Keras深度学习

--入门、实战与进阶

分享人：谢佳标

个人简介

谢佳标

- 曾任职于康拓普、乐逗游戏、平安人寿等电力、游戏、金融等领域企业，熟悉不同行业的数据特点，有丰富的利用Python和R语言进行数据挖掘实战经验
- 超过13年的数据挖掘与分享相关工作经验，擅长数据分析、数据挖掘、数据额可视化等领域；
- 2017-2021年微软数据科学和AI方向最有价值专家(微软MVP)
- 《中国现场统计研究会大数据统计分会》第一届理事
- 历届中国R语言大会特邀演讲嘉宾
- 已出版：《Keras深度学习：入门、实践到进阶》、《R语言游戏数据分析与挖掘》、《R语言与数据挖掘》、《数据实践之美：31位大数据专家的方法、技术与思想》
- 即将出版：《R语言数据处理及挖掘》（微课）、《Python深度学习实战：基于Keras》

目录

01

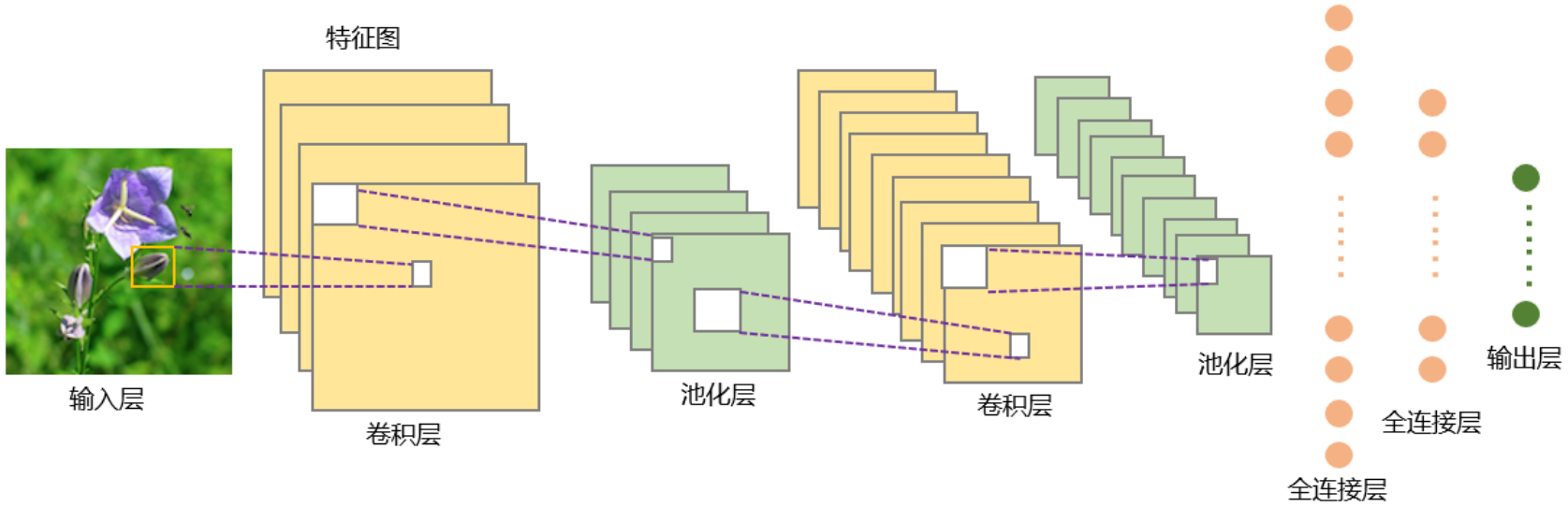
深度学习理论基础

机器学习与深度学习

- 人工智能、机器学习、深度学习是近年来非常流行的三个词语。三者之间的关系如下图所示：

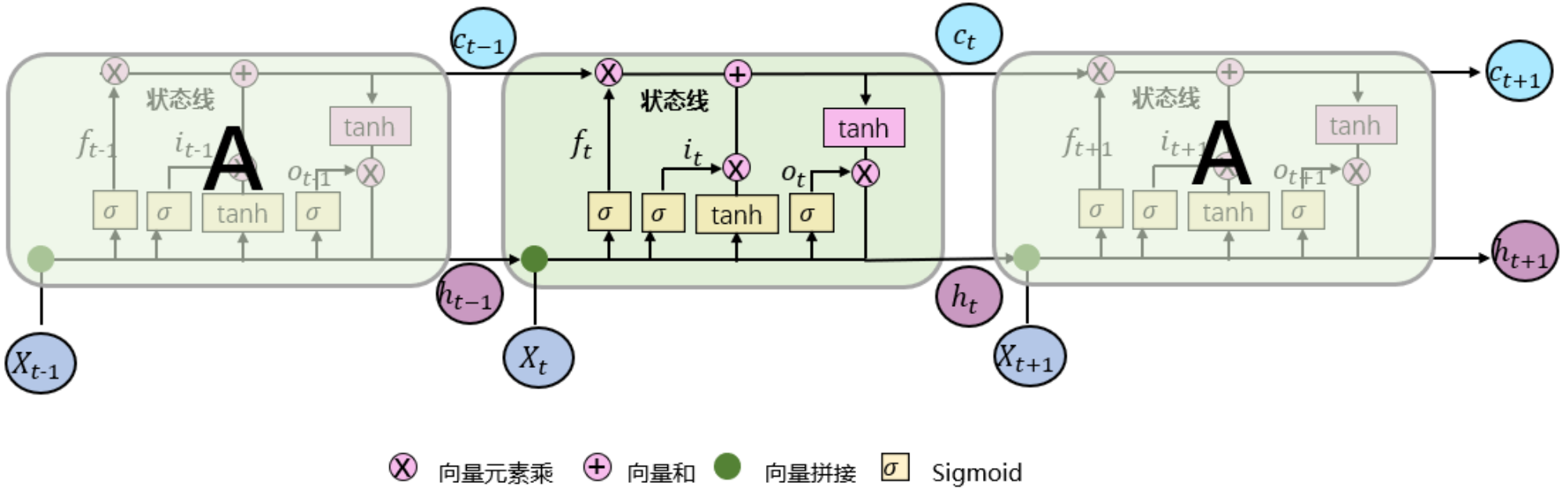
卷积神经网络

- 在理论上，卷积神经网络是一种特殊的多层感知器或前馈神经网络。标准的卷积神经网络一般由输入层（input layer）、交替的卷积层（convolution layer）和池化层（pooling layer）、全连接层（fully connected layer）和输出层（output layer）构成。



