Tarea9_SupportVectorMachine

December 14, 2017

```
In [2]: import tensorflow as tf
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.svm import SVC
```

Para datos linealmente separables, comparamos el desempeño del perceptrón contra una máquina de soporte vectorial.

```
In [3]: dataNl = pd.read_csv('https://raw.githubusercontent.com/ClaseML-2017/MaterialyTareas/mas
        dataNl.describe()
```

```
Out[3]:              X1        X2         y
        count  7.000000  7.000000  7.000000
        mean   1.000000  0.857143  0.428571
        std    1.154701  1.345185  0.534522
        min    0.000000 -1.000000  0.000000
        25%    0.000000  0.000000  0.000000
        50%    1.000000  1.000000  0.000000
        75%    1.500000  1.500000  1.000000
        max    3.000000  3.000000  1.000000
```

Primero, construimos el perceptron.

```
In [74]: X=np.asarray([dataNl["X1"],dataNl["X2"]]).T
         Y=np.asarray(dataNl["y"]).reshape(7,1)

         input_size=2
         output_layer_size=1

         x = tf.placeholder(tf.float32, [None, input_size])
         y_ = tf.placeholder(tf.float32, [None, output_layer_size])
```

Se obtiene la variable que se va calculando y modificando en el camino.

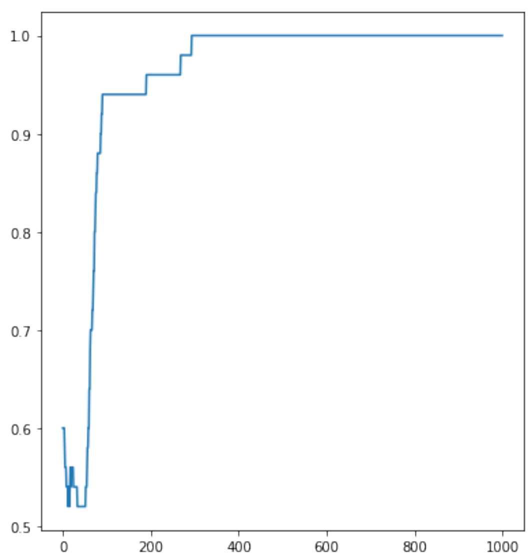
```
In [75]: W_layer1=tf.Variable(tf.random_uniform([input_size,output_layer_size], -1, 1), name="W_
         b_layer1 = tf.Variable(tf.zeros([output_layer_size]), name="b_layer1")
```

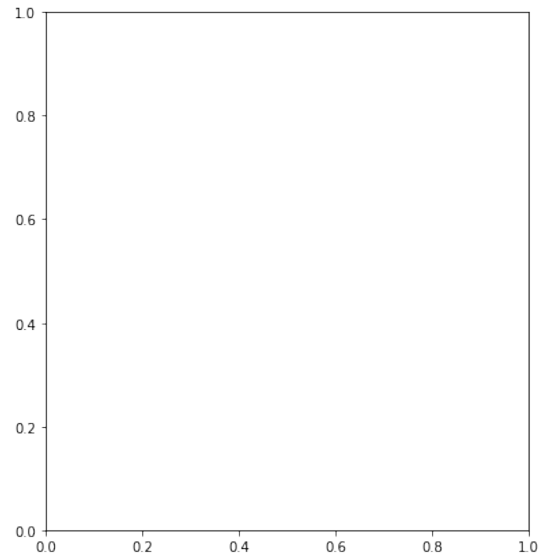
Operaciones, grafo:

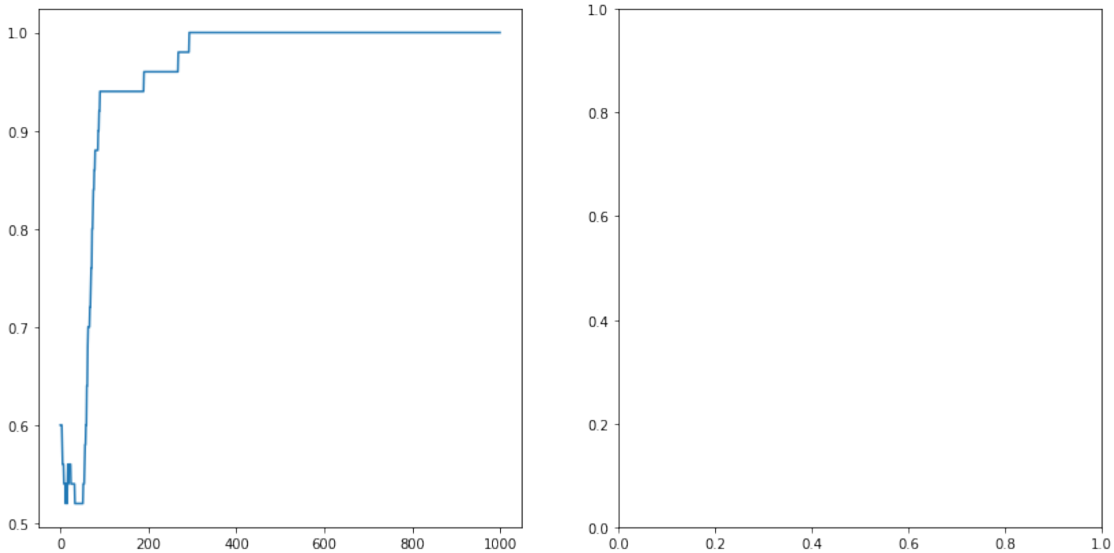
```
In [76]: y = tf.nn.sigmoid(tf.matmul(x,W_layer1)+b_layer1)
         lossfn = tf.reduce_mean(tf.reduce_sum((y_-y)**2))
         train_step = tf.train.GradientDescentOptimizer(0.01).minimize(lossfn)
```

