Clasificación de imágenes de letras

Aprendizaje Estadístico

Laura Pérez - Gabriel Raby

14 de Octubre 2021

Introducción

A partir de un **conjunto de datos de imágenes**, se buscará aplicar técnicas de aprendizaje estadístico para **clasificarlas según un objetivo** o criterio específico.

Dataset

Conjunto de 86 imágenes de letras manuscritas

- **Formato**: imágenes en formato JPG o PNG
- **Tamaño**: 200 x 200 px

Distribucion de letras

Agregar grafico con distribucion de clases en el conjunto de datos.

Objetivo

Para simplificar la tarea de clasificacion en lugar de clasificacion multi-clase en primera instancia se decidio defininir un objetivo de clasificacion binaria. Se pretende generar un modelo estadístico que pueda clasificar las imágenes de letras según el siguiente criterio:

- Criterio positivo: **es** una *letra a*
- Criterio negativo: **no es** una *letra a*

Técnicas

Se utilizarán las siguientes técnicas de aprendizaje estadístico:

- **Regresión logística**
- **Clasificación K-NN**

Metodología de Trabajo

El proceso de trabajo será el siguiente:

- **Pre-procesamiento**: el conjunto de datos será transformado de modo tal que pueda ser utilizado durante el modelado
- **Definición de modelos**: se ajustarán los modelos definidos para cada técnica con los datos de entrenamiento
- **Evaluación de modelos**: a partir de los modelos ajustados, se estimarán resultados de clasificación sobre los datos de evaluación
- **Conclusión**: se compararán los resultados de clasificación de ambas técnicas

Preprocesamiento

En esta etapa del flujo de trabajo se transformará el conjunto de datos de modo tal que pueda ser utilizado durante el modelado

Formato de imágenes

Se utilizará la librería **ImageR**.

Esta librería utiliza el formato estándar de imágenes **CImg**.

Este formato representa las imágenes como un arreglo numérico $4D$ (x, y, z, c)

- **x**, **y**: dimensiones espaciales
- **z**: profundidad o tiempo (en secuencia de imágenes)
- **c**: color

Carga de datos

Se cargan las 86 imágenes en una lista

```
imgList <- load.dir("./letras")

glimpse(imgList[1:5])
```

```
## List of 5
##  $ a (1).jpg  : 'cimg' num [1:200, 1:200, 1, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
##  $ a (2).jpg  : 'cimg' num [1:200, 1:200, 1, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
##  $ a_1 (1).jpg: 'cimg' num [1:200, 1:200, 1, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
##  $ a_1.jpg    : 'cimg' num [1:200, 1:200, 1, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
##  $ a_10.jpg   : 'cimg' num [1:200, 1:200, 1, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "class")= chr [1:2] "imlist" "list"
```

Transformaciones (1 de 2)

Se realizan las siguientes transformaciones a la lista de imágenes:

- uniformar y reducir tamaño de imgs – [100,100,z,c]
- eliminar canales de color (grayscale) – [x,y,1,1]

```
imgList %<>%
  map(resize, 100, 100) %>%
  map(function(x) as.cimg(as.vector(round(x, digits = 0)))) %>%
  map(grayscale)
```

Transformaciones (2 de 2)

Se crea matriz de imágenes adecuada para los modelos:

- generar tags de letras a partir de nombres de archivos
- armar matriz de trabajo [img, v1, . . . , v_i, . . . , v_(x*y)]

```
# fc para leer tags a partir de los nombres de archivos
getBinaryLabel <- function(input) {
  ret <- str_split(input, "/")
  ifelse(str_sub(ret[[1]][length(ret[[1]])], 1, 1) == "a", 1, 2 )
}

binLabels <- names(imgList) %>% map(getBinaryLabel)

#matriz de imágenes en formato adecuado para modelos
imgMatrix <- as_tibble(cbind(unlist(binLabels),
                             matrix(unlist(imgList),
                                    nrow=length(imgList), byrow=TRUE)))

imgMatrix <- rename(imgMatrix, letra = V1) %>%
  mutate(letra = if_else(letra==1, "aes", "noaes"),
         letra = factor(letra, levels = c("aes", "noaes")))

glimpse(imgMatrix[1:5, 1:10])
```

```
## Rows: 5
## Columns: 10
## $ letra <fct> aes, aes, aes, aes, aes
## $ V2    <dbl> 1, 1, 1, 1, 1
## $ V3    <dbl> 1, 1, 1, 1, 1
## $ V4    <dbl> 1, 1, 1, 1, 1
## $ V5    <dbl> 1, 1, 1, 1, 1
## $ V6    <dbl> 1, 1, 1, 1, 1
## $ V7    <dbl> 1, 1, 1, 1, 1
## $ V8    <dbl> 1, 1, 1, 1, 1
## $ V9    <dbl> 1, 1, 1, 1, 1
## $ V10   <dbl> 1, 1, 1, 1, 1
```

