logistic_regression_by_hand

October 28, 2018

0.1 Formula

Para optimizar los parametros de la regresion, debemos hacer:

$y = \frac{1}{(1+e^{-(b1x_1+b2x_2+c)})}$
Cada dato : $p(y = "c"|experimentos) = p(y_1 = "C") * p(y_2 = "C")$
$p(y = 1|d) = 1/n \prod_{i=1}^{n}(1 - p(y))^{1-y}.(p(y))^y$
$l = \prod_{i=1}^{n}(1 - \frac{1}{(1+e^{-(b1x+b2x_2+c)})})^{(1-y)} * (\frac{1}{(1+e^{-(b1x+b2x_2+c)})})^y$
$\log(l) = \sum_{i=1}^{n} +(1 - y)\log(1 - \frac{1}{(1+e^{-(b_1x+b_2x_2+c)})}) + y\log(\frac{1}{(1+e^{-(b_1x+b_2x_2+c)})})$
$\frac{dl}{db_1} = \frac{dl}{du}\frac{du}{db_1}$
$\frac{dl}{db_1} = y\log([1 + e^{-(b_1x+b_2x_2+c)}]^{-1}) + (1 - y)\log(\frac{e^{-(b_1x+b_2x_2+c)}}{1+e^{-(b_1x+b_2x_2+c)}})$
$\frac{dl}{db_1} = -y\log(1 + e^{-(b_1x+b_2x_2+c)}) + (1 - y)[\log(e^{-(b_1x+b_2x_2+c)}) - \log(1 + e^{-(b_1x+b_2x_2+c)})]$
$\frac{dl}{db_1} = \log(e^{-(b_1x+b_2x_2+c)}) - \log(1 + e^{-(b_1x+b_2x_2+c)}) - y\log(e^{-(b_1x+b_2x_2+c)})$
$\frac{dl}{db_1} = \frac{-(b_1x_1+b_2x_2+c)}{db_1} - \frac{\log(1+e^{-(b_1x_1+b_2x_2+c)})}{db_1} - \frac{y(-(b_1x_1+b_2x_2+c))}{db_1}$
$= -x_1 - (1 + e^{-(b_1x_1+b_2x_2+c)})^{-1}(e^{-(b_1x_1+b_2x_2+c)})(-x_1)) + yx_1$
$= -x_1 - (-x_1)(\frac{1}{1+e^{(b_1x_1+b_2x_2+c)}}) + yx_1$
$= -x_1 - (-x_1)(\frac{1}{1+e^{(b_1x_1+b_2x_2+c)}}) + yx_1$
$= \frac{-x_1+x_1+-x_1e^{(b_1x_1+b_2x_2+c)}}{1+e^{(b_1x_1+b_2x_2+c)}}) + yx_1$
$= x_1(y - \frac{e^{(b_1x_1+b_2x_2+c)}}{1+e^{(b_1x_1+b_2x_2+c)}})$
$\frac{dl}{db_1} = x_1(y - \frac{1}{1+e^{-(b_1x_1+b_2x_2+c)}})$

