Gradient Descent, Decision Boundary, and Logistic Loss

Shirong Xu

October 11, 2024

This PDF aims to provide an example of using surrogate loss to find a classifier and demonstrates that each classifier corresponds to a decision boundary. Different function classes will have different type of decision boundaries.

Step 1. Setup: Suppose a classification dataset is available $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. Here $\boldsymbol{x}_i = (x_{i1}, x_{i2})$ is 2-dimensional feature and $y_i \in \{-1, 1\}$ is binary label. The objective is find a classifier f that can predict the label of a new $\boldsymbol{x}$, say $\boldsymbol{x}_{new}$. In the following Python codes, I generate such a dataset for simulation:

```python
import numpy as np
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
np.random.seed(42) # Set random seed for reproducibility
X, y = make_classification(n_samples=100, n_features=2, n_informative=2, flip_y=0.15,class_sep=0.2,
                           n_redundant=0, n_clusters_per_class=1, random_state=42)
y = 2*y-1 # let y take -1 or 1
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis') # there are two classes in this dataset in different colors
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
plt.grid()
```

Listing 1: Data Generation

In this dataset, there are 100 samples with the true relationship between $\boldsymbol{X} = (X_1, X_2)$ and Y being unknown. Here X_1 means the first dimension of feature and X_2 means the second dimension of feature. Run the above codes, you will get a classification dataset as in Figure 3.

Step 2. Empirical Risk Minimization Let us utilize logistic loss for finding the classifier:

$$\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \log\left(1+\exp(-f(\boldsymbol{x}_i)y_i)\right) = \min_{f\in\mathcal{F}} \frac{1}{100}\sum_{i=1}^{100} \log\left(1+\exp(-f(\boldsymbol{x}_i)y_i)\right)$$

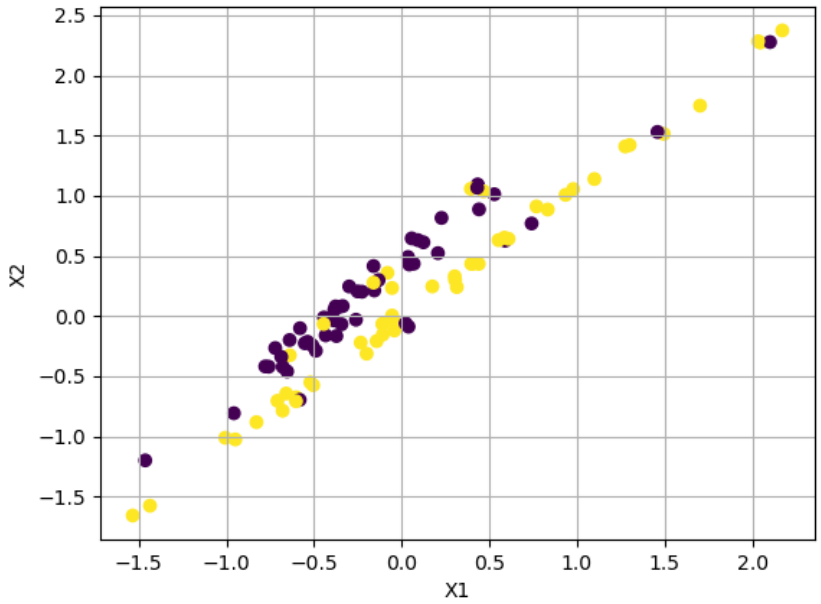

Figure 1: The sampels of two classes.

Suppose we choose a function class $\mathcal{F} = \{f(\boldsymbol{x}_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} : \beta_0, \beta_1, \beta_2 \in \mathbb{R}\}$ where $\boldsymbol{x}_i = (x_{i1}, x_{i2})$. Then we can re-write the **averaged logistic loss** as

$$\min_{\beta_0,\beta_1,\beta_2} L(\beta_0, \beta_1, \beta_2) = \min_{\beta_0,\beta_1,\beta_2} \frac{1}{100} \sum_{i=1}^{100} \log\left(1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) y_i)\right).$$

The next question is **how to minimize $L(\beta_0, \beta_1, \beta_2)$ with respect to β_0, β_1, and β_2**. We take the partial derivative of $L(\beta_0, \beta_1, \beta_2)$ with β_0, β_1, and β_2:

$$\begin{aligned}
\frac{\partial L(\beta_0, \beta_1, \beta_2)}{\partial \beta_0} &= -\frac{1}{100} \sum_{i=1}^{100} \frac{\exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) y_i)}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) y_i)} \cdot y_i \\
\frac{\partial L(\beta_0, \beta_1, \beta_2)}{\partial \beta_1} &= -\frac{1}{100} \sum_{i=1}^{100} \frac{\exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) y_i)}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) y_i)} \cdot y_i x_{i1} \\
\frac{\partial L(\beta_0, \beta_1, \beta_2)}{\partial \beta_2} &= -\frac{1}{100} \sum_{i=1}^{100} \frac{\exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) y_i)}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) y_i)} \cdot y_i x_{i2}
\end{aligned}$$