Modelado

Durante esta etapa se ajustarán los modelos definidos para cada técnica con los datos de entrenamiento

Tidymodels

Este framework incluye un conjunto de librerías para modelado y machine learning, y permitirá definir y evaluar los modelos previstos.

Partición de observaciones

Se divirá el conjunto de imágenes en un conjunto de entrenamiento y otro de evaluación en una proporción de 70 a 30

```
set.seed(4096)

#se dividen los datos en entrenamiento y prueba
letras_split <- initial_split(imgMatrix,
                              prop = 0.70,
                              strata = letra)

letras_train <- training(letras_split)
letras_test <- testing(letras_split)

# nro de filas en los datos de entrenamiento y prueba
nrow(letras_train);nrow(letras_test)
```

```
## [1] 59
```

```
## [1] 27
```

Definición y Ajuste del modelo: Regresión Logística (1 de 2)

Usando funciones de *tidymodels* se define el modelo de regresión logística

1. Se llama la función `logistic_reg()` correspondiente al modelo de regresión logística.
2. Se define el *motor* de modelado con la función `set_engine()` ; en este caso, se agrega **glm** ya que la regresión logística se incluye dentro de los GLM.
3. Se define el modo en el que funcionará el modelo con la función `set_mode()`, en este caso es una clasificación.
4. Se usa la función `fit()` para ajustar el modelo a partir de los datos de entrenamiento.

Definición y Ajuste del modelo: Regresión Logística (2 de 2)

```
fitted_logistic_model <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification") %>%
  fit(letra ~., data = letras_train)

tidy(fitted_logistic_model, exponentiate=TRUE)
```

```
## # A tibble: 10,001 x 5
##    term          estimate std.error  statistic p.value
##    <chr>            <dbl>     <dbl>      <dbl>   <dbl>
##  1 (Intercept)  4.34e-266 12882145. -0.0000474    1.00
##  2 V2          NA                NA  NA           NA
##  3 V3          NA                NA  NA           NA
##  4 V4          NA                NA  NA           NA
##  5 V5          NA                NA  NA           NA
##  6 V6          NA                NA  NA           NA
##  7 V7          NA                NA  NA           NA
##  8 V8          NA                NA  NA           NA
##  9 V9          NA                NA  NA           NA
## 10 V10         NA                NA  NA           NA
## # ... with 9,991 more rows
```

Predicción del modelo: Regresión Logística

```
pred_class <- predict(fitted_logistic_model,
                      new_data = letras_test,
                      type = "class")

pred_proba <- predict(fitted_logistic_model,
                      new_data = letras_test,
                      type = "prob")

letras_results <- letras_test %>%
  select(letra) %>%
  bind_cols(pred_class, pred_proba)

letras_results[1:5, ]
```

```
## # A tibble: 5 x 4
##   letra .pred_class .pred_aes .pred_noaes
##   <fct> <fct>           <dbl>       <dbl>
## 1 aes   noaes        2.90e-12        1.00
## 2 aes   noaes        2.90e-12        1.00
## 3 aes   noaes        2.90e-12        1.00
## 4 aes   noaes        2.90e-12        1.00
## 5 aes   noaes        2.90e-12        1.00
```

Evaluación del modelo: Regresión Logística (1 de 2)

Se presenta matriz de confusión

```r
conf_mat(letras_results, truth = letra,
         estimate = .pred_class)
```

```
##           Truth
## Prediction aes noaes
##      aes     2     3
##      noaes   6    16
```

Evaluación del modelo: Regresión Logística (2 de 2)

Métricas de evaluación: Accuracy y Sensibilidad

```r
accuracy(letras_results, truth = letra,
         estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.667
```

```r
sens(letras_results, truth = letra,
         estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary          0.25
```

Definición y ajuste del modelo: Clasificación K-NN (1 de 2)

También se utilizan funciones de *tidymodels* para realizar una clasificación K-NN

1. Se llama la función `nearest_neighbor()` para definir un modelo que usa las K observaciones más similares del conjunto de entrenamiento para predecir nuevas muestras.
2. Se define el *motor* de modelado con la función `set_engine()` ; en este caso, se utiliza **kknn**.
3. Se define el modo en el que funcionará el modelo con la función `set_mode()`, en este caso es una clasificación.
4. Se usa la función `fit()` para ajustar el modelo a partir de los datos de entrenamiento.