0.2 Calcular C.

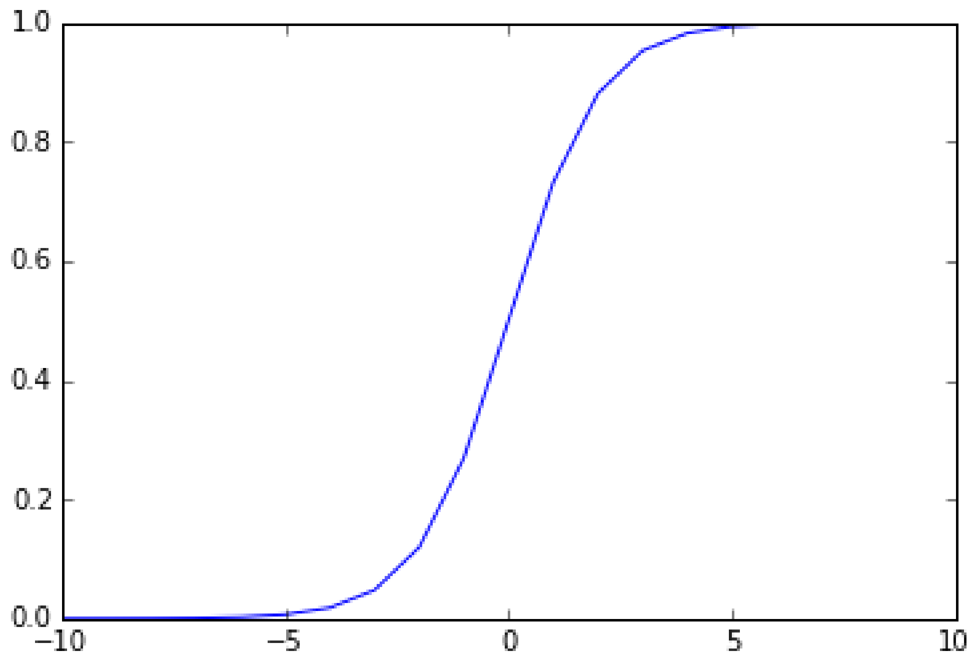
$\frac{dl}{dc} = \log(e^{-(b_1x_1+b_2x_2+c)}) - \log(1 + e^{-(b_1x_1+b_2x_2+c)}) - y\log(e^{-(b_1x+b_2x_2+c)})$
$\frac{dl}{dc} = \frac{-(b_1x_1+b_2x_2+c)}{dc} - \frac{\log(1+e^{-(b_1x_1+b_2x_2+c)})}{dc} - \frac{y(-(b_1x_1+b_2x_2+c))}{dc}$
$\frac{dl}{dc} = -1 - (-1)(\frac{1}{1+e^{(b_1x_1+b_2x_2+c)}})$ +y
$\frac{dl}{dc} = y - \frac{e^{(b_1x_1+b_2x_2+c)}}{1+e^{(b_1x_1+b_2x_2+c)}})$
$\frac{dl}{dc} = y - \frac{1}{1+e^{-(b_1x_1+b_2x_2+c)}}$

In [1]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math
import random
import pandas as pd
import numpy as np
%matplotlib inline
```

0.3 Funcion Sigmoid

Esta funcion es en forma de S, el rango va de 0 a 1. Y el objetivo es encontrar la funcion que mejor precide los valores de y

In [2]:
```
f = lambda x : 1/(1+math.exp(-x))
plt.plot(range(-10,10),[f(d) for d in range(-10,10)])
plt.show()
```



0.4 Caso 1

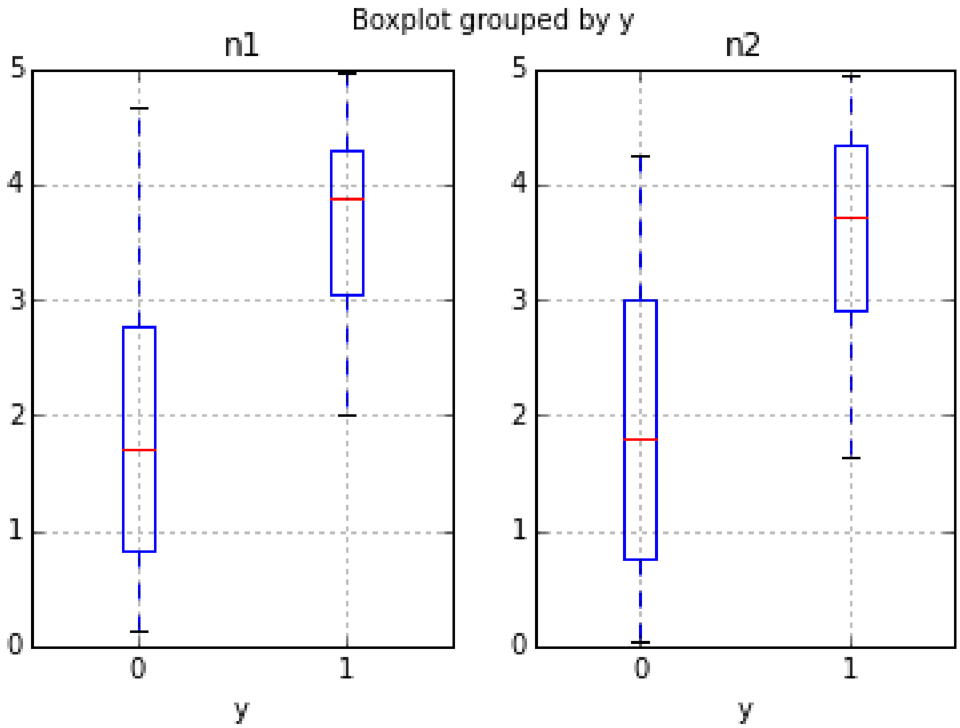
En este ejemlo vamos a generar los valores de Y. De esta forma esperamos que la regresion encuentre los coeficientes. En este caso vamos a predicr si un estudiante pasara o no el proximo examen, basado en las 2 ultimas notas.

In [10]:
```
n_est = 100
x_1 = [random.random()*5.0 for i in range(n_est)]
x_2 = [random.random()*5.0 for i in range(n_est)]
y = [1 if (x1_i+ x2_i)/2>3.0 else 0 for x1_i,x2_i in zip(x_1,x_2)]
df = pd.DataFrame({"n1":x_1,"n2":x_2,"y":y})
df.head(10)
```

Out[10]:

	n1	n2	y
0	1.958658	2.712304	0
1	4.283459	1.815374	1
2	1.999215	1.369591	0
3	4.577981	0.853161	0
4	2.615377	0.079121	0
5	2.479763	3.845374	1
6	0.131860	0.491915	0
7	3.190390	1.831196	0
8	1.637544	4.090928	0
9	0.355322	2.701056	0

In [11]:
```
fig, ax = plt.subplots(nrows=1, ncols=2)
df.boxplot(column='n1',by='y',ax=ax[0])
df.boxplot(column='n2',by='y',ax=ax[1])
plt.show()
```



In [5]:
```
class LogisticRegression():