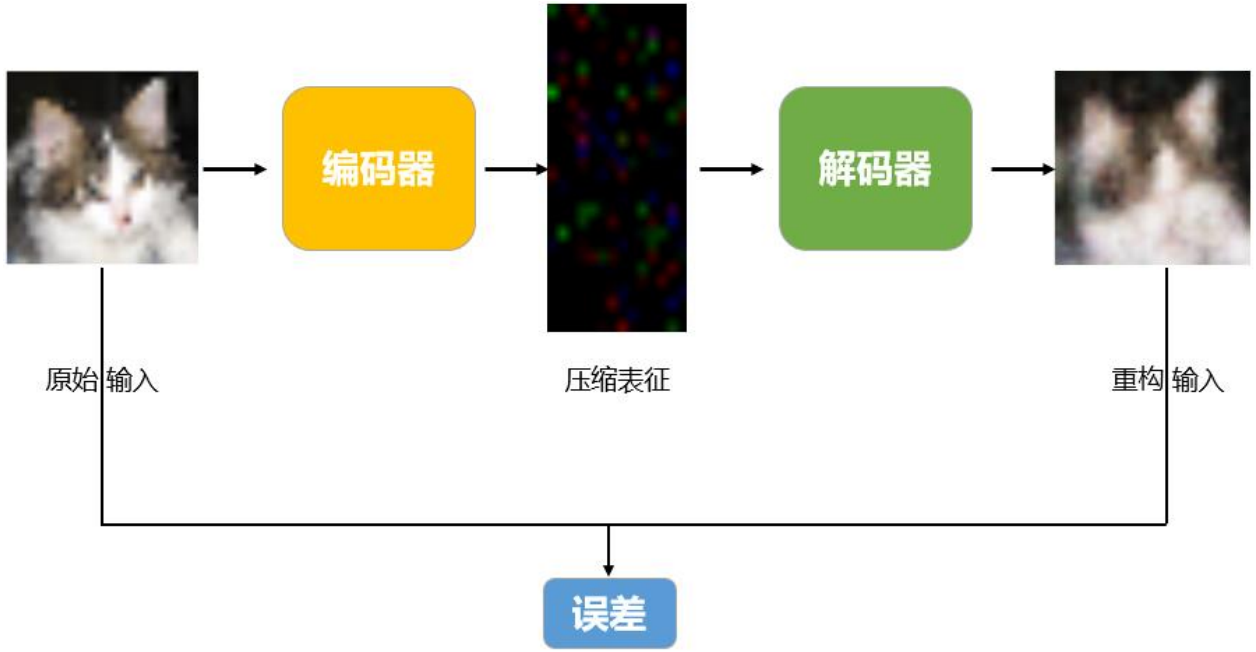
循环神经网络-LSTM

- 长短期记忆网络（Long Short-Term Memory，LSTM）是循环神经网络的一个变体，旨在避免长期依赖性问题，它具有长期记忆信息的能力，可以有效地解决简单循环网络的梯度爆炸或梯度消失问题。
- 所有递归神经网络都具有重复的神经网络模块的链式结构。在标准RNN中，该重复模块仅有一个非常简单的机构，例如单个tanh层。



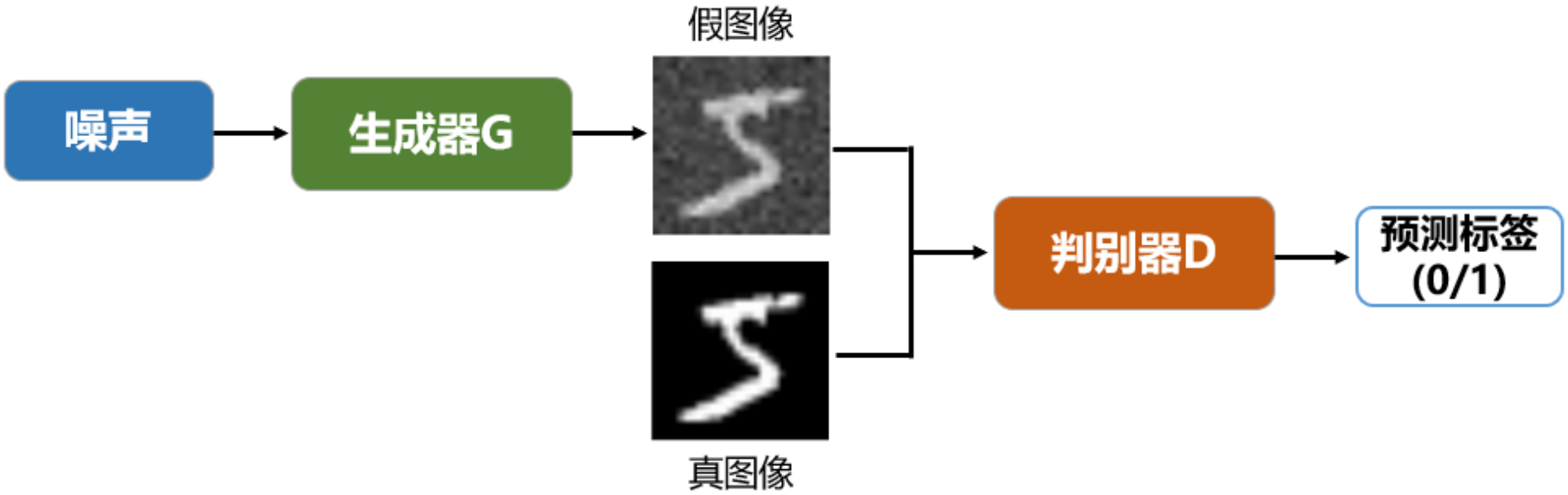
自编码网络

- 自编码器（Auto Encoder，AE）是一种基于无监督学习的数据维度压缩和特征表达方法，就是一种利用反向传播算法使得输出值等于输入值的神经网络，它先将输入压缩成潜在空间表征，然后将这种表征重构为输出。
- 通过以下三步骤可以构建一个自编码器：搭建编码器、搭建解码器、设定一个用于衡量由于压缩而丢失信息的损失函数。编码器和解码器一般都是参数化的方程，并关于损失函数可导。典型情况是使用神经网络搭建编码器和解码器，并通过最小化损失函数来优化编码器和解码器参数，所以自编码器又称为自编码网络。



生成式对抗网络

- 生成式对抗网络（GAN）由生成器（Generator）和判别器（Discriminator）两个神经网络生成。GAN受博弈论中的零和博弈启发，生成器和判别器这两个神经网络拥有不一样的目标，相互进行对抗和博弈，最终能够更精准地完成任务。



迁移学习

- 在计算机视觉领域中，迁移学习通常是通过使用预训练模型来表示的。预训练模型是在大型基准数据集上训练的模型，用于解决相似的问题。由于训练这种模型的计算成本较高，因此，导入已发布的成果并使用相应的模型是比较常见的做法。

