

memoriap2.pdf



postdata9



Aprendizaje Automatico



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada



¿Atascado con tu TFG?

Aquí tenemos la solución
Trabajos universitarios por encargo





Menú
Doble Texas®

Menú
Long Chicken®



27€
x 2

Menú Crispy
Chicken® BBQ
con queso

Menú
Big King®



AUTO KING



PARA LLEVAR



RESTAURANTE

Válido hasta 10/05/21. Elección de 2 menús pequeños entre menú Crispy Chicken® BBQ con queso, Big King®, Doble Texas o Long Chicken®. Por +0,50€/menú mediano, por +1€/menú grande. Patatas Supreme excepto para menús pequeños. Agua de 0,33cl en menú pequeño y 0,5cl en el resto de menús. Excluidos refrescos embotellados. Cerveza no disponible en menús pequeños. Restaurantes no adheridos en www.burgerking.es. COCA-COLA® y COCA-COLA ZERO® son marcas registradas de THE COCA-COLA COMPANY. TM Burger King Corporation. © 2021 Burger King Europe GmbH. BURGER KING® se reserva el derecho a ampliar el periodo promocional. Todos los derechos reservados.

**FORMACIÓN ONLINE Y
PRESENCIAL EN GRANADA**

**Clases de Inglés B1, B2, C1
DELF B1 y DELF B2 de Francés**

Centro Preparador
PREMIUM
Sede de Matriculaciones
Sede de Exámenes Oficiales

EXAMS
ANDALUCIA
English Qualification



**PUERTA
REAL**

Academia de Enseñanza

academia-granada.es



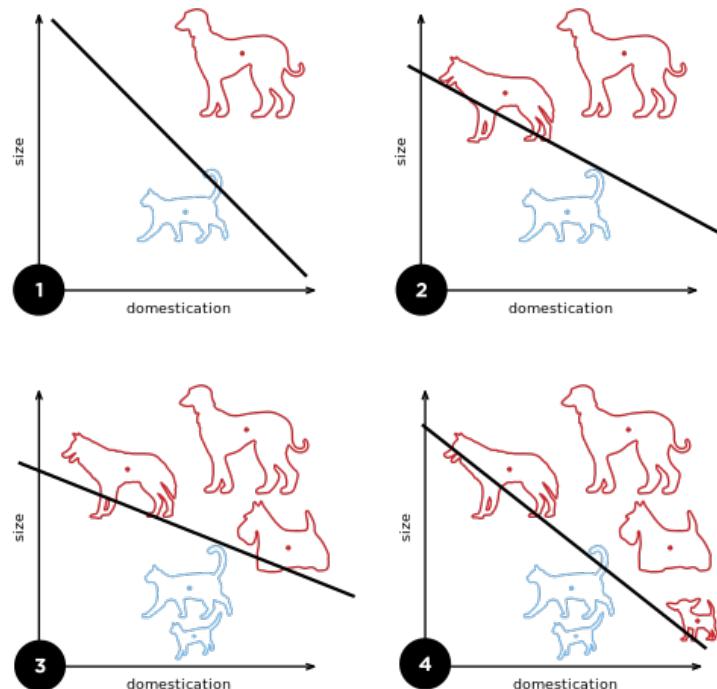
**MEJORA TU
INGLÉS**
PLAZAS DISPONIBLES

ASIGNATURAS DE UNIVERSIDAD:
HACEMOS GRUPOS
PARA CLASES DE APOYO

APRENDIZAJE AUTOMÁTICO PRÁCTICAS

TRABAJO 2

PROGRAMACIÓN



2018 - 2019

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

ÍNDICE:

- **Ejercicio sobre la complejidad de H y el ruido**
 - Ejercicio 1
 - Apartado a)
 - Apartado b)
 - Ejercicio 2
 - Apartado a)
 - Apartado b)
 - Ejercicio 3
- **Modelos Lineales**
 - Algoritmo Perceptron
 - Apartado a)
 - Apartado a)
 - Apartado b)
 - Apartado b)
 - Regresión Logística
 - Apartado a)
 - Apartado b)

COMPLEJIDAD DE H Y EL RUIDO

Para la realización de los ejercicios siguientes, nos facilitan cuatro funciones auxiliares, las cuales son:

- **simula_unif(*N, dim, rango*)**, que calcula una lista de N vectores de dimensión dim . Cada vector contiene dim números aleatorios uniformes en el intervalo $rango$;
- **simula_gaus(*N, dim, sigma*)**, que calcula una lista de longitud N de vectores de dimensión dim , donde cada posición del vector contiene un número aleatorio extraído de una distribución Gaussiana de media 0 y varianza dada, para cada dimensión, por la posición del vector $sigma$;
- **simula_recta(*intervalo*)**, que simula de forma aleatoria los parámetros, $v = (a, b)$ de una recta, $y = ax + b$, que corta al cuadrado $[-50, 50] \times [-50, 50]$; g
- **plot_datos_cuad(*X, y, fz, title, xaxis, yaxis*)**, dibuja los datos X con la etiqueta y y clasificándolos según la función fz .



