BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ

MÔN HỌC: DATA MINING

Đề tài: Digit Recognizer

GVHD:	ThS. Quách Đình Hoàng	
SVTH1:	Trần Trọng Nghĩa	(17133040)
SVTH2:	Vũ Anh Thái Dương	(17133010)
SVTH3:	Nguyễn Quang Nhật	(17133043)
SVTH4:	Đỗ Viết Hải	(17133017)

THÀNH PHỐ HỒ CHÍ MINH, 1/2021

Đề Tài Digit Recognizer

Nhóm 8 (Nguyễn Quang Nhật, Trần Trọng Nghĩa, Vũ Anh Thái Dương, Đỗ Viết Hải)

2021/01/25

1. Tóm tắt đề tài Nhận dạng chữ số viết tay (https://www.kaggle.com/c/digit-recognizer)

Trong đề tài này, mục tiêu là xác định chính xác các chữ số từ bộ dữ liệu gồm hàng chục ngàn hình ảnh viết tay từ bộ dữ liệu gốc MNIST được chúng tôi chia thành 2 tệp train và test để thuận lợi cho quá trình huấn luyện và dự đoán.

Chúng tôi thử nghiệm huấn luyện và đưa ra dự đoán với 3 thuật toán nổi tiếng trong học máy là K-nearest neighbors(KNN), random forest và SVM. Qua thực nghiệm chúng tôi rút ra được mô hình KNN có hiệu suất dự đoán cao nhất(), sau đó đến random forest và cuối cùng là SVM. Qua thực nghiệm chúng tôi cũng nhận thấy rằng các thuật toán gặp khó khăn ở các chữ số 8 và 3 hoặc 9 và 4. Link mô tả kết quả trên Kagger: https://www.kaggle.com/haidoviet/digit-recognizer-k-nearest-neighbors-knn

2. Giới thiệu

Nhận dạng chữ số viết tay là cần thiết và được ứng dụng rộng rãi trong nhiều lĩnh vực như nhận dạng các chữ số trên chi phiếu ngân hàng, mã số trên bì thư của dịch vụ bưu chính, hay các chữ số trên các biểu mẫu nói chung. Vấn đề nhận dạng chữ viết tay là một thách thức lớn đối với các nhà nghiên cứu. Bài toán lớn luôn đặt ra vì sự phức tạp của việc nhận dạng chữ viết phụ thuộc nhiều vào phong cách viết và cách thể hiện ngôn ngữ của người viết. Chúng ta không thể luôn luôn viết một ký tự chính xác theo cùng một cách. Do vậy, xây dựng hệ thống nhận dạng chữ viết có thể nhận dạng bất cứ ký tự nào một cách đáng tin cậy trong tất cả các ứng dụng là điều không dễ dàng.

3. Dữ liệu

Input:

Mỗi hình ảnh có chiều cao 28 pixel và chiều rộng 28 pixel, với tổng số 784 pixel. Mỗi pixel có một giá trị pixel duy nhất được liên kết với nó, biểu thị độ sáng hoặc tối của pixel đó, với số cao hơn có nghĩa là tối hơn. Giá trị pixel này là một số nguyên nằm trong khoảng từ 0 đến 255.

Tập dữ liệu huấn luyện, (train.csv), có 785 cột. Cột đầu tiên, được gọi là "label" mang các giá trị từ 0 đến 9, là chữ số được vẽ bởi người dùng. Các cột còn lại chứa các giá trị pixel của hình ảnh.

Tập dữ liệu kiểm tra, (test.csv), có 784 cột chứa các giá trị pixel thể hiện độ xám của hình ảnh.

Output:

Output của bài toán là kết quả dự đoán các chữ số trong tập test sử dụng các thuật toán: KNN, Random Forest và SVM. Kết quả là tệp csv gồm 2 cột: ImageID(từ 1-28000) và label(các chữ số từ 0-9)

4 Giải pháp

4.1 Thuật toán KNN K-nearest neighbors là thuật toán học máy có giám sát, không quá khó và dễ triển khai. Thường được dùng trong các bài toán phân loại và hồi quy.

Thuật toán KNN cho rằng những dữ liệu tương tự nhau sẽ tồn tại gần nhau trong một không gian, từ đó công việc của chúng ta là sẽ tìm k điểm gần với dữ liệu cần kiểm tra nhất. Việc tìm khoảng cách giữa 2 điểm củng có nhiều công thức có thể sử dụng, tùy trường hợp mà chúng ta lựa chọn cho phù hợp. Đây là một trong các cách cơ bản để tính khoảng cách 2 điểm dữ liệu x, y có k thuộc tính:

Khoảng cách Euclidean được định nghĩa như sau:

$$\sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$$

4.2 Thuật toán SVM (Support Vector Machine) Support Vector Machine (SVM) là một thuật toán thuộc nhóm Supervised Learning (học có giám sát) dùng để phân chia dữ liệu (classification) thành các nhóm riêng biệt. Thuật toán để ánh xạ bộ data đó vào không gian nhiều chiều hơn (n chiều), từ đó tìm ra siêu mặt phẳng (hyperplane) để phân chia.

Margin là khoảng cách giữa siêu phẳng (trong trường hợp không gian 2 chiều là đường thẳng) đến 2 điểm dữ liệu gần nhất tương ứng với 2 phân lớp. Nó được tính là khoảng cách gần nhất từ 1 điểm tới mặt đó (bất kể điểm nào trong hai classes) bằng công thức như sau:

$$\text{margin } = \min_n \frac{y_n \left(\mathbf{w}^T \mathbf{x}_n + b\right)}{\|\mathbf{w}\|_2}$$

Với x_n và y_n à tọa độ của một điểm dữ liệu bất kì.

Bài toán tối ưu trong SVM chính là bài toán tìm w và b sao cho margin này đạt giá trị lớn nhất, công thức này được định nghĩa như sau:

$$(\mathbf{w}, b) = \arg\max_{\mathbf{w},b} \left\{ \min_n \frac{y_n \left(\mathbf{w}^T \mathbf{x}_n + b\right)}{\|\mathbf{w}\|_2} \right\} = \arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|_2} \min_n y_n \left(\mathbf{w}^T \mathbf{x}_n + b\right) \right\}$$

4.3. Thuật toán random forest

Random forest là một thuật toán học có giám sát. Như tên gọi của nó, random forest sử dụng các cây (tree) để làm nền tảng. Random forest là một tập hợp của các decision tree, mà mỗi cây được chọn theo một thuật toán dựa vào ngẫu nhiên.

Random forest có điểm mạnh:

Random forest algorithm có thể sử dụng cho cả bài toán classification và regression.

Random forest làm việc được với dữ liệu thiếu giá trị.

Khi forest có nhiều cây hơn, chúng ta có thể tránh được việc **overfitting** với tập dữ liệu.

Có thể tạo mô hình cho các giá trị phân loại.

Mã giả cho hoạt động của thuật toán random forest:

Bước 1: Chọn ngẫu nhiên "k" features từ tập "m" **features**. (lưu ý k « m).