表 14-1 Keras 内置的预训练模型信息

函数	模型	大小	Top1准确率	Top5准确率	参数数目	深度
application_xception()	Xception	88MB	0790	0.945	22,910,480	126
application_vgg16()	VGG16	528MB	0.713	0.901	138,357,544	23
application_vgg19()	VGG19	549MB	0.713	0.900	143,667,240	26
application_resnet50()	ResNet50	99MB	0.749	0.921	25,636,712	168
application_inception_v3()	InceptionV3	92MB	0.779	0.937	23,851,784	159
application_inception_resnet_v2()	InceptionResNetV2	215MB	0.803	0.953	55,873,736	572
application_mobilenet()	MobileNet	16MB	0.704	0.895	4,253,864	88
application_mobilenet_v2()	MobileNetV2	14MB	0.713	0.901	3,538,984	88
application_densenet121()	DenseNet121	33MB	0.750	0.923	8,062,504	121
application_densenet169()	DenseNet169	57MB	0.762	0.932	14,307,880	169
application_densenet201()	DenseNet201	80MB	0.773	0.936	20,242,984	201
application_nasnet()	NASNetMobile	23MB	0.744	0.919	5,326,716	-
application_nasnetlarge()	NASNetLarge	343MB	0.825	0.960	88,949,818	-

02

Keras开发深度学习模型

Keras模型生命周期

- Keras模型生命周期包括以下5个步骤：定义网络、编译网络、训练网络、评估网络、做出预测。

数据预处理

- 泰坦尼克号数据集共有1309个样本，11个字段。各字段说明如表所示。

字段	字段说明	数据说明
pclass	舱等级	1=头等舱，2=二等舱，3=三等舱
survived	是否生存	0=否，1=是
name	姓名	
sex	性别	female=女性、male=男性
age	年龄	
sibsp	手足或配偶也在船上数量	
parch	双亲或子女也在船上数量	
ticket	船票号码	
fare	旅客费用	
cabin	舱位号码	
embarked	登船港口	C=Cherbourg，Q=Queenstown，S=Southampton

经过数据预处理后会产生feature（共有9个特征字段）与label标签字段（是否生存？1：是，2：否）。

定义网络

- 构建深度学习网络的第一步是定义神经网络。神经网络在Keras中被定义为层序列。这些图层的容器是Sequential类。

```
> library(keras)
> model <- keras_model_sequential()
> model %>% layer_dense(units = 40, input_shape = c(9),
+                       kernel_initializer = 'normal',
+                       activation = 'relu' )
> model %>% layer_dense(units = 1, activation = 'sigmoid' )
> # 查看模型摘要
> summary(model)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 40)                400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 41
=================================================================
Total params: 441
Trainable params: 441
Non-trainable params: 0
_________________________________________________________________
```

编译网络

- 一旦我们定义了网络，在训练模型前就必须编译它，编译是提高效率的一个步骤。
- 我们使用compile()函数对训练模型进行设置。

```
> model %>% compile(loss = 'binary_crossentropy',
+                   optimizer = 'adam',
+                   metrics = c('accuracy'))
```

训练网络

- 一旦网络结构编译完成，就可以进行模型训练了。这意味着在训练数据集上调整模型权重，训练网络需要指定训练数据，包括输入矩阵X和匹配输出数组y。使用反向传播算法训练网络，并根据编译模型时指定的优化算法和损失函数进行优化。
- 使用fit()函数进行训练，并将训练过程存储在history变量中。

```
> history <- model %>% fit(
+   x = x_train_scale,
+   y = y_train,
+   validation_split = 0.1,
+   epochs = 10,
+   batch_size = 32,
+   verbose = 2)
```

```
Train on 943 samples, validate on 105 samples
Epoch 1/10
943/943 - 3s - loss: 0.6629 - accuracy: 0.6394 - val_loss: 0.6249 - val_accuracy: 0.7238
Epoch 2/10
943/943 - 0s - loss: 0.5993 - accuracy: 0.7752 - val_loss: 0.5574 - val_accuracy: 0.7905
Epoch 3/10
943/943 - 0s - loss: 0.5481 - accuracy: 0.7826 - val_loss: 0.5028 - val_accuracy: 0.8000
Epoch 4/10
943/943 - 0s - loss: 0.5071 - accuracy: 0.7900 - val_loss: 0.4691 - val_accuracy: 0.8095
Epoch 5/10
943/943 - 0s - loss: 0.4808 - accuracy: 0.7922 - val_loss: 0.4527 - val_accuracy: 0.7905
Epoch 6/10
943/943 - 0s - loss: 0.4653 - accuracy: 0.7922 - val_loss: 0.4450 - val_accuracy: 0.7905
Epoch 7/10
943/943 - 0s - loss: 0.4561 - accuracy: 0.7837 - val_loss: 0.4402 - val_accuracy: 0.7905
Epoch 8/10
943/943 - 0s - loss: 0.4497 - accuracy: 0.7837 - val_loss: 0.4385 - val_accuracy: 0.7905
Epoch 9/10
943/943 - 0s - loss: 0.4455 - accuracy: 0.7858 - val_loss: 0.4346 - val_accuracy: 0.8000
Epoch 10/10
943/943 - 0s - loss: 0.4416 - accuracy: 0.7943 - val_loss: 0.4319 - val_accuracy: 0.8095
```

评估网络

- 一旦网络训练完成，就可以对其进行评估。我们可以利用在训练期间没有用到的测试集来评估模型的性能。这样才能真实的反映我们训练的这个网络结构的真实情况。evalute()函数按照批次计算在某些输入数据上模型的误差。

```
> score <- model %>% evaluate(x_test_scale, y_test, verbose = 0)
> score
$loss
[1] 0.4723684

$accuracy
[1] 0.7854406
```

做出预测

- 一旦我们对训练模型的性能感到满意，就可以用它来预测新数据。调用模型的predict()函数实现。

```
> predictions <- model %>% predict(x_test_scale,verbose = 0)
> head(predictions)
          [,1]
[1,] 0.9309239
[2,] 0.2624388
[3,] 0.9582632
[4,] 0.9527556
[5,] 0.6007361
[6,] 0.5770388
```

- 对于分类问题，我们还可以使用predict_classes()函数，它会自动将预测转换为概率值最大对应的类别值。

```
> pred_label <- model %>% predict_classes(x_test_scale,verbose = 0)
> head(pred_label)
     [,1]
[1,]    1
[2,]    0
[3,]    1
[4,]    1
[5,]    1
[6,]    1
```

