Tarea 7 - Reconocimiento estadístico de patrones

Erick Salvador Alvarez Valencia

CIMAT A.C.,
erick.alvarez@cimat.mx

Resumen En el presente reporte se hablará sobre la resolución de los ejercicios de la séptima tarea de reconocimiento de patrones, de la misma manera se presentará el código de los mismos y los resultados que fueron obtenidos en los problemas que lo requieren.

1. Problema 1

En el contexto de clasificación binaria, verifica que si $\hat{f} = argmin_f E(\exp(-Yf(x)))$, $\hat{y} = sgn(\hat{f}(x))$ asigna x a la clase más probable.

Solución : Empezamos desarrollando la esperanza, recordemos que $Y = \{-1, 1\}$:

$$E(\exp(-Yf(x))) = E_X E_{Y|X=x} \tag{1}$$

Podemos notar en el desarrollo anterior de la esperanza que para todo X basta minimizar $E_{Y|X=x}$ por lo que nuestro problema se convierte a:

$$argmin_f E_{Y|X=x} \exp(-Yf(x))) \tag{2}$$

Desarrollamos la esperanza, posteriormente la derivamos y la igualamos a cero para encontrar el punto crítico.

$$\begin{aligned}
E_{Y|X=x} \exp(-Yf(x))) &= P(Y=1|X=x)\exp(-f(x)) + P(Y=-1|X=x)\exp(f(x)) \\
\frac{\partial E_{Y|X=x}}{\partial f(x)} &= -P(Y=1|X=x)\exp(-f(x)) + P(Y=-1|X=x)\exp(f(x)) = 0 \\
P(Y=-1|X=x)\exp(f(x)) &= P(Y=1|X=x)\exp(-f(x)) \\
\frac{\exp(f(x))}{\exp(-f(x))} &= \frac{P(Y=1|X=x)}{P(Y=-1|X=x)} \\
\exp(2f(x)) &= \frac{P(Y=1|X=x)}{P(Y=-1|X=x)} \\
f(x) &= \frac{1}{2}\log(\frac{P(Y=1|X=x)}{P(Y=-1|X=x)})
\end{aligned} \tag{3}$$

Hemos encontrado el punto crítico $f(x)$ de la ecuación anterior, para demostrar que es un mínimo local usaremos el criterio de la segunda derivada

$$\frac{\partial^2 E_{Y|X=x}}{\partial f(x)^2} = P(Y=1|X=x)\exp(-f(x)) + P(Y=-1|X=x)\exp(f(x)) \quad (4)$$

Sustituimos el punto crítico encontrado en la derivada anterior

$$\begin{aligned}
\frac{\partial^2 E_{Y|X=x}}{\partial f(x)^2} &= P(Y=1|X=x)\exp(-\frac{1}{2}\log(\frac{P(Y=1|X=x)}{P(Y=-1|X=x)}))+ \\
&\quad P(Y=-1|X=x)\exp(\frac{1}{2}\log(\frac{P(Y=1|X=x)}{P(Y=-1|X=x)})) \\
&= P(Y=1|X=x)\exp(\log(\frac{P(Y=1|X=x)}{P(Y=-1|X=x)})^{-\frac{1}{2}})+ \\
&\quad P(Y=-1|X=x)\exp(\log(\frac{P(Y=1|X=x)}{P(Y=-1|X=x)})^{\frac{1}{2}}) \\
&= \frac{P(Y=1|X=x)}{\sqrt{\frac{P(Y=1|X=x)}{P(Y=-1|X=x)}}} + P(Y=-1|X=x)\sqrt{\frac{P(Y=1|X=x)}{P(Y=-1|X=x)}}
\end{aligned} \quad (5)$$

Podemos ver que la segunda derivada de la función evaluada en el punto crítico obtenido es una expresión que depende únicamente de las probabilidades condicionales, y como estas son números reales positivos la ecuación anterior es positiva y por lo tanto el punto encontrado es un mínimo local.
Para terminar, podemos ver que el punto encontrado corresponde al clasificador bayesiano óptimo. Esto lo podemos ver fácilmente verificando los casos:

1. Si $P(Y=1|X=x) > P(Y=-1|X=x)$ entonces el cociente $\frac{P(Y=1|X=x)}{P(Y=-1|X=x)}$ es mayor que 1 y por lo tanto el logaritmo de dicho cociente es positivo y el signo de dicho número positivo asignará a 1 como la clase predicha. Podemos ver que mltiplicar $\frac{1}{2}$ por el logaritmo no afecta a la predicción por lo cual lo podemos descartar.