Bước 2: Từ tập "k" **features**, tính toán ra node "d" là tốt nhất cho node phân loại.

Bước 3: Chia các node con theo node tốt nhất vừa tìm được.

Bước 4: Lặp lại bước 1-3 cho đến khi đạt đến k node.

Bước 5: Lặp lại bước 1-4 để tạo ra "n" cây.

Sau các bước trên, chúng ta đã tạo ra được một mô hình random rorest. Thuât toán random forest precdiction (dự đoán) sẽ được thực hiện qua các bước sau:

Bước 1: Lấy các test features và sử dụng các Cây quyết định đã tạo ra để dự đoán kết quả, lưu nó vào một danh sách.

Bước 2: Tinh toán số lượng vote trên toàn bộ forest cho từng kết quả.

Bước 3: Lấy kết quả có số lượng vote lớn nhất làm kết quả cuối cho mô hình.

4.4. Độ đo Confusion Matrix Confusion Matrix là một bảng được sử dụng để mô tả hiệu suất của một mô hình phân loại. Với các đầu ra được thể hiện trong ví dụ ở hình dưới.

	Reference	
Predicted	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

$$Sensitivity = A/(A+C)$$

$$Specificity = D/(B+D)$$

$$Prevalence = (A+C)/(A+B+C+D)$$

$$PPV = (sensitivity * prevalence)/((sensitivity * prevalence) + ((1-specificity) * (1-prevalence)))$$

$$NPV = (specificity * (1-prevalence))/(((1-sensitivity) * prevalence) + ((specificity) * (1-prevalence)))$$

$$DetectionRate = A/(A+B+C+D)$$

$$DetectionPrevalence = (A+B)/(A+B+C+D)$$

$$BalancedAccuracy = (sensitivity + specificity)/2$$

$$Precision = A/(A+B)$$

$$Recall = A/(A+C)$$

Figure 1: A caption

**** 4.5. k-Fold Cross-Validation**** Cross validation là một kỹ thuật lấy mẫu để đánh giá mô hình học máy trong trường hợp dữ liệu không được dồi dào cho lắm.

Tham số quan trọng trong kỹ thuật này là k, đại diện cho số nhóm mà dữ liệu sẽ được chia ra. Vì lý do đó, nó được mang tên k-fold cross-validation. Khi giá trị của k được lựa chọn, người ta sử dụng trực tiếp giá trị đó trong tên của phương pháp đánh giá. Ví dụ với k=10, phương pháp sẽ mang tên 10-fold cross-validation.

Kỹ thuật này thường bao gồm các bước như sau:

```
1. Xáo trộn dataset một cách ngẫu nhiên

2. Chia dataset thành k nhóm

3. Với mỗi nhóm:

  - Sử dụng nhóm hiện tại để đánh giá hiệu quả mô hình
```

- Các nhóm còn lại được sử dụng để huấn luyện mô hình

- Huấn luyện mô hình

- Đánh giá và sau đó hủy mô hình

- Tổng hợp hiệu quả của mô hình dựa từ các số liệu đánh giá

Một lưu ý quan trọng là mỗi mẫu chỉ được gán cho duy nhất một nhóm và phải ở nguyên trong nhóm đó cho đến hết quá trình. Các tiền xử lý dữ liệu như xây dựng vocabulary chỉ được thực hiện trên tập huấn luyện đã được chia chứ không được thực hiện trên toàn bộ dataset. Việc hủy mô hình sau mỗi lần đánh giá là bắt buộc, tránh trường hợp mô hình ghi nhớ nhãn của tập test trong lần đánh giá trước. Các lỗi thiết lập này dễ xảy ra và đều dẫn đến kết quả đánh giá không chính xác (thường là tích cực hơn so với thực tế).

Kết quả tổng hợp thường là trung bình của các lần đánh giá. Ngoài ra việc bổ sung thông tin về phương sai và độ lệch chuẩn vào kết quả tổng hợp cũng được sử dụng trong thực tế.

5. Thực nghiệm và kết quả

```
library(readr)
train <- read_csv("D:/KhaiPhaDuLieu/pro_dm/nhan_dang_chu_so_r/train.csv")
```

5.1. Quá trình kiểm tra và tiền xử lý dữ liệu

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
test <- read_csv("D:/KhaiPhaDuLieu/pro_dm/nhan_dang_chu_so_r/test.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
dim(train) ; dim(test)
```

```
## [1] 42000   785
```

```
## [1] 28000   784
```

Tập dữ liệu train có thêm một cột, được gọi là "label", chứa "chữ số" được đại diện bởi 784 cột khác. Các cột 784 (28x28 ô) chứa giá trị từ 0 đến 255. Giá trị này cho chúng ta biết độ đậm hay nhạt của mỗi ô. Đầu tiên ta phải chuyển đổi cột "label" thành giá trị "factor".

```
train[, 1] <- as.factor(train[, 1]$label)  # As Category
```

Giá trị của các cột:

```
head(sapply(train[1,], class))
```

```
##     label    pixel0    pixel1    pixel2    pixel3    pixel4
##  "factor" "numeric" "numeric" "numeric" "numeric" "numeric"
```

Chúng tôi nhận thấy nhiều cột có ít giá trị và hầu hết các giá trị chỉ bằng 0; Cho nên phương sai của chúng gần như bằng không. Nó không tốt cho quá trình dự đoán và chúng tôi sẽ thực hiện loại bỏ chúng.

```
train_orig <- train
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
nzv.data <- nearZeroVar(train[,-1],saveMetrics=T,freqCut=10000/1,uniqueCut =1/7)
sum(nzv.data$nzv)
```

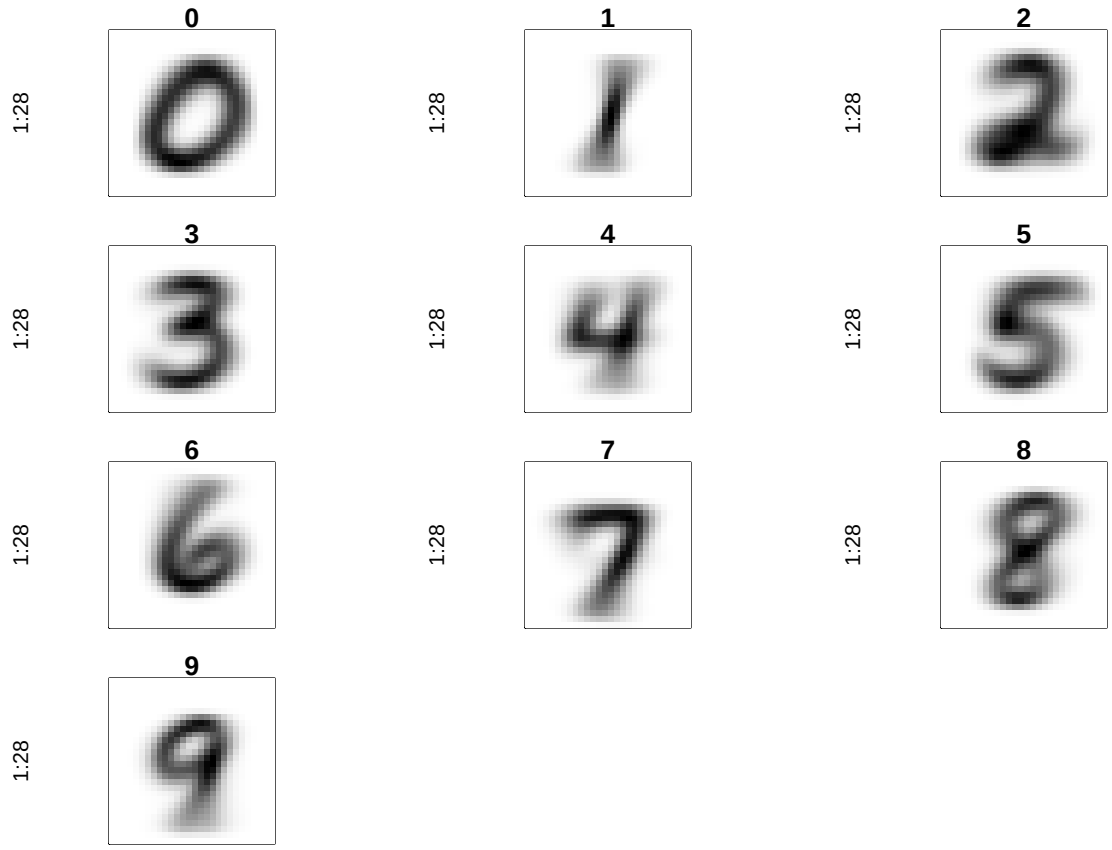
```
## [1] 208
```