FORMACIÓN ONLINE Y PRESENCIAL EN GRANADA

**Clases de Inglés B1, B2, C1
DELF B1 y DELF B2 de Francés**

Centro Preparador
PREMIUM
Sede de Matriculaciones
Sede de Exámenes Oficiales

EXAMS
ANDALUCIA
English Qualification

**PUERTA
REAL**
Academia de Enseñanza
academia-granada.es

Dibujar una gráfica con la nube de puntos de salida correspondiente.

Ejercicio 1

Apartado a)

Considere $N = 50$, $dim = 2$, $rango = [-50, +50]$ con $\text{simula_unif}(N, dim, rango)$.

Llamamos a la función auxiliar `simula_unif`, para que nos devuelva una lista de 50 vectores, donde cada vector se compone de dos elementos, cuyos valores se encuentran entre -50 y 50.

unif - Arreglo de NumPy	
0	1
-8.2978	22.0324
-49.9886	-19.7667
-35.3244	-40.7661
-31.374	-15.4439
-10.3233	3.88167
-8.08055	18.522
-29.5548	37.8117

Formato Redimensionar Color de fondo
Guardar y Cerrar Cerrar

Primeros valores extraídos, a)

gauss - Arreglo de NumPy	
0	1
0.705781	-5.35024
-0.684693	2.19061
0.514507	2.01609
-0.497141	-0.531156
0.417164	1.08489
0.443412	0.314867
-1.49965	0.99894

Formato Redimensionar Color de fondo
Guardar y Cerrar Cerrar

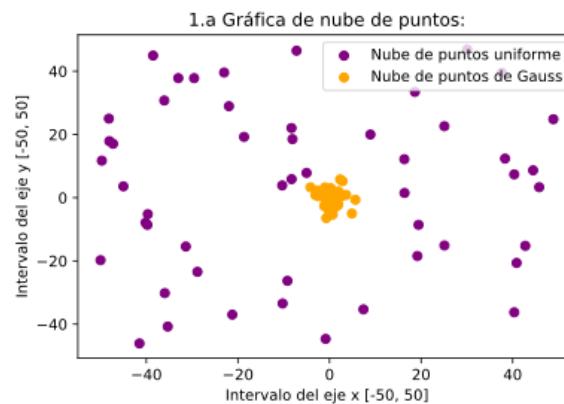
Primeros valores extraídos, b)

Apartado b)

Considere $N = 50$, $dim = 2$, $sigma = [5, 7]$ con $\text{simula_gaus}(N, dim, sigma)$.

Llamamos a la función auxiliar `simula_gaus`, para que nos devuelva una lista de 50 vectores. Cada vector se compone de dos elementos, cuyos valores son aleatoriamente extraídos de la distribución Gaussiana.

La gráfica correspondiente a ambas salida es la siguiente:



Como se observa, la nube de puntos uniforme se encuentra distribuida por toda la gráfica, entre los valores [-50,50], mientras que la nube perteneciente a la distribución de Gauss permanece concentrada en una zona, alrededor del punto [0,0].

Ejercicio 2

Con ayuda de la función `simula_unif()` generar una muestra de puntos 2D a los que vamos a añadir una etiqueta usando el signo de la función $f(x,y) = y - ax - b$, es decir, el signo de la distancia de cada punto a la recta simulada con `simula_recta()`.

Apartado a)

Dibujar una gráfica donde los puntos muestren el resultado de su etiqueta, junto con la recta usada para ello. (Observe que todos los puntos están bien clasificados respecto de la recta).

Para obtener la pendiente de la recta y el término independiente de forma aleatoria, utilizo la función `simula_recta`, pasándole el intervalo [-50,50]. Como ya tenemos los valores de a y b , podemos utilizar la recta para clasificar los datos obtenidos en el ejercicio anterior, esto es, obtener el signo que toma el punto en la función $y - ax - b$. Para ello, he implementado una función auxiliar, `fun_etq`, que dados el punto, la pendiente y el término independiente calcula el signo. Con un `for`, voy llamando a la función con cada punto de la lista de `unif` y lo voy almacenando en un array `etiq`.

Para poder realizar la gráfica, hice este apartado de dos formas distintas. La primera forma es la más larga, y luego caí en que podía hacerlo de una forma más sencilla.

PRIMER PROCEDIMIENTO:

Realicé un bucle `for` en el cual en cada iteración comprobaba si la etiqueta era positiva o negativa. Si la etiqueta era positiva, pintaba el punto con `plt.scatter` de color morado; si era negativa, de color naranja. En cada iteración, utilizaba el `plt.scatter` con el argumento `label`, por lo que al ejecutar el programa, la leyenda tenía el tamaño del vector de `unif`, había tantas etiquetas en la leyenda como puntos había en el vector de `unif`.

