ANN_Basic_2

2018 年 7 月 18 日

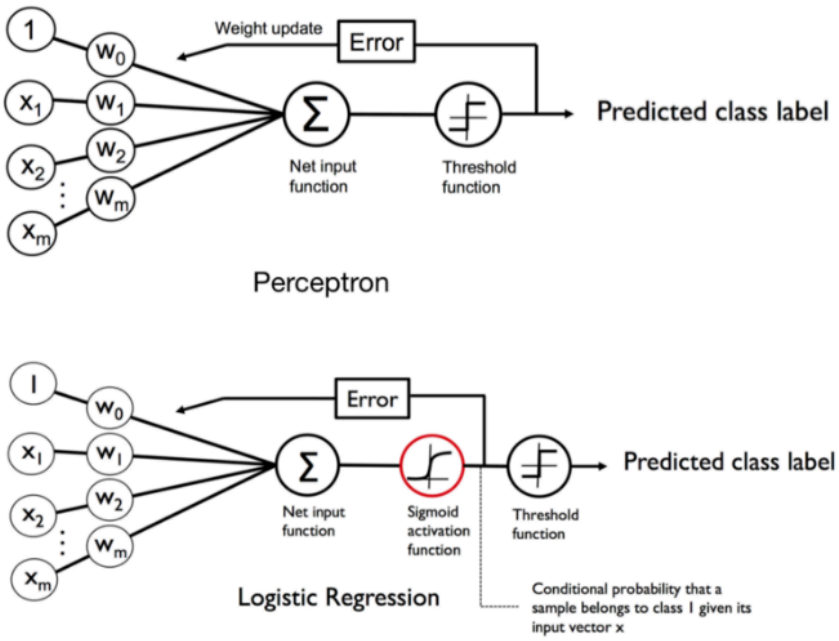
1　初探神經網路 (邏輯斯迴歸)

1.1　介紹

前面我們說的問題 1，對於解釋性的部分我們的感知器有點稍嫌弱，有人就提出一個變體，我們在進到我們的分類前，先轉換成一個機率值

那怎麼轉換機率值呢？我們經過一個 0~1 連續分布的函數不就得了

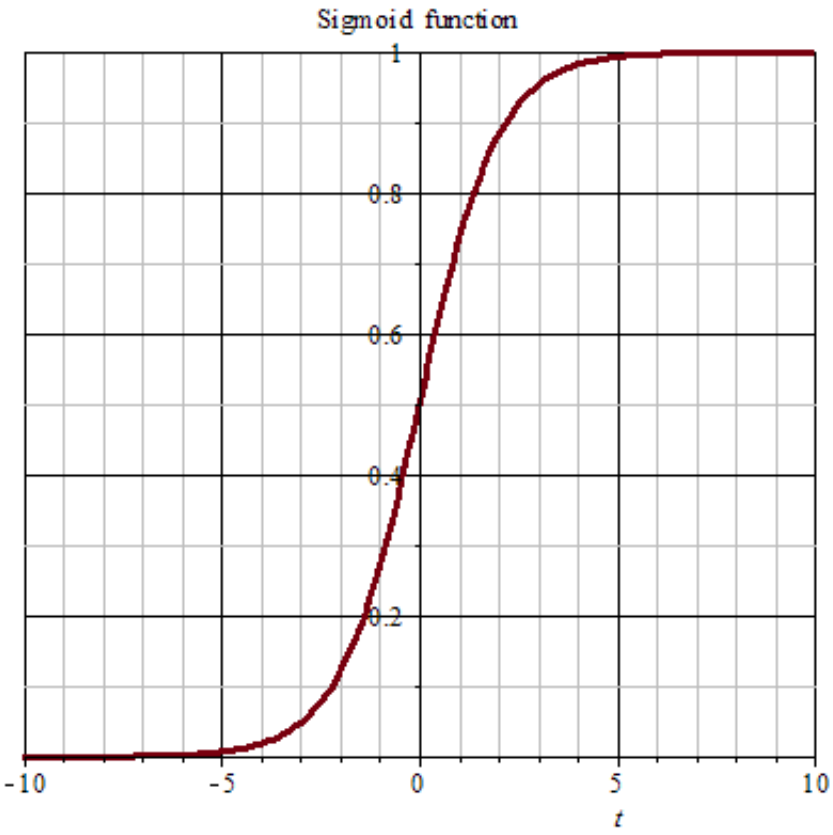
所以我們把我們的模型轉換成下面這樣



Perceptron以及Logistic Regression模型的差異

這個函數我們常用所謂的 S 函數 (sigmoid function) 來轉換，之所以叫 S 函數是因為他長得像 S 型，同時他也有個別名叫做 Logistic 函數

$$S(t) = \frac{1}{1 + e^{-t}}.$$



經過這樣轉換，我們就也可以跟人家說出機率了!
這個就叫我們的邏輯斯迴歸 (Logistic Regression)

注意：邏輯斯回歸雖然有個迴歸但是他是一個分類模型！這迴歸字眼指的是我們用這個 S 函數去迴歸 (擬合) 的意思

所以步驟是這樣的

1. 我們有一個特徵向量 [a, b, c]
2. 乘上係數 score = w1 * a + w2 * b + w3 * c
3. (跟之前不一樣的一步) 再將我們算出的 score 經過 S 函數，得到一個新 score 值，這值就可以拿來當作機率了
4. 讓我們的 score 經過一個啟動函數 (Activation Function)，這啟動函數我們先設定成最基本的超過一定的量輸出 1，否則輸出 0