```
drop.cols <- rownames(nzv.data)[nzv.data$nzv == TRUE]
train <- train[,!names(train) %in% drop.cols]
```

Bây giờ, chúng ta hãy thực hiện một số phân tích để trực quan hóa dữ liệu và EDA:(Tham khảo từ nguồn: http://rstudio-pubs-static.s3.amazonaws.com/6287_c079c40df6864b34808fa7ecb71d0f36.html).

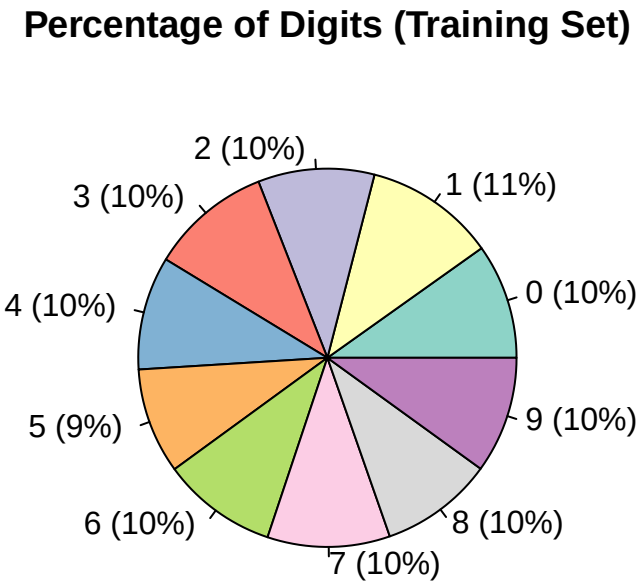
```
library(RColorBrewer)
BNW <- c("white", "black")
CUSTOM_BNW <- colorRampPalette(colors = BNW)
par(mfrow = c(4, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
images_digits_0_9 <- array(dim = c(10, 28 * 28))
for (digit in 0:9) {
  images_digits_0_9[digit + 1, ] <- apply(train_orig[train_orig[, 1] == digit, -1], 2, sum)
  images_digits_0_9[digit + 1, ] <- images_digits_0_9[digit + 1, ]/max(images_digits_0_9[digit + 1, ]) *
  z <- array(images_digits_0_9[digit + 1, ], dim = c(28, 28))
  z <- z[, 28:1]
  image(1:28, 1:28, z, main = digit, col = CUSTOM_BNW(256))
}
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



Chúng tôi nhận ra có nhiều hình mờ và các hình này khả năng bị nhầm lẫn nhiều hơn. Ví dụ: số 9 và số 4 hoặc số 8 và số 3. Điều đó có nghĩa là có khả năng cao hơn dự đoán sai về những con số như vậy. Chúng tôi sẽ khám phá điều này chi tiết hơn khi chúng tôi dự đoán tập dữ liệu của mình. Tỉ lệ của mỗi chữ số trong tập train là:

```
CUSTOM_BNW_PLOT <- colorRampPalette(brewer.pal(10, "Set3"))
LabTable <- table(train_orig$label)
par(mfrow = c(1, 1))
percentage <- round(LabTable/sum(LabTable) * 100)
labels <- paste0(row.names(LabTable), " (", percentage, "%) ")
pie(LabTable, labels = labels, col = CUSTOM_BNW_PLOT(10), main = "Percentage of Digits (Training Set)")
```



