Logistic Regression I

COSC 410: Applied Machine Learning
Fall 2025
Prof. Forrest Davis

September 30, 2025

Warm-up

1. Fall is here! Discuss with your neighbor the best type of apple.
2. Complete the following model instance of PyTorch by adding a single parameter that will be learned and applied (as a scalar) to the input to forward to yield a prediction.

```
    class Model(torch.nn.Module):
        def __init__(self):

        def forward(self, X):


```

Listing 1: Partial PyTorch class

Logistics

- Midterm I Thursday Oct 2
  - Possible topics included on the website
  - Please write to me if you have accommodations
- Lab Wednesday will be partial a conversation with Richard Maxwell on Robotics in Theater and then we will have time for some final review
- Extra time for Codelet 3

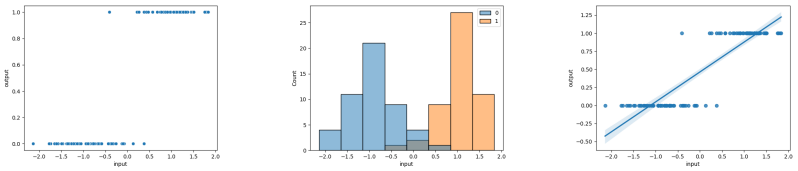
Learning Objectives

- Describe the move from linear regression to logistic regression
- Understand the logistic function
- Apply log loss
- Set up multi-class classification from the system of equations, to the relevant activation function, to the loss function

Summary: We return to classification via changing the treatment of parameters and output from linear regression. Along the way we cover logistic regression, softmax, and cross-entropy loss, which will become increasingly important as we shift to neural networks.

From Modeling Regression to Classification

WE WILL TRANSFORM LINEAR REGRESSION, a model-based regression model, to classification.



(a) Raw Data (b) Distribution of Labels (c) Fitted Line

Figure 1: Sample classification data in a regression setting

Given a line how do we use if for classification? Let's focus on data with one input feature and one output feature. Figure 1 shows an example and representations of it. We can visually see that around 0.0 is a good decision point. However, regression finds the line that is closest to all the points and treats the output as continuous. We need to map from our continuous output space (from our line) to a discrete label.

Which Side Are You On?

- What distinguishes a sequence of points as part of a line?
- Well one way to think about it is:
  - Say we toss on a vector perpendicular to a single point on a line

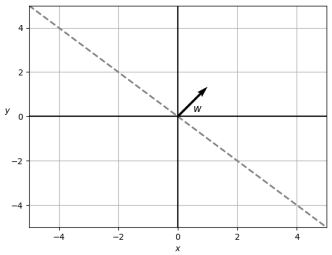


Figure 2: Normal vector $\mathbf{w}$ for the dotted line.

  - All other points that make up the line are perpendicular to this same vector

- Let's call our perpendicular vector, **w**
- This **w** is called the **normal vector** of the line (or plane in higher dimensions)
- In other words a line (or plane) is a surface with the same *normal vector* over all subsets of its points

Question

We gain a nice thing by considering the normal vector of our decision boundary. Namely, we can determine whether a point occurs above or below the line. How do we do this? (Hint: Consider Figure 3 and the mathematical operations we have been using for vectors).

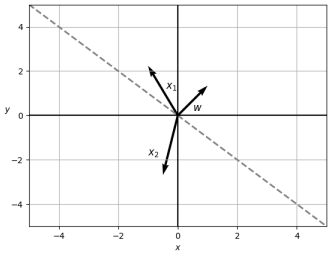


Figure 3: Sample vectors for points x_1 and x_2 above and below the dashed line.

Lack of Confidence in the Center

Fitting a line that satisfies the criteria above (that is, that groups points on different sides of the line), yields Figure 4.

Question

As we approach the line in Figure 4, are we more confident or less confident that those points are correctly labeled? Does our model capture this intuition?

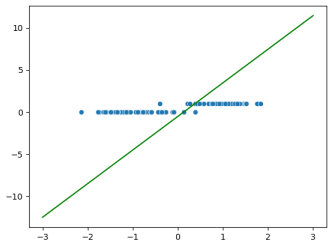


Figure 4: A line fit to group labels onto other sides of the line.

- **Assumption**: Our data is linearly sepearable
- **Classification rule**: For a test input $\mathbf{x}_{\text{test}}$, assign a label of 1, if it's predicted value is greater than or equal to 0.5, otherwise assign the label 0
- **Formal definition**: Denote our test point as $\mathbf{x}_{\text{test}}$, our parameters as **w** (including the bias term), and our target output y (whose values are 0 or 1). Additionally, let $\sigma(t) = \frac{1}{1+e^{-t}}$. Then our classifier h applied to our test point, $h(x_{\text{test}})$ is

$$1 \quad \text{if} \quad \sigma(\mathbf{w} \cdot \mathbf{x}_{\text{test}}) \geq 0.5 \tag{1}$$

$$0 \quad \text{if} \quad \sigma(\mathbf{w} \cdot \mathbf{x}_{\text{test}}) < 0.5 \tag{2}$$

$$\tag{3}$$

$\sigma(t)$ is called the logistic function.