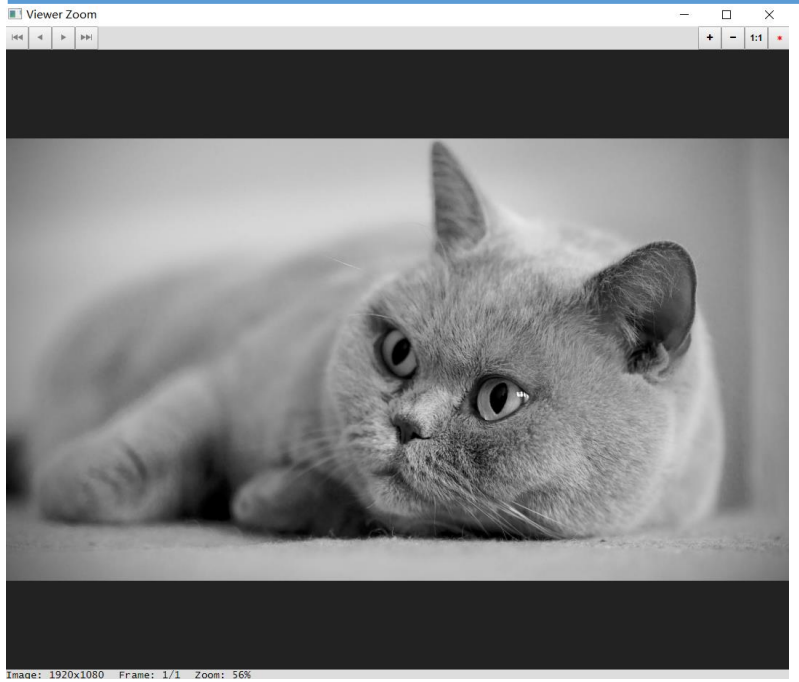
03

图像识别实战

图像读取与保存：EBImage包

- EBImage包在Bioconductor中，通过以下命令进行安装。
  - install.packages("BiocManager")
  - BiocManager::install("EBImage")
- EBImage的基本功能包括图像的读取、显示和写入。使用readImage()函数读取图像，函数中的参数files表示需要读取的文件名或URL，参数type表示读取的图像文件格式，目前支持jpeg、png和tiff三种图像文件格式。

```
> img <- readImage('../images/cat.jpg')
> display(img ,method = 'browser')
```




```
> imgcol <- readImage('../images/cat-color.jpg')
> display(imgcol,method = 'raster')
```

图像读取与保存：Keras

- Keras中的图像读取函数image_load()

```
> imgcol <- image_load('../images/cat-color.jpg')
```

- imgcol是PIL.Image类的一个实例对象，所以具备Image类相关的方法和属性。

```
> imgcol$format
[1] "JPEG"
> imgcol$mode
[1] "RGB"
> imgcol$size
[[1]]
[1] 603

[[2]]
[1] 402
```

```
> plot(as.raster(imgcol_tensor[,,],max = 255))
```

图像缩放

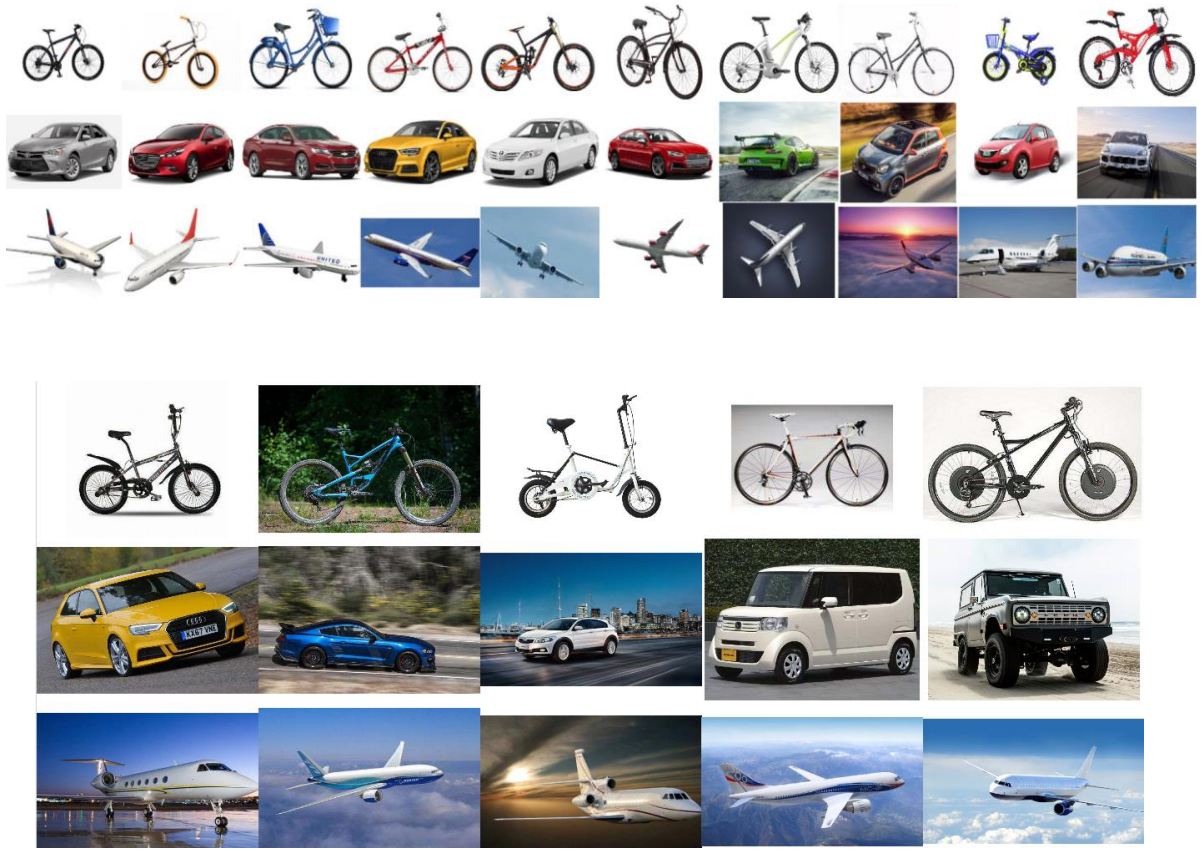
- 使用resize()函数可以将图像进行缩放，如果仅提供宽度或者高度之一，则将自动计算另一个尺寸并保持原始宽高比。以下代码实现将img、imgcol图像的宽、高均设置为256。

```
> # 调整图像尺寸
> img_resize <- resize(img,w = 256,h = 256)
> imgcol_resize <- resize(imgcol,w = 256,h = 256)
> par(mfrow=c(1,2))
> plot(img_resize)
> plot(imgcol_resize)
> par(mfrow=c(1,1))
```

导入本地小样本图像数据

- 在本地文件夹image1保存了自行车、汽车、飞机三种彩色图像，各有15张，一共有45张彩色图像。
- 使用EBImage包的readImage()函数将image1文件夹中的所有彩色图像读取到R中。

```
> library(keras)
> library(EBImage)
>
> setwd('../image1')
> # 读入图像数据
> pic1 <- paste0(rep(c('b','c','p'),each = 10),
+                1:10,'.jpg')
> train <- list()
> for (i in 1:length(pic1)) {train[[i]] <- readImage(pic1[i])}
> pic2 <- paste0(rep(c('b','c','p'),each = 5),
+                11:15,'.jpg')
> test <- list()
> for (i in 1:length(pic2)) {test[[i]] <- readImage(pic2[i])}
> # 绘制train数据的图像
> par(mfrow=c(3,10))
> for(i in 1:30) plot(train[[i]])
> par(mfrow=c(1,1))
> # 绘制test数据的图像
> par(mfrow=c(3,5))
> for(i in 1:15) plot(test[[i]])
> par(mfrow=c(1,1))
```



