Deber 2

Clasificacion por neurona simple

Roberto Alvarado (00206411)

Universidad San Francisco
Matematica
Redes Neuronales Artificiales
Universidad San Francisco de Quito
Ecuador
25 de Septiembre

Problema 1

Solución La función esta en documento de jupyter notebook que esta adjuntado en el archivo. Aqui esta descrita

```python
def modeloPolinomial(x,y,n,gamma= 0.001,cf=np.array([]),tolerancia = 10e-8, iteraciones= 10000):
  #Crea coeficientes random si no se especifican
  if(cf.size == 0):
      cf = np.random.rand(n+1)-0.5

  #Tensor de estimulos
  J = np.ones([1,n+1,len(x)])
  for i in range(n,-1,-1):
      J[0,i,:] = x**i

  yt = np.matmul(cf,J[0])
  e = y-yt
  it = 0

  #Descenso del gradiente
  while(np.linalg.norm(e)>= tolerancia) and (it < iteraciones):
      it+=1
      grad = []
      yt = np.matmul(cf,J)
      for i in range(n+1):
          grad.append(-2*np.sum((y-yt)*J[0,i,:]))
      grad = np.array(grad)
      cf = cf - grad*gamma
      e = y-yt
  #Devuelven los coeficientes de X^n a X0 ya que la funcion poly1D(como polyval en matlab) los necesita en ese orden
  return (cf[::-1],np.linalg.norm(e))
```

Problema 2

Solución El codigo se encuentra en el notebook, utilice una forma 3x2 por motivos de edicion

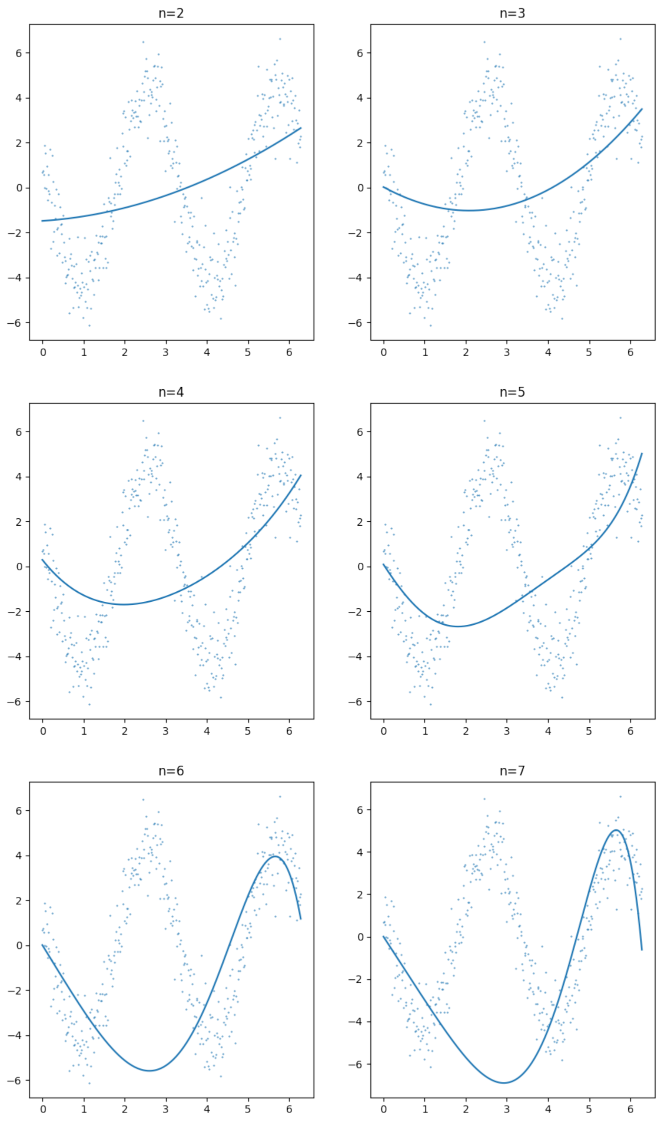

Figure 1: Grafica de las funciones segun nuestra aproximacion

Problema 3

Solución Antes de iniciar las preguntas, iniciemos visualizando los datos para entender la clasificacion de las clases

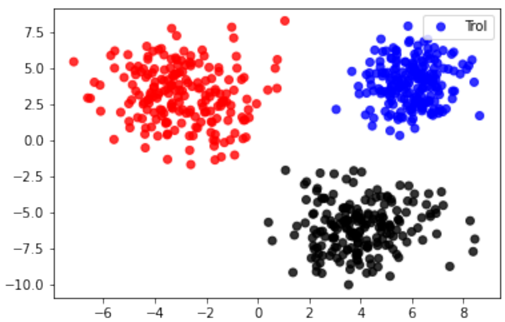

Figure 2: Tribus.csv

a) Simplemente tenemos que hacer las variables dummies para cada clase, para eso hacemos uso del siguiente codigo

```
tribus = pd.read_csv("data/Tribus.csv")
YT = np.zeros(600)
YT[tribus['Tribu']=='Trol']=1
YE = np.zeros(600)
YE[tribus['Tribu']=='Elfo']=1
YEn = np.zeros(600)
YEn[tribus['Tribu']=='Enano']=1
```

b) Para la creacion de los modelos vamos a hacer uso del siguiente perceptron

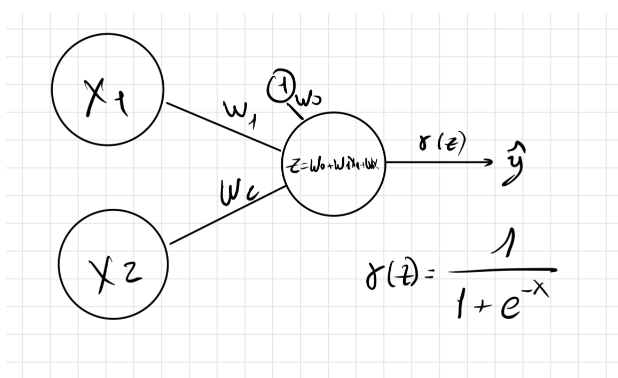

Figure 3: Perceptron del modelo

Entonces nuestro modelo puede ser descrito segun

$$\begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \end{bmatrix} = \begin{bmatrix} z_1 & z_2 & \dots & z_n \end{bmatrix}$$

Este es simplemente una generalizacion del modelo que tenemos que utilizar para cada clase. Entonces primera para elfos. En este proceso utilizamos nuestra funcion de activacion

$$f(x) = \frac{1}{1+e^{-x}}$$

Y el gradiente que previamente ya se calculo

$$< -2\sum_{i=0}^{k}(Y - \frac{1}{1+e^{-x}}) * (\frac{e^{-x}}{(1+e^{-x})^2}), -2\sum_{i=0}^{k} x_1 * (Y - \frac{1}{1+e^{-x}}) * (\frac{e^{-x}}{(1+e^{-x})^2}), -2\sum_{i=0}^{k} x_2 * (Y - \frac{1}{1+e^{-x}}) * (\frac{e^{-x}}{(1+e^{-x})^2}) >$$

Entonces podemos hacer uso del siguiente codigo

```
#Modelo para Elfos
w0 = np.random.rand()
w1 = np.random.rand()
w2 = np.random.rand()
W1 = np.array([w0,w1,w2])
gamma = 0.00001
nmax = 20000
ite =0
while (ite<nmax):
    ite += 1
    W1 = W1 -gamma *grad(W1, tensor ,YE)
print(W1)

Yp = f(np.matmul(W1, tensor[0,:,:]))
mfd(YE,Yp)
Yr =np.round(Yp)
Yr= np.ones(600)
Yr[Yp<0.5]=0
print(sum(Yr==YE)/600)
```

Con esto conseguimos los siguientes valores de los pesos, w0,w1,w2

$$[-1.15216466, 0.15040429, -1.12533629]$$

Y resulta en que 99.6% de los datos que se presentaron estan bien clasificados por nuestros pesos! y con esto podemos ver nuestro clasificador en el siguiente grafico

