

MA 506 Probability and Statistical Inference

Lecture 21: Logistic Regression Introduction

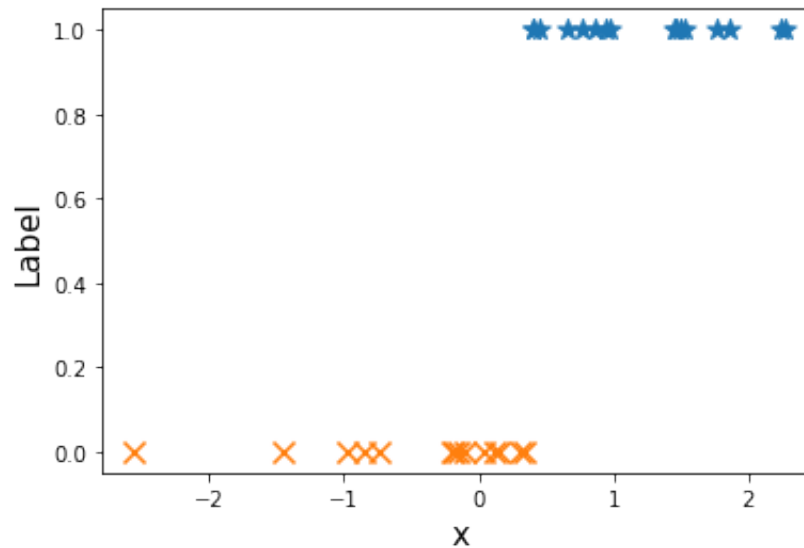
```
In [3]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
```

Logistic Regression model

Lets generate some synthetic data

```
In [4]: np.random.seed(0)
l = 30
xs = np.random.randn(l)
ys = np.ones(l)
ys[xs < 0.4] = 0
```

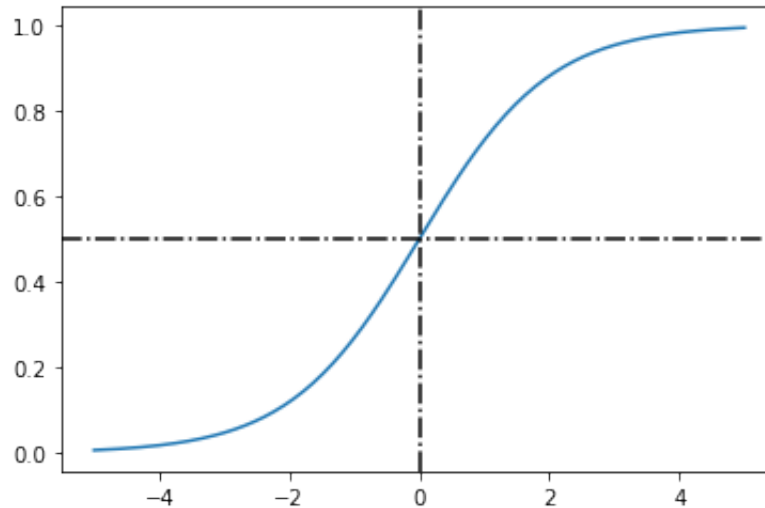
```
In [5]: plt.scatter(xs[ys == 1],ys[ys == 1],marker = '*',s = 100)
plt.scatter(xs[ys == 0],ys[ys == 0],marker = 'x',s = 100)
plt.xlabel('x',size = 15)
plt.ylabel('Label',size = 15)
plt.show()
```



Since this is a classification problem, instead of fitting a function directly and predicting the y values, we fit a special kind of function which can predict a discrete label $y = 0/1$. **We fit a logistic or a sigmoid function**

$$y = \frac{1}{1 + e^{-x}}$$

```
In [4]: x = np.linspace(-5,5,200)
y = 1/(1 + np.exp(-1*x))
plt.plot(x,y)
plt.axvline(x = 0,color = 'k',linestyle = '-.')
plt.axhline(y = 0.5,color = 'k',linestyle = '-.')
plt.show()
```