2. Si $P(Y=1|X=x) < P(Y=-1|X=x)$ entonces el cociente $\frac{P(Y=1|X=x)}{P(Y=-1|X=x)}$ es menor que 1 y por lo tanto el logaritmo de dicho cociente es negativo y el signo de dicho número positivo asignará a -1 como la clase predicha.

2. Problema 2

En la clase introducimos modelos de regresión logística usando como codificación de y, 0 y 1. Si a cambio usamos -1 y 1, entonces verifica que la función de logverosimilitud (condicional) será de la forma:

$$\sum_i \log \frac{1}{1+\exp(-y_i f(x_i))},\ f(x) = \alpha + \beta^t x$$

Es decir, con el método de máxima verosimilitud para la estimación de los parámetros se usa como función de costo:

$$log(1 + \exp(yf(x)))$$

Solución : Sean

$$\begin{aligned} \pi = P(Y = 1|X = x) &= \frac{1}{1 + \exp(-yf(x))} \\ 1 - \pi = P(Y = -1|X = x) &= \frac{1}{1 + \exp(yf(x))} \end{aligned} \tag{6}$$

Tal como fueron definidas en clase, ahora podemos definir la función de probabilidad como:

$$p(x, y) = \pi^{\frac{1+y}{2}}(1 - \pi)^{\frac{1-y}{2}} \tag{7}$$

De esta forma vemos que dependiendo el valor que tome y sólamente quedará en pie una de las dos probabilidades condicionales, ya sea $P(Y = 1|X = x)$ ó $P(Y = -1|X = x)$. Ahora podemos escribir la función de verosimilitud como:

$$L = \prod_{i=1}^{n} \pi^{\frac{1+y_i}{2}}(1 - \pi)^{\frac{1-y_i}{2}} \tag{8}$$

Y por lo tanto la log-verosimilitud como:

$$\begin{aligned} l &= \log(\prod_{i=1}^{n} \pi^{\frac{1+y_i}{2}}(1 - \pi)^{\frac{1-y_i}{2}}) \\ l &= \sum_{i=1}^{n} log(\pi^{\frac{1+y_i}{2}}(1 - \pi)^{\frac{1-y_i}{2}}) \\ l &= \sum_{i=1}^{n} \frac{1 + y_i}{2} log(\pi) + \frac{1 - y_i}{2} log(1 - \pi) \\ l &= \frac{1}{2} \sum_{i=1}^{n} (1 + y_i) log(\pi) + (1 - y_i) log(1 - \pi) \end{aligned} \tag{9}$$

Ahora analizamos por casos la ecuación anterior.

1. Si $y_i = 1$: Podemos ver que la log-verosimilitud se convierte en

$$l = \frac{1}{2} \sum_{i=1}^{n} 2log(\frac{1}{1 + \exp(-f(x_i))}) = \sum_{i=1}^{n} log(\frac{1}{1 + \exp(-f(x_i))})$$

2. Si $y_i = -1$: Podemos ver que la log-verosimilitud se convierte en

$$l = \frac{1}{2}\sum_{i=1}^{n} 2log(\frac{1}{1+\exp(f(x_i))}) = \sum_{i=1}^{n} log(\frac{1}{1+\exp(f(x_i))})$$

Se puede apreciar que lo único que cambia en ambos casos es el signo de $f(x_i)$, esto dependiendo de y_i. De esta forma podemos escribir la log-verosimilitud como:

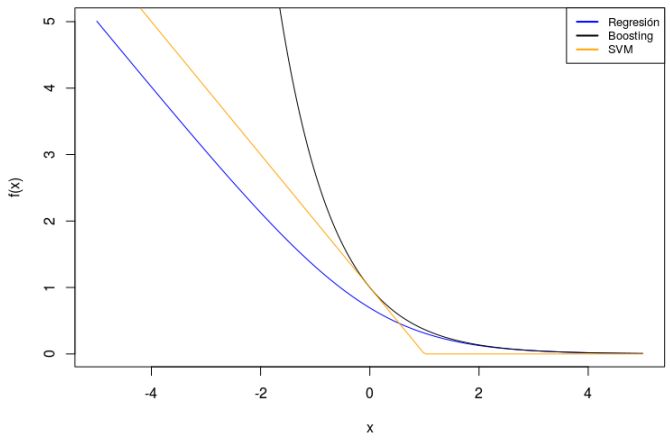
$$l = \sum_{i=1}^{n} log(\frac{1}{1+\exp(-y_i f(x_i))}) \tag{10}$$

Grafica lo anterior en función de $yf(x)$. ¿Cómo se compara con la función de costo de una SVM y de Boosting?

Solución : Se hicieron las gráficas para las tres funciones de costo, las cuales son las siguientes:

1. Regresión: $C(x, y) = log(1 + \exp(-yf(x)))$
2. Boosting: $C(x, y) = \exp(-yf(x))$
3. SVM: $C(x, y) = max(0, 1 - yf(x))$

A continuación se mostrarán las gráficas de las funciones de costo anteriores hechas en R:



(a) Figura 1. Gráfica de las funciones de costo anteriores.

Lo primero que podemos ver es que el método de Boosting penaliza mucho más que los otros dos aunque al final todos tienden a cero, pero lo interesante es que el único modelo que no penaliza una vez que se está clasificando bien es la SVM, los otros dos siguen penalizando en una cierta área donde las clases se están prediciendo de manera correcta.

3. Problema 3

Implementa en R la idea de random forests a partir de la libreria *rpart* para árboles de decisión. Usalo para construir clasificadores para los datos SPAM. Compáralo con Boosting.

Solución : Para este problema se comparó el modelo de random forest contra el de boosting, por lo cual se hicieron unas pruebas con un conjunto de datos de emails que son y no son SPAM, en dicho conjunto habían filas en donde tenían ciertas características medidas como números reales y al final se indicaba la clase del email, si era o no SPAM. Para construir el random forest se usó la librería -part"la cual construye un árbol de decisión en base a un conjunto de datos, dicho árbol elige con un algoritmo los predictores que considera más importantes para hacer la clasificación, pero la idea del random forest es entrenar un conjunto de árboles de decisión y usar un método de votación con la respuesta de todos los árboles para elegir la clase final para esa observación, además de que cada árbol se entrena con diferentes predictores, y eso hace la diferencia al método de boosting el cual los árboles siempre trabajan con los mismos predictores. Otra cuestión en la parte del entrenamiento es que a cada árbol se entrena con diferentes datos, esto para evitar el sobreajuste.
A continuación se muestran los resultados obtenidos al aplicar el método de rando forest al conjunto de datos en donde se usaron los siguientes parámetros:

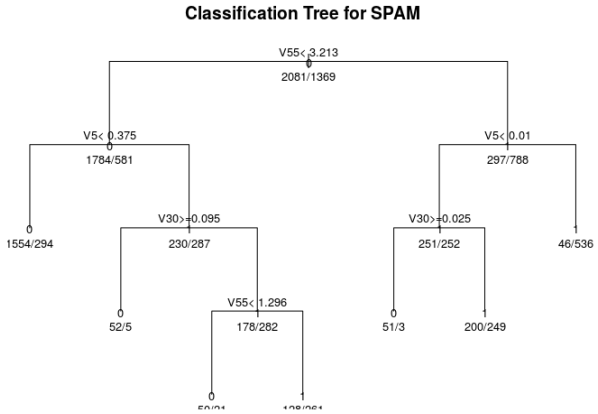
1. **Porcentaje de datos de entrenamiento** : 50 %.
2. **Número de árboles en el random forest** : 100.
3. **Número de predictores para cada árbol** : 7.