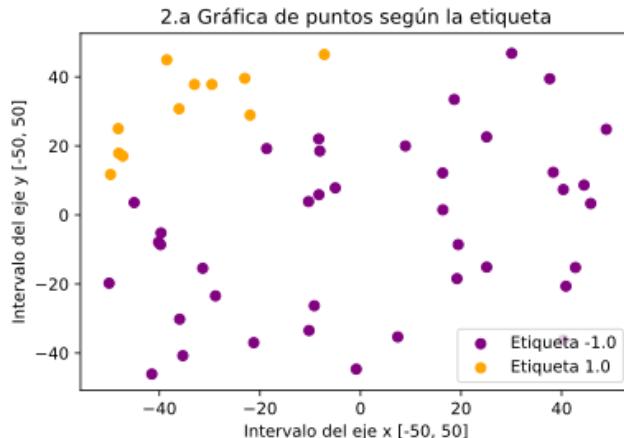
Para solucionar esto, pensé en pintar la etiqueta cuando se fuera a dibujar el último punto de `unif`. Para ello, contabilicé el número de etiquetas positivas y negativas, y al `for` anterior le añadí la comprobación de que cuando fuera el último elemento de la clase negativa o positiva, en el `plt.scatter` se usara `label`.

Después de realizarlo de esta manera, caí en la cuenta que podía pintar la etiqueta no cuando fuera a dibujar el último punto sino el primero, con lo que me ahorraría el contabilizar los elementos que había de cada clase.

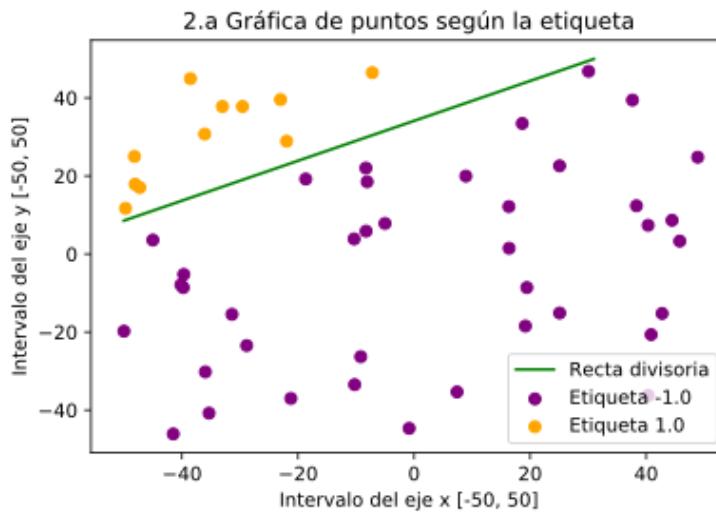
SEGUNDO PROCEDIMIENTO:

Esta segundo procedimiento es dibujar la gráfica de forma indexada. Para esto me creo dos listas, en una almaceno aquellos puntos con etiqueta positiva y en la otra aquellos puntos con etiquetas negativas.

Para pintar con `plt.scatter`, primero le paso la coordenada de la x , que es la primera columna de la lista, y después la coordenada de la y , que es la segunda columna.



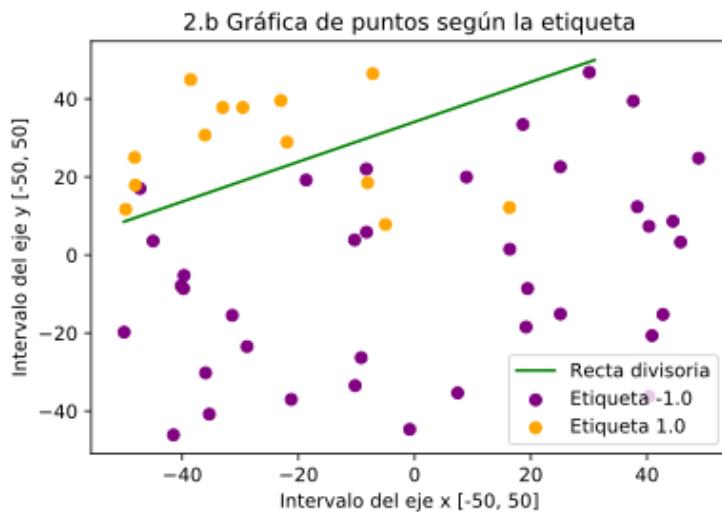
Con esto he juntado en una gráfica todos los puntos diferenciados por la clase a la que pertenecen, que se aprecia por el color. Para poder visualizar la recta que separa a ambos tipos, necesito 2 puntos, el inicial y el final. Sabemos que la x_i inicial es -50 y la x_f final 50. El valor de la y asociada a dichas x , la conseguimos despejando de $f(x,y) = y - ax - b$, de forma que nos quedaría $y = ax + b$. Sustituyendo en esta función los valores de a y b por los ya calculados, y el valor de la x por -50 y 50, tenemos ambas y . Utilizamos plot para pintar, plt.plot((x_i, x_f) , (y_i, y_f) , color, etiqueta).