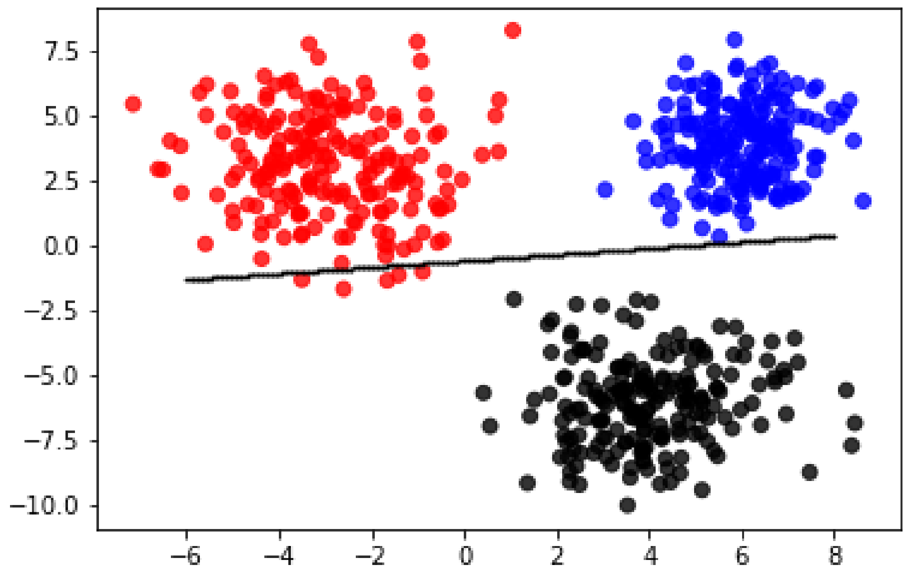

Figure 4: Clasificador lineal para los elfos

Con este mismo proceso podemos calcular para enanos y trols, vamos a presentar en una tabla los resultados

Class	w0	w1	w2
Elfos	-1.15216466	0.15040429	-1.12533629
Enanos	0.27264105	-1.11343464	0.41442398
Trols	-2.16473195	0.51744283	0.42070561

Y podemos ver en los siguientes graficos los valores clasificados

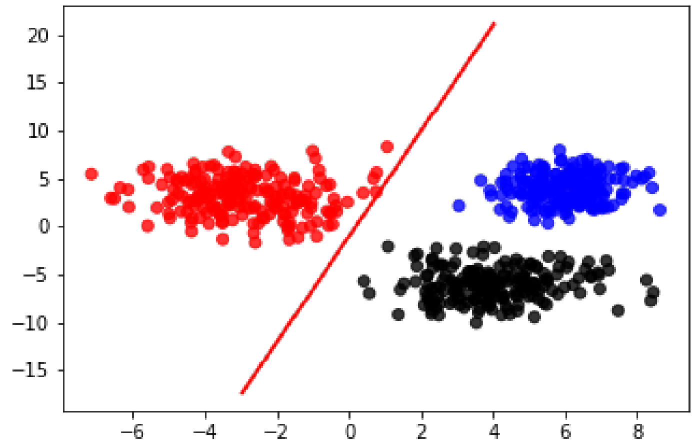

Figure 5: Clasificador lineal para los Enanos

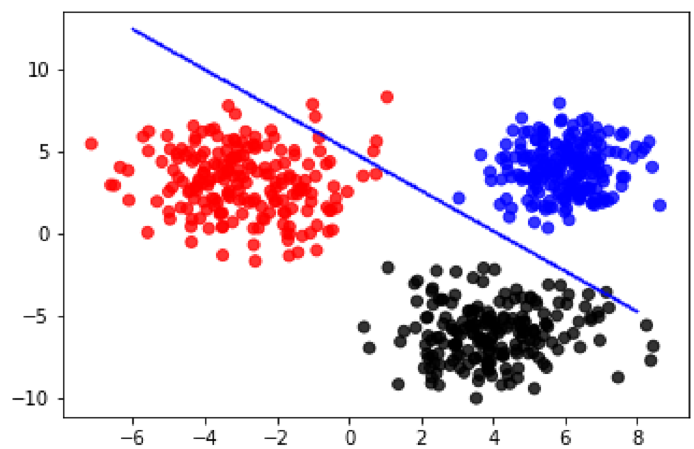

Figure 6: Clasificador lineal para los Trols

Y finalmente podemos ver todos los clasificadores juntos

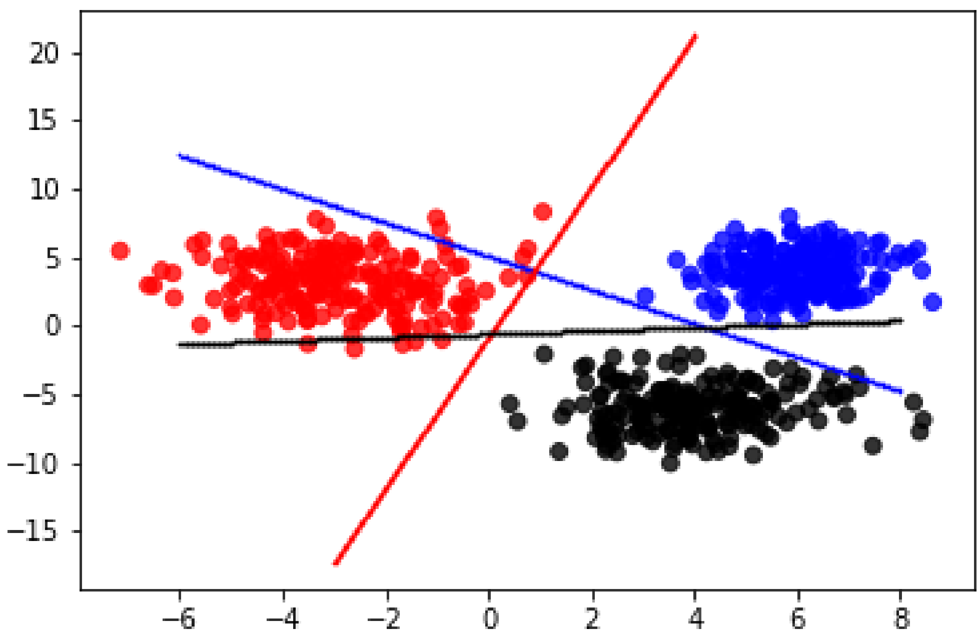

Figure 7: Clasificador lineal para los Trols

c) y d) Ahora como ya vimos que la clasificacion de esa forma es mas complicado podemos hacer uso del perceptron presentado en el archivo. Para eso tenemos que hacer uso de la funcion de perdida presentada y para eso debemos calcular su derivada para cada peso.

$$L = \frac{1}{k+1}\sum_{j=0}^{k} ||Y_j - \bar{Y})_j||^2$$

Sabiendo que

$$\bar{Y_j} = \frac{1}{1+e^{-wj0-wj1x_1-wj2x_2}}$$

Entonces

$$\frac{\delta L}{\delta wij} = \frac{\delta}{\delta wij}\frac{1}{k+1}\sum_{j=0}^{k} ||Y_j - \bar{Y})_j||^2$$

$$\frac{\delta L}{\delta wij} = \frac{1}{k+1}\sum_{j=0}^{k}\frac{\delta}{\delta wij} ||Y_j - \bar{Y})_j||^2$$

$$\frac{\delta L}{\delta wij} = \frac{1}{k+1}\sum_{j=0}^{k}\frac{\delta}{\delta wij}[(Y_{En} - \bar{Y_0})^2 + (Y_{El} - \bar{Y_1})^2 + (Y_T - \bar{Y_2})^2]$$