Qua biểu đồ ta có thể nhận thấy tất cả các chữ số đều đóng góp gần như bằng nhau vào tập dữ liệu. Chúng tôi chọn KNN (K-near k-Nearest Neighbors) là mô hình đầu tiên phù hợp cho các bài toán phân loại. Hãy xem KNN hoạt động như thế nào trong Phân loại nhận dạng số. Để tăng tốc độ chạy mô hình ban đầu của chúng tôi, tôi chỉ sử dụng 10% của toàn bộ tập train (~ 4200) để đào tạo và sử dụng kích thước kết quả(label) tương tự để thực hiện đánh giá.

```
set.seed(1234)
library(class)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
trainIndex <- createDataPartition(train$label, p = 0.1, list = FALSE, times = 1)
allindices <- c(1:42000)
training <- train[trainIndex,]
```

```
validating <- train[-trainIndex,]
vali0_index <- allindices[! allindices %in% trainIndex]
validIndex <- createDataPartition(validating$label, p = 0.11, list = FALSE, times = 1)
validating <- validating[validIndex,]
original_validindex <- vali0_index[validIndex]
training
```

```
## # A tibble: 4,206 x 577
##    label pixel41 pixel42 pixel43 pixel63 pixel64 pixel65 pixel66 pixel67 pixel68
##    <fct>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 4           0       0       0       0       0       0       0       0       0
##  2 6           0       0       0       0       0       0       0       0       0
##  3 7           0       0       0       0       0       0       0       0       0
##  4 1           0       0       0       0       0       0       0     191     191
##  5 4           0       0       0       0       0       0       0       0       0
##  6 1           0       0       0       0       0       0       0       0       0
##  7 6           0       0       0       0       0       0       3      45     222
##  8 4           0       0       0       0       0       0       0       0       0
##  9 6           0       0       0       0       0       0       0       0       0
## 10 7           0       0       0       0       0       0       0       0       0
## # ... with 4,196 more rows, and 567 more variables: pixel69 <dbl>,
## #   pixel70 <dbl>, pixel71 <dbl>, pixel72 <dbl>, pixel73 <dbl>, pixel74 <dbl>,
## #   pixel75 <dbl>, pixel76 <dbl>, pixel77 <dbl>, pixel78 <dbl>, pixel90 <dbl>,
## #   pixel91 <dbl>, pixel92 <dbl>, pixel93 <dbl>, pixel94 <dbl>, pixel95 <dbl>,
## #   pixel96 <dbl>, pixel97 <dbl>, pixel98 <dbl>, pixel99 <dbl>, pixel100 <dbl>,
## #   pixel101 <dbl>, pixel102 <dbl>, pixel103 <dbl>, pixel104 <dbl>,
## #   pixel105 <dbl>, pixel106 <dbl>, pixel107 <dbl>, pixel108 <dbl>,
## #   pixel116 <dbl>, pixel117 <dbl>, pixel118 <dbl>, pixel119 <dbl>,
## #   pixel120 <dbl>, pixel121 <dbl>, pixel122 <dbl>, pixel123 <dbl>,
## #   pixel124 <dbl>, pixel125 <dbl>, pixel126 <dbl>, pixel127 <dbl>,
## #   pixel128 <dbl>, pixel129 <dbl>, pixel130 <dbl>, pixel131 <dbl>,
## #   pixel132 <dbl>, pixel133 <dbl>, pixel134 <dbl>, pixel135 <dbl>,
## #   pixel136 <dbl>, pixel137 <dbl>, pixel144 <dbl>, pixel145 <dbl>,
## #   pixel146 <dbl>, pixel147 <dbl>, pixel148 <dbl>, pixel149 <dbl>,
## #   pixel150 <dbl>, pixel151 <dbl>, pixel152 <dbl>, pixel153 <dbl>,
## #   pixel154 <dbl>, pixel155 <dbl>, pixel156 <dbl>, pixel157 <dbl>,
## #   pixel158 <dbl>, pixel159 <dbl>, pixel160 <dbl>, pixel161 <dbl>,
## #   pixel162 <dbl>, pixel163 <dbl>, pixel164 <dbl>, pixel165 <dbl>,
## #   pixel171 <dbl>, pixel172 <dbl>, pixel173 <dbl>, pixel174 <dbl>,
## #   pixel175 <dbl>, pixel176 <dbl>, pixel177 <dbl>, pixel178 <dbl>,
## #   pixel179 <dbl>, pixel180 <dbl>, pixel181 <dbl>, pixel182 <dbl>,
## #   pixel183 <dbl>, pixel184 <dbl>, pixel185 <dbl>, pixel186 <dbl>,
## #   pixel187 <dbl>, pixel188 <dbl>, pixel189 <dbl>, pixel190 <dbl>,
## #   pixel191 <dbl>, pixel192 <dbl>, pixel193 <dbl>, pixel194 <dbl>,
## #   pixel199 <dbl>, pixel200 <dbl>, pixel201 <dbl>, ...
```

```
validating
```

```
## # A tibble: 4,162 x 577
##    label pixel41 pixel42 pixel43 pixel63 pixel64 pixel65 pixel66 pixel67 pixel68
##    <fct>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 9           0       0       0       0       0       0       0       0       0
```

```
##  2 1          0      0      0      0      0      0      0      0      0
##  3 2          0      0      0      0      0      0      0      0      0
##  4 5          0      0      0      0      0      0      0      0      0
##  5 6          0      0      0      0      0      0      0      0      0
##  6 3          0      0      0      0      0      0      0      0      0
##  7 9          0      0      0      0      0      0      0      0      0
##  8 2          0      0      0      0      0      0      0      0      0
##  9 9          0      0      0      0      0      0      0      0      0
## 10 9          0      0      0      0      0      0      0      0      0
## # ... with 4,152 more rows, and 567 more variables: pixel69 <dbl>,
## #   pixel70 <dbl>, pixel71 <dbl>, pixel72 <dbl>, pixel73 <dbl>, pixel74 <dbl>,
## #   pixel75 <dbl>, pixel76 <dbl>, pixel77 <dbl>, pixel78 <dbl>, pixel90 <dbl>,
## #   pixel91 <dbl>, pixel92 <dbl>, pixel93 <dbl>, pixel94 <dbl>, pixel95 <dbl>,
## #   pixel96 <dbl>, pixel97 <dbl>, pixel98 <dbl>, pixel99 <dbl>, pixel100 <dbl>,
## #   pixel101 <dbl>, pixel102 <dbl>, pixel103 <dbl>, pixel104 <dbl>,
## #   pixel105 <dbl>, pixel106 <dbl>, pixel107 <dbl>, pixel108 <dbl>,
## #   pixel116 <dbl>, pixel117 <dbl>, pixel118 <dbl>, pixel119 <dbl>,
## #   pixel120 <dbl>, pixel121 <dbl>, pixel122 <dbl>, pixel123 <dbl>,
## #   pixel124 <dbl>, pixel125 <dbl>, pixel126 <dbl>, pixel127 <dbl>,
## #   pixel128 <dbl>, pixel129 <dbl>, pixel130 <dbl>, pixel131 <dbl>,
## #   pixel132 <dbl>, pixel133 <dbl>, pixel134 <dbl>, pixel135 <dbl>,
## #   pixel136 <dbl>, pixel137 <dbl>, pixel144 <dbl>, pixel145 <dbl>,
## #   pixel146 <dbl>, pixel147 <dbl>, pixel148 <dbl>, pixel149 <dbl>,
## #   pixel150 <dbl>, pixel151 <dbl>, pixel152 <dbl>, pixel153 <dbl>,
## #   pixel154 <dbl>, pixel155 <dbl>, pixel156 <dbl>, pixel157 <dbl>,
## #   pixel158 <dbl>, pixel159 <dbl>, pixel160 <dbl>, pixel161 <dbl>,
## #   pixel162 <dbl>, pixel163 <dbl>, pixel164 <dbl>, pixel165 <dbl>,
## #   pixel171 <dbl>, pixel172 <dbl>, pixel173 <dbl>, pixel174 <dbl>,
## #   pixel175 <dbl>, pixel176 <dbl>, pixel177 <dbl>, pixel178 <dbl>,
## #   pixel179 <dbl>, pixel180 <dbl>, pixel181 <dbl>, pixel182 <dbl>,
## #   pixel183 <dbl>, pixel184 <dbl>, pixel185 <dbl>, pixel186 <dbl>,
## #   pixel187 <dbl>, pixel188 <dbl>, pixel189 <dbl>, pixel190 <dbl>,
## #   pixel191 <dbl>, pixel192 <dbl>, pixel193 <dbl>, pixel194 <dbl>,
## #   pixel199 <dbl>, pixel200 <dbl>, pixel201 <dbl>, ...
```

5.2.Mô hình 1: FNN (Fast K-Nearest Neighbor)