1.2 需要函式庫

1. mlxtend: 可以幫我們快速畫出分類線的函式庫

1.3 資料集

我們利用之前已經用過的鳶尾花數據集，但我們這次做一點比較特別的事
我們只用花瓣和花萼的長度來分類就好
因為我們希望讓你看到決策的邊界

1.4 開始撰寫程式

由於步驟和之前差不多，這裡就直接從頭寫到結束，首先先準備資料集

```
In [1]: from sklearn.datasets import load_iris
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

        # 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示 10 個
        # 大家練習的時候可以去掉下面兩行
        pd.set_option('display.max_rows', 15)
        pd.set_option('display.max_columns', 10)

        # 為了顯示的漂亮，有時候 sklearn 會提示一下 Future Warning
        # 我也把關掉了
        import warnings
        warnings.filterwarnings('ignore')

        # 使用 scikit-learn 提供的鳶尾花資料庫
        iris = load_iris()
        df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
        df["target"] = iris["target"]
        df = df.drop(["petal width (cm)", "sepal width (cm)"], axis = 1)

        # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練
        from sklearn.model_selection import train_test_split
        # 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
        data = df.drop(["target"], axis = 1)
        data_train, data_test, target_train, target_test = train_test_split(data,
                                                                            df['target'],
                                                                            test_size=0.1)
```

接著訓練我們的 Logistic Regression

```
In [4]: from sklearn.linear_model import LogisticRegression
        clf = LogisticRegression()
        clf = clf.fit(data_train, target_train)
```

預測試試看，但這裡我們順便把機率印出來

```
In [6]: from sklearn.metrics import accuracy_score

        predict = clf.predict(data_test)
        print("預測:", list(predict))
        print("正確標籤:", list(target_test))
        print("正確率: ", accuracy_score(target_test, predict) * 100, "%")
```

```
預測: [2, 1, 0, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 0]
正確標籤: [1, 1, 0, 2, 1, 2, 1, 0, 1, 0, 1, 2, 2, 0, 0]
正確率:  86.6666666667 %
```

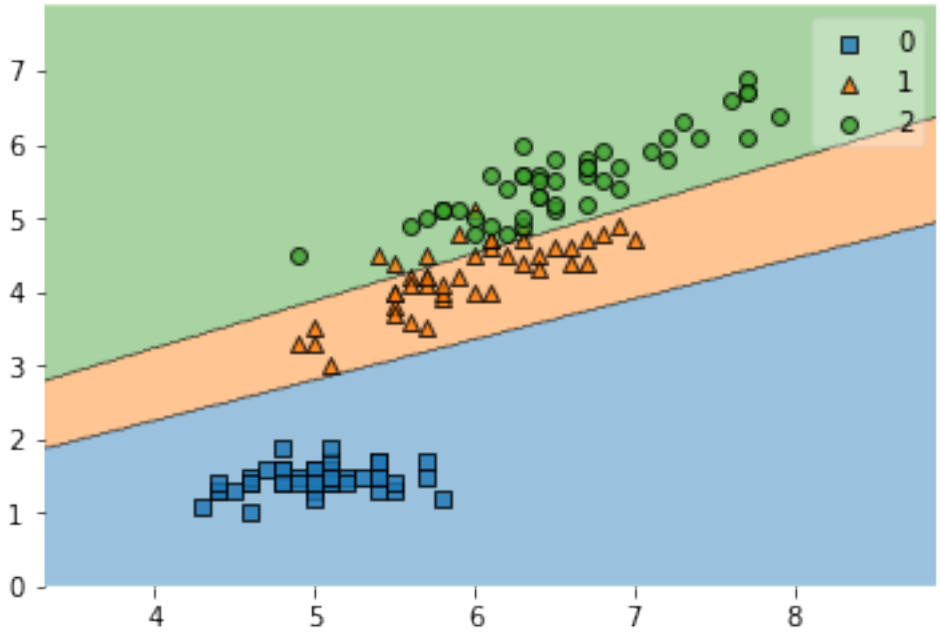
```
In [8]: # 你可以看到我們的 sample 分屬 0, 1, 2 的機率
        proba = pd.DataFrame(clf.predict_proba(data_test))
        pd.DataFrame(proba)
```

Out[8]:

	0	1	2
0	0.009954	0.482254	0.507793
1	0.015853	0.500694	0.483452
2	0.821032	0.178457	0.000511
3	0.010367	0.494120	0.495513
4	0.029885	0.528887	0.441228
5	0.000611	0.336665	0.662724
6	0.006807	0.418578	0.574614
7	0.823367	0.176253	0.000380
8	0.068456	0.627174	0.304370
9	0.821987	0.177550	0.000463
10	0.053154	0.634643	0.312203
11	0.000623	0.327043	0.672334
12	0.002899	0.379808	0.617293
13	0.821987	0.177550	0.000463
14	0.841304	0.158456	0.000240

```
In [9]: from mlxtend.plotting import plot_decision_regions
        import numpy as np
        plot_decision_regions(X=np.array(data_train),
                              y=np.array(target_train),
                              clf=clf)
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x10aedf048>
```



1.5 結論

透過上面的決策邊界，你可以看到，Logistic Regresson 雖然解決了第一個問題，但卻沒解決我們的第二個問題，非線性的分類。所以實用性也不算高，也屬於精神象徵類型，下一章節我們繼續往前邁進，對於非線性分類給出一個好的答案!