Definición y ajuste del modelo: Clasificación K-NN (2 de 2)

```r
fitted_knn_model <- nearest_neighbor(neighbors = 2) %>%
  set_engine("kknn") %>%
  set_mode("classification") %>%
  fit(letra ~., data = letras_train)

fitted_knn_model
```

```
## parsnip model object
##
## Fit time:  7.5s
##
## Call:
## kknn::train.kknn(formula = letra ~ ., data = data, ks = min_rows(2,     data, 5))
##
## Type of response variable: nominal
## Minimal misclassification: 0.7118644
## Best kernel: optimal
## Best k: 2
```

Predicción del modelo: Clasificación K-NN

```
pred_class <- predict(fitted_knn_model,
                      new_data = letras_test,
                      type = "class")

pred_proba <- predict(fitted_knn_model,
                      new_data = letras_test,
                      type = "prob")

letras_results <- letras_test %>%
  select(letra) %>%
  bind_cols(pred_class, pred_proba)

letras_results[1:5, ]
```

```
## # A tibble: 5 x 4
##   letra .pred_class .pred_aes .pred_noaes
##   <fct> <fct>           <dbl>       <dbl>
## 1 aes   aes             1           0
## 2 aes   aes             1           0
## 3 aes   noaes           0.153       0.847
## 4 aes   aes             1           0
## 5 aes   aes             1           0
```

Evaluación del modelo: Clasificación K-NN (1 de 2)

Se presenta matriz de confusión

```
conf_mat(letras_results, truth = letra,
         estimate = .pred_class)
```

```
##           Truth
## Prediction aes noaes
##      aes     7    17
##      noaes   1     2
```

Evaluación del modelo: Clasificación K-NN (2 de 2)

Métricas de evaluación: Accuracy y Sensibilidad

```
accuracy(letras_results, truth = letra,
         estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.333
```

```
sens(letras_results, truth = letra,
         estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.875
```

Conclusiones

Conclusiones (1 de 3)

- Elegimos representar las imágenes como un vector de pixeles, donde cada dimensión representa una posición o pixel de la imagen. De esta manera nuestro conjunto de variables independientes es igual al numero de elementos de este vector (es decir, 10.000 en nuestra implementación).
- Probamos con dos algoritmos de clasificación supervisados: **regresión logística** y **K-NN**. En el caso de regresión logística, hemos visto que una gran parte de las variables explicativas obtiene un coeficiente distinto de NA. Lo cual consideramos adecuado dado que para ciertas variables (es decir, elementos del vector) todas las instancias de datos tienen el mismo valor.

Conclusiones (2 de 3)

En términos de performance

- **Regresión logística**: el modelo obtiene un valor de *accuracy* de **0.67** indicando una buena performance. Sin embargo, cuando miramos el estadístico *sensibilidad*, este es bajo **0.25**; dado el desbalance de datos (mayor cantidad de observaciones de la clase *noaes*) el modelo focaliza su ajuste en esta clase.
- **Clasificación K-NN**: el modelo logra una clasificación relativamente buena para la clase *aes* (*sensibilidad* **0.85**) pero obtiene una mala performance clasificando el resto de las letras. Esta pobre clasificación de *noaes* se puede relacionar a que el conjunto de *noaes* está formado por el complemento de letras (e, i, o, u) las cuales pueden tener distancias lejanas entre ellas y por lo tanto no se clasificadas correctamente dentro de la misma clase.

Conclusiones (3 de 3)

Otras consideraciones finales incluyen:

- Pruebas preliminares en **K-NN multi-clase** (es decir, todas las posibles letras) arroja similar sesgo del modelo hacia la clase mas numerosa (es decir, letra a).
- Pruebas con **Support Vector Machines (SVM)** no fueron concluidas por error arrojado por la función ksvm() durante el ajuste. Según nuestro entendimiento puede haber alguna incompatibilidad con el vector de pixeles 0s y 1s. Una alternativa a probar como trabajo futuro es con un vector con otro formato.
- Otra pista a explorar en trabajo futuro es aumentar el conjunto de datos inicial con imágenes sintéticas. Estas pueden ser creadas automáticamente aplicando *perturbaciones* tales como rotación o difuminación.