    def __init__(self, lr=0.1, is_norm=False):
        self.lr = 0.1
        self.b = [0.0, 0.0]
        self.c = 0.0
        self.is_norm = is_norm

    def train(self, x , y):
        # Copiamos las variables y hacemos la actualziacion despues de calcular
        b = self.b[:]
        c = self.c
        if np.array(x).ndim < 2 :
            b[0] = b[0] + self.lr * x[0]*(y - self.sigmoid(x))
            b[1] = b[1] + self.lr * x[1]*(y - self.sigmoid(x))
            if not self.is_norm:
                c = c + self.lr * (y - self.sigmoid(x))

        else:
            n_row = np.array(x).shape[0]
            b[0] = b[0] + self.lr *(sum([x[i][0]*(y[i] -self.sigmoid(x[i]))
                        for i in range(n_row)])/float(n_row))
```

```
                b[1] = b[1] + self.lr *(sum([x[i][1]*(y[i] -self.sigmoid(x[i]))
                            for i in range(n_row)])/float(n_row))
                if not self.is_norm:
                    c = c + self.lr *(sum([y[i] -self.sigmoid(x[i])
                                for i in range(n_row)])/float(n_row))
            self.b = b
            self.c = c

        def predict(self,x):
            y_hat = None
            if np.array(x).ndim < 2 :
                y_hat = 1 if self.predict_proba(x)>0.5 else 0
            else:
                y_hat = [1 if ypr_i>0.5 else 0  for ypr_i in self.predict_proba(x)]
            return y_hat

        def predict_proba(self,x):
            y_prob = []
            if np.array(x).ndim < 2 :
                y_prob = self.sigmoid(x)
            else:
                for x_i in x:
                    y_prob.append(self.sigmoid(x_i))


            return y_prob

        def sigmoid(self,x):
            return  1.0/(1.0+math.exp(-1*(self.b[0]*x[0]+self.b[1]*x[1]+self.c)))

        def __repr__(self):
            return "Log model param:{}, constant: {}".format(self.b, self.c)
```

0.5 Entrenamiento

Entrenaremos el modelo con una observacion. Comparado con todas las observaciones varias interaciones. Ademas, dividiremos los datos en 80% porciento para entrenamiento y el 20% para test

In [15]:
```
def measure_acc(y_pred, y_test):
    m_acc = np.mean(np.array([1.0 if y_h == y_t else 0.0
                    for y_h, y_t in zip(y_hat,y_test) ]))
    return m_acc

def norm_x(x):
    x_mean, x_std = np.mean(x), np.std(x)
    norm_x = [(x_i - x_mean)/x_std for x_i in x]

    return norm_x

x = list(zip(x_1,x_2))
ix = list(range(len(x)))
random.shuffle(ix)
cut = int(len(x)*0.8)
x_train = x[:cut]
y_train = y[:cut]
```