Los resultados obtenidos fueron los siguientes:

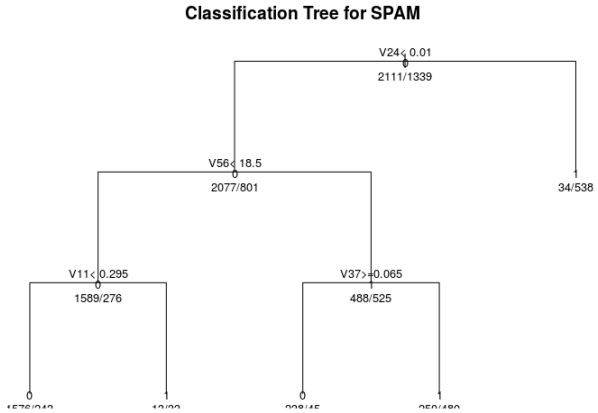
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 679 120
         1  11 341

               Accuracy : 0.8862
                 95% CI : (0.8664, 0.904)
    No Information Rate : 0.5995
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7533
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9841
            Specificity : 0.7397
         Pos Pred Value : 0.8498
         Neg Pred Value : 0.9688
             Prevalence : 0.5995
         Detection Rate : 0.5899
   Detection Prevalence : 0.6942
      Balanced Accuracy : 0.8619

       'Positive' Class : 0
```

(b) Figura 2. Precisión del modelo con los datos de prueba.

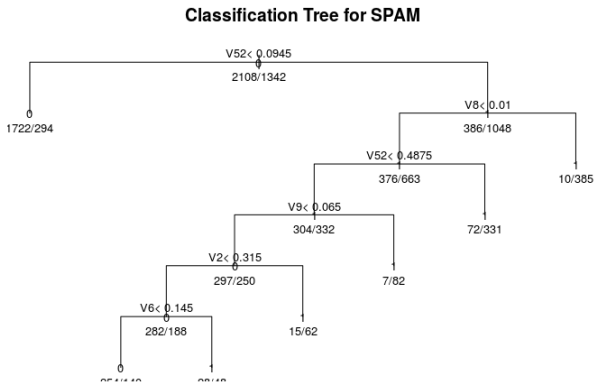
En la Figura 2. se muestra la matriz de confusión generada con la librería *caret*. En dicha matriz vemos una precisión final de 88.62 % de datos correctamente clasificados, un porcentaje algo bajo según lo esperado, además se tiene que hubo más falsos negativos que falsos positivos. Como comentario el número de predictores elegidos de manera aleatoria fue seleccionado como la raíz cuadrada del número de predictores totales tal como indica el libro de *Caret* aunque el árbol determina cuáles de esos predictores generan mayor importancia en la clasificación. A continuación se muestran algunos de los árboles de decisión generados para demostrar esto anterior.



(c) Figura 3. Precisión del modelo con los datos de prueba.



(d) Figura 4. Precisión del modelo con los datos de prueba.



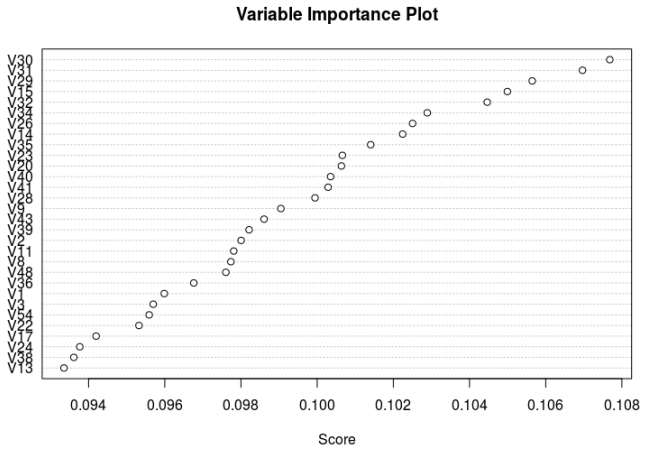
(e) Figura 5. Precisión del modelo con los datos de prueba.

En las 3 Figuras anteriores se muestran 3 árboles que se generaron con diferentes predictores, y lo que se puede apreciar es que de los 7 predictores que se pasaban al modelo sólamente se usaban entre 4 y 6.
Posteriormente se aplicó el mismo conjunto de datos a un boosting para ver los datos que se obtenían, se utilizó la librería *Adaboost* la cual ya tiene implementado el algoritmo, simplemente se entrenó el modelo y se generaron las predicciones. A continuación se muestra una matriz de confusión con los datos obtenidos.

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 515  21
         1  15 313

               Accuracy : 0.9583
                 95% CI : (0.9428, 0.9706)
    No Information Rate : 0.6134
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9119
 Mcnemar's Test P-Value : 0.4047

            Sensitivity : 0.9717
            Specificity : 0.9371
         Pos Pred Value : 0.9608
         Neg Pred Value : 0.9543
             Prevalence : 0.6134
         Detection Rate : 0.5961
   Detection Prevalence : 0.6204
      Balanced Accuracy : 0.9544

       'Positive' Class : 0
```