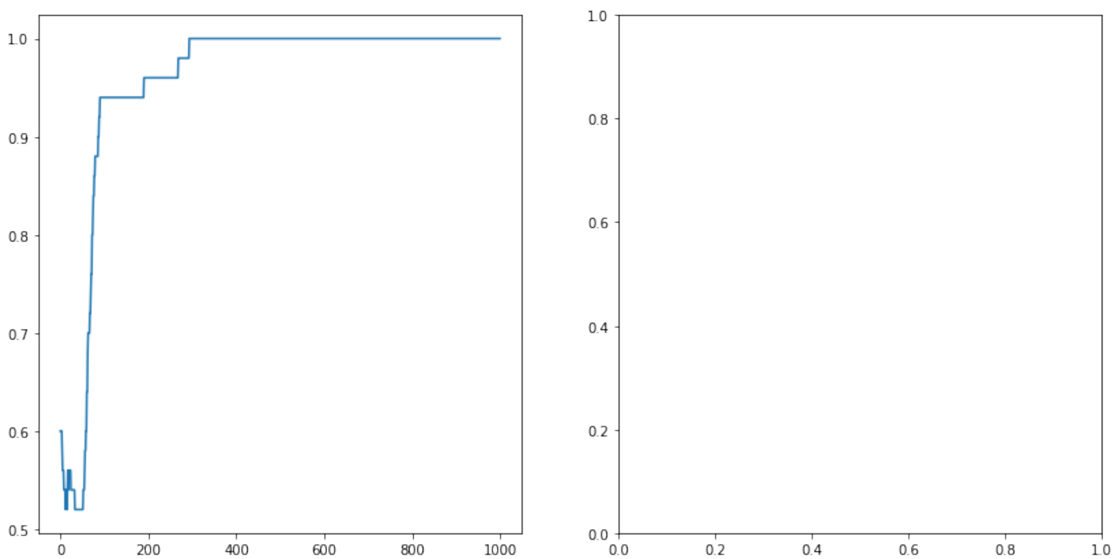
Entrenamos y graficamos.

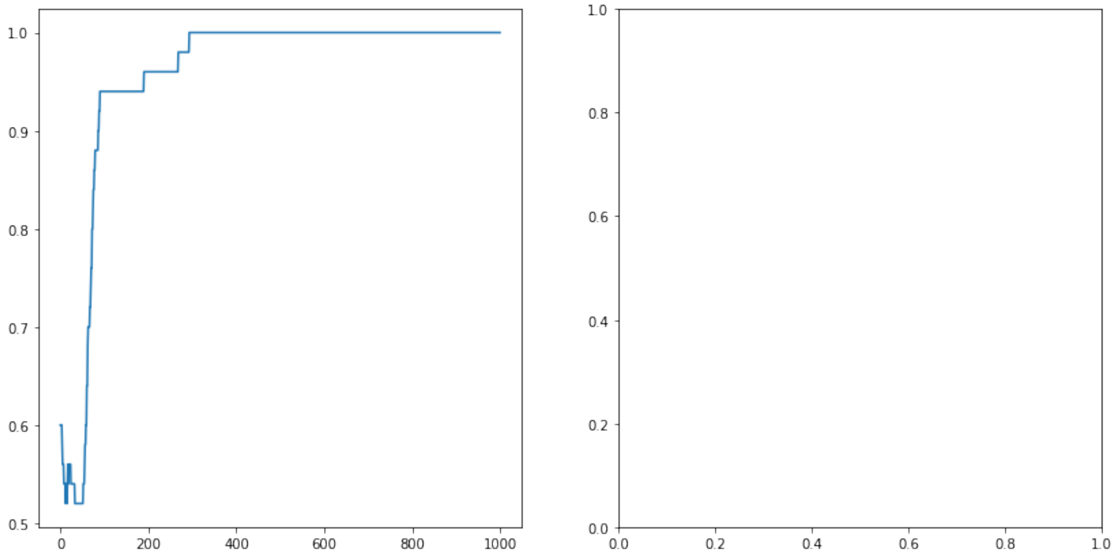
```
In [77]: init = tf.global_variables_initializer()
         sess = tf.Session()
         sess.run(init)
         for i in range(5000):
             sess.run(train_step, feed_dict={x: X, y_: Y})


         correct_prediction = tf.equal(tf.round(y),y_)
         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
         print("Precision: ",sess.run(accuracy, feed_dict={x: X, y_: Y})*100,"%")
         w=[sess.run(b_layer1, feed_dict={x: X, y_: Y})[0]]+[i[0] for i in sess.run(W_layer1, fe
         m=((w[0]/w[2]))/((-w[0]/w[1]))
         plt.scatter(X[:,0],X[:,1],c=['green' if i==1 else 'blue' for i in Y])
         x2 = np.linspace(-5.2,5.2,100) # 100 numeros espaciados
         plt.plot(x2,-w[0]/w[2]+m*x2,color='green')
         plt.show()

('Precision: ', 100.0, '%')
```