Apartado b)

Modifique de forma aleatoria un 10% etiquetas positivas y otro 10% de negativas y guarde los puntos con sus nuevas etiquetas. Dibuje de nuevo la gráfica anterior. (Ahora hay puntos mal clasificados respecto de la recta).

Como ya tenemos dos listas cada una con los puntos de cada clase, realizo una copia de ambas para trabajar sobre ellas. Calculo el número de elementos que supone el 10% de cada clase y desordeno ambas listas, para que se modifique de forma aleatoria. Guardo los primeros n valores que suponen el 10% y los elimino de la lista, para después insertarlos en la lista opuesta. De esta forma, he insertado en cada lista elementos que no son de su clase, es decir, hay puntos mal clasificados como se puede visualizar en la gráfica. Encontramos puntos de color amarillo por debajo de la línea divisoria y un punto morado por encima.



Ejercicio 3

Supongamos ahora que las siguientes funciones definen la frontera de clasificación de los puntos de la muestra en lugar de una recta

- $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$
- $f(x, y) = 0,5(x + 10)^2 + (y - 20)^2 - 400$
- $f(x, y) = 0,5(x - 10)^2 - (y + 20)^2 - 400$
- $f(x, y) = y - 20x^2 - 5x + 3$

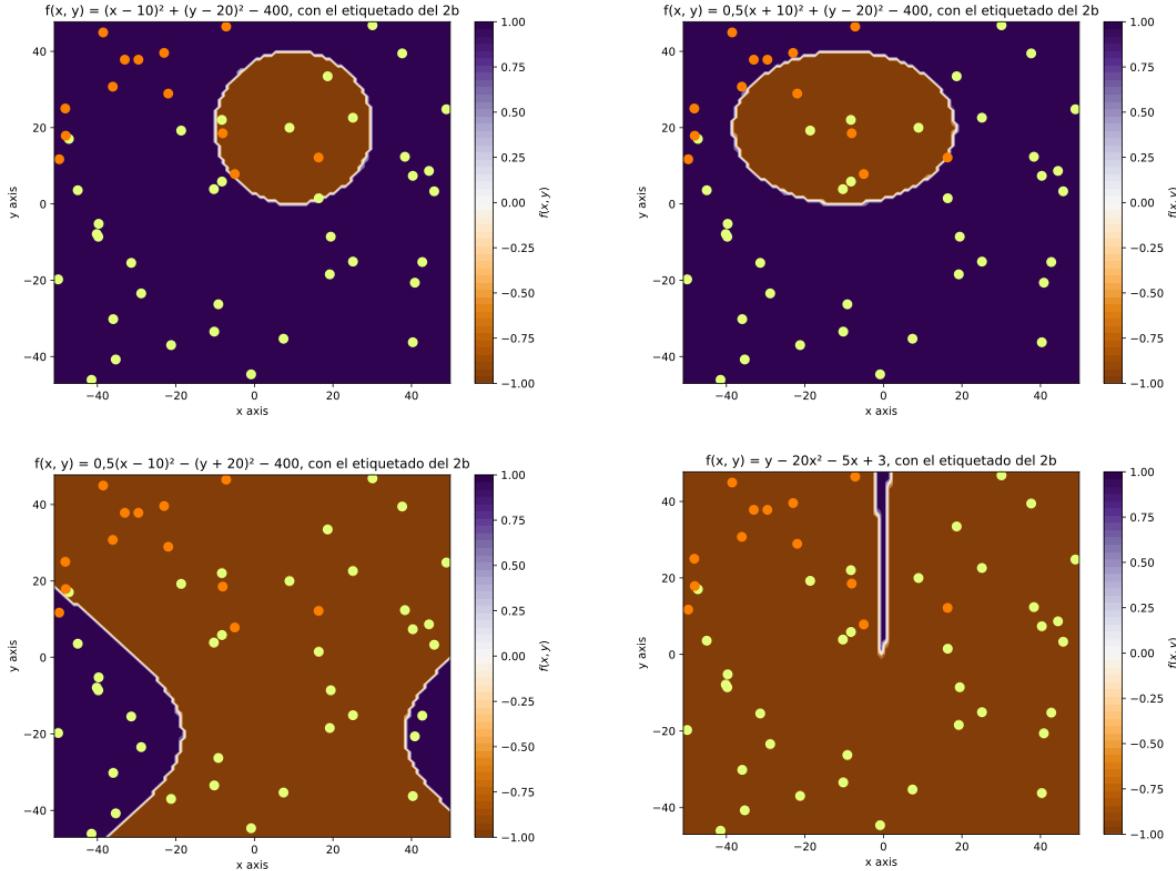
Visualizar el etiquetado generado en 2b junto con cada una de las gráficas de cada una de las funciones. Comparar las formas de las regiones positivas y negativas de estas nuevas funciones con las obtenidas en el caso de la recta. ¿Son estas funciones más complejas mejores clasificadores que la función lineal? ¿En qué ganan a la función lineal? Explicar el razonamiento.

