2 Modelos lineales

Descripción del algoritmo de aprendizaje del perceptrón

Este algoritmo determinará un vector de pesos $w \in \mathbb{R}^d$ ajustado a través de los como sigue:

Sea S un conjunto de pares (x_i, y_i) con $i < |S|$ natural y denotando por S el tamaño de muestra.

Pues el algoritmo es el siguiente:

```
hay_cambio = True
w inicializada

mientras hay_cambio:

    hay_cambio = False

    Para todo (x_i, y_i) en S:
        si signo(w^T x_i) != yi:
            w = w + y_i x_i
            hay_cambio = True

devolver w
```

Si los datos son separables este algoritmo nos asegura la convergencia.

Sin embargo saber con certeza que un conjunto de datos es separable es un hipótesis bastante fuerte y por ejemplo, en el caso de $\mathbb{R}^d$ sabemos que existen configuraciones de $d + 1$ puntos que ya no son clasificables.

Es por ello que para nuestro problema hemos añadido además otro criterio de parada: cuando alcance un número de iteraciones máximo.

Esta solución introduce dos nuevos problemas:

1. Que el conjunto sea separable y converja pero se pare antes de alcanzar dicha solución.
2. Que el w final no sea el mejor de todos los que hemos calculado.

Para resolverlos, se podría plantear una solución en la que se tenga una función para medir el error de cierto w (por ejemplo la contadora de número de datos más clasificados), una variable para guardar el valor del w de menor error encontrado y una condición de parada nueva que combine la monotomía del error y el número de iteraciones.

Apartado 2.a.1.b

Se ha utilizado un máximo de 500 iteraciones como heurística a ver que ninguno las incumple, tras 10 iteraciones el número de iteraciones obtenido en cada uno de ellas es: 257, 43, 231, 71, 76, 59, 274, 235, 257, 74.

El número de iteraciones medio es de 157.7 y una desviación típica de 94.175, de aquí deducimos que que nuestro vector inicial sea el nulo es una buena heurística. Además podemos observar que la desviación típica es bastante grande en comparación con los datos que tenemos, esto nos hace pensar que en el valor inicial tiene relevancia a la hora del número de pasos necesarios.

Analicemos con más detalle el experimiento,

En esta tabla se han recogido los datos por número de pasos necesarios,

w_0 es el vector de pesos inicial y w_f el final

numero pasos	w_0	w_f
43	[0.228 0.664 0.497]	[464.228 15.388 23.746]
59	[0.032 0.093 0.065]	[558.032 19.363 29.714]
71	[0.996 0.816 0.594]	[663.996 23.150 31.898]
74	[0.476 0.013 0.353]	[673.476 22.585 31.349]
76	[0.975 0.902 0.596]	[661.976 24.899 36.1992]
231	[0.519 0.175 0.571]	[1078.519 39.474 53.764]
235	[0.168 0.973 0.767]	[1089.168 39.447 53.534]
257	[0.574 0.349 0.056]	[1115.574 43.477 62.122]
257	[0.824 0.633 0.669	[1148.824 39.897 60.948]
274	[0.452 0.375 0.975]	[1145.452 40.279 60.814]

Otro detalle interesante es que aunque se hable de convergencia de w, esto no es a una función concreta, si no a una familia de soluciones que cumple la propiedad de separar tales datos. Esto es notable en que ninguna de las w_f es igual.

Apartado 2.a.2

Sabemos que ahora los datos no son separables, luego por más pasos que demos estos no convergirán.

Además como el ruído introducido es del 10% la precisión máxima a la que podemos aspirar es a 90%.

Como este algoritmo no es de regresión, no se está minimizando ningún valor, simplemete se iteran los datos y oscilamos entorno a la solución. Para observar esto mejor he planteado el siguiente experimento,manteniendo los vectores iniciales del apartado anterior variaremos el número de épocas y veremos el vector final y su precisión.

Para $max_iter = 100$:

numero_pasos	w_0	w_f	Precisión (%)
100	[0.574, 0.349, 0.057]	[461.574, 29.602, 54.869]	86.0
100	[0.229, 0.664, 0.497]	[484.229, 28.285, 51.766]	86.0
100	[0.519, 0.175, 0.571]	[480.519, 28.202, 55.13]	86.0
100	[0.997, 0.817, 0.594]	[454.997, 29.599, 42.999]	82.0
100	[0.976, 0.902, 0.596]	[460.976, 22.188, 58.932]	83.0
100	[0.032, 0.094, 0.065]	[456.032, 4.774, 54.175]	76.0
100	[0.452, 0.375, 0.975]	[456.452, 10.36, 45.774]	79.0
100	[0.168, 0.973, 0.767]	[451.168, 29.135, 56.879]	85.0
100	[0.824, 0.633, 0.669]	[455.824, 25.473, 48.574]	86.0
100	[0.477, 0.013, 0.353]	[459.477, 0.021, 24.57]	77.0

Para max_iter = 200:

numero pasos	w_0	w_f	Precisión (%)
200	[0.574, 0.349, 0.057]	[484.574, 21.757, 62.042]	82.0
200	[0.229, 0.664, 0.497]	[473.229, -1.77, 19.977]	75.0
200	[0.519, 0.175, 0.571]	[493.519, 23.984, 58.465]	83.0
200	[0.997, 0.817, 0.594]	[480.997, 26.531, 30.417]	86.0
200	[0.976, 0.902, 0.596]	[503.976, 4.218, 21.352]	81.0
200	[0.032, 0.094, 0.065]	[485.032, 1.791, 36.703]	74.0
200	[0.452, 0.375, 0.975]	[478.452, 23.616, 47.269]	86.0
200	[0.168, 0.973, 0.767]	[494.168, 33.638, 50.445]	82.0
200	[0.824, 0.633, 0.669]	[486.824, 24.497, 38.54]	86.0
200	[0.477, 0.013, 0.353]	[476.477, 23.059, 63.463]	83.0

Para max_iter = 300:

numero_pasos	w_0	w_f	Precisión (%)
300	[0.574, 0.349, 0.057]	[495.574, 27.744, 53.525]	86.0
300	[0.229, 0.664, 0.497]	[484.229, 20.829, 64.077]	81.0
300	[0.519, 0.175, 0.571]	[501.519, 28.838, 52.773]	86.0
300	[0.997, 0.817, 0.594]	[488.997, 6.238, 49.528]	76.0
300	[0.976, 0.902, 0.596]	[491.976, 25.317, 35.578]	88.0
300	[0.032, 0.094, 0.065]	[490.032, 22.965, 49.033]	86.0
300	[0.452, 0.375, 0.975]	[487.452, 30.27, 53.141]	86.0
300	[0.168, 0.973, 0.767]	[486.168, 28.835, 55.224]	86.0
300	[0.824, 0.633, 0.669]	[480.824, 15.16, 55.684]	80.0
300	[0.477, 0.013, 0.353]	[493.477, 35.702, 52.526]	82.0

Regresión logística