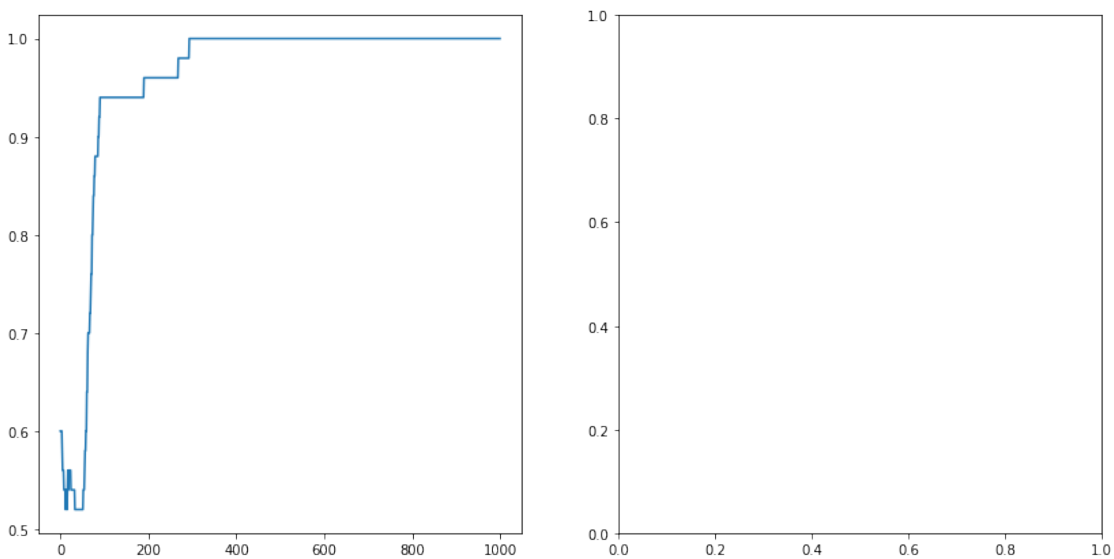


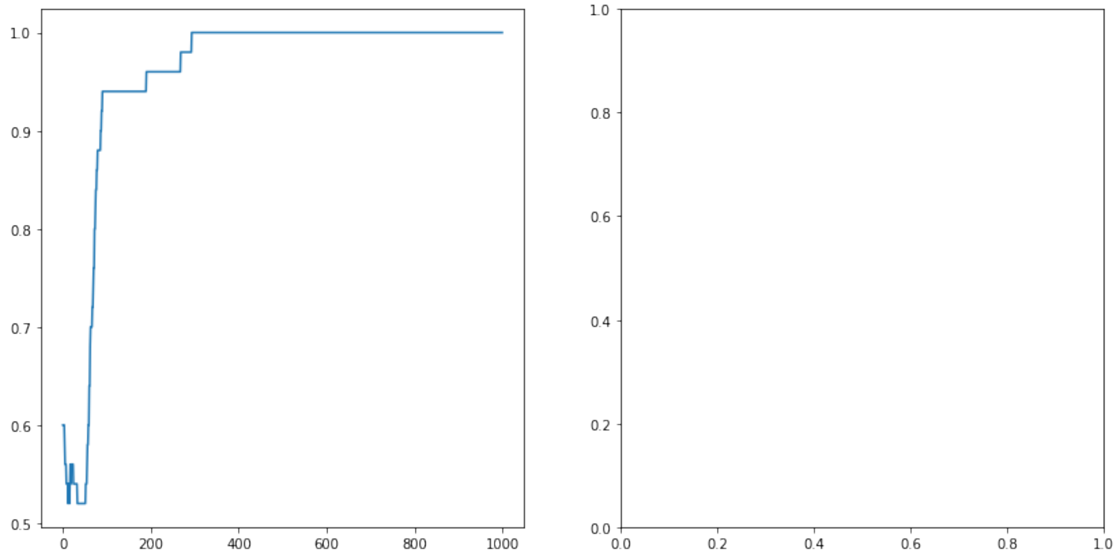


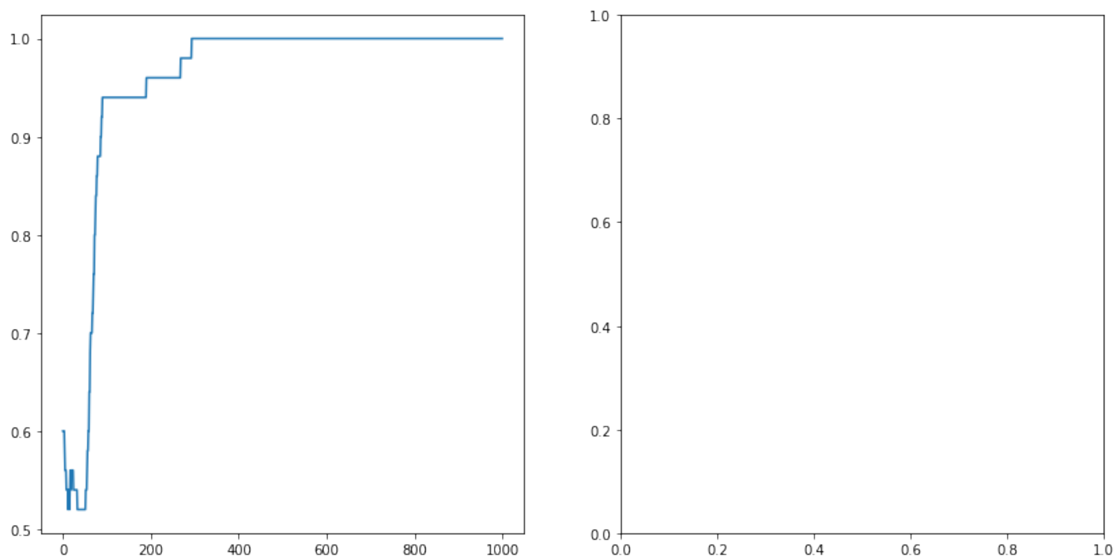

1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

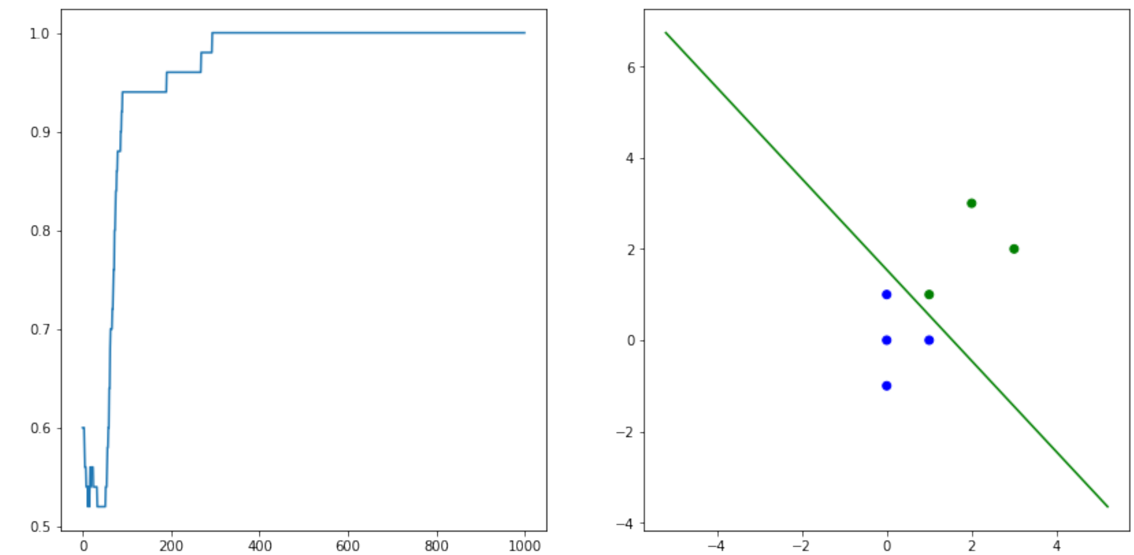

1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

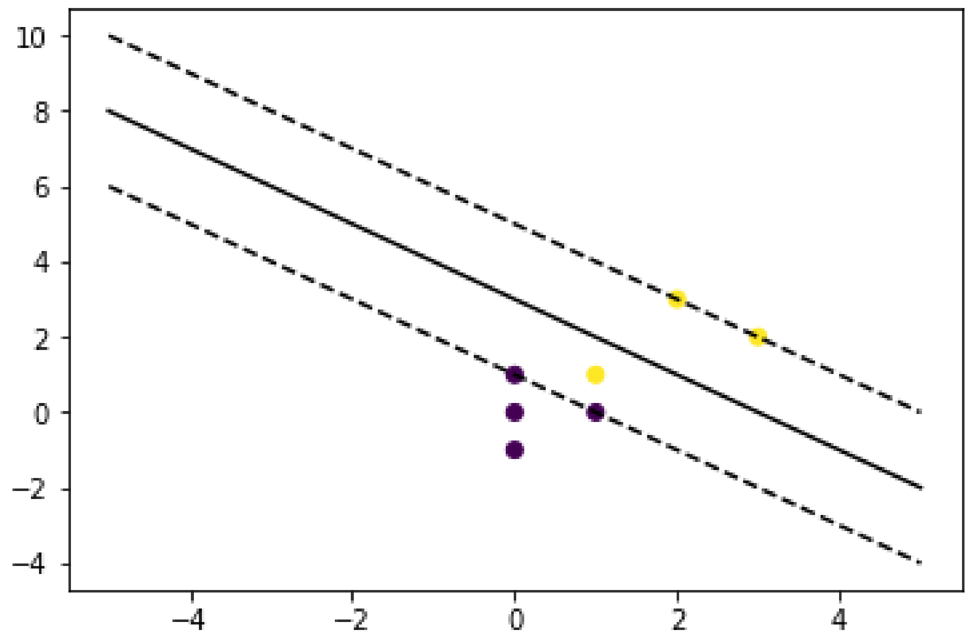



```
In [78]: sess.close()
```

Ahora construímos la SVM para una sola dimensión.

```
In [79]: X=np.asarray([dataNl["X1"],dataNl["X2"]]).T
         Y=np.asarray(dataNl["y"])

         clf = SVC(C=1.0, kernel="linear")
         clf.fit(X, Y)



         w = clf.coef_[0]
         a = -w[0] / w[1]
         xx = np.linspace(-5, 5)