Therefore, we randomly choose an initial point for $(\beta_0^{(0)}, \beta_1^{(0)}, \beta_2^{(0)}) = (0, 0, 0)$ and step size $\lambda = 0.05$, and do the following updates $t = 1, 2, \ldots, 30000$ (We repeat gradient descent 30000

times):

$$\beta_0^{(t)} = \beta_0^{(t-1)} + \lambda \frac{1}{100} \sum_{i=1}^{100} \frac{\exp(-(\beta_0^{(t-1)} + \beta_1^{(t-1)} x_{i1} + \beta_2^{(t-1)} x_{i2}) y_i)}{1 + \exp(-(\beta_0^{(t-1)} + \beta_1^{(t-1)} x_{i1} + \beta_2^{(t-1)} x_{i2}) y_i)} \cdot y_i$$

$$\beta_1^{(t)} = \beta_1^{(t-1)} + \lambda \frac{1}{100} \sum_{i=1}^{100} \frac{\exp(-(\beta_0^{(t-1)} + \beta_1^{(t-1)} x_{i1} + \beta_2^{(t-1)} x_{i2}) y_i)}{1 + \exp(-(\beta_0^{(t-1)} + \beta_1^{(t-1)} x_{i1} + \beta_2^{(t-1)} x_{i2}) y_i)} \cdot y_i x_{i1}$$

$$\beta_1^{(t)} = \beta_0^{(t-1)} + \lambda \frac{1}{100} \sum_{i=1}^{100} \frac{\exp(-(\beta_0^{(t-1)} + \beta_1^{(t-1)} x_{i1} + \beta_2^{(t-1)} x_{i2}) y_i)}{1 + \exp(-(\beta_0^{(t-1)} + \beta_1^{(t-1)} x_{i1} + \beta_2^{(t-1)} x_{i2}) y_i)} \cdot y_i x_{i2}$$

The above algorithm is summarized as follows:

```
# Gradient Descent
beta_0 = 0. # The intercept term
beta_12 = [0,0] # Coefficient term: beta = (beta_1,beta_2)
lamb = 0.05 # step size 0.1 for gradient descent
for rep in range(30000):
    Temp =((beta_12 * X).sum(axis=1)+beta_0) * y
    Temp_2 = np.exp(-Temp)/(1+np.exp(-Temp)) * y
    beta_0 = beta_0 + lamb * (Temp_2 ).mean(axis=0)
    beta_12[0] = beta_12[0] + lamb * (Temp_2 * X[:,0]).mean(axis=0)
    beta_12[1] = beta_12[1] + lamb * (Temp_2 * X[:,1]).mean(axis=0)
print(beta_0,beta_12) # beta_0 = 1.366 beta_12 = [7.471, -6.757]
```

Listing 2: Logistic Loss Mnimization

The resulting estimated parameters are

$$(\widehat{\beta}_0, \widehat{\beta}_1, \widehat{\beta}_2) = (1.366, 7.471, -6.757).$$

That means that the prediction model $\widehat{f}$ for a $\boldsymbol{x} = (x_1, x_2)$ is

$$\widehat{f}(\boldsymbol{x}) = 1.366 + 7.471 x_1 - 6.757 x_2.$$

The **corresponding decision boundary** characterized by this prediction model $\widehat{f}$ is

$$\widehat{f}(\boldsymbol{x}) = 0 \Longleftrightarrow 1.366 + 7.471 x_1 - 6.757 x_2 = 0.$$

If we plot $1.366 + 7.471 x_1 - 6.757 x_2 = 0$, we get

$$x_2 = \frac{1.366}{6.757} + \frac{7.471}{6.757} x_1.$$

For plotting samples and decision boundary, we can use the following codes:

```
# plot the decision boundary...
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis') # there are two classes
    in this dataset in different colors
```

```
plt.xlabel('X1')
plt.ylabel('X2')
X_new = np.linspace(min(X[:,0]),max(X[:,0]),1000)
YY = 1.366/6.757 +7.471/6.757 * X_new
plt.plot(X_new,YY,label='Decision Boundary')
plt.grid()
plt.legend()
```