Sabemos que $Y_{En} - \bar{Y_0}$ es $Y_{En} - \frac{1}{1+e^{-w00-w01x_1-w02x_2}}$ entonces lo que nos damos cuenta es que para cualquier derivada que no sea para las varible w_{01}, w_{02}, w_{03} es 0, por lo que para cada variable hay dos terminos que se convierten en 0, entonces se vuelve la derivada similar al de la funcion de perdida de la primera pregunta, entonces tenemos el siguiente gradiente

$$\nabla L^T = \begin{bmatrix}
\frac{-2}{1+k}\sum_{j=0}^{k}(Y_{En} - \frac{1}{1+e^{-w_{00}-w_{01}x_1-w_{02}x_2}}) * (\frac{e^{-w_{00}-w_{01}x_1-w_{02}x_2}}{(1+e^{-w_{00}-w_{01}x_1-w_{02}x_2})^2}) \\
\frac{-2}{1+k}\sum_{j=0}^{k}x_1 * (Y_{En} - \frac{1}{1+e^{-w_{00}-w_{01}x_1-w_{02}x_2}}) * (\frac{e^{-w_{00}-w_{01}x_1-w_{02}x_2}}{(1+e^{-w_{00}-w_{01}x_1-w_{02}x_2})^2}) \\
\frac{-2}{1+k}\sum_{j=0}^{k}x_2 * (Y_{En} - \frac{1}{1+e^{-w_{00}-w_{01}x_1-w_{02}x_2}}) * (\frac{e^{-w_{00}-w_{01}x_1-w_{02}x_2}}{(1+e^{-w_{00}-w_{01}x_1-w_{02}x_2})^2}) \\
\frac{-2}{1+k}\sum_{j=0}^{k}(Y_T - \frac{1}{1+e^{-w_{10}-w_{11}x_1-w_{12}x_2}}) * (\frac{e^{-w_{10}-w_{11}x_1-w_{12}x_2}}{(1+e^{-w_{10}-w_{11}x_1-w_{12}x_2})^2}) \\
\frac{-2}{1+k}\sum_{j=0}^{k}x_1 * (Y_T - \frac{1}{1+e^{-w_{10}-w_{11}x_1-w_{12}x_2}}) * (\frac{e^{-w_{10}-w_{11}x_1-w_{12}x_2}}{(1+e^{-w_{10}-w_{11}x_1-w_{12}x_2})^2}) \\
\frac{-2}{1+k}\sum_{j=0}^{k}x_2 * (Y_T - \frac{1}{1+e^{-w_{10}-w_{11}x_1-w_{12}x_2}}) * (\frac{e^{-w_{10}-w_{11}x_1-w_{12}x_2}}{(1+e^{-w_{10}-w_{11}x_1-w_{12}x_2})^2}) \\
\frac{-2}{1+k}\sum_{j=0}^{k}(Y_{El} - \frac{1}{1+e^{-w_{20}-w_{21}x_1-w_{22}x_2}}) * (\frac{e^{-w_{20}-w_{21}x_1-w_{22}x_2}}{(1+e^{-w_{20}-w_{21}x_1-w_{22}x_2})^2}) \\
\frac{-2}{1+k}\sum_{j=0}^{k}x_1 * (Y_{El} - \frac{1}{1+e^{-w_{20}-w_{21}x_1-w_{22}x_2}}) * (\frac{e^{-w_{20}-w_{21}x_1-w_{22}x_2}}{(1+e^{-w_{20}-w_{21}x_1-w_{22}x_2})^2}) \\
\frac{-2}{1+k}\sum_{j=0}^{k}x_2 * (Y_{El} - \frac{1}{1+e^{-w_{20}-w_{21}x_1-w_{22}x_2}}) * (\frac{e^{-w_{20}-w_{21}x_1-w_{22}x_2}}{(1+e^{-w_{20}-w_{21}x_1-w_{22}x_2})^2})
\end{bmatrix}$$

Entonces solamente nos falta implementar el codigo

```
#lectura de datps
tribus = pd.read_csv("data/Tribus.csv")
x1 = np.array(tribus.Loc1, dtype=np.float128)
x2 = np.array(tribus.Loc2, dtype=np.float128)
coeficientes = np.array(np.random.rand(3,3),dtype = np.float128)
J = np.ones([1,3,len(x1)])
```

```
J[0,1,:] = x1
J[0,2,:] = x2

#dummyvariables
YT = np.zeros(600, dtype=np.float128)
YT[tribus['Tribu']=='Trol']=1
YE = np.zeros(600, dtype=np.float128)
YE[tribus['Tribu']=='Elfo']=1
YEn = np.zeros(600, dtype=np.float128)
YEn[tribus['Tribu']=='Enano']=1
Y = np.array([YE,YT,YEn])

#definicion del gradiente
def grad(W,J,Y):
    z = np.matmul(W,J[0,:,:])
    sp1 = (Y[0,:]-1/(1+np.exp(-z)))*np.exp(-z)/(1+np.exp(-z))**2
    sp2 = (Y[1,:]-1/(1+np.exp(-z)))*np.exp(-z)/(1+np.exp(-z))**2
    sp3 = (Y[2,:]-1/(1+np.exp(-z)))*np.exp(-z)/(1+np.exp(-z))**2
    w00 = -2*(1/len(x1))*np.sum(sp1*1)
    w01 = -2*(1/len(x1))*np.sum(sp1*J[0,1,:])
    w02 = -2*(1/len(x1))*np.sum(sp1*J[0,2,:])
    w10 = -2*(1/len(x1))*np.sum(sp2*1)
    w11 = -2*(1/len(x1))*np.sum(sp2*J[0,1,:])
    w12 = -2*(1/len(x1))*np.sum(sp2*J[0,2,:])
    w20 = -2*(1/len(x1))*np.sum(sp3*1)
    w21 = -2*(1/len(x1))*np.sum(sp3*J[0,1,:])
    w22 = -2*(1/len(x1))*np.sum(sp3*J[0,2,:])
    return np.array([[w00,w01,w02],[w10,w11,w12],[w20,w21,w22]])

#random weights
coeficientes = (np.array(np.random.rand(3,3),dtype = np.float128) -0.5)*2
it = 10000
gamma = 0.001

#descenso del gradiente
for i in range(it):
    coeficientes = coeficientes - gamma*grad(coeficientes,J,Y)


#clasificacion
Yp = sigma(np.matmul(coeficientes,J[0,:,:]),1)
suma = []
for i in range(len(Yp[0,:])):
    suma.append(np.sum(Yp[:,i]))

y0t = []
y1t = []
y2t = []
for i in range(len(Yp[0,:])):
    y0t.append(Yp[0,i]/suma[i])
    y1t.append(Yp[1,i]/suma[i])
    y2t.append(Yp[2,i]/suma[i])

Yp = [y0t,y1t,y2t]
```

```
Yr = np.round(Yp)
print(Yr)

#calculo del error
print(np.sum(Yr==Y)/1800)
```

Todo este proceso resulta en los siguientes pesos

$[[0.427516482.9529813-6.05260981][-0.272877351.16503370.99420651][0.57046952-5.227574782.87130223]]$

Que llevan a que un 98.77% de individuos estan bien clasificados por nuestro modelo!

d) Finalmente presentamos los datos clasificados por secciones, y podemos ver comoo cada punto del plano esta clasificado

Chévere!!

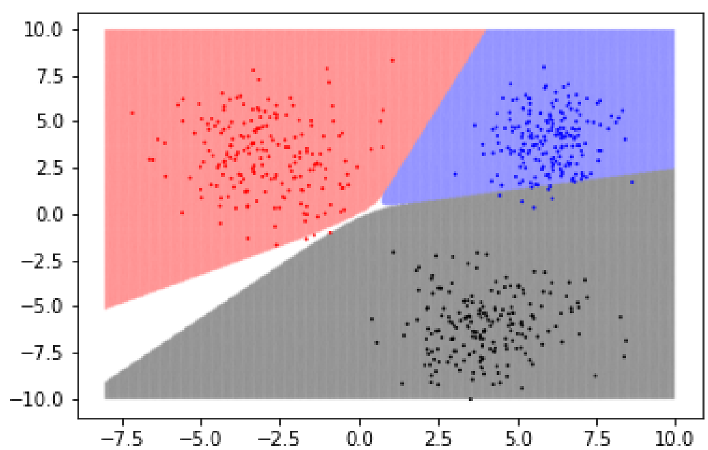

Figure 8: Clasificacion de datos

Los puntos dentro del plano que estan en blanco son puntos que pueden entrar a dos clasificadores o no entran a ninguno, gran parte de nuestro error se encuentra en los puntos que recaen en este segmento.