         yy = a * xx - (clf.intercept_[0]) / w[1]
         margin = 1 / np.sqrt(np.sum(clf.coef_ ** 2))
         yy_down = yy - np.sqrt(1 + a ** 2) * margin
         yy_up = yy + np.sqrt(1 + a ** 2) * margin

         e = np.mean((clf.predict(X)-Y)**2)*100
         print("Error cuadrado medio:",e,"%")

         plt.clf()
         plt.plot(xx, yy, 'k-')
         plt.plot(xx, yy_down, 'k--')
         plt.plot(xx, yy_up, 'k--')
         plt.scatter(X[:,0],X[:,1],c=Y)

         plt.show()
```

```
('Error cuadrado medio:', 14.285714285714285, '%')
```

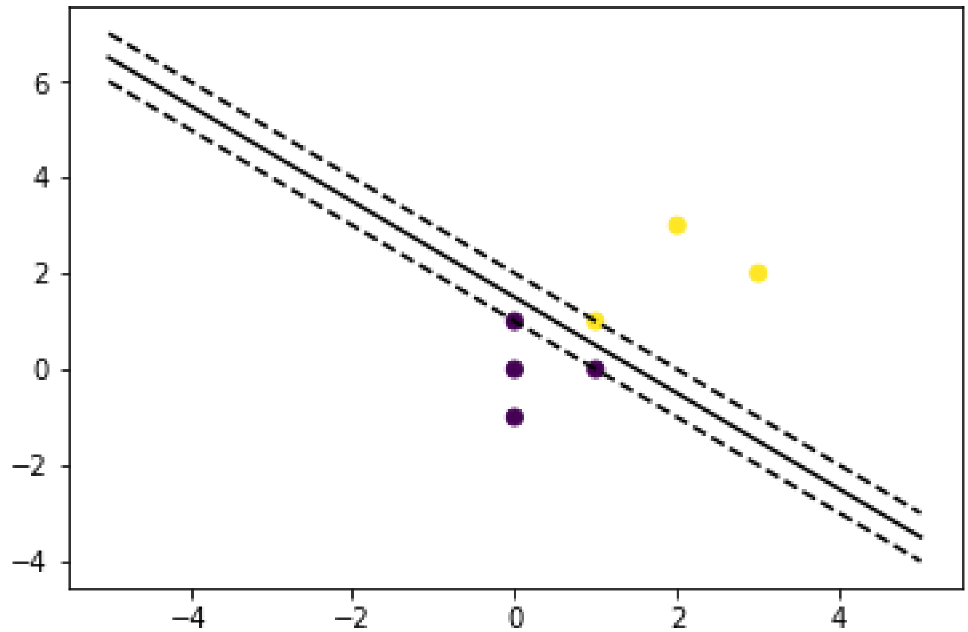


Ahora aumentamos la dimensión de la SVM a 100.

```
In [80]: X=np.asarray([dataNl["X1"],dataNl["X2"]]).T
         Y=np.asarray(dataNl["y"])

         clf = SVC(C=100.0, kernel="linear")
         clf.fit(X, Y)



         w = clf.coef_[0]
         a = -w[0] / w[1]
         xx = np.linspace(-5, 5)
         yy = a * xx - (clf.intercept_[0]) / w[1]
         margin = 1 / np.sqrt(np.sum(clf.coef_ ** 2))
         yy_down = yy - np.sqrt(1 + a ** 2) * margin
         yy_up = yy + np.sqrt(1 + a ** 2) * margin

         e = np.mean((clf.predict(X)-Y)**2)*100
         print("Error cuadrado medio:",e,"%")

         plt.clf()
         plt.plot(xx, yy, 'k-')
         plt.plot(xx, yy_down, 'k--')
```