Listing 3: Plot Decision Boundary

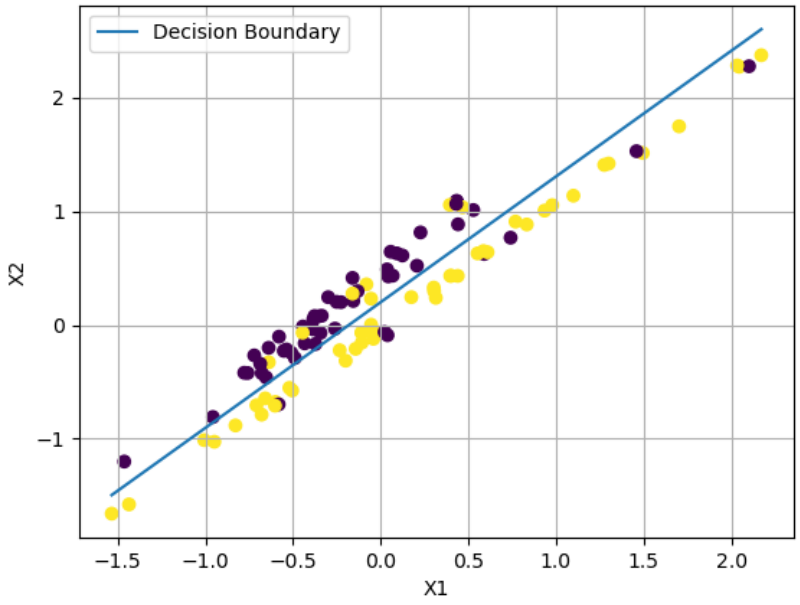

Figure 2: The sampels of two classes and Decision Boundary.

It is worth noting that if we change the function class $\mathcal{F}$, we will get a completely different decision boundary. For example, if we consider

$$\mathcal{F} = \{f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1^2 + \beta_2 x_2\}$$

Using the same logistic loss and dataset, we will have

```
# consider a quadratic function class x_1^2 + x_2
X_tilde = np.array([X[:,0]**2,X[:,1]]).T
beta_0 = 0. # The intercept term
beta_12 = [0,0] # Coefficient term: beta = (beta_1,beta_2)
lamb = 0.05 # step size 0.1 for gradient descent
for rep in range(30000):
    Temp =((beta_12 * X_tilde).sum(axis=1)+beta_0) * y
    Temp_2 = np.exp(-Temp)/(1+np.exp(-Temp)) * y
    beta_0 = beta_0 + lamb * (Temp_2).mean(axis=0)
```

```
    beta_12[0] = beta_12[0] + lamb * (Temp_2 * X_tilde[:,0]).mean(axis=0)
    beta_12[1] = beta_12[1] + lamb * (Temp_2 * X_tilde[:,1]).mean(axis=0)
print(beta_0,beta_12) # beta_0 = -0.2603652125568569 beta_12 = [0.5571,
   -0.1683]
# plot the decision boundary...
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')
plt.xlabel('X1')
plt.ylabel('X2')
X_new = np.linspace(min(X[:,0]),max(X[:,0]),1000)
YY = -0.26038/0.1683 +0.5571/0.1683 * X_new **2
plt.plot(X_new,YY,label='Decision Boundary')
plt.grid()
plt.legend()
```

Listing 4: Plot Decision Boundary

The estimated model is

$$\widehat{f}(\boldsymbol{x}) = -0.2603 + 0.5571x_1^2 - 0.1683x_2.$$

If we set $\widehat{f}(\boldsymbol{x}) = 0$, we will get the decision boundary

$$x_2 = -\frac{0.2603}{0.1683} + \frac{0.5571}{0.1683}x_1^2$$

The corresponding decision boundary is given as follosw:

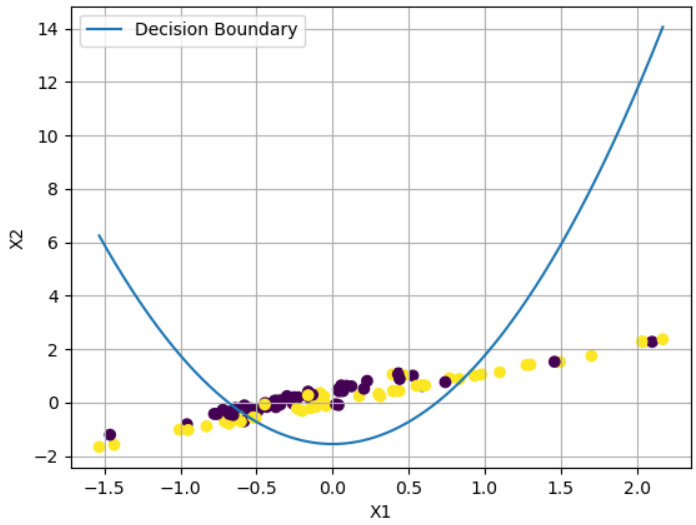

Figure 3: The sampels of two classes and Decision Boundary.