Aprendizaje Automático

Jorge Gangoso Klöck 49398653N
Práctica 2 - Entrega: 01/05/2021

1. Ejercicio sobre la complejidad de H y el ruido

1.1 Dibujar gráficas con las nubes de puntos

Procedemos sencillamente a generar los puntos siguiendo las distribuciones probabilísticas uniformes y gaussianas (haciendo uso de las funciones dadas: "*simula_unif*" y "*simula_gaus*") y realizaremos un scatter plot de ambos utilizando la primera columna de **x** como primera coordenada y la segunda como segunda coordenada de los puntos.

1.2 Valorar la influencia del ruido en la selección de la complejidad de la clase.

1.2.a Graficar el etiquetado de la función f(x) y la recta de clasificación

Creamos una recta mediante la función "*simula_recta*" proporcionada en la plantilla y la usamos para clasificar los puntos aleatorios generados mediante "*simula_unif*" una vez más.
Hacemos un plot diferenciando mediante colores los puntos que han sido clasificados como positivos y aquellos clasificados como negativos.
Añadimos al plot la recta de forma ax+b que hemos creado anteriormente para observar como todos los puntos están perfectamente clasificados según si se encuentran encima de la recta o debajo.
Finalmente, escribimos por pantalla el porcentaje de error de clasificación cometido por nuestra función, que en este caso al ser la propia recta es de 0%.

1.2.b Graficar el etiquetado de la función f(x) y la recta de clasificación sobre un etiquetado con ruido.

Repetimos el proceso anterior pero añadimos ruido a las etiquetas tras su cálculo en un 10% de estas. En esta ocasión, al comprobar el porcentaje de error vemos como aproxima a 10%.
Al tratarse de tan pocos elementos y ser la cantidad de elementos en la muestra un número entero, el porcentaje no siempre puede ser del 10% exacto, y ya que el redondeo se lleva a cabo hacia abajo, se suele obtener un error del 9%.

1.2.c Suponer funciones complejas como fronteras de clasificación.

Para este ejercicio creamos las 4 funciones descritas en el enunciado mediante funciones lambda y le pasamos a la función "*plot_datos_cuad*" del template las funciones junto al etiquetado obtenido en el punto 1.2.b y a la nube de puntos original.
Además, calculamos las etiquetas que recibirán los elementos según estas nuevas fronteras y calculamos el porcentaje de error cometido por cada una de ellas.
Los resultados son los siguientes (para la semilla inicial):

Función 1: Misclassification rate 41%
Función 2: Misclassification rate 49%
Función 3: Misclassification rate 84%
Función 4: Misclassification rate 73%

Podemos ver como el porcentaje aumenta drásticamente. Los valores también se vuelven inconsistentes entre ejecuciones con semillas diferentes y se puede adjudicar en la mayoría de ocasiones un porcentaje de error de como mínimo el 50%, indicando que la función usada no es mejor que cualquier función aleatoria que podríamos escoger.
Vamos a comprobar los porcentajes de error para 10 simulaciones con semillas diferentes y así comprobar de entre las 4 funciones cual tiende a tener menos error dentro del caos que supone esta estrategia.
Los resultados son los siguientes (semillas 1-10):

Función 1: Misclassification rate 54.3%
Función 2: Misclassification rate 55.1%
Función 3: Misclassification rate 63.3%
Función 4: Misclassification rate 52.1%

Como era de esperar, todas ellas rondan el 50%. Incluso en las iteraciones con menor porcentaje ninguna de las funciones es capaz de bajar de entorno al 30% de error, lo cual demuestra que haber utilizado simplemente una ecuación de recta aleatoria, aunque de media también tendría un error considerable, tiene mucha mejor probabilidad de acabar clasificando bien los datos.
Sin embargo, hay que considerar también que las iteraciones en las que se ha conseguido mejores valores de Misclassification rate para las funciones complejas, eran aquellos que daban una mayor tasa de error para la función lineal simple, demostrando la puntual capacidad de estas funciones de clasificar datos que no son linealmente separables, y demostrando que, con un ajuste correcto, una función compleja puede llegar a clasificar nubes de puntos que un ajuste de recta lineal no puede.

2. Modelos Lineales

2.a Algoritmo Perceptron

Implementamos el algoritmo de aprendizaje para el Perceptron (PLA). El pseudocódigo empleado es el que aparece en el documento “Trabajo2.pdf”.
El algoritmo recibe como parámetros una matriz de datos estructurada de forma que cada elemento en X tiene la forma:
$x = x_0\{1\} + x_1 + x_2$ [Para nuestro ejemplo, aunque puede tener m

componentes]
Unos pesos iniciales que se irán ajustando en la forma:
$w = w_0 + w_1 + w_2$ [Para nuestro ejemplo, aunque tendrá m componentes como

x]
Un vector de etiquetas, donde para cada elemento de X le asigna un valor:
$y = \{-1|1\}$
Y un número máximo de iteraciones tras el cual, si el algoritmo no ha llegado a converger parará.