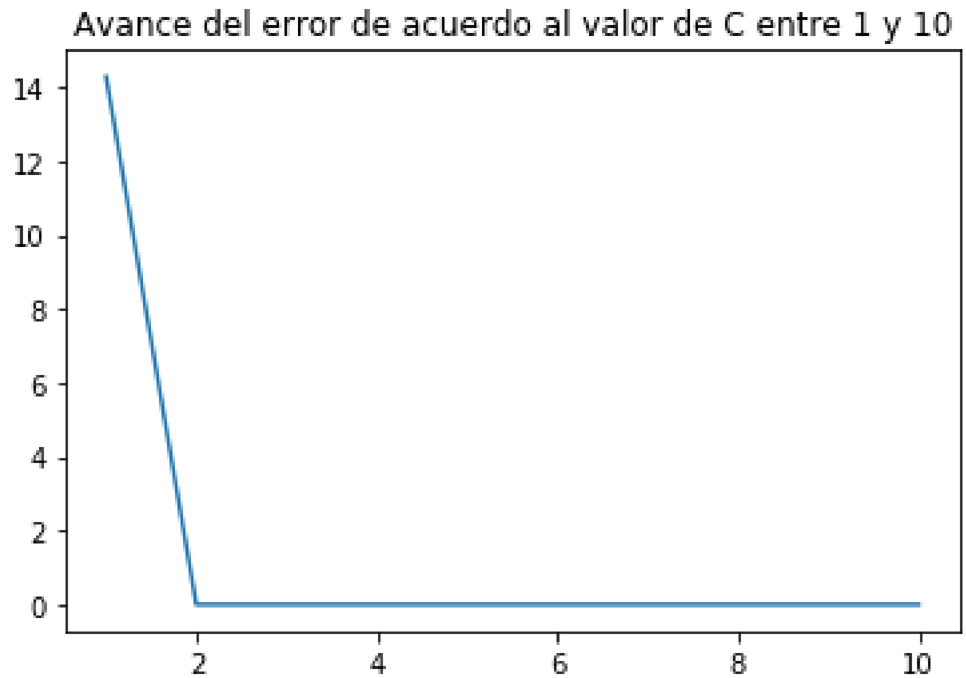
```
        plt.plot(xx, yy_up, 'k--')
        plt.scatter(X[:,0],X[:,1],c=Y)

        plt.show()
```

```
('Error cuadrado medio:', 0.0, '%')
```



El error disminuye a medida que vamos agregando dimensiones.

```
In [81]: X=np.asarray([dataNl["X1"],dataNl["X2"]]).T
         Y=np.asarray(dataNl["y"])
         n = 10
         e = np.zeros(n)
         for i in range(1,n):
             clf = SVC(C=i, kernel="linear")
             clf.fit(X, Y)
             e[i-1] = np.mean((clf.predict(X)-Y)**2)*100

         plt.plot(range(1,n+1),e)
         plt.title("Avance del error de acuerdo al valor de C entre 1 y 10")
         plt.show()
```



Finalmente, resta comparar el desempeño de las máquinas de soporte vectorial con las redes neuronales para datos que no son linealmente separables.

Primero, generamos los datos.

```
In [82]: npuntos = 50
         mu = 0
         var = 1
         X = np.random.normal(mu, var, [npuntos,2])
         Y = 1.0*np.array(X[:,0]**2 + X[:,1]**2 < var**2).reshape(npuntos,1)
```

Construimos la red neuronal.

```
In [83]: input_size=2
         hiden_size=4
         output_size=1

         x = tf.placeholder(tf.float32, [None, input_size])
         y_ = tf.placeholder(tf.float32, [None, output_size])


         weights = {
             'w_h': tf.Variable(tf.random_uniform([input_size, hiden_size], -1, 1)),
             'w_out': tf.Variable(tf.random_uniform([hiden_size, output_size], -1, 1))
         }
         biases = {
             'b_h': tf.Variable(tf.zeros([hiden_size])),
```

```
        'b_out': tf.Variable(tf.zeros([output_size]))
    }
```

```
In [84]: hlayer = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['w_h']), biases['b_h']))
```

Salida completamente conectada (una salida)

```
In [85]: y = tf.nn.sigmoid(tf.matmul(hlayer, weights['w_out']) + biases['b_out'])
```

Funcion de pérdida

```
In [86]: lossfn = tf.reduce_mean(tf.reduce_sum((y_-y)**2)) #cuadratico
```