Question

1. What happens to the logistic function as t becomes a larger and larger positive value?
2. What happens to the logistic function as t becomes a larger and larger negative value?
3. How does the proposed modification to our classification problem address the issues with the points near the threshold (consider Figure 5).

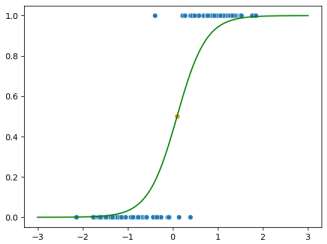


Figure 5: A model fit with to satisfy the logistic regression criteria.

Learning Classification: Log Loss

LOGISTIC REGRESSION USES A LOSS FUNCTION called the **log loss** or **logistic loss**.

$$\min_{\mathbf{w}} -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\hat{p}^{(i)}) + (1-y^{(i)})\log(1-\hat{p}^{(i)})] \quad \text{where} \quad \hat{p}^{(i)} = \sigma(\mathbf{w}\cdot\mathbf{x}^{(i)})$$

Practice Problems

What is the log loss associated with the following classifier:

sample	$\hat{p}$	y
1	0.8	0
2	0.6	1
3	0.2	0
4	0.4	1

Multi-Class Classification Formulation

WE WILL NOW MODIFY THIS APPROACH to work in multi-class classification settings. **Multi-class classification** is an approach to modeling classification with more than a binary output. That is, we might be interested in modeling:

- Whether a cat or a dog or both occurs in an image
- What word is likely to come next
- Whether an email is Spam, Marketing, or Important
- ...

System of Equations

$$o_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x + 3 + w_{14}x_4 + b_1 \tag{4}$$
$$o_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x + 3 + w_{24}x_4 + b_2 \tag{5}$$
$$o_3 = w_{31}x_1 + w_{32}x_2 + w_{33}x + 3 + w_{34}x_4 + b_3 \tag{6}$$

$$\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Where, we have:

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{pmatrix}$$

Multi-Class Classification Output

GIVEN OUR VECTOR OF OUTPUTS $\mathbf{o}$, we could apply the logistic function to each output independently. This would be encoding an assumption that each of the output labels for each input are independent:

label	raw output	σ(output)
cat	12	0.9
dog	1	0.5
lizard	-100	0.01

A New Activation Function

There are many problems where we **do not** want to treat the multi-class classification output as independent. Consider the choice of the next word in a sentence, for example. Only one word can occur, and we expect some organization among the output (e.g., if the word 'eating' is likely in the context *Bob is* than probably 'hungry' is also likely and not something like 'cat').

We need a new function to accomplish this. Note, we call these functions which apply to the raw output of a model **activation functions**. A common one is **softmax**, we applies element-wise.[1]

$$\text{Softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{7}$$

[1] In the softmax equation, z_j is the jth element of the output vector $\mathbf{z}$ which has K dimensions (e.g., $\mathbf{z} \in \mathbb{R}^K$).

Practice Problems

Apply softmax to the following output from a model for one sample:

label	raw output	Softmax(output)
cat	2	
dog	1	
lizard	-1	

A New Loss Function

Finally, we need a new loss function to learn in the multi-class classification domain. Our aim, in multi-class classification is to maximize the probability of our data (just like we discussed recently with regression).

$$P(\mathbf{Y}|\mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}).$$

Notice that our typically dataset looks something like below:

input	output
$\mathbf{x}^{(1)}$	cat
$\mathbf{x}^{(2)}$	dog
$\mathbf{x}^{(3)}$	cat
$\mathbf{x}^{(4)}$	kangaroo

When we take in a sample like $\mathbf{x}^{(1)}$, we don't actually have a probability that the label is cat, for example. We just have a sample. To get around this, we employ a trick, called **one hot encoding**. That is, we create a vector representation of our multi-class output with an index for each possible output. We put a one for the observed label and a zero everywhere else. For example,

label	one-hot
cat	[1, 0, 0]
dog	[0, 1, 0]
kangaroo	[0, 0, 1]

We are allowed to use the factorization since we assume that each label is drawn independently from its respective distribution $P(\mathbf{y}|\mathbf{x}^{(i)})$. Since maximizing the product of terms is awkward,

We make this an optimization problem (in particular, a minimization problem) using the negative log-likelihood.

$$-\log P(\mathbf{Y}|\mathbf{X}) = -\sum_{i=1}^{m} \log P(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) = \sum_{i=1}^{m} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}),$$

where for any pair of label $\mathbf{y}$ and model prediction $\hat{\mathbf{y}}$ over K classes, the loss function l is

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{K} y_j \log \hat{y}_j.$$

All together, we typically take the average loss over our data to find the best parameters.

$$\min_{\mathbf{W}} -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} y_j^{(i)} \log \hat{y}_j^{(i)} \quad \text{where} \quad \hat{\mathbf{y}}^{(i)} = \mathbf{W}\mathbf{x}^{(i)}$$

Before Next Class

- Study for the Midterm