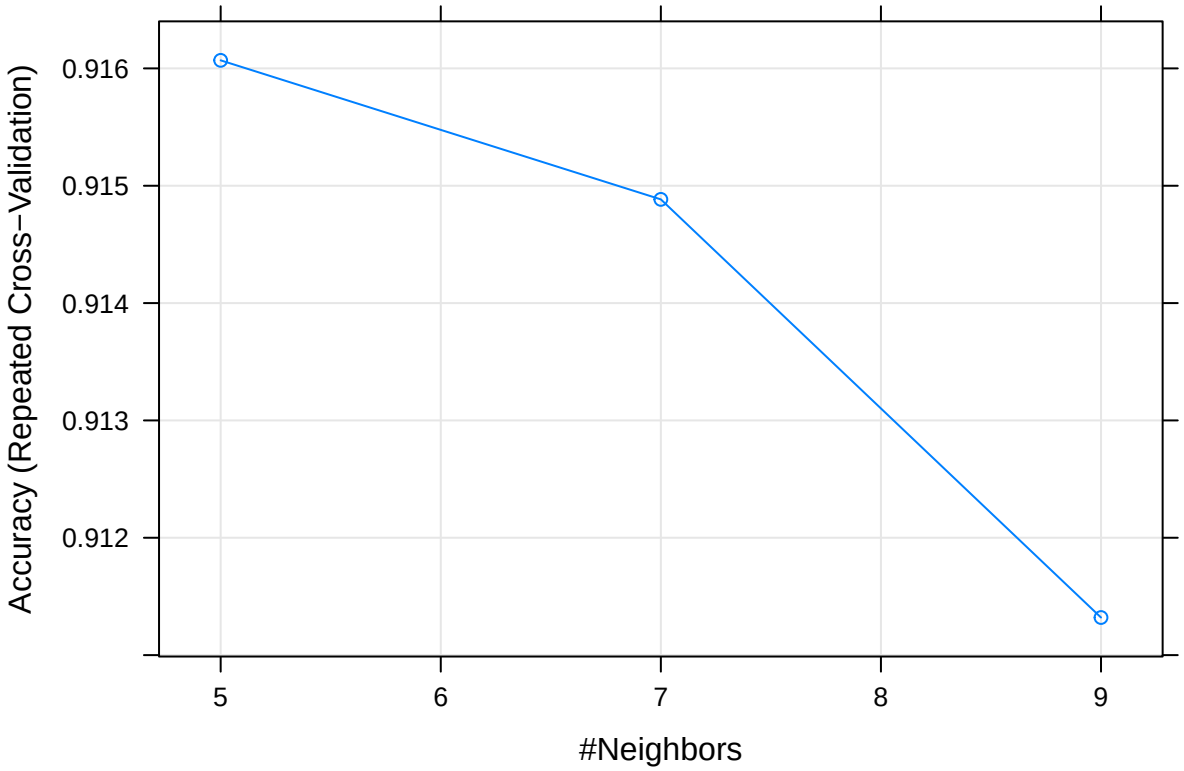
5.2.1. Huấn luyện tập train K-Nearest Neighbor (KNN) là một thuật toán không quá khó nhưng độ chính xác rất cao để giải quyết vấn đề Nhận dạng chữ số. Để dự đoán một cá thể mới, KNN tính toán Khoảng cách Euclid giữa cá thể mới và tất cả các cá thể trong toàn bộ tập huấn luyện. Sau đó, thuật toán tìm kiếm K trường hợp gần nhất (tương tự nhất) hàng đầu và đưa ra lớp có tần suất cao nhất (nhiều phiếu bầu nhất) như dự đoán. Chúng tôi sử dụng K-fold cross validation(k=5) để chọn giá trị tốt nhất cho K mang lại độ chính xác cao nhất. Tôi sử dụng gói CARET để đào tạo KNN và chọn K.

```
ctrl <- trainControl(method="repeatedcv",repeats = 1, number = 5, verboseIter = T, allowParallel = T)
knnFit <- train(label ~ ., data = training, method = "knn", trControl = ctrl)
```

```
## + Fold1.Rep1: k=5
## - Fold1.Rep1: k=5
## + Fold1.Rep1: k=7
## - Fold1.Rep1: k=7
## + Fold1.Rep1: k=9
```

```
## - Fold1.Rep1: k=9
## + Fold2.Rep1: k=5
## - Fold2.Rep1: k=5
## + Fold2.Rep1: k=7
## - Fold2.Rep1: k=7
## + Fold2.Rep1: k=9
## - Fold2.Rep1: k=9
## + Fold3.Rep1: k=5
## - Fold3.Rep1: k=5
## + Fold3.Rep1: k=7
## - Fold3.Rep1: k=7
## + Fold3.Rep1: k=9
## - Fold3.Rep1: k=9
## + Fold4.Rep1: k=5
## - Fold4.Rep1: k=5
## + Fold4.Rep1: k=7
## - Fold4.Rep1: k=7
## + Fold4.Rep1: k=9
## - Fold4.Rep1: k=9
## + Fold5.Rep1: k=5
## - Fold5.Rep1: k=5
## + Fold5.Rep1: k=7
## - Fold5.Rep1: k=7
## + Fold5.Rep1: k=9
## - Fold5.Rep1: k=9
## Aggregating results
## Selecting tuning parameters
## Fitting k = 5 on full training set
```

```
plot(knnFit)
```



Vì vậy, ta chọn K = 5. Qua tham khảo Tôi thấy rằng "kd-tree" là lựa chọn tốt nhất.(Thuật toán Kd-tree dùng để tìm kiếm các dữ liệu gần, liên quan nhất (neighbouring data points) trong miền không gian 2 chiều, hoặc nhiều chiều). Sử dụng khoảng cách là trung vị(median) giữa các điểm để thực hiện phân cụm.

```
library(FNN)
```

```
##
## Attaching package: 'FNN'
```

```
## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```
fnn.kd1 <- FNN::knn(training[,-1], validating[,-1], training$label, k=5, algorithm = c("kd_tree"))
fnn.kd.pred1 <- as.numeric(fnn.kd1)-1
```

```
g <- union(fnn.kd.pred1, validating$label)
h <- table(factor(fnn.kd.pred1, g), factor(validating$label, g))
confusionMatrix(h)
```

```
## Confusion Matrix and Statistics
##
##
##       9   1   2   5   6   3   7   4   8   0
```

```
##   9 372   0   1   6   0   2   8  30   7   0
##   1   7 461  12   4   4   4   9  10  12   0
##   2   0   1 384   1   0   3   0   1   2   0
##   5   2   0   0 335   5   5   0   1  13   3
##   6   0   1   3   6 391   1   0   1   3   4
##   3   2   0   3  17   0 406   0   0  15   0
##   7  19   0   8   1   0   4 414   1   2   0
##   4   9   1   1   2   1   0   4 360   5   0
##   8   0   0   1   0   1   5   0   0 344   1
##   0   4   0   1   4   8   1   1   0   0 401
##
## Overall Statistics
##
##                Accuracy : 0.9294
##                  95% CI : (0.9212, 0.937)
##     No Information Rate : 0.1115
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9215
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 9 Class: 1 Class: 2 Class: 5 Class: 6 Class: 3
## Sensitivity           0.89639   0.9935  0.92754  0.89096  0.95366  0.94200
## Specificity           0.98559   0.9832  0.99787  0.99234  0.99494  0.99008
## Pos Pred Value        0.87324   0.8815  0.97959  0.92033  0.95366  0.91648
## Neg Pred Value        0.98849   0.9992  0.99204  0.98920  0.99494  0.99328
## Prevalence            0.09971   0.1115  0.09947  0.09034  0.09851  0.10356
## Detection Rate        0.08938   0.1108  0.09226  0.08049  0.09395  0.09755
## Detection Prevalence  0.10235   0.1257  0.09419  0.08746  0.09851  0.10644
## Balanced Accuracy     0.94099   0.9884  0.96270  0.94165  0.97430  0.96604
##                      Class: 7 Class: 4 Class: 8 Class: 0
## Sensitivity           0.94954  0.89109  0.85360  0.98044
## Specificity           0.99061  0.99388  0.99787  0.99494
## Pos Pred Value        0.92205  0.93995  0.97727  0.95476
## Neg Pred Value        0.99407  0.98836  0.98451  0.99786
## Prevalence            0.10476  0.09707  0.09683  0.09827
## Detection Rate        0.09947  0.08650  0.08265  0.09635
## Detection Prevalence  0.10788  0.09202  0.08457  0.10091
## Balanced Accuracy     0.97007  0.94248  0.92573  0.98769
```