Optimizamos

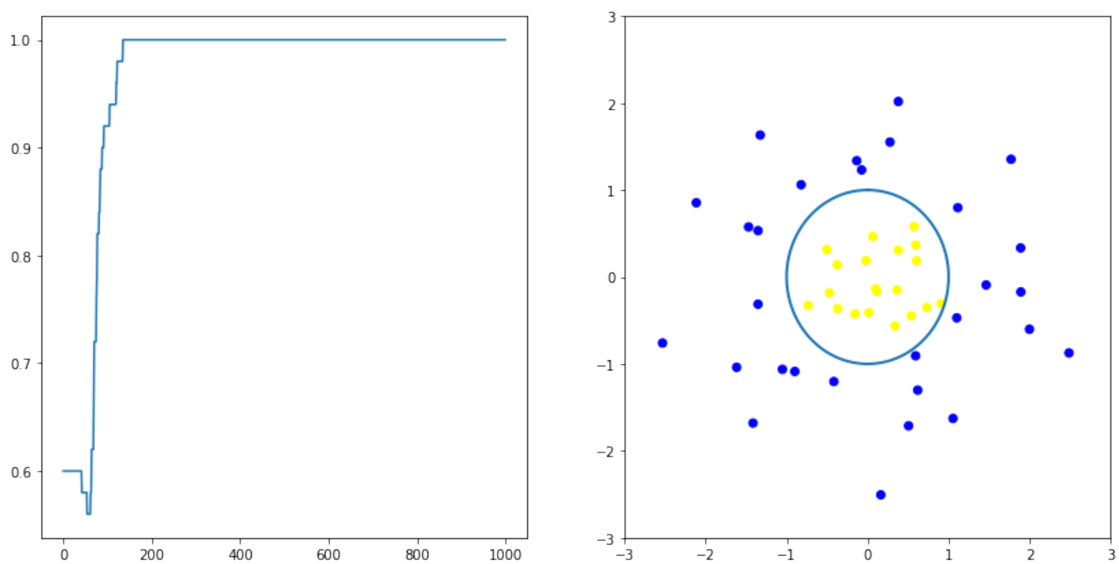
```
In [87]: train_step = tf.train.GradientDescentOptimizer(0.1).minimize(lossfn)


         init = tf.global_variables_initializer()
         sess = tf.Session()

         correct_prediction = tf.equal(tf.round(y),y_)
         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

         sess.run(init)
         n = 1000
         acc = np.zeros(n)
         for i in range(n):
             aux, acc[i] = sess.run([train_step,accuracy], feed_dict={x: X, y_: Y})
```

Graficamos los resultados.

```
In [88]: print("Capa oculta de",hiden_size," neuronas")
         accG = sess.run(accuracy, feed_dict={x: X, y_: Y})*100
         print("Exactitud: ",accG,"%")

         #Progreso de accuracy:
         plt.figure(figsize=(14,7))
         plt.subplot(1,2,1)
         plt.plot(range(n), acc)


         #Partición del espacio:
         bs=[sess.run(biases["b_h"], feed_dict={x: X, y_: Y})][0]
         ws=[sess.run(weights["w_h"], feed_dict={x: X, y_: Y})][0]

         plt.subplot(1,2,2)

         ng=100
```

```
x2 = np.linspace(-2*var,2*var,ng)
Y2 = sess.run(tf.round(y), feed_dict={x: X, y_: Y})



plt.scatter(X[:,0],X[:,1],c=['yellow' if i==1 else 'blue' for i in Y])

#Circulo:
xC = np.linspace(0,2*3.1415,ng)
plt.plot(var*np.sin(xC),var*np.cos(xC), linewidth=2)

plt.xlim(-3*var,3*var)
plt.ylim(-3*var,3*var)
plt.show()
```

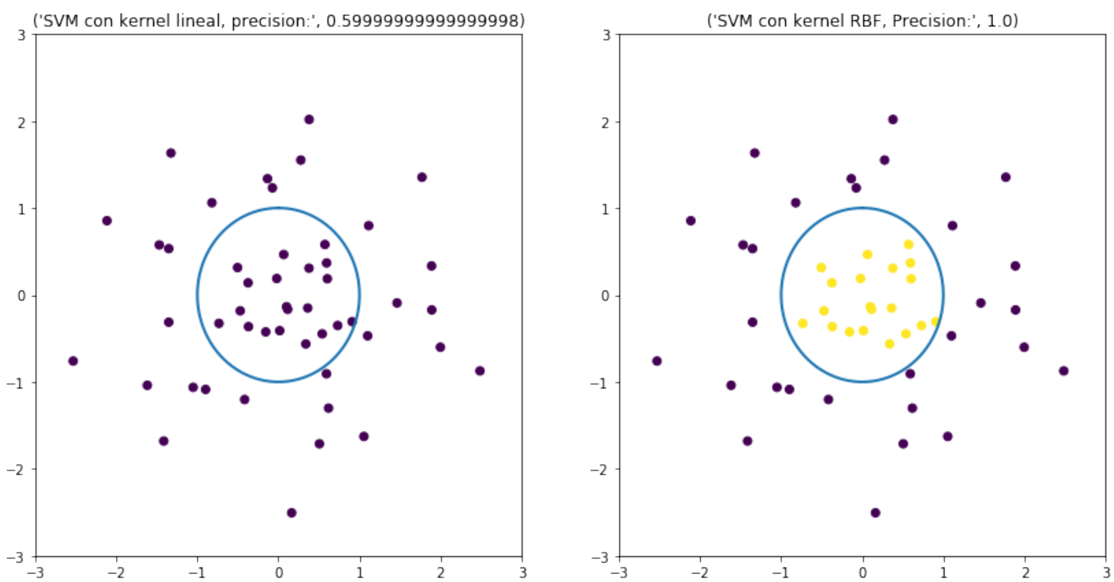
```
('Capa oculta de', 4, ' neuronas')
('Exactitud: ', 100.0, '%')
```