数据预处理

- 利用resize()函数将图像的像素高度、像素宽度都设置为100。使用combine()函数将图像合并为图像序列。使用to_categorical()函数对标签数据进行独热编码处理（One-Hot Encoding）。

```
> # 重新定义图像大小
> for (i in 1:30) {train[[i]] <- resize(train[[i]], 100, 100)}
> for (i in 1:15) {test[[i]] <- resize(test[[i]], 100, 100)}
> # 图像合并
> trainx <- combine(train)
> testx <- combine(test)
> dim(trainx)
[1] 100 100   3  30
> dim(testx)
[1] 100 100   3  15
> # 绘制经过形状处理的图像
> display(tile(trainx))
> display(tile(testx,nx = 5))
> # 调整维度顺序
> trainx <- aperm(trainx,c(4,1,2,3))
> testx <- aperm(testx,c(4,1,2,3))
> dim(trainx)
[1]  30 100 100   3
> dim(testx)
[1]  15 100 100   3
> trainLabels <- to_categorical(trainy)
> testLabels <- to_categorical(testy)
```

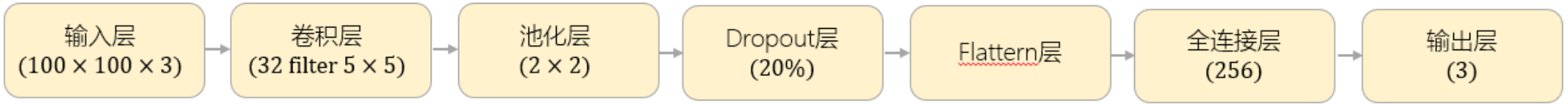
建立全连接神经网络模型识别小数据集图像

- 在这个例子中，使用两层完全连接的网络结构，使用ReLU作为前两层的激活函数，使用Softmax作为输出层的激活函数。第一个隐藏层有256个神经元，使用30000（100×100×3）个输入变量；第二个隐藏层有128个神经元，最后输出层有3个神经元来预测数据结果。

```
> # 构建全连接神经网络模型函数
> build_mlp <- function() {
+   model <- keras_model_sequential() %>%
+     layer_dense(units = 256, activation = 'relu', input_shape =
c(30000)) %>%
+     layer_dense(units= 128, activation = 'relu') %>%
+     layer_dense(units = 3, activation = 'softmax')
+   # Compile
+   model %>% compile(
+     loss = 'categorical_crossentropy',
+     optimizer = optimizer_rmsprop(),
+     metrics = 'accuracy')
+   model
+ }
> # 训练模型
> trainx _flattern <- array_reshape(trainx,dim =
c(nrow(train),100*100*3))
> testx _flattern <- array_reshape(testx,dim =
c(nrow(test),100*100*3))
> mlp_model <- build_mlp()
> history <- mlp_model %>% fit(
+   trainx _flattern,
+   trainLabels,
+   epochs = 50,
+   validation_split = 0.2)
```

模型在测试集的整体准确率仅有47%，效果欠佳。

建立简单卷积神经网络识别小数据集图像

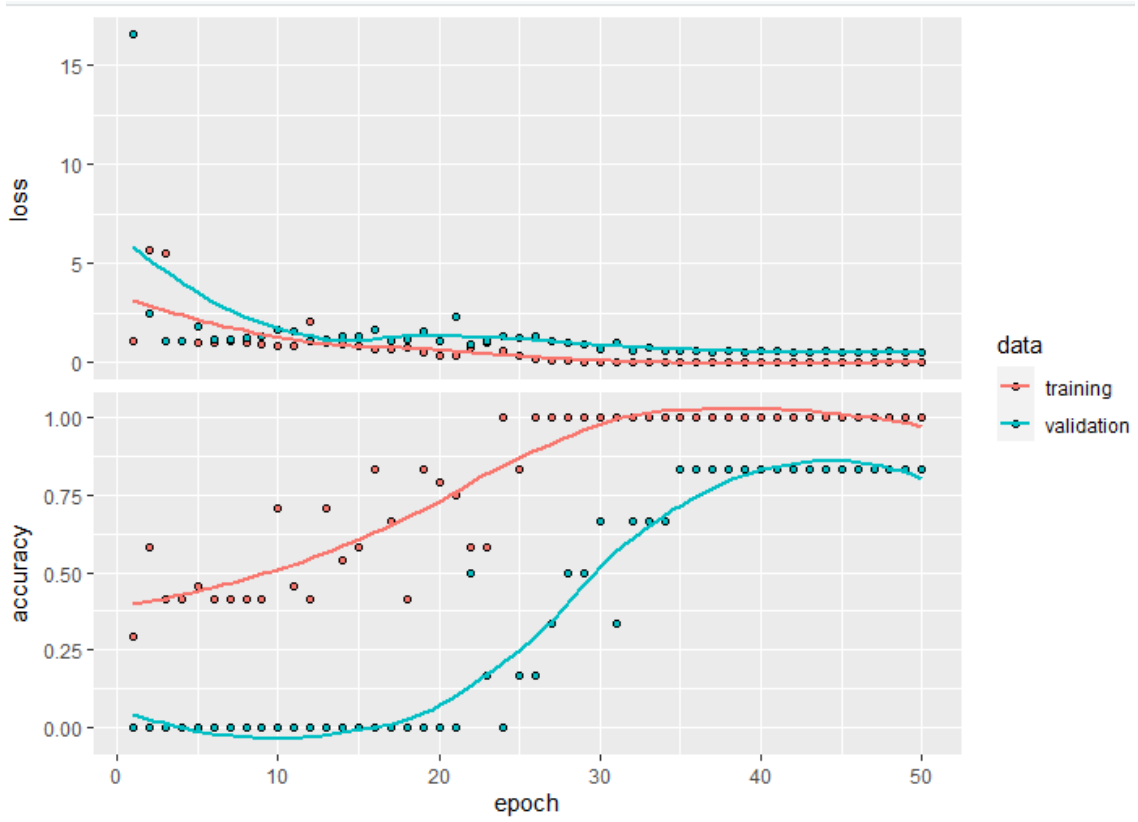



```
> # 构建simple_cnn模型函数
> build_simple_cnn <- function(X=trainx) {
+   model <- keras_model_sequential() %>%
+     layer_conv_2d(filters = 32,
+                   kernel_size = c(5,5),
+                   activation = 'relu',
+                   input_shape = dim(X)[-1]) %>%
+     layer_max_pooling_2d(pool_size = c(2,2)) %>%
+     layer_dropout(rate = 0.2) %>%
+     layer_flatten() %>%
+     layer_dense(units = 256, activation = 'relu') %>%
+     layer_dense(units = 3, activation = 'softmax')
+   # Compile
+   model %>% compile(
+     loss = 'categorical_crossentropy',
+     optimizer = optimizer_rmsprop(),
+     metrics = 'accuracy')
+   model
+ }
> # 训练模型
> simple_cnn_model <- build_simple_cnn()
> history <- simple_cnn_model %>%
+   fit(trainx,
+       trainLabels,
+       epochs = 50,
+       validation_split = 0.2)
```