Problema 4

Solución Sea

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

sabemos que

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Entonces

$$tanh(x) = \frac{e^x - e^{-x} + 2e^{-2} - 2e^{-2}}{e^x + e^{-x}}$$

$$tanh(x) = \frac{e^x + e^{-x} - 2e^{-2}}{e^x + e^{-x}}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x - e^{-x}} + \frac{2e^{-x}}{e^x + e^{-x}}$$

$$tanh(x) = 1 - \frac{2e^{-x}}{e^x + e^{-x}}$$

$$tanh(x) = 1 - \frac{2e^{-x} * e^{-x}}{(e^x + e^{-x}) * e^{-x}}$$

$$tanh(x) = 1 - \frac{2}{e^{2x} + 1}$$

$$tanh(x) = 1 - \frac{2}{e^{2x} + 1}$$

$$tanh(x) = 1 - 2\sigma(-2x)$$

Como

$$1 - \sigma(x) = 1 - \frac{1}{1+e^{-x}}$$

$$1 - \sigma(x) = \frac{1 - 1 - e^{-x}}{1+e^{-x}}$$

$$1 - \sigma(x) = \frac{-e^{-x}}{1+e^{-x}}$$

$$1 - \sigma(x) = \frac{-e^{-x} * e^{-x}}{1+e^{-x} * e{-x}}$$

$$1 - \sigma(x) = \frac{-1}{\frac{1}{e^{-x}} + 1}$$

$$1 - \sigma(x) = \frac{-1}{1+e^{x}}$$

$$1 - \sigma(x) = \sigma(-x)$$

Por lo tanto

$$tanh(x) = 1 - 2(1 - \sigma(2x))$$

$$tanh(x) = 2\sigma(2x) - 1$$

■

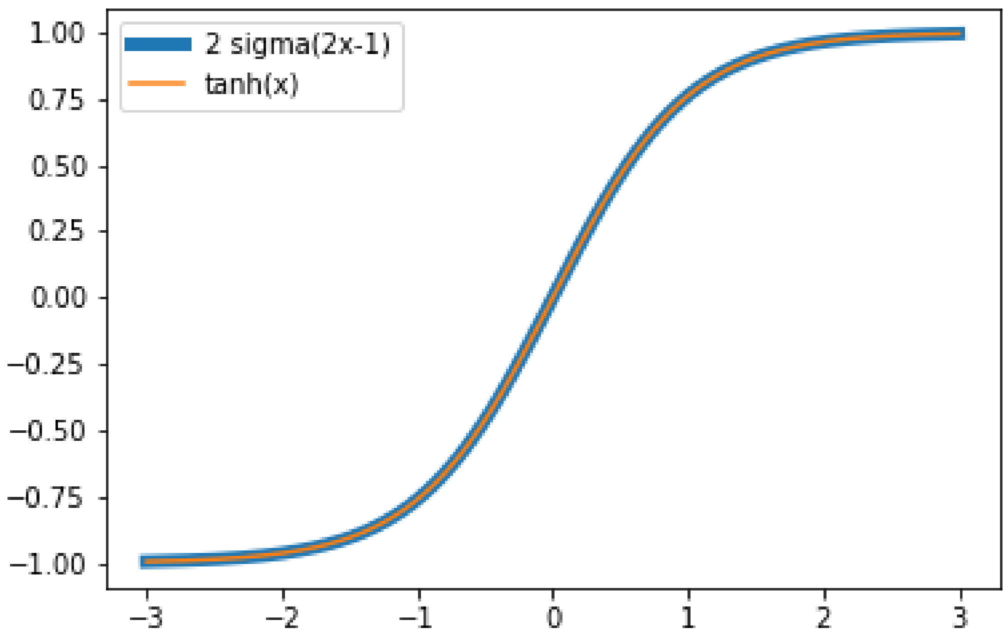

Figure 9: Grafico de tanh y función sigma

Problema 5

Solución Para resolver los literales primero analicemos el modelo y para eso primero visualicemos los datos, tenemos el conjunto de datos iris2 que estan distribuidos de esta forma

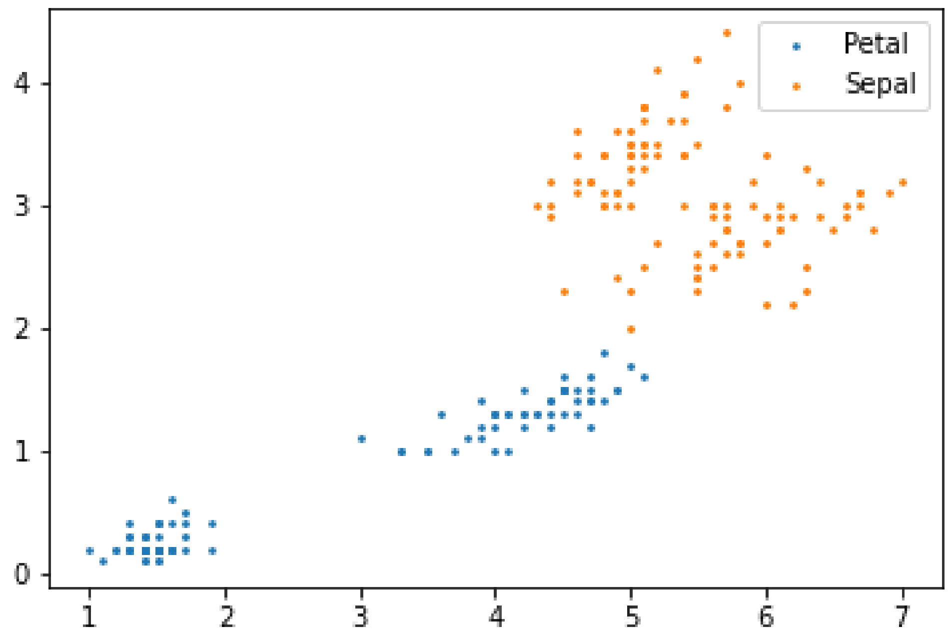

Figure 10: Grafico de dataFrame iris2

Sabemos que esta grafica no muestra lo que en realidad es el conjunto de datos porque petal y sepal no es que sean diferentes individuos, pero como no podemos graficar en 4 dimensiones esta representacion nos va a ayudar a entender un poco como funciona. Buscamos clasificar en dos posibles resultado **versicolor** y **setosa**, en el siguiente grafico tenemos en azul las que son versicolor y rojo las setosas

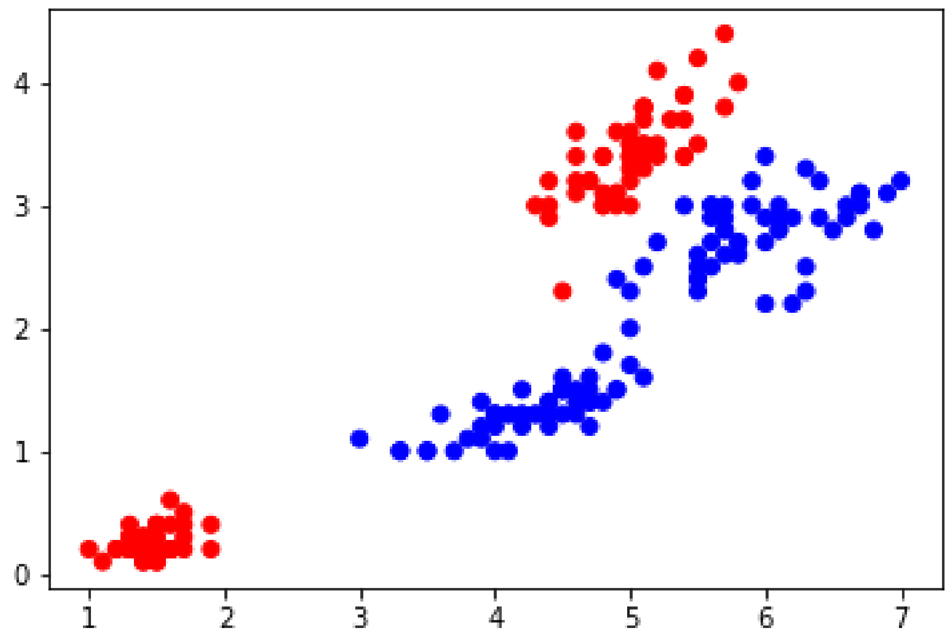

Figure 11: Grafico de dataFrame iris2, clasificados

Sabemos que vamos a trabajar con un modelo logístico con una función de activación tanh, (podriamos utilizar a la función $\sigma(x) = \frac{1}{1+e^{-x}}$), pero por el ejercicio 4 entendemos que no es necesario. Tenemos la función de perdida asociada dada por ejericio

$$L = -\sum_{i=1}^{m} log(|\frac{y_i}{2} - \frac{1}{2} - \widehat{y_i}|)$$

Como

$$\widehat{y_i} = \sigma(\sum_{j-1}^{n} w_i X_{ij})$$

$$tanh(z) = 2\sigma(2z) - 1$$

Para simplificar el codigo se utiliza la siguiente notacion

$$\sum_{j-1}^{n} w_i X_{ij} = x$$

Tenemos que

$$L = -\sum_{i=1}^{m} log(|\frac{y_i}{2} - \frac{1}{2} + \widehat{y_i}|)$$

como

$$\sigma(2x) = \frac{tanh(2x) - 1}{2}$$

Entonces

$$L = -\sum_{i=1}^{m} log(|\frac{y_i}{2} - \frac{1}{2} + (\frac{tanh(x/2) - 1}{2a}|))$$

$$L = -\sum_{i=1}^{m} log(|\frac{y_i}{2} + \frac{tanh(x/2)}{2}|)$$

Con lo que conseguimos una función de perdida equivalente a la planteada pero tiene el beneficio que es mucho mas facil de analizar y de conseguir sus derivadas