Para este ejercicio se nos facilita la función `plot_datos_cuad` para dibujar la nube de puntos con su etiqueta y para clasificar los puntos con la función que deseemos. Tiene como argumento `fz` una función, esto es, `fz` es un método que devuelve una de las funciones anteriores, por lo que me declaré cuatro funciones auxiliares, una para cada $f(x,y)$.

Esta función requiere además como argumento los puntos del ejercicio anterior 2b, con sus respectivas etiquetas. Ambas listas de puntos según su clase, sólo tienen el punto, esto es, la coordenada x y la coordenada y , pero no tiene su etiqueta asociada, y para pasarlle como argumento es necesario. A cada lista le añado una columna de 1.0, -1.0 según si es positiva o negativa.

Una vez tengo ambas listas de puntos junto su etiqueta asociada, las uno en una sola con la función `concatenate`. Con esta única lista, ya puedo llamar a la función `plot_datos_cuad` con cada función. Como primer parámetro, le paso las dos primeras columnas de esa lista, como segundo parámetro, la tercera columna que son las etiquetas y por último le paso la función clasificadora.

Obtenemos las siguientes gráficas con la nube de puntos del ejercicio 2b.



FORMACIÓN ONLINE Y PRESENCIAL EN GRANADA

Clases de Inglés B1, B2, C1
DELF B1 y DELF B2 de Francés

Centro Preparador
PREMIUM
Sede de Matriculaciones
Sede de Exámenes Oficiales

EXAMS
ANDALUCIA
English Qualification®

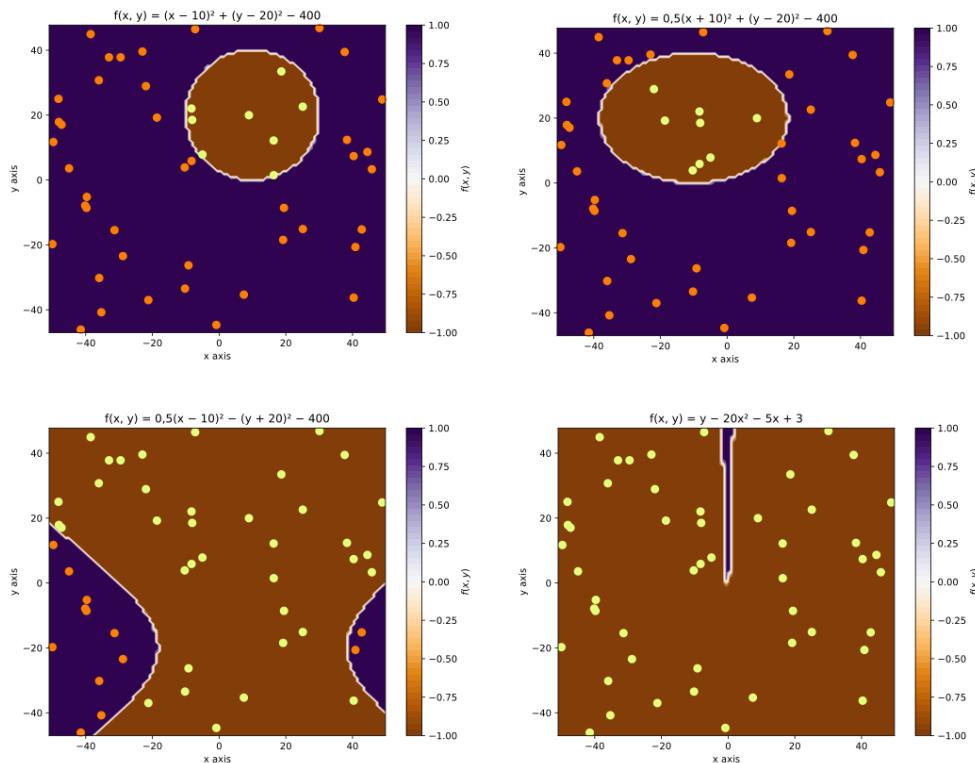


PUERTA REAL

Academia de Enseñanza

academia-granada.es

Para obtener las gráficas anteriores cuya nube de puntos queremos clasificarla según la función, cuando llamamos a `plot_datos_cuad` para pintar la gráfica, en vez de pasarle la tercera columna de la lista, le paso el signo de la función que tendría en los puntos. De esta forma, las gráficas obtenidas son las siguientes:



Como podemos apreciar en las gráficas cuyas etiquetas son del ejercicio 2.b), están etiquetadas según una función lineal, las funciones cónicas que tenemos no clasifican bien los puntos. Que las funciones sean más complejas no implica que sean mejores clasificadoras. La ventaja de estas funciones cónicas con respecto a las lineales es que permiten ordenar los datos en aquellos casos en los que no son linealmente separables, es decir, que la nube de datos no tiene por qué ser separable por un hiperplano para poder ser clasificada correctamente.