简单卷积神经网络对测试集的预测准确率为74%，比前面全连接神经网络模型的47%有显著提升。

建立复杂卷积神经网络识别小数据集图像

```
> build_complex_cnn <- function(X=trainx) {
+   model <- keras_model_sequential() %>%
+     layer_conv_2d(filters = 32,
+                   kernel_size = c(3,3),
+                   activation = 'relu',
+                   input_shape = dim(X)[-1]) %>%
+     layer_conv_2d(filters = 32,
+                   kernel_size = c(3,3),
+                   activation = 'relu') %>%
+     layer_max_pooling_2d(pool_size = c(2,2)) %>%
+     layer_dropout(rate = 0.25) %>%
+     layer_conv_2d(filters = 64,
+                   kernel_size = c(3,3),
+                   activation = 'relu') %>%
+     layer_conv_2d(filters = 64,
+                   kernel_size = c(3,3),
+                   activation = 'relu') %>%
+     layer_max_pooling_2d(pool_size = c(2,2)) %>%
+     layer_dropout(rate = 0.25) %>%
+     layer_flatten() %>%
+     layer_dense(units = 256, activation = 'relu') %>%
+     layer_dense(units = 128, activation = 'relu') %>%
+     layer_dense(units = 3, activation = 'softmax')
+   # Compile
+   model %>% compile(
+     loss = 'categorical_crossentropy',
+     optimizer = optimizer_rmsprop(),
+     metrics = 'accuracy')
+   model
+ }
```



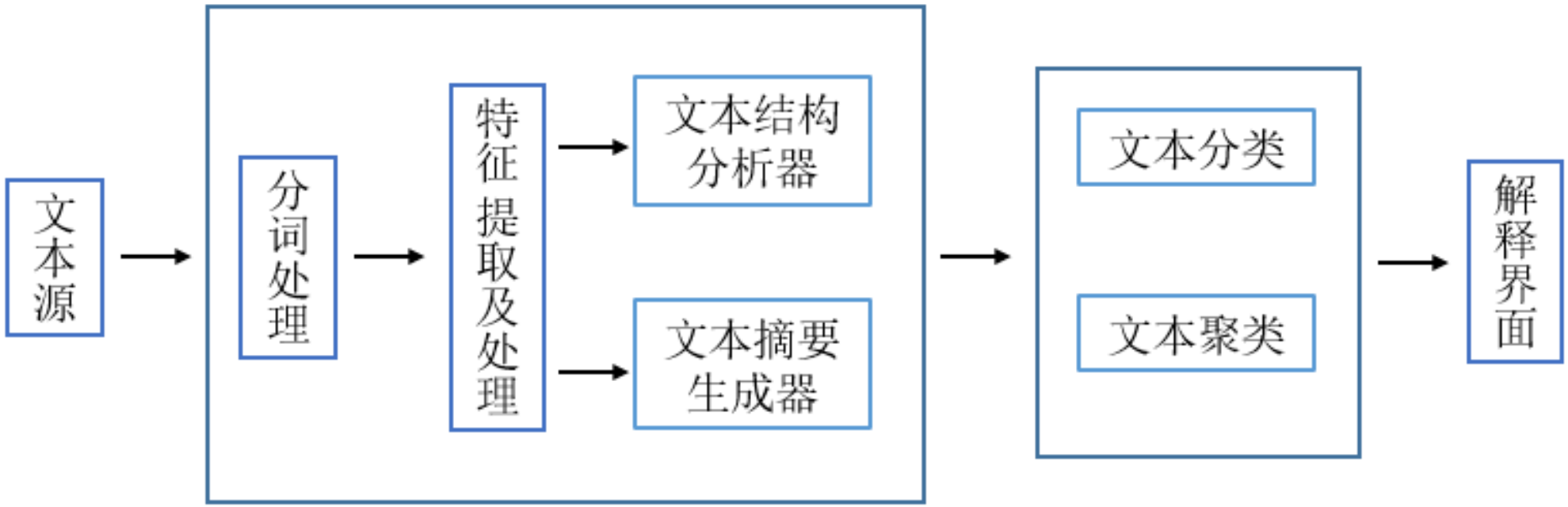
简单卷积神经网络对测试集的预测准确率为93%，比前面简单卷积神经网络的74%又有明显提升。

04

文本挖掘实战

文本挖掘流程

- 文本挖掘被描述为“自动化或半自动化处理文本的过程”，中文分词的结果就可以直接用来建立文本对象，最常用的结构就是词条与文档的关系矩阵，利用这个矩阵可以使用很多文本挖掘的算法来得到不同的结果，包括相似度计算、文本聚类、文本分类、主题模型、情感分析等。



相关R包简介

- R语言提供了丰富的扩展包来完成文本分词和建模。常用的有tm、tmcn、RwordSeg、jiebaR、tidytext等扩展包。

tm包	tmcn包	Rwordseg包	jiebaR包	tidytext包
tm包是文本挖掘的基础框架包，它有很多NLP相关的包。tm.plugin.dc、tm.plugin.mail、tm.plugin.factiva是针对tm包的扩展，可以用来分布式存储语料、处理邮件文本、获取Factiva语料。	tm包中文支持不是很友好，tmcn试图去解决这些问题。**中文编码：**各种编码的识别和UTF-8之间的转换；中文简体字和繁体字之间的切换；增强了tau包中的一些功能**中文语料资源：**例如GBK字符集及中文停止词等	Rwordseg是一个R环境下的中文分词工具，使用rJava调用Java分词工具Ansj。Ansj是一个开源的Java中文分词工具，基于中科院的ictclas中文分词算法，采用隐式马尔可夫模型（Hidden Markov Model，HMM）。	结巴分词（jiebaR）是一款高效的R语言中文分词包，底层使用的是C++，通过Rcpp进行高效调用。jiebaR支持最大概率法，隐式马尔科夫模型，索引模型，混合模型，共四种模式。	由于文本是非结构化数据，常规的数据处理及可视化方法不适合做文本分析。tidytext包提供了相对简单却实用的功能，其将文本视为单个单词的数据框架可以使我们轻松地对文本进行操作、汇总和可视化，这使得许多文本挖掘任务变得更容易高效。