Entonces para el literal a vamos a calcular $\frac{\delta}{\delta w_i}$ para eso

$$\frac{\delta L}{\delta w_i} = \frac{\delta}{\delta w_i}(-\sum_{i=1}^{m} log(|\frac{y_i}{2} + \frac{tanh(x/2)}{2}|))$$

$$\frac{\delta L}{\delta w_i} = -\sum_{i=1}^{m} \frac{\delta}{\delta w_i}(log(|\frac{y_i}{2} + \frac{tanh(x/2)}{2}|))$$

$$\frac{\delta L}{\delta w_i} = -\sum_{i=1}^{m} \frac{1}{\frac{y_i}{2} + \frac{tanh(x/2)}{2}} * \frac{\delta L}{\delta w_i}((|\frac{y_i}{2} + \frac{tanh(x/2)}{2}|))$$

$$\frac{\delta L}{\delta w_i} = -\sum_{i=1}^{m} \frac{1}{|\frac{y_i}{2} - \frac{tanh(x/2)}{2}|} * \frac{\frac{\delta}{\delta w_i}(x/2) * sech^2(\frac{x}{2}) * (\frac{y_i}{2} + \frac{tanh(\frac{x}{2}}{2}))}{2 * |\frac{y_i}{2} + \frac{tanh(x/2)}{2}|}$$

De forma que

$$\frac{\delta L}{\delta w_i} = -\sum_{i=1}^{m} \frac{\frac{\delta}{\delta w_i}(x/2) * sech^2(x/2)}{2 * (\frac{y_i}{2} + \frac{tanh(x/2)}{2})}$$

Ahora, lo unico que tenemos que ver para terminar nuestra derivada es darnos cuenta que la derivada de x ($\sum_{j-1}^{n} w_i X_{ij}$) es una combinacion lineal de las variables w0,w1,...,wn entonces la derivada segun w_i sera igual a x_i respectivamente. por lo que

$$\frac{\delta L}{\delta w_i} = -\sum_{i=1}^{m} \frac{x_i * sech^2(x/2)}{(y_i + tanh(x/2))}$$

Entonces para responder la pregunta b solamente tenemos que presentar nuestros resultado en forma de que utilicemos el metodo del descenso del gradiente.

La ecuacion iterativa es

$$\begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_2 \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_n \end{bmatrix} - gamma * \begin{bmatrix} -\sum_{i=1}^{m} \frac{sech^2(x/2)}{(y_i + tanh(x/2))} \\ -\sum_{i=1}^{m} \frac{x_1 * sech^2(x/2)}{(y_i + tanh(x/2))} \\ \dots \\ -\sum_{i=1}^{m} \frac{x_n * sech^2(x/2)}{(y_i + tanh(x/2))} \end{bmatrix}$$

En el primer gradiente podemos ver que no hay x_i ya que siempre $x_0 = 1$

Teniendo esto es momento de iniciar nuestro modelo. Para eso primero analizamos un poco el perceptrón del cual queremos crear el modelo

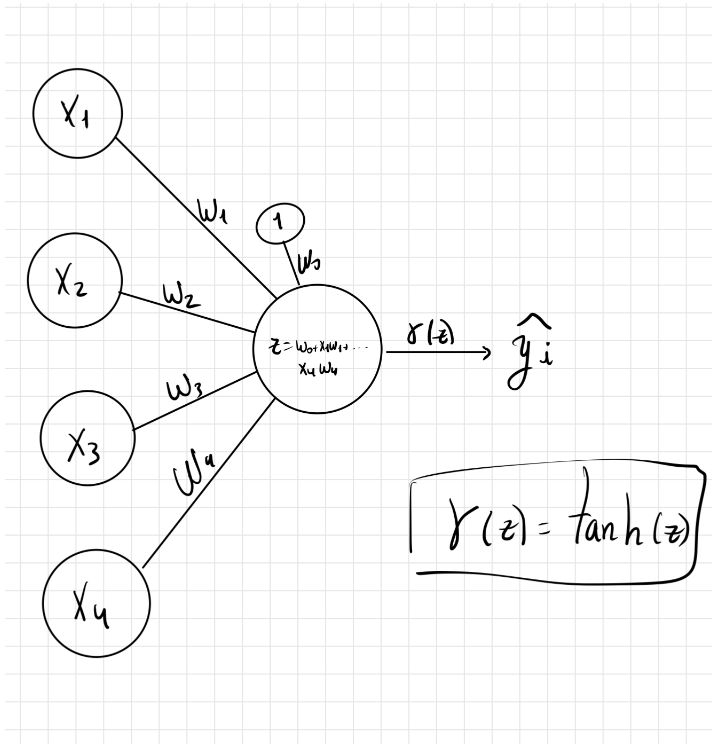

Figure 12: Perceptrón del modelo

Entonces podemos hacer un tensor de estimulos para entender como va a funcionar el modelo

$$\begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix} \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ x_{31} & x_{32} & \dots & x_{3n} \\ x_{41} & x_{42} & \dots & x_{4n} \\ x_{51} & x_{52} & \dots & x_{5n} \end{bmatrix} = \begin{bmatrix} z_1 & z_2 & \dots & z_n \end{bmatrix}$$

Finalmente tenemos que aplicar lo que ya aprendimos de otros ejercicios, que es el proceso de disminucion del gradiente con la función de perdida y el gradiente que recientemente encontramos, entonces lo aplicamos en python

```
#previamente ya esta inicializada el data frame de iris2

#valores para el tensor
x1 = np.array(df["PetalLength"])
x2 = np.array(df["PetalWidth"])
x3 = np.array(df["SepalLength"])
x4 = np.array(df["SepalWidth"])
y = np.array(df["Species"])

#Instanciacion del tensor
J= np.ones([1,5,len(x1)])

#Construccion del tensor
J[0,1,:] = x1
J[0,2,:] = x2
J[0,3,:] = x3
```

```python
J[0,4,:] = x4

#definicion de la variable dummy tal que versicolor = -1 y setosa = 1
Y = np.ones(len(x1))
print(J[0,0,:])
Y[y=='versicolor'] =-1

#coeficientes
W = np.random.rand(5)

#funcion gradiente como la que conseguimos
# en vez de sech(x) se utiliza 1/cosh(x)
def grad(W,J,y):
    zt = np.matmul(W,J)
    w0 = - np.sum(((1/(np.cosh(zt/2))**2)*J[0,0,:])/(y + np.tanh(zt/2)))
    w1 = - np.sum(((1/(np.cosh(zt/2))**2)*J[0,1,:])/(y + np.tanh(zt/2)))
    w2 = - np.sum(((1/(np.cosh(zt/2))**2)*J[0,2,:])/(y + np.tanh(zt/2)))
    w3 = - np.sum(((1/(np.cosh(zt/2))**2)*J[0,3,:])/(y + np.tanh(zt/2)))
    w4 = - np.sum(((1/(np.cosh(zt/2))**2)*J[0,4,:])/(y + np.tanh(zt/2)))
    return  np.array([w0,w1,w2,w3,w4])


#espectros
gamma = 0.0000001
it = 50000

#proceso de descenso del gradiente
for i in range(it):
    W = W - gamma*grad(W,J,Y)

#calculo del error
Yp = np.matmul(W,J[0,:,:])
Yr = np.tanh(Yp)

for i in range(len(Yr)):
    if(Yr[i]>=0):
        Yr[i] = 1
    else:
        Yr[i] = -1

#imprimir el error
np.sum(Y==Yr)/100
```

Con esto con seguimos los siguientes valores para los pesos, w0,w1,w2,w3,w4 respectivamente

$$0.34627295, -0.29130158, 0.30365354, -0.18628593, 0.34360007$$

Y lo interesante y muy bonito (me disculpo el uso de estos terminos pero me emocione) es que el error fue de 0. Por lo que un 100% de los puntos estan clasificados de manera correcta
Entonces logramos conseguimos crear un modelo logístico que clasifica cada punto en su tipo correcto.