MEJORA TU
INGLÉS
PLAZAS DISPONIBLES

ASIGNATURAS DE UNIVERSIDAD:
HACEMOS GRUPOS
PARA CLASES DE APOYO

MODELOS LINEALES

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

ALGORITMO PERCEPTRON

Ejercicio 1

Implementar la función `ajusta_PLA(datos, label, max_iter, vini)` que calcula el hiperplano solución a un problema de clasificación binaria usando el algoritmo PLA. La entrada `datos` es una matriz donde cada ítem con su etiqueta está representado por una fila de la matriz, `label` el vector de etiquetas (cada etiqueta es un valor +1 o -1), `max_iter` es el número máximo de iteraciones permitidas y `vini` el valor inicial del vector. La función devuelve los coeficientes del hiperplano.

Para implementar el algoritmo Perceptron me basé en el pseudocódigo de las transparencias del tema 2 de teoría:

```
- Given the data set  $(\mathbf{x}_n, y_n), n = 1, 2, \dots, N$ 
- Step.1: Fix  $\mathbf{w}_{\text{ini}} = 0$ 
- Step.2: Iterate on the  $\mathcal{D}$ -samples improving the solution:
  - repeat
    For each  $\mathbf{x}_i \in \mathcal{D}$  do
      if:  $\text{sign}(\mathbf{w}^T \mathbf{x}_i) \neq y_i$  then
        update  $\mathbf{w}$ :  $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + y_i \mathbf{x}_i$ 
      else continue
    End for
  - Until No changes in a full pass on  $\mathcal{D}$ 
```

La cabecera de la función es: `ajuste_PLA(puntos, etiquetas, max_iter, w_act)`, donde los puntos son los puntos, una matriz de 2 columnas en la que la primera columna se corresponde con la coordenada x del punto mientras que la segunda columna se refiere a la coordenada y del punto; a estos puntos le añado una columna de 1 por la cual se va a multiplicar.. `Etiquetas` son las etiquetas de dichos puntos que serán 1.0 ó -1.0, `max_iter` es el número máximo de iteraciones que va a realizar el algoritmo y `w_act` es el vector inicial por el cual se empezará.

El algoritmo iterará en un bucle while mientras no se llegue al máximo número de iteraciones y mientras el vector de pesos actual y el anterior no sean iguales. Dentro de este bucle, actualizo el vector de pesos anterior con el actual, aumento el número de iteraciones y, para cada punto, calculo su etiqueta. Si es distinta de la que le corresponde, actualizo el vector de pesos actual. Además comprobar si el error es igual a 0, que de ser así salgo del bucle retornando el vector de pesos actual y el número de iteraciones, ya que no seguirá iterando.

El algoritmo devuelve el número de iteraciones y el vector de pesos actual.

Apartado a)

Ejecutar el algoritmo PLA con los datos simulados en los apartados 2a de la sección.1. Inicializar el algoritmo con:

a) el vector cero

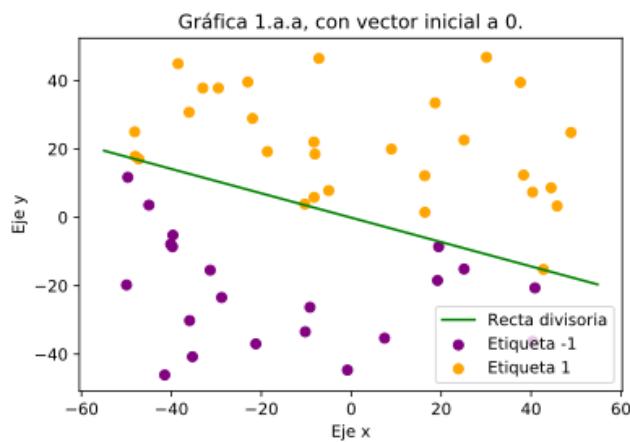
Al tener que realizar este ejercicio en otro fichero, volvemos a repetir lo realizado en el ejercicio 1 de la sección anterior. Obtenemos los puntos con la función `simula_unif`; obtenemos la pendiente y el término independiente de una recta con la función `simula_recta` y sacamos las etiquetas de los puntos respecto a la función $f(x, y) = y - a*x - b$.

Como ya tenemos los datos junto sus etiquetas, llamamos a la función `ajuste_PLA` con los datos, las etiquetas, un máximo de iteraciones de 50 y el vector inicial de 0.

Pintamos los puntos con `plt.scatter`, mientras que para pintar la recta llamo a la función auxiliar `pinta_recta`. Esta función la he implementado para que, dado el vector de pesos del algoritmo Perceptron, dibuje la línea que divide la nube de puntos en 2. Esta función recibe como parámetro la salida del algoritmo Perceptron, y a partir de dicho vector calcula la pendiente de la recta y el término independiente para después dibujar la recta. Además recibe también como parámetro un índice que determina si establece la etiqueta al realizar el plot.

Como se puede visualizar en la gráfica siguiente, el algoritmo ha clasificado bien los puntos ya que la recta separa aquellos puntos amarillos cuya etiqueta es positiva, de aquellos puntos morados cuya etiqueta es negativa.

Con el vector de 0, el algoritmo realiza 14 iteraciones.