Confusion Matrix cung cấp cho chúng ta những thông tin có giá trị. Trước hết, KNN đã làm rất tốt chỉ với 10% của toàn bộ dữ liệu. Nó dự đoán bộ xác thực với độ chính xác 92,9%. Độ chính xác cao nhất thuộc về các nhãn "1", "0" và "6". Tuy nhiên, nó gặp khó khăn với việc dự đoán các chữ số "8", "5" và "9". Ví dụ: trong một số trường hợp "5" và "9", "3" và "8" hoặc "3" và "9" bị phân loại sai. Giá trị đặc trưng (specificity value) trong Confusion Matrix cho biết mỗi chữ số bị nhầm lẫn với các nhãn khác như thế nào. Hãy xem tính đặc trưng (specificity) của chữ số "1". Nó có giá trị thấp nhất trong số tất cả các chữ số. Có nghĩa là ~ 0,93% mô hình tính nhầm các chữ số khác là "1" (tức là chữ số không phải là 1, mà được dự đoán là 1). Ngược lại, chữ số "2" có độ đặc trưng(spensificity) cao nhất. Rất ít khả năng các chữ số khác được dự đoán sai là "2" (khi chúng thực sự không phải là số 2).

Thay đổi giá trị của k=1 để xem sự thay đổi của hiệu suất của quá trình huấn luyện.

```
library(FNN)
fnn.kd2 <- FNN::knn(training[,-1], validating[,-1], training$label, k=1, algorithm = c("kd_tree"))
fnn.kd.pred2 <- as.numeric(fnn.kd2)-1
```

```
i <- union(fnn.kd.pred2, validating$label)
k <- table(factor(fnn.kd.pred2, i), factor(validating$label, i))
confusionMatrix(k)
```

```
## Confusion Matrix and Statistics
##
##
##       9   1   2   5   6   3   4   7   8   0
##   9 371   2   2   5   0   4  31  13   7   1
##   1   2 459   7   2   2   1   5   6   8   0
##   2   0   1 382   0   0   4   0   1   4   0
##   5   4   1   0 336   3   6   0   0  10   0
##   6   1   1   1   6 396   1   1   0   5   4
##   3   2   0   5  15   0 404   0   0  14   2
##   4  17   0   2   0   2   0 366   3   3   0
##   7  16   0  10   4   0   3   1 412   1   0
##   8   1   0   3   7   1   8   0   0 350   0
##   0   1   0   2   1   6   0   0   1   1 402
##
## Overall Statistics
##
##                Accuracy : 0.9318
##                  95% CI : (0.9237, 0.9392)
##     No Information Rate : 0.1115
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9241
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 9 Class: 1 Class: 2 Class: 5 Class: 6 Class: 3
## Sensitivity           0.89398   0.9892  0.92271  0.89362  0.96585  0.93735
## Specificity           0.98265   0.9911  0.99733  0.99366  0.99467  0.98982
## Pos Pred Value        0.85092   0.9329  0.97449  0.93333  0.95192  0.91403
## Neg Pred Value        0.98819   0.9986  0.99151  0.98948  0.99626  0.99274
## Prevalence            0.09971   0.1115  0.09947  0.09034  0.09851  0.10356
## Detection Rate        0.08914   0.1103  0.09178  0.08073  0.09515  0.09707
## Detection Prevalence  0.10476   0.1182  0.09419  0.08650  0.09995  0.10620
## Balanced Accuracy     0.93831   0.9902  0.96002  0.94364  0.98026  0.96359
##                      Class: 4 Class: 7 Class: 8 Class: 0
## Sensitivity           0.90594  0.94495  0.86849  0.98289
## Specificity           0.99282  0.99061  0.99468  0.99680
## Pos Pred Value        0.93130  0.92170  0.94595  0.97101
## Neg Pred Value        0.98992  0.99354  0.98602  0.99813
## Prevalence            0.09707  0.10476  0.09683  0.09827
## Detection Rate        0.08794  0.09899  0.08409  0.09659
## Detection Prevalence  0.09443  0.10740  0.08890  0.09947
```

```
## Balanced Accuracy     0.94938  0.96778  0.93158  0.98984
```

k=1 có một sự cải thiện đáng kể cho hiệu suất của mô hình(93%). Như vậy, quá trình sử dụng k-Fold Cross-Validation để xác định K = 5 chưa thật sự là tốt nhất. Để đánh giá trực quan hơn ta cùng áp dụng mô hình trên tập test.

```
# KNN method at k = 5
pc <- proc.time()
fnn.kd <- knn(train_orig[,-1], test, train_orig$label , k=5 , algorithm = c("kd_tree"))
proc.time() - pc
```

5.2.2.Dự đoán bộ thử nghiệm test

```
##    user  system elapsed
## 1462.23    7.09 1541.19
```

Xuất kết quả

```
submission <- data.frame(ImageId=1:nrow(test), Label=fnn.kd)
head(submission)
```

```
##   ImageId Label
## 1       1     2
## 2       2     0
## 3       3     9
## 4       4     9
## 5       5     3
## 6       6     7
```

```
write_csv(submission, "KQ_KNN.csv")
```

Tập kết quả của bộ dữ liệu test

```
KQ <- read_csv("D:/KhaiPhaDuLieu/K-NN/Do_not_submit.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   ImageId = col_double(),
##   Label = col_double()
## )
```