Problema 6

Solución Antes de iniciar vamos a analizar los datos

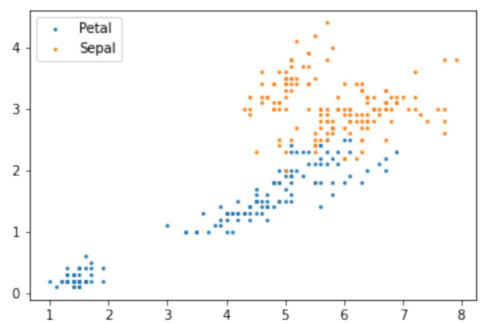

Figure 13: Datos del dataFrame iris

Ahora vamos analizar separado por la clase. Versicolor es azul, virginica es verde y setosa es rojo

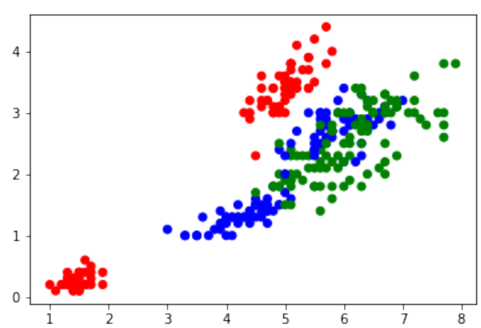

Figure 14: Datos del dataFrame iris

Ahora aumentó la complejidad, entonces vamos a hacer uso de algunas funciones pasadas, no es necesario volver a realizarlas. Por ejemplo la función de activación $f(z) = tanh(z)$ se sigue utilizando, por lo que vamos a volver a hacer uso del gradiente. Aqui tenemos que hacer tres modelos logísticos para cada clasificador y el valor mas grande de f(z) es la clase del individuo. Entonces comencemos analizando el perceptrón del modelo.
En realidad algo que va a pasar es que este va a ser el mismo que la anterior pregunta ya que estamos haciendo 3 veces el mismo modelo. Entonces lo que vamos a presentar es un pseudoperceptrón que analiza el modelo que vamos a crear

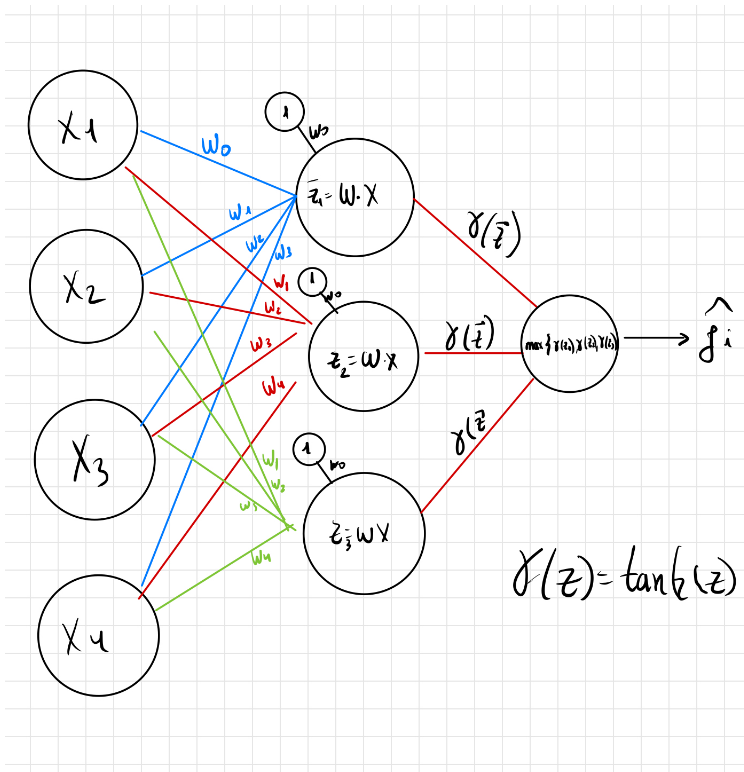

Figure 15: Perceptrón para modelo

Si nos damos cuenta los pesos entre cada valor de z1, tienen el mismo nombre de la variable pero eso no significa que son iguales, ya que son diferentes modelos
Entonces comencemos con el codigo

```
#base del tensor de estimulos, el objeto df es el datafram de iris
x1 = np.array(df["Petal.Length"])
x2 = np.array(df["Petal.Width"])
x3 = np.array(df["Sepal.Length"])
x4 = np.array(df["Sepal.Width"])
y = np.array(df["Species"])

#inicializacion del tensor
J= np.ones([1,5,len(x1)])

#creo el tensor
J[0,1,:] = x1
J[0,2,:] = x2
J[0,3,:] = x3
J[0,4,:] = x4

#coeficientes random
Ws = np.random.rand(5)
Wver = np.random.rand(5)
Wvir = np.random.rand(5)

#escalas
gamma = 0.000001
```

```
it = 40000


#proceso para setosa
for i in range(it):
    Ws = Ws - gamma*grad(Ws,J,ys)
#proceso para virginia
for i in range(it):
    Wvir = Wvir - gamma*grad(Wvir,J,yvir)
#proceso para versicolor
for i in range(it):
    Wver = Wver - gamma*grad(Wver,J,yver)
```

Este codigo es exactamente el mismo que hemos realizado en la actividad 5, simplemente que lo realizamos 3 veces para conseguir tres vectores de pesos distintos
¿Entonces comó clasificar y ver el error?, para eso hacemos uso de lo que nos da el problema, donde vamos a clasificar tal que la clase de un individup es igual al $max\{tanh(y_i)\}$ donde $y_i = W_i * X$ donde W_i es igual a los pesos segun cada clase y X es nuestro tensor de estimulos. Entonces lo que estamos haciendo es clasificar cada individuo para cada clase, y tomando la que tiene una mayor seguridad de pertenecer a esa clase.Utilizamos el siguiente codigo

```
#calculo de los valores de y_i
ysA = np.tanh(np.matmul(Ws,J[0,:,:]))
yvirA = np.tanh(np.matmul(Wvir,J[0,:,:]))
yverA = np.tanh(np.matmul(Wver,J[0,:,:]))

#asignamos cada conjunto dee individuos un clasificador segun cual es mayor
ans = []
for i in range(len(x1)):
    maxi = (ysA[i],"setosa")
    if(maxi[0]<yvirA[i]):
        maxi = (yvirA[i],"virginica")
    if(maxi[0]<yverA[i]):
        maxi = (yverA[i],"versicolor")
    ans.append(maxi[1])

#imprimimos pesos y error
print(Ws)
print(Wvir)
print(Wver)
print(np.sum(y==ans)/150)
```

Esto resulta en los siguientes valores para los pesos segun cada modelo!
Primero para el modelo de las setosas

$$0.26021362, -1.08148421, 0.03459942, 0.04584251, 0.78645334]$$

para las virgnicas

$$[-0.35232071, 0.8390362, 0.87373084, -0.74071012, -0.18745811]$$

para las versicolor

$$[0.53949486, 0.22594454, -0.03373467, -0.17321971, -0.24297752]$$

Y finalmente tenemos un porcentaje de individuos que estan bien clasificados de 0.946666 que es un porcentaje muy alto!!

