Binary Classification Networks

Machine Learning: From Scratch to Mastery

Yasser Hifny

Faculty of Computers and Artificial Intelligence
Capital University

Overview

- Logistic regression as a probabilistic binary classification model.
- The single-output network and sigmoid activation function.
- The binary cross-entropy learning objective.
- Derivation of the gradients and gradient-descent updates.
- Threshold-based classification and the linear decision boundary.
- Binary classification evaluation: confusion matrix, precision, recall, F1.
- F1-based threshold tuning.
- A complete AND-gate example and Python implementation.
- Assignment: spam-email detection with Keras and the Spambase dataset.

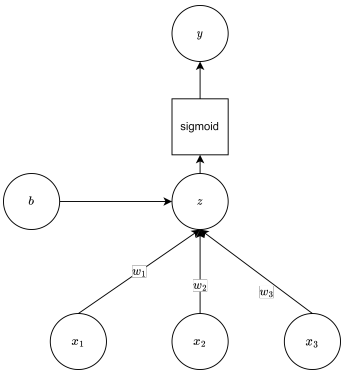
Binary Classification Model

- Logistic regression is used to predict a binary dependent variable from one or more input variables.
- The model has many input variables and exactly one output probability variable.
- Inputs may be float, binary, or integer values.
- The output is a probability bounded between 0 and 1.

Core idea

First compute a linear score z, then transform it into a probability using the sigmoid activation function.

Logistic Regression Network



A binary classification model has one output variable and an optional number of input variables.

Linear Part of the Model

For three input variables, the intermediate variable is

$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + b.$$

The general form is

$$z = \mathbf{w}^T \mathbf{x} + b,$$

where

$$\mathbf{x} \in \mathbb{R}^d, \qquad \mathbf{w} \in \mathbb{R}^d, \qquad b \in \mathbb{R}.$$

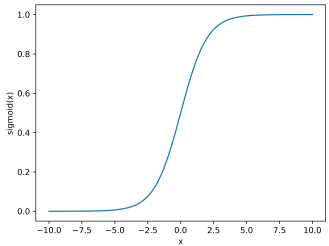
- $\mathbf{w}$ and b are the model parameters.
- They are estimated from the training data.

Sigmoid Activation

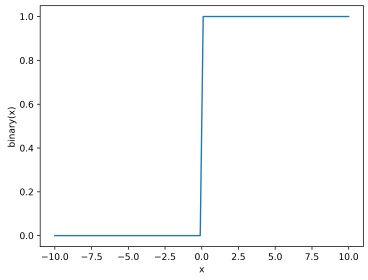
The output probability is

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}.$$

- Maps any real-valued z to $(0, 1)$.
- Provides a continuous and differentiable transformation.
- Makes gradient-based learning possible.



Why Not a Binary Activation Function?



- A binary activation maps its input directly to 0 or 1.
- The function is not continuous.
- It is not differentiable at 0.
- Consequently, it does not provide the smooth gradient needed by the learning procedure developed in this work.

The Learning Problem

Given training data

$$\{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}, \qquad t_n \in \{0, 1\},$$

the objective is to estimate $\mathbf{w}$ and b so that the predicted probability agrees with the target.

$$y^n = \sigma\left(\mathbf{w}^T \mathbf{x}_n + b\right).$$

Optimization view

Define a loss over the training set and minimize it with respect to the model parameters.

Binary Cross-Entropy Loss

For one training example,

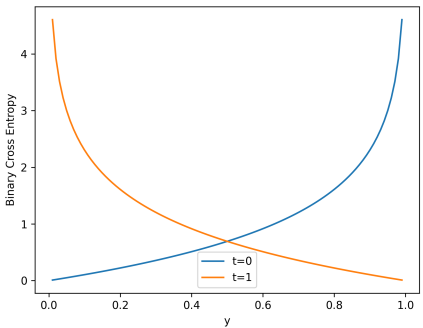
$$E^n(\mathbf{w}, b) = -\left[t^n \log y^n + (1 - t^n) \log(1 - y^n)\right].$$

The average loss over N examples is

$$E(\mathbf{w}, b) = \frac{1}{N} \sum_{n=1}^{N} E^n.$$

- This is the binary cross-entropy (BCE) objective.
- The model learns by minimizing $E(\mathbf{w}, b)$.