Đánh giá mô hình với tập kết quả bằng Confusion Matrix

```
z <- union(fnn.kd, KQ$Label)
r <- table(factor(fnn.kd, z), factor(KQ$Label, z))
confusionMatrix(r)
```

```
## Confusion Matrix and Statistics
##
##
##        2    0    9    3    7    5    4    1    6    8
##   2 2695    2    3   16    6    3    2   10    1   14
##   0   25 2745    7    2    1    9    2    1    5   11
##   9    2    1 2656   12   30   10   48    3    0   32
##   3   11    0   18 2686    1   31    1    2    0   47
##   7   32    3   25   22 2804    5    6    5    0   16
##   5    2    4    8   31    1 2419    0    0    7   47
##   4    5    0   34    1   10    6 2663    4    1   17
##   1   28    5   11    5   38    6   20 3168   11   40
##   6    6    9    2    2    0   24    8    0 2713   11
##   8    7    2    6   13    1    5    2    0    1 2527
##
## Overall Statistics
##
##                Accuracy : 0.967
##                  95% CI : (0.9648, 0.9691)
##     No Information Rate : 0.114
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9633
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 2 Class: 0 Class: 9 Class: 3 Class: 7 Class: 5
## Sensitivity           0.95805  0.99062  0.95884  0.96272   0.9696  0.96068
## Specificity           0.99774  0.99750  0.99453  0.99560   0.9955  0.99608
## Pos Pred Value        0.97929  0.97756  0.95061  0.96031   0.9609  0.96030
## Neg Pred Value        0.99533  0.99897  0.99548  0.99587   0.9965  0.99611
## Prevalence            0.10046  0.09896  0.09893  0.09964   0.1033  0.08993
## Detection Rate        0.09625  0.09804  0.09486  0.09593   0.1001  0.08639
## Detection Prevalence  0.09829  0.10029  0.09979  0.09989   0.1042  0.08996
## Balanced Accuracy     0.97789  0.99406  0.97669  0.97916   0.9825  0.97838
##                      Class: 4 Class: 1 Class: 6 Class: 8
## Sensitivity           0.96766   0.9922  0.99051  0.91492
## Specificity           0.99691   0.9934  0.99755  0.99853
## Pos Pred Value        0.97154   0.9508  0.97766  0.98557
## Neg Pred Value        0.99648   0.9990  0.99897  0.99076
## Prevalence            0.09829   0.1140  0.09782  0.09864
## Detection Rate        0.09511   0.1131  0.09689  0.09025
## Detection Prevalence  0.09789   0.1190  0.09911  0.09157
## Balanced Accuracy     0.98229   0.9928  0.99403  0.95673
```

Với k = 5 chúng tôi đạt được Accuracy = 96,67%. Như vậy chỉ có ~3,3% số các chữ số mà mô hình đoán là sai. Một chỉ số khá là tốt với mô hình đầu tiên mà chúng tôi chọn để đánh giá. Chúng ta cùng xem với k = 1 liệu hiệu suất của mô hình có được cải thiện như khi chúng ta huấn luyện tập train.

```
fnn.kd1 <- knn(train_orig[,-1], test, train_orig$label , k=1 , algorithm = c("kd_tree"))
```

```
submission1 <- data.frame(ImageId=1:nrow(test), Label=fnn.kd1)
head(submission1)
```

```
##   ImageId Label
## 1       1     2
## 2       2     0
## 3       3     9
## 4       4     0
## 5       5     3
## 6       6     7
```

```
write_csv(submission1, "KQ_KNN_k1.csv")
```

```
a <- union(fnn.kd1, KQ$Label)
b <- table(factor(fnn.kd1, a), factor(KQ$Label, a))
confusionMatrix(b)
```

```
## Confusion Matrix and Statistics
##
##
##        2    0    9    3    7    5    4    1    6    8
##   2 2721    2    0   11    9    1    3    9    0    9
##   0   20 2746    5    1    1    7    1    1   11    8
##   9    1    1 2661    8   26    7   60    2    0   29
##   3   10    0    9 2685    1   36    1    1    0   40
##   7   31    5   44   19 2829    1   14    8    0   11
##   5    0    4    5   43    0 2423    0    1    5   46
##   4    6    0   35    1    6    6 2648    4    2   10
##   1   16    4    6    6   17    2   13 3167    9   26
##   6    3    7    1    2    0   27    9    0 2709    8
##   8    5    2    4   14    3    8    3    0    3 2575
##
## Overall Statistics
##
##                Accuracy : 0.9701
##                  95% CI : (0.9681, 0.9721)
##     No Information Rate : 0.114
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9668
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 2 Class: 0 Class: 9 Class: 3 Class: 7 Class: 5
## Sensitivity           0.96729  0.99098  0.96065  0.96237   0.9782  0.96227
## Specificity           0.99825  0.99782  0.99469  0.99611   0.9947  0.99592
## Pos Pred Value        0.98409  0.98036  0.95206  0.96479   0.9551  0.95884
## Neg Pred Value        0.99635  0.99901  0.99568  0.99584   0.9975  0.99627
## Prevalence            0.10046  0.09896  0.09893  0.09964   0.1033  0.08993
## Detection Rate        0.09718  0.09807  0.09504  0.09589   0.1010  0.08654
```

```
## Detection Prevalence  0.09875  0.10004  0.09982  0.09939   0.1058  0.09025
## Balanced Accuracy     0.98277  0.99440  0.97767  0.97924   0.9865  0.97910
##                      Class: 4 Class: 1 Class: 6 Class: 8
## Sensitivity           0.96221   0.9919  0.98905  0.93230
## Specificity           0.99723   0.9960  0.99774  0.99834
## Pos Pred Value        0.97425   0.9697  0.97939  0.98395
## Neg Pred Value        0.99589   0.9989  0.99881  0.99263
## Prevalence            0.09829   0.1140  0.09782  0.09864
## Detection Rate        0.09457   0.1131  0.09675  0.09196
## Detection Prevalence  0.09707   0.1166  0.09879  0.09346
## Balanced Accuracy     0.97972   0.9939  0.99340  0.96532
```

Như vậy với k=1 cho ta một kết quả tối ưu hơn (97,01% số dự đoán là đúng) so với khi ta chọn k=5 (96,67%). Nhưng quá trình huấn luyện cũng như dự đoán khá là tốn kém về mặt thời gian cũng như tài nguyên.