Problema 7

Solución Antes de iniciar visualicemos los datos para entenderlos

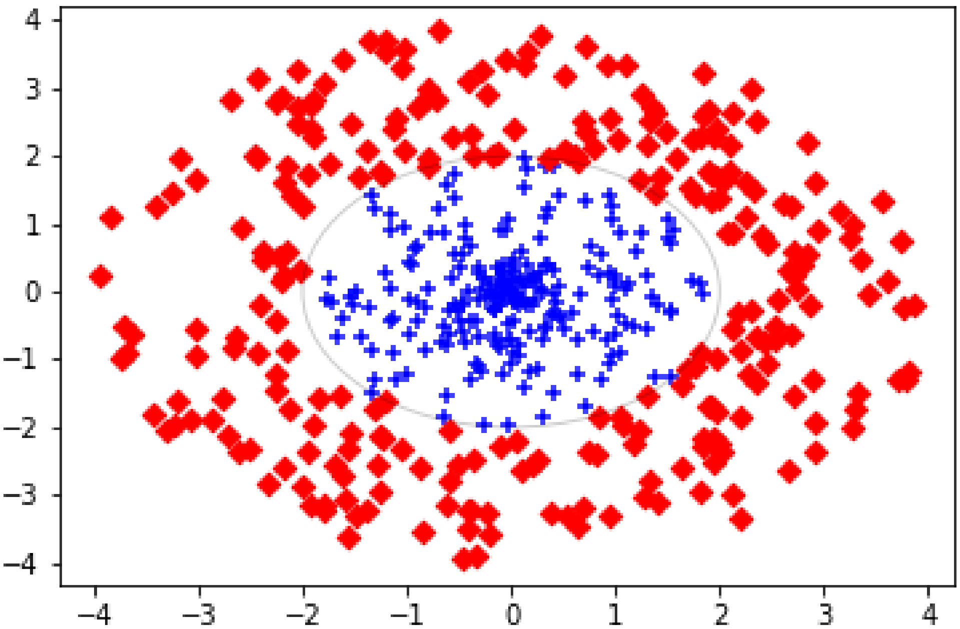

Figure 16: Distribucion de puntos

Este problema es sencillo pero sinceramente si no fuera por el ejercicio 8 fuera muy dificil de entender su utilidad. Primeramente nosotros al momento de analizar y de utilizar un perceptron con los valores de x,y teoricamente, manejamos un sistema de resultados lineales, es muy dificil que tengamos un porcentaje alto de individuos correctamente clasificados con datos que estan clasificados según una función no lineal, entonces en este ejercicio vamos a realizar el modelo sin ninguna alteración para analizar el error, y en el próximo analizamos considerando que no es lineal. Para este modelo entonces tenemos el siguiente perceptrón

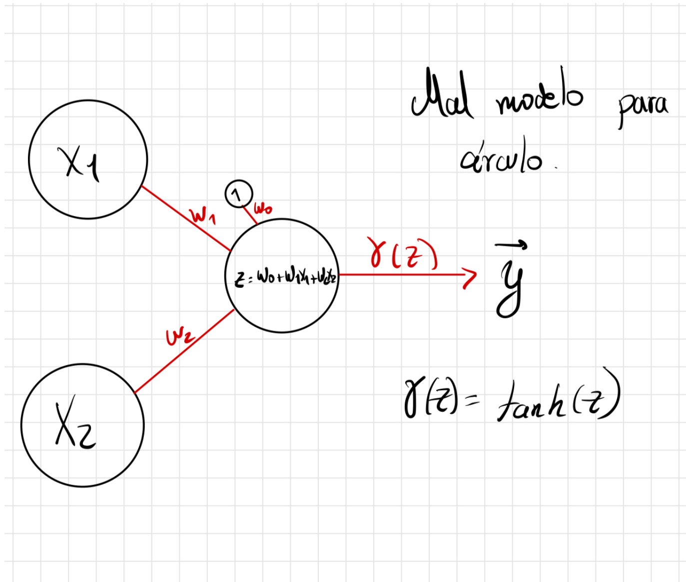

Figure 17: Perceptron de modelo logistico

teniendo estos valores sabemos que nuestro vector de resultantes viene dado por

$$\begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \end{bmatrix} = \begin{bmatrix} z_1 & z_2 & \dots & z_n \end{bmatrix}$$

Como funcion de activacion tanto para el problema 7 y 8 se hace uso de la funcion $f(z) = tanh(z)$ con su respectivo gradiente previamente calculado entonces, hagamos el código para este proceso

```
#modelo
x1 = np.array(dataFrame.x)
x2 = np.array(dataFrame.y)
y = dataFrame["state"]

#tensor
J=np.ones([1,3,len(x1)])
J[0,1,:] = x1
J[0,2,:] = x2

#circle out = -1
#circle in = 1
Y = -np.ones(len(x1))
Y[y=='circleIn']=1

#coeficientes
W = np.random.rand(3)

#definicion del gradiente
def grad(W,J,y):
    zt = np.matmul(W,J)
    w0 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,0,:])/(y - np.tanh(-zt)))
    w1 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,1,:])/(y - np.tanh(-zt)))
    w2 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,2,:])/(y - np.tanh(-zt)))
    return np.array([w0,w1,w2])
```

```

#escalas
it = 10000
gamma = 0.000
#calculo

#Descenso del gradiente
for i in range(it):
    W = W - gamma*grad(W,J,Y)

#Analisis de error
Yp = np.tanh(np.matmul(W,J[0,:,:]))
Yr = Yp
for i in range(len(Yr)):
    if(Yr[i]>=0):
        Yr[i] = 1
    else:
        Yr[i] = -1
print(W)
print(np.sum(Y==Yr)/500)

#Estamos trabajando una formula lineal entonces no va a funcionar, en el
    siguiente problema tendremos un mejor resultado
```

Tenemos entonces que el vector de pesos es igual a

$$[0.99769098, 0.67892594, 0.56848014]$$

y el porcetaje de puntos correctamente clasificados es de 0.588, un porcentaje muy bajo para un algoritmo clasificador.

Analicemos con otros puntos que creamos nosotros dentro del rectangulo [-5,5]x[-5,5] los puntos aleatorios que conseguimos estan presentados en el gráfico

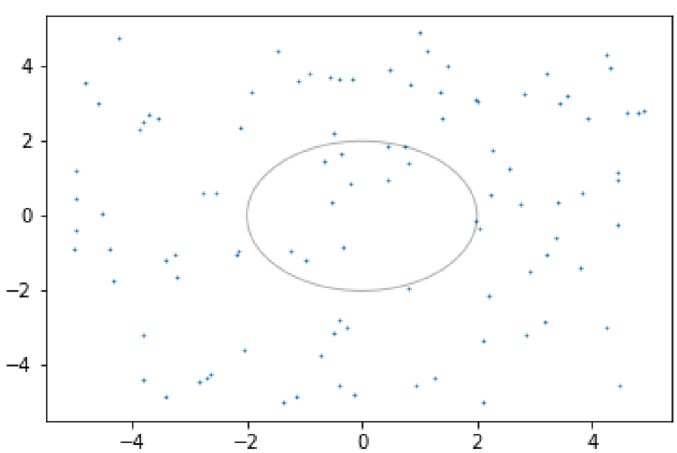

Figure 18: Puntos aleatorios dentro de [-5,5]x[-5,5]

Realizamos el mismo proceso con el siguiente codigo

```
#matriz aleatoria
ran = np.random.uniform(low=-5, high=5, size=(2,100))

#matrix de referencia de los valores
y = np.array(ran[0,:])**2 + np.array(ran[1,:])**2
#clasificamos
y[y<=4] = 1
y[y>4] = -1

#utilizando los pesos que encontramos anterior mente ver nuestro error

J = np.ones([1,3,100])
J[0,1,:] = ran[0,:]
J[0,2,:] = ran[1,:]

Yp = np.tanh(np.matmul(W,J[0,:,:]))
Yr = Yp
for i in range(len(Yr)):
    if(Yr[i]>=0):
        Yr[i] = 1
    else:
        Yr[i] = -1
print(W)
print(np.sum(y==Yr)/100)
#Es un porcentaje muy bajo pero nosotros sabemos que en la pregunta 8 todo va
    a estar mejor
```

Entonces finalmente esto nos da un porcentaje de puntos correctamente clasificados de 0.41, muy bajo, entonces vamos a analizar en el siguiente ejercicio como hacerlo mejor
Presentamos el clasificador que nos damos cuenta que no hace un buen trabajo clasificando los puntos

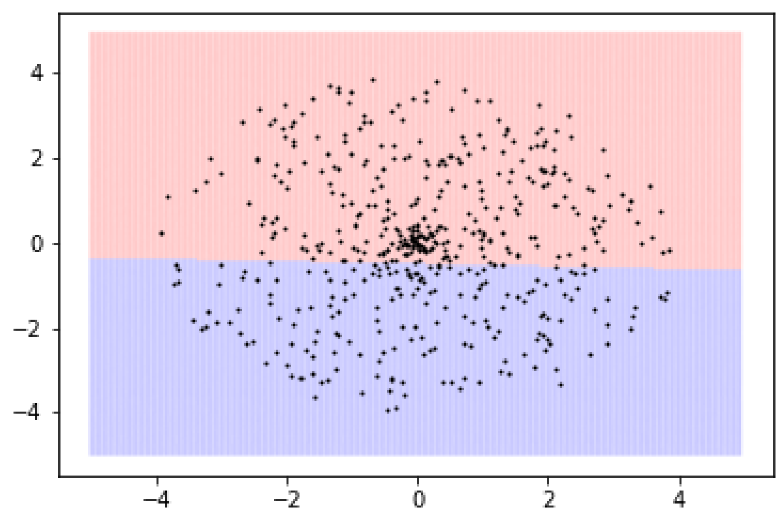

Figure 19: Clasificador

Siempre la clasificacion se va a dar de esta forma, va a dividir al plano, con una recta que pasa muy cerca del (0,0), por la naturaleza de nuestro clasificador lineal.