Binary Cross-Entropy Objective



The BCE loss increases strongly when the predicted probability of the true class approaches zero.

Gradient of the BCE Loss: Chain Rule

For a weight w_i,

$$\frac{\partial E^n}{\partial w_i} = \frac{\partial E^n}{\partial y^n} \frac{\partial y^n}{\partial z^n} \frac{\partial z^n}{\partial w_i}.$$

The three factors are

$$\begin{aligned}
\frac{\partial E^n}{\partial y^n} &= \frac{y^n - t^n}{y^n(1 - y^n)}, \\
\frac{\partial y^n}{\partial z^n} &= y^n(1 - y^n), \\
\frac{\partial z^n}{\partial w_i} &= x_i^n.
\end{aligned}$$

Simplified Weight and Bias Gradients

Combining the three terms gives

$$\boxed{\frac{\partial E^n}{\partial w_i} = (y^n - t^n)x_i^n}$$

and the bias gradient is

$$\boxed{\frac{\partial E^n}{\partial b} = y^n - t^n}.$$

Important simplification

The sigmoid derivative and the BCE derivative cancel the probability factors, producing a compact error term $(y^n - t^n)$.

Gradient-Descent Updates

For the full training set,

$$w_i^{t+1} = w_i^t - \eta \frac{1}{N} \sum_{n=1}^{N} (y^n - t^n) x_i^n,$$

$$b^{t+1} = b^t - \eta \frac{1}{N} \sum_{n=1}^{N} (y^n - t^n).$$

- η is the learning rate.
- The gradient is averaged over the training examples.
- Mini-batch stochastic gradient descent can be used for large datasets.

Deriving the BCE Loss Gradient

For binary classification, the gradient of the binary cross-entropy (BCE) loss with respect to weight w_i is obtained using the chain rule:

$$\frac{\partial E^n(\mathbf{w}, b)}{\partial w_i} = \frac{\partial E^n}{\partial y^n} \frac{\partial y^n}{\partial z^n} \frac{\partial z^n}{\partial w_i}. \tag{1}$$

The BCE loss for training example n is

$$E^n = -t^n \log y^n - (1 - t^n) \log(1 - y^n).$$

Its derivative with respect to the predicted probability y^n is

$$\begin{aligned} \frac{\partial E^n}{\partial y^n} &= -\frac{t^n}{y^n} + \frac{1 - t^n}{1 - y^n} \\ &= \frac{-t^n(1 - y^n) + y^n(1 - t^n)}{y^n(1 - y^n)} \\ &= \boxed{\frac{y^n - t^n}{y^n(1 - y^n)}}. \end{aligned} \tag{2}$$

Here, $t^n \in \{0, 1\}$ is the target and y^n is the predicted probability.

Gradients of the Sigmoid and Linear Output

The output is obtained from the sigmoid activation:

$$y^n = \sigma(z^n) = \frac{1}{1 + e^{-z^n}}.$$

Differentiating with respect to the logit z^n gives

$$\begin{aligned} \frac{\partial y^n}{\partial z^n} &= \frac{e^{-z^n}}{(1 + e^{-z^n})^2} \\ &= \frac{1}{1 + e^{-z^n}} \frac{e^{-z^n}}{1 + e^{-z^n}} \\ &= y^n(1 - y^n). \end{aligned} \tag{3}$$

For the linear logit

$$z^n = \mathbf{w}^\mathsf{T}\mathbf{x}^n + b,$$

the derivatives with respect to the weight and bias are

$$\frac{\partial z^n}{\partial w_i} = x_i^n, \qquad \frac{\partial z^n}{\partial b} = 1.$$

