

# Распознавание рукописных цифр

АЛЕКСАНДРА КУЗНЕЦОВА

**Аннотация.** Задача распознавания рукописного текста — одна из классических задач машинного обучения, к решению которой применялось такое количество алгоритмов, что она успешно может быть использована в качестве учебной.

В этой заметке мы будем учиться распознавать написанные от руки цифры.

**Ключевые слова:**  $k$  ближайших соседей,  $R$ , машинное обучение, распознавание образов.

Мы обратимся к одному из наиболее известных хранилищ, в котором собраны обработанные изображения цифр, разберём, как эти данные оттуда извлекать, а затем применим к ним алгоритм  $k$  ближайших соседей, используя  $R$ .

## 1 База данных MNIST

Источником данных нам послужит база «MNIST» (LeCun, Cortes, Burges, 2011), в которой хранятся 70 000 изображений цифр, написанных несколькими сотнями разных людей. Каждая картинка в этой базе обработана так, чтобы поместиться в квадратик размером  $28 \times 28$  пикселей. Каждый пиксель представлен числом от 0 до 255, где 0 соответствует белому цвету, а 255 — чёрному.

Все изображения поделены на две части: 60 000 относятся к учебной выборке, а 10 000 — к тестовой. По учебной выборке наш алгоритм будет настраивать свои параметры, а по тестовой мы будем оценивать качество классификации. Такое разбиение нужно для того, чтобы убедиться, что алгоритм не переобучился, то есть не настроен исключительно на учебные примеры и способен правильно классифицировать новые для него изображения.

Каждое из изображений в нашей задаче должно быть отнесено к одному из десяти классов — это цифры от 0 до 9. Для элементов обучающей выборки известно, к какому классу они принадлежат, поэтому настройка параметров классифицирующего алгоритма относится к области *обучения с учителем*. Это значит, что процесс обучения использует известные ответы и стремится за счёт выбора параметров сделать предсказания алгоритма максимально близкими к правильным (кстати, таким образом мы обучали все наши алгоритмы в курсе эконометрики).

Как говорит Борис Борисович, «я-то думала, что .csv-файлы на деревьях, как булки, растут», но всё оказалось совсем не так. Для того чтобы

прочитать IDX-файл, в котором хранятся данные с изображениями цифр в базе «MNIST», мне потребовались Google и кандидат физико-математических наук. Проблема заключается в том, что данные хранятся в этой базе в бинарном виде, и их не открыть в привычных нам приложениях. Вот как это можно сделать в **R**.

(1) Скачав данные с сайта, загружаем файл с учебной выборкой командой `file`:

```
to.read <- file("train-images-idx3-ubyte", "rb")
```

(2) Данные устроены таким образом, что в самом начале мы читаем заголовок из четырёх чисел. Эти первые четыре числа содержат информацию о размерах выборки, упомянутых выше. Считываем их из полученного объекта `to.read` с помощью функции для чтения бинарных данных `readBin`.

```
readBin(to.read, integer(), n = 4, endian="big")
## [1] 2051 60000 28 28
```

(3) Далее для каждой картинке следуют  $28 \times 28$  байт, содержащие информацию о цвете каждого пикселя (числа от 0 до 255) и записанные из изображения построчно. Получаем большой массив `TRAIN` из 60 000 таблиц размером  $28 \times 28$ : берём первые 28 чисел и кладём их в первый столбец первой таблицы, продолжаем, пока не дойдём до 28-го столбца, затем приступаем к следующей таблице — и так до конца.

Нужно обратить внимание, что изначально данные были разложены в строку, но мы только что разложили их по столбцам, для того чтобы позже было удобнее выводить рисунок.

```
TRAIN <- array(data = NA, dim = c(28,28,60000))

for(i in 1:60000)
{
  TRAIN[,i] <- matrix(readBin(to.read,integer(), size = 1,
    n = 28*28, signed = FALSE, endian = "big"), 28, 28);
}
```

(4) Перед тем как начинать работать с данными и оценивать по ним какие-либо модели, обычно бывает полезно взглянуть на них и построить описательные статистики. В данном случае для этой цели может послужить сама картинка.

```
layout(matrix(c(1:36), 6, 6, byrow = TRUE),
        widths = lcm(rep(2.5,36)), heights = lcm(rep(2.5,36)))
par(mar=c(0,0,0,0))

for(i in 1:36)
{
  image(TRAIN[,28:1,i])
}
```



Рис. 1. Пример входных данных

Нарисуем цифры по первым 36 таблицам из массива **TRAIN**. Нужно обратить внимание, что, нарисуй мы сейчас всё как есть, мы бы получили перевёрнутые цифры (это можно проверить), потому что функция **image** будет соотносить первый столбец с нулевой ординатой на картинке, но мы помним, что первый столбец соответствовал верхней строке рисунка. Для того чтобы избежать этой проблемы, будем рисовать столбцы в обратном порядке, от 28-го к 1-му: **TRAIN[,28:1,]**.