```
    x_test = x[cut:]
    y_test = y[cut:]
    model_1 = LogisticRegression(lr=0.05)
    for i in range(10000):
        model_1.train(x_train,y_train)

    y_hat = []
    y_hat = model_1.predict(x_test)
    print("Model 1", measure_acc(y_hat, y_test))
    print("Model1", model_1)

    model_2 = LogisticRegression(lr=0.05)
    for i in range(1000):
        for x_i, y_i in zip(x_train,y_train):
            model_2.train(x_i, y_i)

    y_hat = model_2.predict(x_test)
    print("Model 2", measure_acc(y_hat, y_test))
    print("Model2", model_2)
```

```
('Model 1', 1.0)
('Model1', Log model param:[2.664710440133677, 2.565073675138286], constant: -15.3843208369)
('Model 2', 1.0)
('Model2', Log model param:[5.89520089619068, 5.578391555676436], constant: -34.5040970603)
```

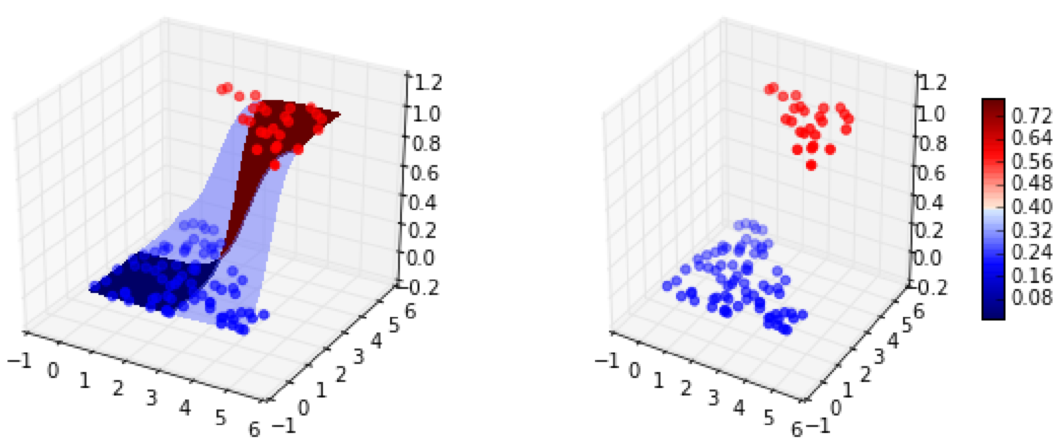
0.6 Pintando la funcion

Despues de aprender, vamos a ver como clasifica

```
In [7]: from mpl_toolkits.mplot3d import Axes3D
        from matplotlib import cm
        from matplotlib.ticker import LinearLocator, FormatStrFormatter
        from matplotlib.colors import LinearSegmentedColormap

        def create_surface(model,x_1, x_2):
            g_x1 = np.arange(min(x_1), max(x_1), 0.25)
            g_x2 = np.arange(min(x_2), max(x_2), 0.25)
            g_x1, g_x2 = np.meshgrid(g_x1, g_x2)
            g_z = np.array(zip(g_x1, g_x2))
            print (np.array(g_z).shape)
            f_sigmoid = np.apply_along_axis(lambda x: model.predict_proba(x), 1, g_z)
            return g_x1, g_x2, f_sigmoid


        def plot_matplolib(model_1, x_1, x_2, y):

            g_x1, g_x2, f_sigmoid = create_surface(model_1, x_1, x_2)
            fig = plt.figure(figsize=plt.figaspect(0.4))
            ax = fig.add_subplot(1, 2, 1, projection='3d')
            ax2 = fig.add_subplot(1, 2, 2, projection='3d')
            color = ['red' if y_i == 1 else 'blue' for y_i in y]
            cdict4 = {'red':  ((0.0, 0.0, 0.0),
                               (0.25, 0.0, 0.0),
                               (0.5, 0.8, 1.0),
                               (0.75, 1.0, 1.0),
```