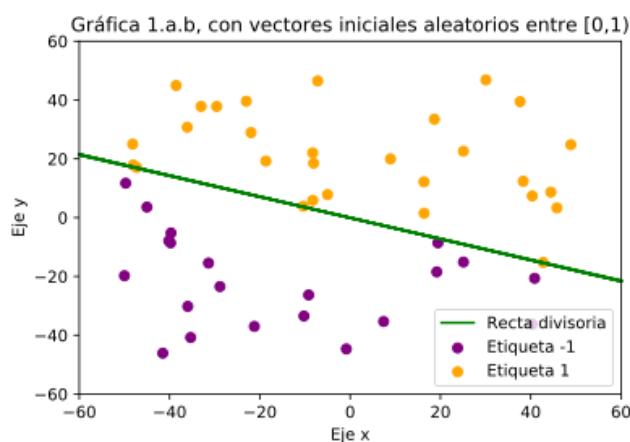
b) con vectores de números aleatorios en [0, 1] (10 veces).

Para realizar la ejecución del algoritmo 10 veces con 10 vectores iniciales distintos, he realizado un for que realice 10 iteraciones en el cual almaceno el vector de pesos resultante del algoritmo en una lista.

Dentro de este bucle, inicializo un vector inicial con dimensión 3 a cero, y a cada elemento de dicho vector le asigno un valor aleatorio usando la función de numpy random.rand. Una vez tengo el vector inicial llamo a *ajusta_PLA* y guardo el resultado en una lista.

Para dibujar la gráfica, pinto los datos con plt.scatter diferenciando si la etiqueta es negativa o positiva y pinto la recta con un bucle for llamando a la función *pinta_recta* con índice 1. Antes del for pinto la primera línea con la etiqueta, es decir, llamando a la función con índice 0.

Para este apartado, con las 10 ejecuciones del algoritmo, se calcula una media de 9.6 iteraciones para converger.



En este caso, el número de iteraciones para el algoritmo con un vector inicial de 0 es mayor que cuando se empieza con un vector aleatorio, pero esto también depende de la semilla que se le fije al principio.



FORMACIÓN ONLINE Y PRESENCIAL EN GRANADA

**Clases de Inglés B1, B2, C1
DELF B1 y DELF B2 de Francés**

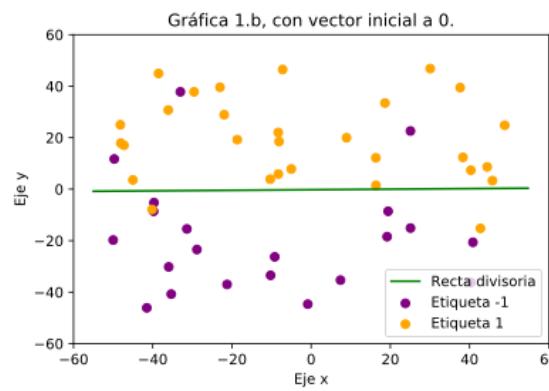
Centro Preparador
PREMIUM
Sede de Matriculaciones
Sede de Exámenes Oficiales

EXAMS
ANDALUCIA
English Qualification®

**PUERTA
REAL**
Academia de Enseñanza
academia-granada.es

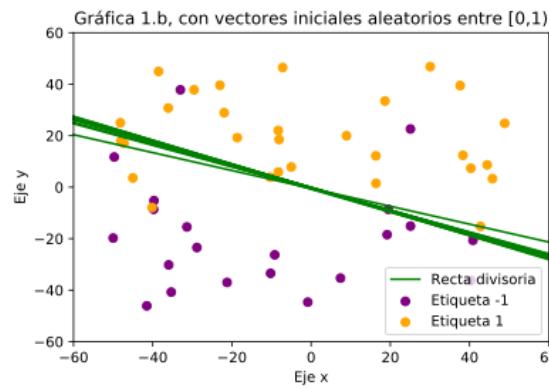
Para este apartado realizo exactamente lo mismo que el apartado anterior, pero usando los datos con un 10% de ruido en cada clase.

La gráfica obtenida para el vector inicial de 0 es:



Vemos que clasifica bastante bien, aunque no tiene en cuenta el ruido. El número de iteraciones que realiza es el máximo que se le pasa como argumento, he realizado varias ejecuciones con distintos valores de `max_iter`, y en todas ellas alcanza el máximo número de iteraciones.

La gráfica obtenida para los vectores iniciales aleatorios es:



Se visualiza que las distintas rectas varían, no son la misma, aunque varíen muy poco. El número de iteraciones medio obtenido coincide con el número máximo de iteraciones.

La cuestión de que el algoritmo ejecute tantas iteraciones como se le permitan, se debe al ruido, a que no encuentra cómo clasificar con el ruido que se le ha insertado a los datos, por lo que busca todo lo que posible para converger.

REGRESIÓN LOGÍSTICA

Ejercicio 2

En este ejercicio crearemos nuestra propia función objetivo f (una probabilidad en este caso) y nuestro conjunto de datos D para ver cómo funciona regresión logística. Supondremos por simplicidad que f es una probabilidad con valores 0/1 y por tanto que la etiqueta y es una función determinista de x .