5.3. Sử dụng mô hình Random Forest

5.3.1. Quá trình huấn luyện Áp dụng Randomforest cho tập train với ntree=200

```
pc <- proc.time()
rf2 <- randomForest(train_orig[,-1], train_orig$label, ntree=200)
proc.time() - pc
```

```
##    user  system elapsed
## 3986.00   17.41 4179.97
```

```
pred1 <- predict(rf2, newdata=train_orig)
confusionMatrix(pred1, train_orig$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 4132    0    0    0    0    0    0    0    0    0
##          1    0 4684    0    0    0    0    0    0    0    0
##          2    0    0 4177    0    0    0    0    0    0    0
##          3    0    0    0 4351    0    0    0    0    0    0
##          4    0    0    0    0 4072    0    0    0    0    0
##          5    0    0    0    0    0 3795    0    0    0    0
##          6    0    0    0    0    0    0 4137    0    0    0
##          7    0    0    0    0    0    0    0 4401    0    0
##          8    0    0    0    0    0    0    0    0 4063    0
##          9    0    0    0    0    0    0    0    0    0 4188
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9999, 1)
##     No Information Rate : 0.1115
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                  Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           1.00000   1.0000  1.00000   1.0000  1.00000  1.00000
## Specificity           1.00000   1.0000  1.00000   1.0000  1.00000  1.00000
## Pos Pred Value        1.00000   1.0000  1.00000   1.0000  1.00000  1.00000
## Neg Pred Value        1.00000   1.0000  1.00000   1.0000  1.00000  1.00000
## Prevalence            0.09838   0.1115  0.09945   0.1036  0.09695  0.09036
## Detection Rate        0.09838   0.1115  0.09945   0.1036  0.09695  0.09036
## Detection Prevalence  0.09838   0.1115  0.09945   0.1036  0.09695  0.09036
## Balanced Accuracy     1.00000   1.0000  1.00000   1.0000  1.00000  1.00000
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            1.0000   1.0000  1.00000  1.00000
## Specificity            1.0000   1.0000  1.00000  1.00000
## Pos Pred Value         1.0000   1.0000  1.00000  1.00000
## Neg Pred Value         1.0000   1.0000  1.00000  1.00000
## Prevalence             0.0985   0.1048  0.09674  0.09971
## Detection Rate         0.0985   0.1048  0.09674  0.09971
## Detection Prevalence   0.0985   0.1048  0.09674  0.09971
## Balanced Accuracy      1.0000   1.0000  1.00000  1.00000
```

```
KQ <- read_csv("D:/KhaiPhaDuLieu/K-NN/Do_not_submit.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   ImageId = col_double(),
##   Label = col_double()
## )
```

Từ quá trình huấn luyện ta sẽ thực hiện dự đoán trên tập test

```
pred2 <- predict(rf2, newdata=test)
```

```
submission2 <- data.frame(ImageId=1:nrow(test), Label=pred2)
head(submission2)
```

```
##   ImageId Label
## 1       1     2
## 2       2     0
## 3       3     9
## 4       4     9
## 5       5     3
## 6       6     7
```

```
write_csv(submission2, "KQ_Randomforest.csv")
```

```
c <- union(pred2, KQ$Label)
d <- table(factor(pred2, c), factor(KQ$Label, c))
confusionMatrix(d)
```

```
## Confusion Matrix and Statistics
##
##
##        2    0    9    3    7    5    4    1    6    8
##   2 2718    5    5   34   30    5    4   23    3   22
##   0   13 2725    7    2    2    9    4    1    7    6
##   9    3    2 2645   11   33    9   45    5    0   34
##   3   14    0   33 2667    1   23    0    6    0   20
##   7   16    2   14   26 2798    2    3    4    0    4
##   5    3    4   12   23    0 2418    0    6   19   25
##   4   16    0   29    1   18    7 2666    5    5   16
##   1    2    2    8    4    6    5    4 3135    9   19
##   6   10    9    2    2    0   21   18    3 2687   15
##   8   18   22   15   20    4   19    8    5    9 2601
##
## Overall Statistics
##
##                Accuracy : 0.9664
##                  95% CI : (0.9643, 0.9685)
##     No Information Rate : 0.114
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9627
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 2 Class: 0 Class: 9 Class: 3 Class: 7 Class: 5
## Sensitivity           0.96623  0.98340  0.95487  0.95591  0.96750  0.96029
## Specificity           0.99480  0.99798  0.99437  0.99615  0.99717  0.99639
## Pos Pred Value        0.95402  0.98163  0.94905  0.96491  0.97525  0.96335
## Neg Pred Value        0.99622  0.99818  0.99504  0.99513  0.99626  0.99608
## Prevalence            0.10046  0.09896  0.09893  0.09964  0.10329  0.08993
## Detection Rate        0.09707  0.09732  0.09446  0.09525  0.09993  0.08636
## Detection Prevalence  0.10175  0.09914  0.09954  0.09871  0.10246  0.08964
## Balanced Accuracy     0.98051  0.99069  0.97462  0.97603  0.98233  0.97834
##                      Class: 4 Class: 1 Class: 6 Class: 8
## Sensitivity           0.96875   0.9818  0.98101  0.94171
## Specificity           0.99616   0.9976  0.99683  0.99525
## Pos Pred Value        0.96489   0.9815  0.97109  0.95590
## Neg Pred Value        0.99659   0.9977  0.99794  0.99363
## Prevalence            0.09829   0.1140  0.09782  0.09864
## Detection Rate        0.09521   0.1120  0.09596  0.09289
## Detection Prevalence  0.09868   0.1141  0.09882  0.09718
## Balanced Accuracy     0.98245   0.9897  0.98892  0.96848
```

Với mô hình Random Forest cho chúng tôi Accuracy = 96,66% số dự đoán trên tập test là đúng. Thấp hơn một chút so với mô hình KNN. Nhưng thời gian thì tối ưu hơn mô hình KNN một cách đáng kể

6. Kết luận

Qua bài tìm hiểu nhóm đã khát quát và nêu rõ được bài toán Digit Recognizer, trình bày được các hoạt động của các mô hình KNN, Random Forest. Một số thao tác tiền xử lý dữ liệu như xóa các cột có giá trị bằng 0. Qua đó áp dụng vào giải quyết bài toán với việc huấn luyện tập train và thay đổi các tham số để xem xét sự thay đổi của hiệu suất:

- Thứ nhất, với thuật toán KNN. Sử dụng k-Fold Cross-Validation để tìm ra k phù hợp và tìm được k=5. Nhưng khi ta thay đổi k=1 nhận thấy hiệu suất được cải thiện tốt hơn. Vậy k-fold Cross-Validation không hẳn là tốt nhất trong mọi trường hợp.
- Thứ hai, với thuật toán Random Forest khi ta thay đổi ntree cũng như thay đổi độ lớn của tập mà ta đưa vào huấn luyện thì kết quả khớp là 100%.
- Thứ ba, nhóm sử dụng độ đo Confusion Matrix cho bài toán này. Qua đó đánh giá quá trình huấn luyện cũng như dự đoán một cách rõ ràng và hiệu quả.
  - Từ các kết quả ta có thể thấy thuật toán KNN là thuật toán cho ta kết quả tốt nhất(97,01% với k=1) nhưng đồng thời thời gian chạy của thuật toán khá lâu.
  - Thuật toán KNN là thuật toán không quá khó nhưng đem lại hiệu suất cao.
  - Nếu có thêm thời gian và các nguồn lực khác, nhóm sẽ thử nghiệm và tìm hiểu thêm về các thuật toán khác như: SVM, CNN, Neural Network(NN).

7. Tài liệu tham khảo

[1] Digit Recognizer Learn computer vision fundamentals with the famous MNIST datahttps://www.kaggle.com/c/digit-recognizer

[2] Aditya S Nakate, Digit Recognizer using Random Forest Algorithm, https://rpubs.com/Adityanakate/240343, 7 January 2017

[3] https://www.rdocumentation.org/

[4] Kashish , KNN vs Decision Tree vs Random Forest for handwritten digit recognition, https://medium.com/analytics-vidhya/knn-vs-decision-tree-vs-random-forest-for-handwritten-digit-recognition-470e864c75bc , Apr 17 2020