El pseudocódigo concreto de mi algoritmo es el siguiente (aunque es casi idéntico al del documento):

```
funcion Ajusta_PLA
(MATRIX(n*m) datos, VECTOR(n) label, INT max_iter, VECTOR(m) vini):
      n <- n
      w <- vini
      mientras (iteraciones < max_iter && cambios) hacer:
            para i en (0...n) hacer:
            si (signo(w*datos[i]) es_distinto_de label[i]) hacer:
                  w_despues = w_antes + label[i] * datos[i]
            si (w_despues = w_antes)      cambios = falso
            iteraciones++
      devolver w, iteraciones
```

2.a.1 Ejecutar PLA sobre datos del punto 1.2.a

Ejecutamos el algoritmo presentado anteriormente a partir de los valores de x y sus respectivas etiquetas calculadas durante el apartado 1.2.a (sin ruido). Como máximo de iteraciones establecemos 1000 ya que es un número que ha demostrado ser más que suficientemente elevado para permitir que el algoritmo converja.

A continuación se establecen dos apartados: uno con el vector inicial a 0, y otro con un vector inicial aleatorio con valores en el intervalo [0,1].

a) PLA(datos{1.2.a}, y{1.2.a}, 1000, vini{0,0,0}):
   Converge en la iteracion 75 (con la semilla 1)
   Comprobamos si la convergencia es correcta, para ello usamos los pesos devueltos por el algoritmo PLA y calculamos las etiquetas con estos pesos. El ratio de error en la clasificación es del 0% lo cual indica que en principio ha podido converger correctamente.
   Para asegurar graficamos el frente de clasificación generado por los pesos y lo superponemos a la nube de puntos para comprobar como efectivamente separa perfectamente los elementos catalogados como positivos de los negativos.

b) PLA(datos{1.2.a}, y{1.2.a}, 1000, vini{random[0,1],random[0,1],random[0,1]}):
   Realizamos la prueba 10 veces para comprobar el valor medio de iteraciones que tarda en converger y obtenemos el valor de 159.1 iteraciones (con la semilla 1).
   El valor con respecto al apartado a) se ha duplicado, dando a entender que los pesos iniciales (0,0,0) eran una mejor aproximación que los obtenidos por los valores aleatorios. Por supuesto esto es susceptible a cambio entre iteraciones, pero la gran diferencia de iteraciones demuestra lo sensible que es el algoritmo a la decisión de valores iniciales para los pesos.

2.a.2 Ejecutar PLA sobre datos del punto 1.2.b

Ejecutamos el algoritmo presentado anteriormente a partir de los valores de x y sus respectivas etiquetas calculadas durante el apartado 1.2.b (con ruido).
En este caso, el algoritmo no llega a converger como veremos más adelante, por esto, en la prueba con una sola ejecución he aumentado el máximo de iteraciones a 10.000 para obtener un ratio de error menor y graficar una frontera de decisión más aproximada a la real.

a) PLA(datos{1.2.b}, y{1.2.b}, 1000**0**, vini{0,0,0}):
   Como comentaba anteriormente el algoritmo PLA no converge, para tras las 10.000 iteraciones con una tasa de error del 19% (semilla 1). Si modificamos la cantidad de iteraciones podemos ver como realmente aumentando las iteraciones consigue mejorar normalmente la tasa de error, pero nunca puede llegar a bajar hasta 0%, explicaremos ahora por qué.
   Al graficar el resultado podemos observar como el algoritmo devuelve una frontera de clasificación que es aproximadamente correcta pero no es capaz de clasificar todos los elementos correctamente, sólo una gran parte de ellos.

b) PLA(datos{1.2.b}, y{1.2.b}, 1000, vini{random[0,1],random[0,1],random[0,1]}):
   El valor medio de iteraciones hasta converger es de 1000. Esto indica que en ninguna de las 10 ejecuciones el algoritmo ha llegado a converger antes de la condición de parada.

La razón para esto es bastante sencilla, por cómo se han preparado los datos, estos datos con ruido no son linealmente separables, y por tanto, es imposible que el algoritmo pueda llegar a converger.
Sencillamente no puedes establecer la recta que separa todos los puntos correctamente si no existe ninguna recta que sea capaz de hacer dicha tarea.

2.b Regresión Logística

Este ejercicio consiste en un único experimento, para el cual usaremos el algoritmo de Regresión Logística basado en el pseudocódigo del guión, para aplicarlo a 100 nubes de 100 puntos bidimensionales situados en el entorno X = [0,2] x [0,2] mediante distribución uniforme.
Para cada nube de puntos calcularemos el Eout empleando una nube de 1000 puntos siguiendo la misma distribución y finalmente realizaremos la media de los Eout obtenidos así como de la cantidad de épocas que se han empleado en cada iteración.

De forma adicional, he decidido emplear el algoritmo una vez aislada para graficar la clasificación llevada a cabo por la regresión logística, ver los pesos obtenidos y calcular además su tasa de misclassification.

Este ejercicio me ha llevado algunas complicaciones ya que tengo algunas dudas al respecto del funcionamiento exacto del experimento. Al realizar las comprobaciones sobre cómo clasificaba la regresión logística me di cuenta que los resultados estaban siendo catastróficos, y busque soluciones al algoritmo en busca de errores. La versión definitiva que va a ser entregada implementa una de las soluciones que encontré.
Los pesos únicamente se actualizan si la previsión de la función sigmoidal es incorrecta. En caso de ser correcta no se modifican los pesos para ese elemento. No se si este procedimiento es correcto o no, pero experimentalmente desde que lo incorporé los resultados se volvieron mucho más razonables y he optado por dejarlo así.