Consideremos $d = 2$ para que los datos sean visualizables, y sea $X = [0, 2] \times [0, 2]$ con probabilidad uniforme de elegir cada $x \in X$. Elegir una línea en el plano que pase por X como la frontera entre $f(x) = 1$ (donde y toma valores +1) y $f(x) = 0$ (donde y toma valores -1), para ello seleccionar dos puntos aleatorios del plano y calcular la línea que pasa por ambos. Seleccionar $N = 100$ puntos aleatorios $\{x_n\}$ de X y evaluar las respuestas $\{y_n\}$ de todos ellos respecto de la frontera elegida.

Apartado a)

Implementar Regresión Logística(RL) con Gradiente Descendente Estocástico (SGD) bajo las siguientes condiciones:

- Inicializar el vector de pesos con valores 0.
- Parar el algoritmo cuando $\|w^{(t-1)} - w^{(t)}\| < 0,01$, donde $w^{(t)}$ denota el vector de pesos al final de la época t . Una época es un pase completo a través de los N datos.
- Aplicar una permutación aleatoria, 1, 2, ..., N , en el orden de los datos antes de usarlos en cada época del algoritmo.
- Usar una tasa de aprendizaje de $\eta = 0,01$.

Para poder implementar el algoritmo de Regresión Logística con Gradiente Descendente Estocástico, he necesitado de una serie de funciones auxiliares, las cuales son:

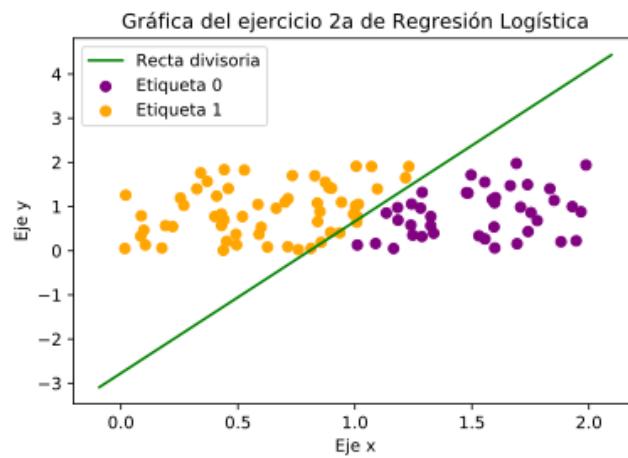
- **pinta_recta_reg(w, i)**, para pintar una recta para la regresión. Recibe como parámetros el vector de pesos resultante del algoritmo y un índice que determina si se dibuja la recta con etiqueta o sin ella;
- **sigmoide(x)**, devuelve la función sigmoide del valor x ;
- **$h(x, w)$** , calculaba el resultado de la función sigmoide aplicada al producto vectorial de w y x , donde w es un vector de pesos de dimensión 3 mientras que x es un punto con dos coordenadas;
- **estocastico(x, y, w)**, calcula el gradiente descendente estocástico en el punto x con su etiqueta y y el vector de pesos w ;
- **error_reg(x, y, w)**, estima el error logístico con el punto de dos coordenadas, su etiqueta y el resultado de realizar la regresión.

La función de **regresion**, la que implementa la Regresión Logística, recibe como parámetros la nube de puntos en un vector que se compone de 3 columnas, la primera en la que se encuentran los valores de la x , la segunda en la que se encuentran los valores de la y y una tercera columna de 1 para poder multiplicar. También se le pasa una y , que es un vector con las etiquetas de los puntos de x ; un número máximo de iteraciones como segunda condición de parada del bucle, y una tasa de aprendizaje.

En dicha función, vamos a ir iterando hasta que lleguemos a la condición de parada que nos especifica el enunciado o hasta que lleguemos al número máximo de iteraciones. En cada iteración, me quedo con una cierta parte de la población de los datos escogida aleatoriamente y calculo el vector de pesos sobre todos dicha selección, además de su error.

Como valores de retorno tenemos una lista de los errores que ha ido tomando el algoritmo y el vector w de pesos.

Para comprobar su correcto funcionamiento, la siguiente gráfica ha sido obtenida con la presente implementación.



Apartado b)

Usar la muestra de datos etiquetada para encontrar nuestra solución g y estimar E_{out} usando para ello un número suficientemente grande de nuevas muestras (>999).

Al necesitar realizar 1000 veces el mismo código, utilizo un for que itere 1000 veces. Dentro de dicho for en cada iteración, voy a utilizar nuevos datos, llamando a la función `simula_unif` y añadiéndole a esos datos una columna de unos. Después, obtengo las etiquetas, que serán un valor de 0 ó 1, y voy almacenando en una lista el error llamando a la función `error_reg`. Dibujo en una gráfica el error según la iteración y obtengo la media con la función `mean` de la librería `statistics`.

El error medio es -93.77822084417849