(5) Описанную выше процедуру повторяем, чтобы считать метки классов для учебной выборки. Заголовок в данном случае состоит из двух чисел, а данные должны восприниматься **R**’ом не как числа, а как категории: этого можно добиться с помощью команды **as.factor**.

```

to.read <- file("train-labels-idx1-ubyte", "rb")
header <- readBin(to.read, integer(), n=2, endian="big")
Train_labels <- readBin(to.read, integer(), size = 1,
                        n = 60000, signed = FALSE, endian="big")
Train_labels <- as.factor(Train_labels)

```

Итак, на данном этапе мы являемся обладателями учебной выборки (но умеем читать и многое другое) и великолепной картинкой, что означает, что пора бы с ними что-нибудь да сделать.

## 2 Алгоритм $k$ ближайших соседей

Одним из простейших алгоритмов классификации является метод  $k$  ближайших соседей ( $k$  Nearest Neighbours, kNN). Основной идеей данного алгоритма является то, что объект, как правило, находится в окружении объектов своего же класса. Таким образом, чтобы классифицировать новый объект, мы должны посмотреть на  $k$  ближайших к нему объектов из учебной выборки и отнести его к тому классу, который среди них чаще встретился.

Первый вопрос, который перед нами встаёт, — это выбор функции расстояния между объектами выборки. В простом случае, когда объекты представлены в виде числовых векторов, пользуются простой евклидовой метрикой,

$$\text{то есть } \rho(a, b) = \sqrt{\sum_{k=1}^m (a_k - b_k)^2}.$$

Второй проблемой является выбор значения  $k$ : сколько объектов нам нужно посмотреть, чтобы наиболее точно классифицировать наш? Единого правила для того, чтобы выбрать  $k$ , не существует, однако нужно учитывать, что при слишком маленьких  $k$  алгоритм будет неустойчивым, а при слишком больших начнёт подстраиваться к шуму и терять обобщающую способность. Конкретное значение  $k$  можно определить опытным путём: попробовать диапазон значений и посмотреть, какое подходит лучше.

### 2.1 Существующие реализации

Для реализации алгоритма  $k$  ближайших соседей в **R** есть как минимум три пакета: **kknn**, **FNN** и **RWeka**, причём в двух последних есть ещё и готовые базы данных.

Пакет **kknn** реализует алгоритм kNN с весами, где голоса «соседей» не равнозначны, а входят в общую сумму с определёнными весами. Мы его реализовывать не будем.

Воспользуемся функцией `install.packages(c("FNN" "RWeka"))`, чтобы загрузить пакеты.

## 2.2 Пакет FNN

Воспользуемся теперь пакетом FNN (*FNN: Fast Nearest Neighbor Search Algorithms and Applications*, 2013), предварительно проделав с данными для тестовой выборки то же, что и с учебными, в итоге получив два больших массива, TRAIN и TEST, а также два вектора ответов, Trainlabels и Testlabels.

Функция knn пакета FNN в качестве аргументов принимает матрицы учебных и тестовых данных, а также вектор ответов для тренировочной выборки, а на выходе отдаёт предсказанную классификацию для тестовой выборки. Эта функция сразу и обучается, и предсказывает — очень удобно.

Сейчас наши данные выглядят не так, как нужно этой функции, поэтому преобразуем их в матрицу, содержащую значения цвета всех пикселей, размещённых в одну строку для каждой картинке. Воспользуемся функцией as.vector, которая будет по очереди брать столбцы из матрицы и выкладывать их в одну строку. Кроме того, для примера мы будем использовать только первую тысячу наблюдений из учебной выборки и пятьсот из тестовой. Если читатель желает получить более точный классификатор, то в этом месте следует использовать все 60 000 и 10 000 соответственно.

```
n <- 1000
train = matrix(data = NA, nrow = n, ncol = 28*28 )

for (i in 1:n)
{
  train[i,] <- as.vector(TRAIN[,i])
}
train_labels <- Train_labels[1:n]
```

Теперь воспользуемся функцией knn, чтобы классифицировать объекты тестовой выборки, и запишем результаты в вектор results. Возьмём, например,  $k = 10$ . Преимуществом функции является высокая скорость подсчёта предсказаний.

```
library(FNN)
results <- (0:9)[knn(train, test, train_labels, k = 10,
                     algorithm = "cover_tree")]
```