```
                     (1.0, 0.4, 1.0)),
           'green': ((0.0, 0.0, 0.0),
                     (0.25, 0.0, 0.0),
                     (0.5, 0.9, 0.9),
                     (0.75, 0.0, 0.0),
                     (1.0, 0.0, 0.0)),
           'blue':  ((0.0, 0.0, 0.4),
                     (0.25, 1.0, 1.0),
                     (0.5, 1.0, 0.8),
                     (0.75, 0.0, 0.0),
                     (1.0, 0.0, 0.0))
          }
    cdict4['alpha'] = ((0.0, 1.0, 1.0),
                    #   (0.25,1.0, 1.0),
                       (0.5, 0.3, 0.3),
                    #   (0.75,1.0, 1.0),
                       (1.0, 1.0, 1.0))

    blue_red1 = LinearSegmentedColormap('BlueRedAlpha', cdict4)

    ax.scatter(x_1, x_2, y, color=color)
    surf = ax.plot_surface(g_x1, g_x2, f_sigmoid, cmap=blue_red1,
                           linewidth=0, antialiased=False)
    fig.colorbar(surf, shrink=0.5, aspect=10)
    # pintar solo los puntos
    ax2.scatter(x_1, x_2, y, color=color)
    return fig

figure = plot_matplolib(model_1, x_1, x_2, y)
plt.show()
```

```
(20, 2, 20)

/Users/millenium/anaconda/lib/python2.7/site-packages/matplotlib/collections.py:590: FutureWarning: ele
  if self._edgecolors == str('face'):
```



In [8]:
```
#Install plotly offline
# download from
#mv  plotly.min.js /Users/millenium/anaconda/lib/python2.7/site-packages/plotly/offline/
def plot_plotly(model_1, x_1, x_2, y):



    def get_text(z,x,y):
        textz = [['x:'+'{:0.5f}'.format(x[i][j])+'<br>y:'+'{:0.5f}'.format(y[i][j])+
                '<br>z:'+'{:0.5f}'.format(z[i][j])
                for j in range(z.shape[1])] for i in range(z.shape[0])]
        return textz

    g_x1, g_x2, f_sigmoid = create_surface(model_1, x_1, x_2)
    color = ['red' if y_i == 1 else 'blue' for y_i in y]
    scatter = dict(
        mode = "markers",
        name = "y",
        type = "scatter3d",
        x = x_1,  y = x_2, z =y,
        marker = dict( size=2, color=color )
    )
    data = [
        scatter,
        go.Surface(
            x=tuple(g_x1),
            y=tuple(g_x2),
            z=tuple(f_sigmoid),
            text= np.array(get_text(f_sigmoid,g_x1,g_x2)),
            hoverinfo='text',
            opacity=0.50
        )

    ]
    layout = go.Layout(
        title='Logistic Function',
        autosize=False,
        width=500,
        height=500,
        margin=dict(
            l=0.75,
            r=0.5,
            b=0.25,
            t=0
        )
    )
    fig = go.Figure(data=data, layout=layout)
    return fig


try:
    import plotly.graph_objs as go
    import plotly.offline as py
    import holoviews as hv
```

```
        import plotly
        py.init_notebook_mode()
        fig = plot_plotly(model_1, x_1, x_2, y)
        py.iplot(fig)
    except:
        print("Error important plotly")
```

```
(20, 2, 20)
```

```
In [9]: model_1 = LogisticRegression(lr=0.05, is_norm=True)
        x1_norm = norm_x(np.array(x_train)[:,0])
        x2_norm = norm_x(np.array(x_train)[:,1])
        x_norm_train = zip(x1_norm, x2_norm)

        x1_norm_t = norm_x(np.array(x_test)[:,0])
        x2_norm_t = norm_x(np.array(x_test)[:,1])
        x_norm_test =  zip(x1_norm_t, x2_norm_t)

        for i in range(10000):
            model_1.train(x_norm_train, y_train)

        y_hat = []
        y_hat = model_1.predict(x_norm_test)
        print("Model 1", measure_acc(y_hat, y_test))
        print("Model1", model_1)

        try:
            import plotly.graph_objs as go
            import plotly.offline as py
            import holoviews as hv
            import plotly
            py.init_notebook_mode()
            fig = plot_plotly(model_1, norm_x(x_1), norm_x(x_2), y)
            py.iplot(fig)

        except:
            print("Error important plotly")
```

```
('Model 1', 0.9000000000000002)
('Model1', Log model param:[1.4937650040463657, 1.5069266841607374], constant: 0.0)

(13, 2, 14)
```

```
In [ ]:
```