The XOR Problem

Wantee Wang

2015-04-27 10:39:02 +0800

Contents

The XOR is an interesting problem, not only because it is a classical example for *Linear Separability*, but also it played a significant role in the history of neutral network research.

1 Probelm

The truth table for XOR is

x	y	x *xor* y
0	0	0
0	1	1
1	0	1
1	1	0

It is impossible for a classifier with linear decision boundary to learn an XOR function. This can be seen easily by the following plot(Figure 1).

Apparently, we can't using a line to separate the two classes.

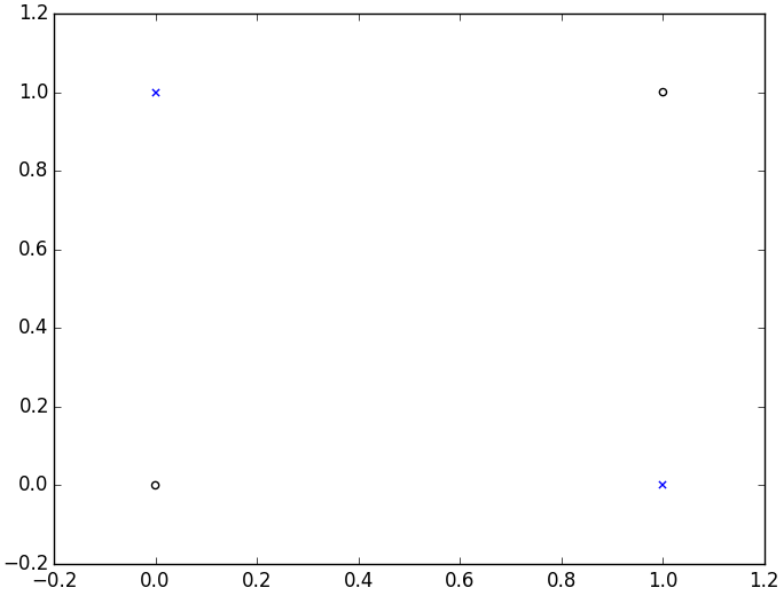

Figure 1: The XOR Problem

2 Non-linear Boundary

If we take a carefully look at the scatter figure, it can be found that it's easy to use an ellipse or hyperbola to separate the classes.

Recall that, the general equation for ellipse or hyperbola is

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \tag{1}$$

Thus, we can just feed those above components(x, y, x^2, y^2, xy) to a linear classifier, and see if the classes can be separated.

We use scikit-learn to perform the experiments. Following shows the code,

```
import sys
import matplotlib.pyplot as plt
import numpy as np
#from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import Perceptron

```

```
X = []
y = []
for i in range(2):
    for j in range(2):
        X.append([i, j])
        y.append(i ^ j)

for x in X:
    x.extend([x[0]*x[0], x[1]*x[1], x[0]*x[1]])

X = np.array(X)
y = np.array(y)

#clf = SGDClassifier(loss='log', n_iter=10, shuffle=False).fit(X, y)
clf = Perceptron(n_iter=10, shuffle=False).fit(X, y)

if clf.score(X, y) != 1.0:
    print 'Failed to fit the data.'
    sys.exit(1)

plt.title("%fx%+fy%+fx^2%+fy^2%+fxy%+f" % (clf.coef_[0, 0], clf.coef_[0, 1],
            clf.coef_[0, 2], clf.coef_[0, 3], clf.coef_[0, 4],
            clf.intercept_))

for i in range(len(y)):
    if y[i] == 1:
        plt.scatter(X[i, 0], X[i, 1], marker=u'x')
    elif y[i] == 0:
        plt.scatter(X[i, 0], X[i, 1], marker=u'o', facecolors='none')

XX, YY = np.mgrid[-2:3:200j, -2:3:200j]
XXX = []
for xs, ys in zip(XX, YY):
    for x_, y_ in zip(xs, ys):
        XXX.append([x_, y_, x_*x_, y_*y_, x_*y_])
Z = clf.decision_function(XXX)

Z = Z.reshape(XX.shape)
plt.contour(XX, YY, Z, levels=[0])
plt.show()
```

