreement))
agreement
   FALSE     TRUE
```

```
  0.16075 0.83925
```

注意，当 Frey 和 Slate 在 1991 年发布该数据集时，他们报告的识别准确度大约为 80%。仅仅使用了几行 R 代码，我们的结果便能够优于他们的结果，不过我们也受益于超过 20 年的额外的机器学习研究。考虑到这一点，我们很有可能可以做得更好。

第 5 步——提高模型的性能

之前的支持向量机模型使用简单的线性核函数。通过使用一个更复杂的核函数，我们可以将数据映射到一个更高维的空间，并有可能获得一个较好的模型拟合度。

然而，从许多不同的核函数进行选择是具有挑战性的。一个流行的惯例就是从高斯 RBF 核函数开始，因为它已经被证明对于许多类型的数据都能运行得很好。我们可以使用 ksvm()函数来训练一个基于 RBF 的支持向量机，如下所示：

```
  > letter_classifier_rbf <- ksvm(letter ~ ., data = lette
rs_train, kernel = "rbfdot")
```

然后，我们像之前一样进行预测：

```
  > letter_predictions_rbf <- predict(letter_classifier_r
bf,letters_test)
```

最后，与我们的线性支持向量机的准确度进行比较：

```
  > agreement_rbf <- letter_predictions_rbf == letters_te
st$letter
  > table(agreement_rbf)
  agreement_rbf
  FALSE  TRUE
    281  3719
  > prop.table(table(agreement_rbf))
```

```
agreement_rbf
  FALSE    TRUE
0.07025 0.92975
```

通过简单地改变核函数，我们可以将字符识别模型的准确度从84%提高到93%。如果这种性能水平对于光学字符识别程序仍不能令人满意，那么你可以测试其他的核函数或者通过改变成本约束参数C来修正决策边界的宽度。