新浪新闻分类实例：SPORT数据集

- 利用tmcn包的SPORT数据集进行中文文本分类实践。SPORT数据集一共收录了2357条体育新闻内容，其中变量class代表新闻内容所属类型，B为篮球，F为足球。我们将对SPORT数据集进行分析，并分别使用机器学习和深度学习建立分类模型对新闻所属类型进行预测。

SPORT 数据集一共有 2357 行 6 列，其中各列描述如下：

- id：新闻序号。
- time：新闻时间。
- title：新闻标题。
- class：新闻类型，“B”表示篮球类，“F”表示足球类。
- abstract：新闻摘要。
- content：新闻内容。

我们只需要利用 SPORT 数据集的变量 class 和 content，其中对 content 进行中文文本分词及探索，并以变量 class 为目标变量建立分类模型，对新闻所属类型进行预测。

新浪新闻分类实例：中文分词及整洁

- 在进行文本挖掘前，先对变量content的文章内容进行初步分析。首先利用tidytext包对SPORT数据集进行整洁，并计算篮球类（B）和足球类（F）的词频统计，并查看前十条记录。

```
> library(tidytext)
> library(jiebaR)
> wk <- worker(bylines = TRUE)
> tidy_data <- SPORT %>%
+   mutate(text = sapply(segment(content,wk),
+
function(x){paste(x,collapse = " ")})) %>%
+   unnest_tokens(word, content)
> # 查看篮球类新闻内容的词频统计
> b_word_count<- tidy_data %>%
+   filter(class=="B") %>%
+   count(word,sort = TRUE)
> # 查看足球类新闻内容的词频统计
> f_word_count <- tidy_data %>%
+   filter(class=="F") %>%
+   count(word,sort = TRUE)
```

```
> # 查看单词数量
> nrow(b_word_count)
[1] 9758
> b_word_count
# A tibble: 9,758 x 2
  word      n
  <chr> <int>
 1 的     9860
 2 场     4878
 3 在     4206
 4 了     4125
 5 赛     3557
 6 比赛   3042
 7 我们   2444
 8 季     2360
 9 我     2331
10 他     2207
# ... with 9,748 more rows
```

```
> # 查看单词数量
> nrow(f_word_count)
[1] 8061
> f_word_count
# A tibble: 8,061 x 2
  word      n
  <chr> <int>
 1 的     8788
 2 在     3343
 3 了     2595
 4 我们   2289
 5 他     1874
 6 比赛   1826
 7 赛     1718
 8 我     1697
 9 尔     1609
10 场     1531
# ... with 8,051 more rows
```

新浪新闻分类实例：词云展示

- 现在利用wordcloud2()函数对篮球新闻分词结果进行词云展示，由于前十记录的单词都是非篮球领域的专业术语，我们从第11条记录开始进行展示。

```
> library(wordcloud2)
> # 对篮球类新闻进行词云展示
> wordcloud2(b_word_count[11:nrow(b_word_count),],
+            shape = "star")
```


```
> library(wordcloud2)
> # 对足球类新闻进行词云展示
> wordcloud2(f_word_count[11:nrow(f_word_count),],
+            shape = "star")
```

新浪新闻分类实例：安装词库

- 为了提高篮球类和足球类新闻的分词效果，我们可以从搜狗网站下载相应的词库进行安装。将篮球【官方推荐】（https://pinyin.sogou.com/dict/cate/index/370）和足球【官方推荐】（https://pinyin.sogou.com/dict/cate/index/372）的词库下载到本地，利用Rwordseg包的installDict()函数将本地词库加载到词典中。

```
> library(Rwordseg)
> installDict(dictpath = "../dict/篮球【官方推荐】.scel",
+             dictname = "basketball")
2195 words were loaded! ... New dictionary 'basketball' was installed!
> installDict(dictpath = "../dict/足球【官方推荐】.scel",
+             dictname = "football")
8806 words were loaded! ... New dictionary 'football' was installed!
```

运行以上代码后，共安装了2195个与篮球相关的专业词语，8806个与足球相关的专业词云。

新浪新闻分类实例：重新分词

- 利用Rwordseg包的segmentCN()函数进行中文分词，运行以下代码分别对篮球类和足球类新闻进行分词，并利用unlist()函数将分词后的列表转为向量。
- 分词后，我们还需要对各词汇的词频进行统计，才能进行词云展示，此处将使用tmcn包的createWordFreq()函数进行词频统计。

```
> # 对新闻进行分词
> b_vec <- unlist(segmentCN(SPORT[SPORT$class=="B","content"]))
> f_vec <- unlist(segmentCN(SPORT[SPORT$class=="F","content"]))
> # 词频统计
> b_vec_count <- createWordFreq(b_vec,
+                               onlyCN = TRUE,
+                               nosymbol = TRUE,
+                               stopwords = stopwordsCN())
> f_vec_count <- createWordFreq(f_vec,
+                              onlyCN = TRUE,
+                              nosymbol = TRUE,
+                              stopwords = stopwordsCN())
```

新浪新闻分类实例：重新绘制词云

- 再次使用wordcloud2包的wordcloud2()函数进行词云展示，篮球类新闻词云结果。

```
> library(wordcloud2)
> # 对篮球类新闻进行词云展示
> wordcloud2(b_word_count[11:nrow(b_word_count),],
+            shape = "star")
```

```
> library(wordcloud2)
> # 对足球类新闻进行词云展示
> wordcloud2(f_word_count[11:nrow(f_word_count),],
+            shape = "star")
```

新浪新闻分类实例：利用机器学习进行文本分类

- 在对数据进行预处理后，将利用caret包的train()函数可以建立多种模型。
- 分别利用朴素贝叶斯、决策树C5.0、随机森林、梯度提升机、神经网络5种算法建立机器学习预测模型。

```
> set.seed(7)
> # 朴素贝叶斯
> modelNavieBayes <- train(class ~ .,
+                          data = train,
+                          method = "nb",
+                          trControl = control)
> # 决策树C5.0
> modelC50 <- train(class ~ .,
+                   data = train,
+                   method = "C5.0",
+                   trControl = control)
> # 随机森林
> modelRF <- train(as.factor(class) ~ .,
+                  data = train,
+                  method = "rf",
+                  trControl = control)># 梯度提升机
>modelGbm <- train(class ~ .,
+                   data = train,
+                   method = "gbm",
+                   trControl = control)
># 神经网络
>modelNnet <- train(class ~ .,
>                   data = train,
>                   method = "nnet",
>                   trControl = control)
```

```
> # 对测试集进行预测
> Accuracy <- NULL
> Sensitivity <- NULL
> Specificity <- NULL
> for(i in 1:5){
+   pred <-
predict(switch(i,modelNavieBayes,modelC50,modelRF,modelGbm,
modelNnet),
+                         newdata = test)
+   t <- table(test$class,pred)
+   Accuracy <- c(Accuracy,sum(diag(t))/sum(t))
+   Sensitivity<- c(Sensitivity,t(1)/(t(1)+t(3)))
+   Specificity <- c(Specificity,t(4)/(t(2)+t(4)))
+ }
> result <- data.frame(Model =
c("NB","C50","RF","GBM","NNET"),
+                      Accuracy,Sensitivity,Specificity)
> # 查看预测结果
> result
  Model    Accuracy    Sensitivity    Specificity
1    NB   0.8407643   0.7389706   0.9798995
2   C50   0.8407643   0.7977941   0.8994975
3    RF   0.8513800   0.7794118   0.9497487
4  GBM    0.8428875   0.7830882   0.9246231
5  NNET   0.8386412   0.7720588   0.9296482
```