After running, we see the final decision boundary(Figure 2),

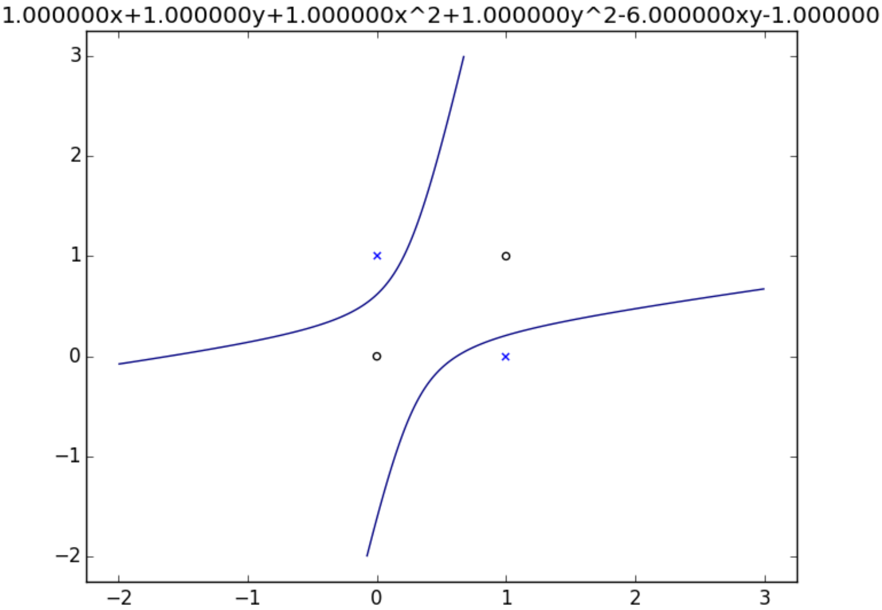

Figure 2: Non-linear boundary

3 Removing Redundant Features

It is can be seen that in the final equation (1), $A = D$ and $C = E$(it will be more clear if we use logistic regression to fit the data). Actually, for boolean features, the high-order polynomial features are useless, because $\forall n, x_i^n = x_i$. So we can only use the interaction features($x_i x_j$). This time we get the features from PolynomialFeatures class of scikit-learn.

```
import sys
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
#from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import Perceptron

X = []
y = []
for i in range(2):
    for j in range(2):
```

```
        X.append([i, j])
        y.append(i ^ j)

X = PolynomialFeatures(interaction_only=True).fit_transform(X)
clf = Perceptron(fit_intercept=False, n_iter=20, shuffle=False).fit(X, y)

if clf.score(X, y) != 1.0:
    print 'Failed to fit the data.'
    sys.exit(1)

plt.title("%fx%+fy%+fxy%+f" % (clf.coef_[0, 1], clf.coef_[0, 2],
            clf.coef_[0, 3], clf.coef_[0, 0]))

for i in range(len(y)):
    if y[i] == 1:
        plt.scatter(X[i, 1], X[i, 2], marker=u'x')
    elif y[i] == 0:
        plt.scatter(X[i, 1], X[i, 2], marker=u'o', facecolors='none')

XX, YY = np.mgrid[-2:3:200j, -2:3:200j]
XXX = []
for xs, ys in zip(XX, YY):
    for x_, y_ in zip(xs, ys):
        XXX.append([1, x_, y_, x_*y_])
Z = clf.decision_function(XXX)

Z = Z.reshape(XX.shape)
plt.contour(XX, YY, Z, levels=[0])
plt.show()
```

Again, we show the final decision boundary(Figure 3),

By adding polynomial features to the model inputs, we are actually mapping the features to higher dimension. This is the SVM's job, here we just choose the features manually.

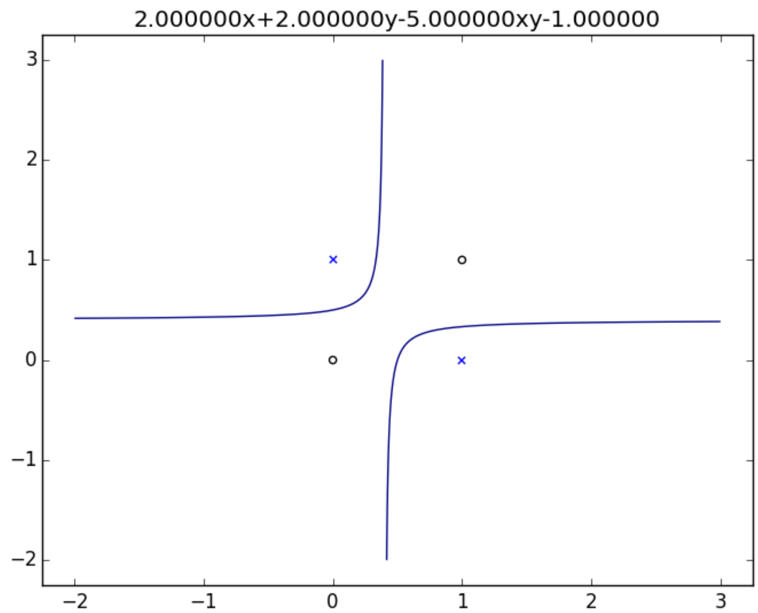

Figure 3: Non-linear boundary using 3-d features