Ahora, comparamos con distintos kernels para las máquinas de soporte vectorial.

```
In [90]: plt.figure(figsize=(14,7))

         clf = SVC(C=1.0, kernel="linear")
         clf.fit(X, Y.ravel())
         #Circulo:
         plt.subplot(1,2,1)
         plt.plot(var*np.sin(xC),var*np.cos(xC), linewidth=2)
         ypred = clf.predict(X)
         plt.scatter(X[:,0],X[:,1],c=ypred)
```

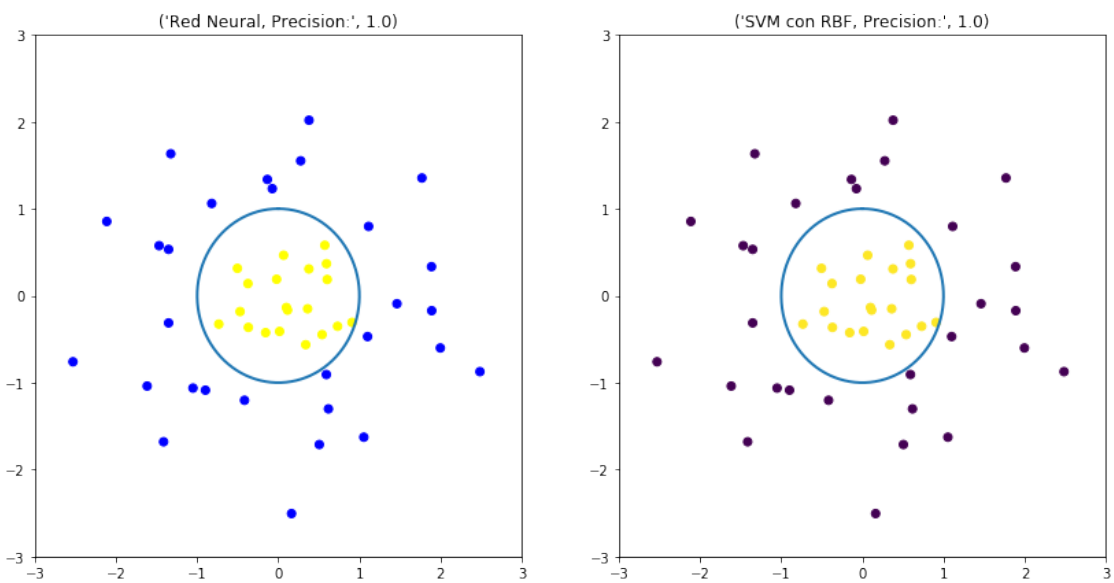
```
acc = np.mean(ypred==Y.T)
plt.xlim(-3*var,3*var)
plt.ylim(-3*var,3*var)
cad = "SVM con kernel lineal, precision:",acc
plt.title(cad)


clf = SVC(C=1.0, kernel="rbf")
clf.fit(X, Y.ravel())
#Circulo:
plt.subplot(1,2,2)
plt.plot(var*np.sin(xC),var*np.cos(xC), linewidth=2)
ypred = clf.predict(X)
plt.scatter(X[:,0],X[:,1],c=ypred)
acc = np.mean(ypred==Y.T)
plt.xlim(-3*var,3*var)
plt.ylim(-3*var,3*var)
cad = "SVM con kernel RBF, Precision:",acc
plt.title(cad)
plt.show()
```



Ahora, tomamos el kernel de RBF y comparamos esta máquina de soporte vectorial con una red neuronal.

```
In [94]: plt.figure(figsize=(14,7))
         plt.subplot(1,2,1)
         ng=100
         x2 = np.linspace(-2*var,2*var,ng)
         Y2 = sess.run(tf.round(y), feed_dict={x: X, y_: Y})
```

```
plt.scatter(X[:,0],X[:,1],c=['yellow' if i==1 else 'blue' for i in Y])

xC = np.linspace(0,2*3.1415,ng) # 100 numeros espaciados
plt.plot(var*np.sin(xC),var*np.cos(xC), linewidth=2)
plt.xlim(-3*var,3*var)
plt.ylim(-3*var,3*var)
cad = "Red Neural, Precision:", (accG/100)
plt.title(cad)


plt.subplot(1,2,2)
plt.plot(var*np.sin(xC),var*np.cos(xC), linewidth=2)
ypred = clf.predict(X)
plt.scatter(X[:,0],X[:,1],c=ypred)
acc = np.mean(ypred==Y.T)
plt.xlim(-3*var,3*var)
plt.ylim(-3*var,3*var)
cad = "SVM con RBF, Precision:",acc
plt.title(cad)
plt.show()
```



De aquí podemos concluir que para este problema ambos métodos son igualmente precisos.