从准确率来看，随机森林的效果最好（0.85138）；从灵敏性来看，决策树的效果最好（0.7978）；从特效性来看，朴素贝叶斯的效果最好（0.9799）。

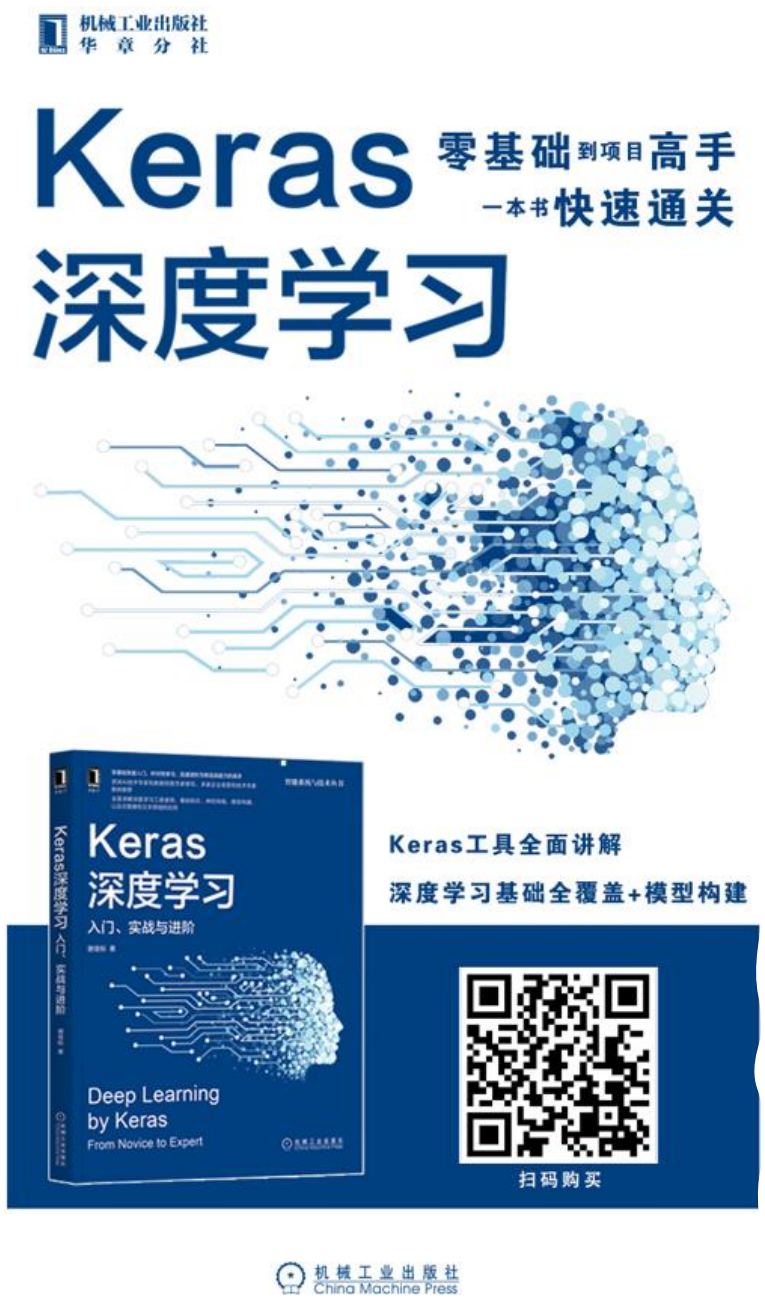
新浪新闻分类实例：利用深度学习进行文本分类

- 数据预处理：利用Rwordseg包的segmentCN()函数对新闻文本进行分词；使用Keras中的text_tokenizer()函数对文档进行令牌化；使用texts_to_sequences()函数可以将tokenizer对象的文本转换为序列；
- 文本分类预测：利用多层感知器、一维卷积神经网络、RNN、LSTM、GRU、双向LSTM等模型进行预测。

```
> 数据预处理
> library(tmcn)
> library(Rwordseg)
> data(SPORT)
> text <- SPORT$content
> d.vec <- segmentCN(text, returnType = "tm")
> library(keras)
> token <- text_tokenizer() %>%
+   fit_text_tokenizer(d.vec)
> X<- pad_sequences(seq, maxlen = 100)
> y <- ifelse(SPORT$class=="B",1,0)
> index <- caret::createDataPartition(y,p=0.8,list=FALSE)
> X_train <- X[index,] # 训练集自变量
> X_test <- X[-index,] # 测试集自变量
> y_train <- y[index] # 训练集因变量
> y_test <- y[-index] # 测试集因变量
```

```
> Accuracy <- NULL
> Sensitivity <- NULL
> Specificity <- NULL
> for(i in 1:6){
+   pred <- switch(i,model_mlp,model_cnn,model_rnn,
+                  model_lstm,model_gru,model_BiLSTM) %>%
+     predict_classes(X_test)
+   t <- table(y_test,pred)
+   Accuracy <- c(Accuracy,sum(diag(t))/sum(t))
+   Sensitivity <- c(Sensitivity,t[1]/(t[1]+t[3]))
+   Specificity <- c(Specificity,t[4]/(t[2]+t[4]))
+ }
> result <- data.frame(Model =
c("MLP","CNN","RNN","LSTM","GRU","Bi_LSTM"),
+                      Accuracy,Sensitivity,Specificity)
> result
     Model  Accuracy    Sensitivity   Specificity
1      MLP 0.9681529   0.9948718     0.9492754
2      CNN 0.9639066   1.0000000     0.9384058
3      RNN 0.9639066   0.9897436     0.9456522
4     LSTM 0.9808917   0.9794872     0.9818841
5      GRU 0.9872611   0.9948718     0.9818841
6 Bi_LSTM 0.8662420   0.9384615     0.8152174
```

从准确率来看，GRU的效果最好（0.987）；从灵敏性来看，CNN的效果最好（1.00）；从特效性来看，LSTM和GRU的效果相同（0.982）。



《Keras深度学习：入门、实战与进阶》——资深专家撰写，多位高管推荐；零基础入门深度学习并迅速成为实战高手；覆盖深度学习工具、基础、神经网络、模型构建、情感分析、预训练模型、文本挖掘、图像处理，配套视频+PPT。优惠购书戳：https://item.jd.com/13467020.html