(f) Figura 6. Matriz de confusión generada por el *boosting*.

Lo que se aprecia en la matriz de confusión anterior es que el algoritmo de boosting generó una mejor precisión de clasificación al conjunto de datos de prueba, la cual fue de un 95 % de datos correctamente clasificados. Esta librería genera una gráfica de los predictores que tuvieron mayor influencia en el entrenamiento, la cual se muestra a continuación.



(g) Figura 7. Importancia de los predictores en el boosring.

En la Figura 7 se muestra el nivel de importancia de cada variable y esto mediante un puntaje que asigna el algoritmo al hace en el entrenamiento.
A continuación se añade el código utilizado para trabajar en este ejercicio, de igual manera se adjuntará el código en los archivos que se enviarán.

```
library("rpart")
library("ada")
library("caret")

setwd("directorio")

data <- as.data.frame(read.table("spambase.data", sep = ",", header = FALSE))
names(data)[length(names(data))] <- "Y"
data$Y <- factor(data$Y)

train_data_per <- 0.75
shuffled_data <- data[sample(nrow(data)),]
smp_size <- floor(train_data_per * nrow(data))
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

train <- shuffled_data[train_ind, ]
```

```
test <- shuffled_data[-train_ind, ]
test_labels <- test$Y
#test$Y <- NULL

#Entrenamiento de los arboles de decision
no_trees <- 100
no_pred <- trunc(sqrt(ncol(train)))
predictions <- matrix(nrow = nrow(test), ncol = no_trees)
for(i in 1 : no_trees) {
  pred <- names(train)[sample(1 : (ncol(train) - 1), no_pred, replace = F)]
  variables <- as.formula(paste("Y ~ ", paste(pred, collapse = "+")))
  sub_set <- train[sample(1 : nrow(train), replace = T), ]
  fit <- rpart(variables, method = "class", data = sub_set)
  predictions[, i] <- predict(fit, newdata = test, type = "class")
  #Plot del modelo cada 10 iteraciones
  if(i %%10 == 0) {
    plot(fit, uniform = TRUE, main = "Classification Tree for SPAM")
    text(fit, use.n = TRUE, all = TRUE, cex = .8)
  }
}
results <- rep(0, nrow(test))
for(i in 1 : nrow(test)) {
  no_c1 <- 0
  no_c2 <- 0
  for(j in 1 : no_trees) {
    if(predictions[i, j] == 1) {
      no_c1 <- no_c1 + 1
    }
    else {
      no_c2 <- no_c2 + 1
    }
  }
  if(no_c1 > no_c2) {
    results[i] <- 0
  }
  else {
    results[i] <- 1
  }
}
results <- factor(results)
confusionMatrix(results, test$Y)

#Hacer boosting
test_data_per <- 0.25
smp_size <- floor(test_data_per * nrow(test))
```

```
test_ind <- sample(seq_len(nrow(test)), size = smp_size)

test_boost <- test[test_ind, ]
test_res <- test[-test_ind, ]
control <- rpart.control(cp = -1, maxdepth = 14, maxcompete = 1, xval = 0)
boost <- ada(Y ~ ., data = train, test.x = test_boost[, -ncol(test_boost)],
test.y = test_boost[, ncol(test_boost)], type ="discrete",
control = control, iter = 70)
pred_boost <- predict(boost, newdata = test_res[, -ncol(test_res)])
test_labels <- test_res[, ncol(test_res)]
confusionMatrix(pred_boost, test_labels)
```