Problema 8

Solución Analicemos para este ejercico primeramente el perceptron que se utiliza

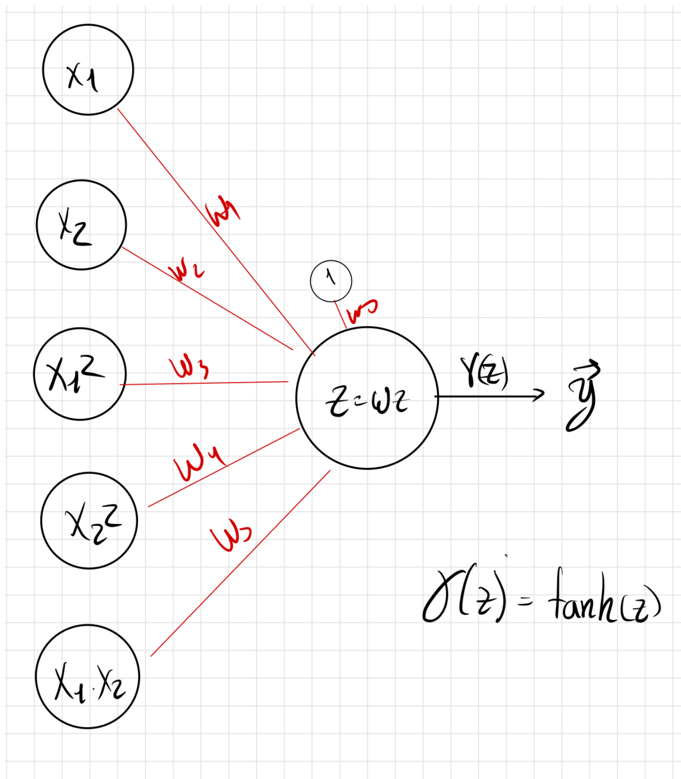

Figure 20: Perceptron para modelo pregunta 8

Podemos ver que se aumento los estimulos, entonces vamos a tener un nuevo modelo, lo analizamos de la siguiente forma!

$$\begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ x_{11}^2 & x_{12}^2 & \dots & x_{1n}^2 \\ x_{21}^2 & x_{22}^2 & \dots & x_{2n}^2 \\ x_{11} * x_{21} & x_{12} * x_{22} & \dots & x_{1n} * x_{2n} \end{bmatrix} = \begin{bmatrix} z_1 & z_2 & \dots & z_n \end{bmatrix}$$

Podemos ver que tanto el tensor de estimulos cambio coomo los pesos ya que necesitamos para cada estimulo un peso. Entonces, comencemos con el codigo.
Como ya se dijo seguimos utilizando la funcion de activacion de tanh(x) con su respectivo gradiente

```
  #modelo
x1 = np.array(dataFrame.x)
x2 = np.array(dataFrame.y)
y = dataFrame["state"]

#tensor
J=np.ones([1,6,len(x1)])
```

```
J[0,1,:] = x1
J[0,2,:] = x2
J[0,3,:] = x1**2
J[0,4,:] = x2**2
J[0,5,:] = x1*x2

#circle out = -1
#circle in = 1
Y = -np.ones(len(x1))
Y[y=='circleIn']=1

#coeficientes
W = np.random.rand(6)

#definicion del gradiente
def grad(W,J,y):
    zt = np.matmul(W,J)
    w0 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,0,:])/(y - np.tanh(-zt)))
    w1 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,1,:])/(y - np.tanh(-zt)))
    w2 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,2,:])/(y - np.tanh(-zt)))
    w3 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,3,:])/(y - np.tanh(-zt)))
    w4 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,4,:])/(y - np.tanh(-zt)))
    w5 = - np.sum(((1/(np.cosh(-zt))**2)*J[0,5,:])/(y - np.tanh(-zt)))
    return np.array([w0,w1,w2,w3,w4,w5])

#escalas
it = 10000
gamma = 0.0001
#calculo

for i in range(it):
    W = W - gamma*grad(W,J,Y)

Yp = np.tanh(np.matmul(W,J[0,:,:]))
Yr = Yp
for i in range(len(Yr)):
    if(Yr[i]>=0):
        Yr[i] = 1
    else:
        Yr[i] = -1
print(W)
print(np.sum(Y==Yr)/500)
```

Teniendo como resultado, un porcentaje de puntos bien clasificados de 0.99, que es un porcentaje muy alto entonces nuestro modelo mejoró

Ahora haciendo nuestros propios puntos analicemos la utilidad de nuestros pesos

```
y = np.array(ran[0,:])**2 + np.array(ran[1,:])**2
y[y<=4] = 1
y[y>4] = -1

J = np.ones([1,6,100])
J[0,1,:] = ran[0,:]
J[0,2,:] = ran[1,:]
J[0,3,:] = np.array(ran[0,:])**2
```

```
J[0,4,:] = np.array(ran[1,:])**2
J[0,5,:] = np.array(ran[1,:])*np.array(ran[0,:])


Yp = np.tanh(np.matmul(W,J[0,:,:]))
Yr = Yp
for i in range(len(Yr)):
    if(Yr[i]>=0):
        Yr[i] = 1
    else:
        Yr[i] = -1
print(W)
print(np.sum(y==Yr)/100)
```

Resultando en los pesos [6.24421406 -0.04341894 -0.08663621 -1.63493172 -1.57236489 0.09970738]
Con esto conseguimos un porcentaje de puntos bien clasificados de aproximadamente 99% un modelo mucho mejor que el anterior!
Podemos ver en el grafico las secciones clasificadas segun neustro clasificador

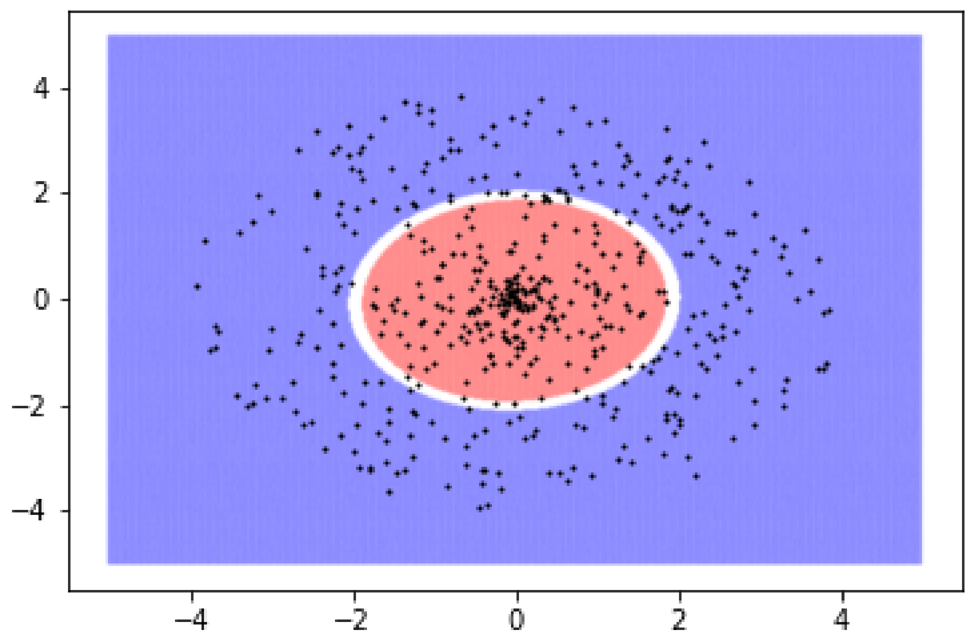

Figure 21: Clasificador