Substituting these results into the chain rule yields

$$\frac{\partial E^n}{\partial w_i} = \frac{y^n - t^n}{y^n(1 - y^n)} y^n(1 - y^n) x_i^n$$

Gradient Descent Updates

For a dataset containing N training examples, the average gradients are

$$\frac{\partial E}{\partial w_i} = \frac{1}{N}\sum_{n=1}^{N}(y^n - t^n)x_i^n, \qquad \frac{\partial E}{\partial b} = \frac{1}{N}\sum_{n=1}^{N}(y^n - t^n).$$

Applying gradient descent, the parameters are updated as

$$w_i^{(t+1)} = w_i^{(t)} - \eta\frac{1}{N}\sum_{n=1}^{N}(y^n - t^n)x_i^n, \tag{5}$$

$$b^{(t+1)} = b^{(t)} - \eta\frac{1}{N}\sum_{n=1}^{N}(y^n - t^n).$$

- η is the learning rate.
- $y^n - t^n$ is the prediction error.
- The weight update scales the error by each input feature x_i^n.
- The bias update uses the average prediction error directly.

From Probability to Class Label

The learned model produces $y \in [0, 1]$, not a discrete class label.

$$\hat{y} = \begin{cases} 1, & y \geq 0.5, \\ 0, & y < 0.5. \end{cases}$$

- The threshold converts the probability into a binary decision.
- We use 0.5 as the default decision threshold.
- The threshold can later be tuned according to evaluation objectives.

Linear Decision Boundary

At the decision boundary,

$$P(y = 1 \mid \mathbf{x}) = P(y = 0 \mid \mathbf{x}) = \frac{1}{2}.$$

For the sigmoid this implies

$$z = \mathbf{w}^T\mathbf{x} + b = 0.$$

For two-dimensional data,

$$w_1 x_1 + w_2 x_2 + b = 0,$$

so

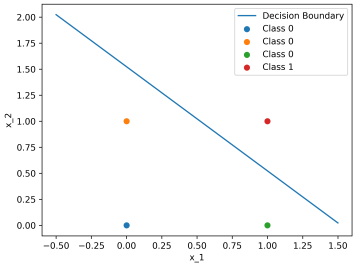
$$x_2 = -\frac{b}{w_2} - \frac{w_1}{w_2} x_1.$$

Consequence

Logistic regression separates the two classes with a linear decision boundary in the input space.

AND-Gate Decision Boundary

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1



The learned linear boundary separates the positive AND-gate example from the three negative examples.

Binary Classification Evaluation

The confusion matrix is the starting point for binary classification metrics.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- TP and TN are correct predictions.
- FP and FN are the two types of classification error.

Precision, Recall, and F1-Score

$$\text{Precision} = \frac{TP}{TP + FP},$$

$$\text{Recall} = \frac{TP}{TP + FN},$$

$$F1 = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

- Precision measures the accuracy of positive predictions.
- Recall measures the ability to identify actual positives.
- F1 is the harmonic mean of precision and recall.

F1 Curve and Threshold Tuning

- A classifier produces a probability score $p \in [0, 1]$.
- Changing the decision threshold changes precision and recall.
- Increasing the threshold generally increases precision but can reduce recall.
- Decreasing the threshold generally increases recall but can reduce precision.
- Plotting F1 against the threshold helps identify a threshold that balances precision and recall for the task.

Generating an F1 Curve

1. Split the data into training, development, and test sets.
2. Train the classifier on the training set.
3. Obtain probability scores on the development set.
4. Define thresholds from 0 to 1.
5. Compute precision, recall, and F1 at each threshold.
6. Plot F1 against the threshold.
7. Select the threshold corresponding to the peak F1 in the development experiment.

Example

The supplied table reports its maximum F1 of 0.81 at threshold 0.4.