Here, if the standard sigmoid above is a best fit to our data, then

1. All the x for which $y > 0.5$ are predicted to have a label 1, i.e $x > 0 \implies y = 1$
2. All the x for which $y < 0.5$ are predicted to have a label 0, i.e $x < 0 \implies y = 0$
3. **Hence for the standard sigmoid $x = 0$ is the breaking point**

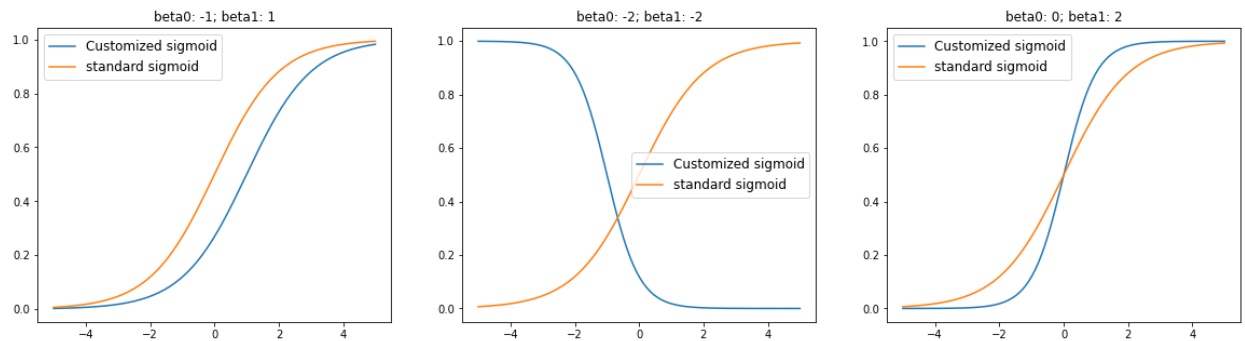
Make the sigmoid function parameterizable to fit to any data

We can put any degree polynomial as a power of e .

```
In [5]: def sigmoid(x):
poly = x
y = 1/(1 + np.exp(-1*(poly)))
return y

def custom_sigmoid(beta0,beta1,x):
poly = beta0 + beta1*x
y = 1/(1 + np.exp(-1*(poly)))
return y
```

```
In [6]: x = np.linspace(-5,5,200)
beta0 = [-1,-2,0]
beta1 = [1,-2,2]
fig = plt.figure(figsize=(20,5))
for i in range(len(beta0)):
    ax = fig.add_subplot(1,3,i+1)
    plt.plot(x,custom_sigmoid(beta0[i],beta1[i],x), label = 'Customize')
    plt.plot(x,sigmoid(x), label = 'standard sigmoid')
    plt.title(f'beta0: {beta0[i]}; beta1: {beta1[i]}')
    plt.legend(prop = {'size':12})
plt.show()
```



Exercise

1. Find the beaking point of x for all 3 customized sigmoids

Hence for any univariate dataset:

1. Optimize for good values of β_0 and β_1 that can fit well to the given data
2. For fitting a more complex function, just increase the complexity of the polynomial in the power of e

For a multivariate dataset

1. just use a multivariate polynomial in the power of e

Using sklearn to fit to the synthetic data

```
In [21]: from sklearn.linear_model import LogisticRegression
```