Посмотрим на долю ошибок, которые мы допустили при классификации. Она достаточно высока, но это можно объяснить тем, что мы использовали слишком маленькое для такого алгоритма число наблюдений.

```
errors <- (sum(results != test_labels))/m
errors

## [1] 0.2
```

### 2.3 Пакет RWeka

Пакет **RWeka** (Hornik, Buchta, Zeileis, 2009) представляет собой целый набор алгоритмов машинного обучения, среди которых есть и нужный нам. В нём замечательно то, что он сам подбирает оптимальное в нашем случае значение  $k$  из заданного диапазона, а потом оценивает полученный классификатор с помощью пятикратной кросс-валидации. Это значит, что он поделит выборку на пять кусочков и по очереди будет использовать их в качестве тестовых, чтобы надёжнее оценить классификацию, исключив возможность того, что хорошие или плохие предсказания получились под влиянием какого-либо конкретного набора данных.

Однако при загрузке пакета нужно убедиться, что на компьютере установлена среда Java, и отдавать себе отчёт, что поиск подходящих параметров и пятикратная оценка классификатора будет требовать значительного времени.

Используем функцию **IBk**. Для этого добавим слева к нашей матрице **train** столбец с ответами и преобразуем матрицу в объект **data.frame**. Выражение  $K = 10$  указывает на то, что алгоритм будет перебирать все значения от 1 до 10. Функция **evaluate\_weka\_classifier** оценивает качество результатов и показывает, какие значения были классифицированы правильно в результате пятикратного разбиения выборки на кусочки и последующего усреднения, а какие — нет.

Обратите внимание на Confusion matrix, которая показывает, с какими именно классами возникали ошибки: можно заметить, что пятёрки и восьмёрки классифицировались как всё что угодно.

```
library(RWeka)

train_weka <- data.frame(train_labels, train)
train_weka[,1] <- as.factor(train_weka[,1])
classifier <- IBk(train_labels~., data = train_weka,
                  control = Weka_control(K = 10, X=TRUE))

classifier

## IB1 instance-based classifier
## using 1 nearest neighbour(s) for classification

evaluate_Weka_classifier(classifier, numFolds = 5)

## === 5 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances      876      87.6    %
## Incorrectly Classified Instances   124      12.4    %
```

```
## Kappa statistic                0.862
## Mean absolute error            0.0267
## Root mean squared error        0.1565
## Relative absolute error        14.8567 %
## Root relative squared error    52.2083 %
## Coverage of cases (0.95 level) 87.6 %
## Mean rel. region size (0.95 level) 10 %
## Total Number of Instances      1000
##
## === Confusion Matrix ===
##
##      a   b   c   d   e   f   g   h   i   j   <-- classified as
##  91    0    0    0    0    1    4    0    0    1 |   a = 0
##    0 114    0    1    0    0    0    1    0    0 |   b = 1
##    2    6   81    1    1    1    2    3    1    1 |   c = 2
##    0    0    3   83    0    2    1    1    1    2 |   d = 3
##    0    3    0    0   89    0    1    0    0   12 |   e = 4
##    0    1    1    5    0   76    3    1    3    2 |   f = 5
##    1    3    0    0    0    0   89    0    1    0 |   g = 6
##    0    5    0    0    2    3    0 100    1    6 |   h = 7
##    0    3    3    3    1    6    1    0   68    2 |   i = 8
##    1    0    0    1    9    0    1    1    2   85 |   j = 9
```

Теперь предскажем значения для тестовой выборки с помощью функции `predict` и посмотрим на долю ошибок.

```
test_weka = data.frame(test)
names(test_weka) = names(train_weka)[-1]

predictions <- predict(classifier, newdata = test_weka)

errors <- (sum(predictions != test_labels))/m
errors

## [1] 0.174
```

### 3 Заключение

Таким образом, мы рассмотрели интересную задачу и отличную, заботливо собранную базу данных, применили к ним один из самых первых и простых алгоритмов машинного обучения, который, однако, часто показывает очень хорошие результаты. Но это был один только пример, а в качестве инструментов в данном случае могут быть использованы разнообразнейшие алгоритмы.

Кроме того, можно подумать над тем, как преобразовать сами изображения для улучшения качества классификации. Удачи!

### Список литературы

- FNN: Fast Nearest Neighbor Search Algorithms and Applications / A. Beygelzimer [и др.]. — 2013. — R package version 1.1.
- Hornik K., Buchta C., Zeileis A.* Open-Source Machine Learning: R Meets Weka // Computational Statistics. — 2009. — Т. 24, № 2. — С. 225–232.
- LeCun Y., Cortes C., Burges C. J. C.* The MNIST database of handwritten digits. — 25 дек. 2011. — URL: <http://yann.lecun.com/exdb/mnist/index.html>.