Example: F1 Versus Threshold

Threshold	Precision	Recall	F1
0.1	0.65	0.95	0.77
0.2	0.70	0.90	0.79
0.3	0.75	0.85	0.80
0.4	0.80	0.82	0.81
0.5	0.82	0.78	0.80
0.6	0.85	0.75	0.80
0.7	0.88	0.70	0.78
0.8	0.90	0.65	0.76
0.9	0.92	0.60	0.73

Observation from the data

The reported peak is $F1 = 0.81$ at threshold 0.4, which is the threshold selected in this example.

Python: Plotting the F1 Curve

```
thresholds = np.linspace(0, 1, 50)
f1_scores = []

for threshold in thresholds:
    predictions = (probabilities >= threshold).astype(int)
    f1 = f1_score(true_labels, predictions)
    f1_scores.append(f1)

plt.plot(thresholds, f1_scores, marker='o', label='F1 Score')
plt.xlabel('Threshold')
plt.ylabel('F1 Score')
plt.legend()
plt.grid()
plt.show()
```

Complete Example: AND Problem

We use the two-dimensional AND gate:

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

- Add a bias input to the design matrix.
- Initialize the parameters to zero.
- Apply gradient descent using BCE.
- Plot the resulting linear decision boundary.

AND Example: Core Python Implementation

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def predict(X, w):
    return sigmoid(np.dot(X, w))

def cost_function(X, t, w):
    N = len(t)
    y = predict(X, w)
    return -1/N * (np.dot(t.T, np.log(y)) +
                   np.dot((1-t).T, np.log(1-y)))

def gradient_descent(X, t, w, alpha, iterations):
    m = len(t)
    E_history = []
    for i in range(iterations):
        y = predict(X, w)
        w = w - alpha * (1/m) * np.dot(X.T, (y-t))
        E_history.append(cost_function(X, t, w))
    return w, E_history
```

AND Example: Data and Training

```
X = np.array([[0,0], [0,1], [1,0], [1,1]])
t = np.array([[0], [0], [0], [1]])

m = len(t)
X = np.hstack((np.ones((m, 1)), X))

n = X.shape[1]
w = np.zeros((n, 1))
alpha = 0.01
iterations = 100000

w_final, E_history = gradient_descent(
    X, t, w, alpha, iterations)

print(predict(X, w_final))
```

AND Example: Plotting the Boundary

```
x_1 = np.linspace(-0.5, 1.5)
x_2 = -(w_final[0] + w_final[1]*x_1) / w_final[2]

plt.plot(x_1, x_2, label='Decision Boundary')
plt.scatter(0, 0, label='Class 0')
plt.scatter(0, 1, label='Class 0')
plt.scatter(1, 0, label='Class 0')
plt.scatter(1, 1, label='Class 1')
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()
plt.savefig('decision_boundary_and.png')
```

What the Example Demonstrates

- A sigmoid converts the linear score into a probability.
- BCE provides the training objective.
- Gradient descent estimates the model parameters.
- The resulting classifier produces a linear decision boundary.
- We notes that repeated runs of the script lead to the same result because the logistic-regression solution is unique.

Summary

- Binary classification uses a single output probability variable.
- The model computes $z = \mathbf{w}^T\mathbf{x} + b$ followed by the sigmoid function.
- Binary cross-entropy measures the disagreement between targets and predicted probabilities.
- The BCE–sigmoid combination gives compact gradients for gradient descent.
- A threshold converts probabilities into binary decisions.
- The resulting decision boundary is linear in the input space.
- Precision, recall, F1, and ROC-AUC provide complementary evaluation views.
- Threshold tuning can be used to study the precision–recall trade-off.

Assignment

Spam Email Detection

Implement binary classification for spam-email detection using Python from scratch, using the Keras deep learning library, and Google Colab.

- Use the Spambase dataset.
- Train a binary classifier.
- Evaluate the classifier using appropriate binary-classification metrics.
- Investigate the effect of the decision threshold on performance.

https://archive.ics.uci.edu/ml/datasets/spambase

Thank You

Questions?