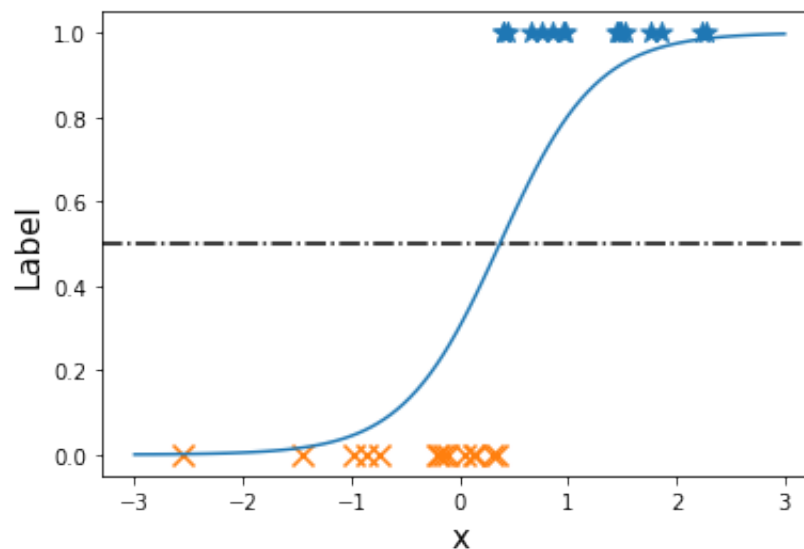
In [8]:

```
clf = LogisticRegression(random_state=0).fit(xs.reshape(-1,1), ys)
```

In [9]:

```
## data
plt.scatter(xs[ys == 1],ys[ys == 1],marker = '*',s = 100)
plt.scatter(xs[ys == 0],ys[ys == 0],marker = 'x',s = 100)
plt.xlabel('x',size = 15)
plt.ylabel('Label',size = 15)

## Prediction
x = np.linspace(-3,3,100)
beta0 = clf.intercept_[0]
beta1 = clf.coef_[0]
y = custom_sigmoid(beta0,beta1,x)
plt.axhline(y = 0.5,color = 'k', linestyle = '-.')
plt.plot(x,y)
plt.show()
```



Hence the x value for which sigmoid function goes above 0.5 is the breakpoint. Hence breakpoint is the solution x to the following problem

$$0.5 = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Getting a different classification dataset

In [10]:

```
D = load_breast_cancer()
```

In [11]:

```
print(D.DESCR)
```

```
.. _breast_cancer_dataset:
```

```
Breast cancer wisconsin (diagnostic) dataset
```

```
-----
```

```
**Data Set Characteristics:**
```

```
    :Number of Instances: 569
```

```
    :Number of Attributes: 30 numeric, predictive attributes and the
    class
```

```
    :Attribute Information:
```

```
        - radius (mean of distances from center to points on the peri
meter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)
```

```
    The mean, standard error, and "worst" or largest (mean of the
three
worst/largest values) of these features were computed for eac
h image,
resulting in 30 features. For instance, field 0 is Mean Radi
us, field
10 is Radius SE, field 20 is Worst Radius.
```

```
    - class:
```

```
        - WDBC-Malignant
        - WDBC-Benign
```

```
    :Summary Statistics:
```

```
=====
```

	Min	Max
radius (mean)	6.081	28.11

```
=====
```

radius (mean):	0.001	20.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208
=====	=====	=====

:Missing Attribute Values: None

:Class Distribution: 212 – Malignant, 357 – Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset S.

<https://goo.gl/U2Uwz2> (<https://goo.gl/U2Uwz2>)

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

.. topic:: References

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

How to test if a model is fitted/performing well

1. Divide the data into training and testing set. Training set has the majority of samples. For example 80%
2. If you want to compare between 2 models. Fit both models on the training set. Then predict on the testing set.
3. The model with better performance on the testing set (unseen samples) is chosen as the best model.

```
In [23]: from sklearn.model_selection import train_test_split
```

```
In [24]: X = D.data
         y = D.target
```

```
In [25]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

```
In [26]: clf = LogisticRegression(random_state=0).fit(X_train,y_train)
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>
(<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
In [27]: clf = LogisticRegression(random_state=0,solver = 'newton-cg',penalty =
```

```
In [29]: pred = clf.predict(X_test)
```

```
In [30]: ## Computing accuracy
```

```
In [31]: from sklearn.metrics import accuracy_score
```

In [32]: `accuracy_score(y_test, pred)`

Out[32]: 0.956140350877193

In []: