# Appendix B: Logistic Loss

## Contents



**DSCI 572**
Supervised Learning II

## Imports

```
import os
import sys

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

sys.path.append(os.path.join(os.path.abspath(".."), "code"))

from sklearn.preprocessing import StandardScaler
import plotly.express as px

%matplotlib inline

DATA_DIR = DATA_DIR = os.path.join(os.path.abspath(".."), "data/")

%config InlineBackend.figure_formats = ['svg']
plt.rcParams.update({'font.size': 12, 'axes.labelweight': 'bold', 'figure.figs
```

# Logistic Regression Refresher

Logistic Regression is a classification model where we calculate the probability of an observation belonging to a class as:

$$z = w^Tx$$

$$\hat{y} = \frac{1}{(1+\exp(-z))}$$

And then assign that observation to a class based on some threshold (usually 0.5):

$$\text{Class }\hat{y} = \begin{cases} 0, & \hat{y} \le 0.5 \\ 1, & \hat{y} > 0.5 \end{cases}$$

# Motivating the Loss Function

- In Lecture 2 we focussed on the mean squared error as a loss function for optimizing linear regression:

$$f(w) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y_i))^2$$

- That won't work for logistic regression classification problems because it ends up being "non-convex" (which basically means there are multiple minima)
- Instead we use the following loss function:

$$f(w) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log\left(\frac{1}{1+\exp(-w^Tx_i)}\right) + (1-y_i)\log\left(1 - \frac{1}{1+\exp(-w^Tx_i)}\right)$$

- This function is called the "log loss" or "binary cross entropy"
- I want to visually show you the differences in these two functions, and then we'll discuss why that loss functions works
- Recall the Pokemon dataset from Lecture 2, I'm going to load that in again (and standardize the data while I'm at it):

```
df = pd.read_csv(DATA_DIR + "pokemon.csv", usecols=['name', 'defense', 'attack
x = StandardScaler().fit_transform(df.drop(columns=["name", "legendary"]))
X = np.hstack((np.ones((len(x), 1)), x))
y = df['legendary'].to_numpy()
df.head()
```

| | name | attack | defense | speed | capture_rt | legendary |
|---|---|---|---|---|---|---|
| **0** | Bulbasaur | 49 | 49 | 45 | 45 | 0 |
| **1** | Ivysaur | 62 | 63 | 60 | 45 | 0 |
| **2** | Venusaur | 100 | 123 | 80 | 45 | 0 |
| **3** | Charmander | 52 | 43 | 65 | 45 | 0 |
| **4** | Charmeleon | 64 | 58 | 80 | 45 | 0 |

```
X
```

```
array([[ 1.        , -0.89790944, -0.78077335, -0.73258737, -0.70940526],
       [ 1.        , -0.49341318, -0.32548801, -0.21387459, -0.70940526],
       [ 1.        ,  0.6889605 ,  1.62573488,  0.47774246, -0.70940526],
       ...,
       [ 1.        ,  0.72007559, -0.65069183, -0.80174908, -1.10265558],
       [ 1.        ,  0.90676617,  0.91028648,  0.44316161, -1.25995571],
       [ 1.        ,  0.53338501,  1.36557183, -0.04097032, -1.25995571]])
```

- The goal here is to use the features (but not “name”, that’s just there for illustration purposes) to predict the target “legendary” (which takes values of 0/No and 1/Yes).
- So we have 4 features meaning that our logistic regression model will have 5 parameters that need to be estimated (4 feature coefficients and 1 intercept)
- At this point let’s define our loss functions:

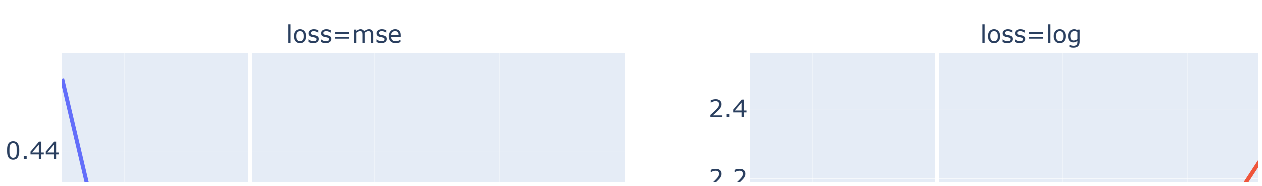
```
def sigmoid(w, x):
    """Sigmoid function (i.e., logistic regression predictions)."""
    return 1 / (1 + np.exp(-x @ w))


def mse(w, x, y):
    """Mean squared error."""
    return np.mean((sigmoid(w, x) - y) ** 2)


def logistic_loss(w, x, y):
    """Logistic loss."""
    return -np.mean(y * np.log(sigmoid(w, x)) + (1 - y) * np.log(1 - sigmoid(w
```

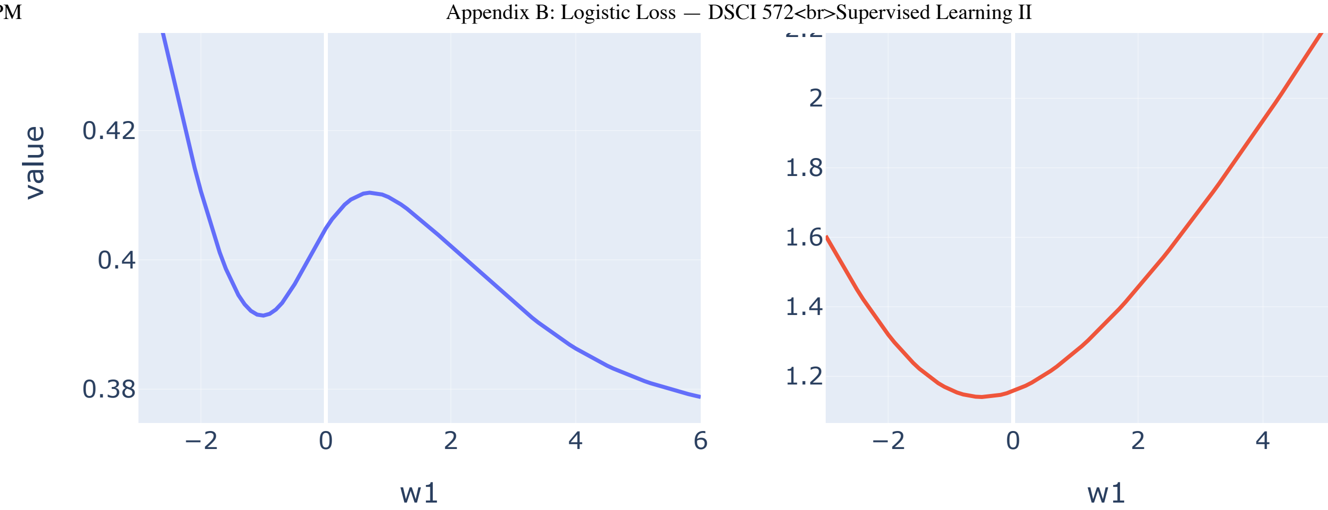
- For a moment, let's assume a value for all the parameters execpt for $w_1$
- We will then calculate the mean squared error for different values of $w_1$ as in the code below

```
w1_arr = np.arange(-3, 6.1, 0.1)
losses = pd.DataFrame({"w1": w1_arr,
                       "mse": [mse([0.5, w1, -0.5, 0.5, -2], X, y) for w1 in w
                       "log": [logistic_loss([0.5, w1, -0.5, 0.5, -2], X, y) f
losses.head()
```

| | w1 | mse | log |
|---|---|---|---|
| **0** | -3.0 | 0.451184 | 1.604272 |
| **1** | -2.9 | 0.446996 | 1.571701 |
| **2** | -2.8 | 0.442773 | 1.539928 |
| **3** | -2.7 | 0.438537 | 1.508997 |
| **4** | -2.6 | 0.434309 | 1.478955 |

```
fig = px.line(losses.melt(id_vars="w1", var_name="loss"), x="w1", y="value", c
fig.update_yaxes(matches=None, showticklabels=True, col=2)
fig.update_xaxes(matches=None, showticklabels=True, col=2)
fig.update_layout(width=800, height=400)
```






- This is a pretty simple dataset but you can already see the "non-convexity" of the MSE loss function.
- If you want a more mathematical description of the logistic loss function, check out [Chapter 3 of Neural Networks and Deep Learning by Michael Nielsen] or [this Youtube video by Andrew Ng].

# Breaking Down the Log Loss Function

- So we saw the log loss before:

$$f(w) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log\left(\frac{1}{1+\exp(-w^T x_i)}\right) + (1-y_i)\log\left(1 - \frac{1}{1+\exp(-w^T x_i)}\right)$$

- It looks complicated but it's actually quite simple. Let's break it down.
- Recall that we have a binary classification task here so $y_i$ can only be 0 or 1.

## When `y = 1`

- When $y_i = 1$ we are left with:

$$f(w) = -\frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{1}{1+\exp(-w^T x_i)}\right)$$

- That looks fine!

- With $y_i = 1$, if $\hat{y}_i = \frac{1}{1+\exp(-w^T x_i)}$ is also close to 1 we want the loss to be small, if it is close to 0 we want the loss to be large, that's where the `log()` comes in:

```
y = 1
y_hat_small = 0.05
y_hat_large = 0.95
```

```
-np.log(y_hat_small)
```

```
np.float64(2.995732273553991)
```

```
-np.log(y_hat_large)
```

```
np.float64(0.05129329438755058)
```

## When `y = 0`

- When $y_i = 1$ we are left with:

$$f(w) = -\frac{1}{n}\sum_{i=1}^{n}\log\left(1 - \frac{1}{1+\exp(-w^T x_i)}\right)$$

- With $y_i = 0$, if $\hat{y}_i = \frac{1}{1+\exp(-w^T x_i)}$ is also close to 0 we want the loss to be small, if it is close to 1 we want the loss to be large, that's where the `log()` comes in:

```
y = 0
y_hat_small = 0.05
y_hat_large = 0.95
```

```
-np.log(1 - y_hat_small)
```

```
np.float64(0.05129329438755058)
```

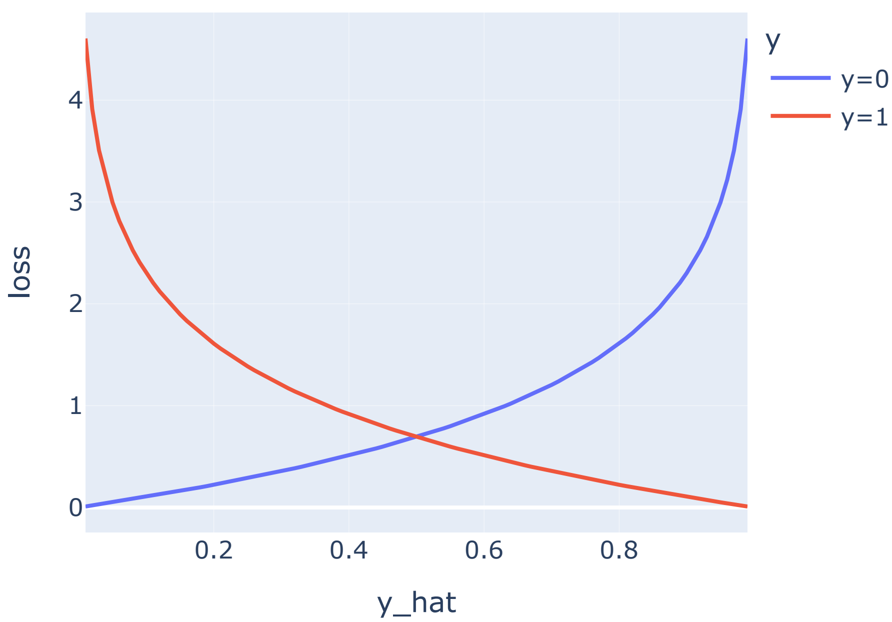
```
-np.log(1 - y_hat_large)
```

```
np.float64(2.995732273553991)
```

# Plot Log Loss

- We know that our predictions from logistic regression $\hat{y}$ are limited between 0 and 1 thanks to the sigmoid function
- So let's plot the losses because it's interesting to see how the worse our predictions are, the worse the loss is (i.e., if $y = 1$ and our model predicts $\hat{y} = 0.05$, the penalty is exponentially bigger than if the prediction was $\hat{y} = 0.90$)

```
y_hat = np.arange(0.01, 1.00, 0.01)
log_loss = pd.DataFrame({"y_hat": y_hat,
                         "y=0": -np.log(1 - y_hat),
                         "y=1": -np.log(y_hat)}).melt(id_vars="y_hat", var_nam
fig = px.line(log_loss, x="y_hat", y="loss", color="y")
fig.update_layout(width=500, height=400)
```



# Log Loss Gradient

- In Lecture 2 we used the gradient of the log loss to implement gradient descent
- Here's the log loss and it's gradient:

$$f(w) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log\left(\frac{1}{1+\exp(-w^T x_i)}\right) + (1-y_i)\log\left(1 - \frac{1}{1+\exp(-w^T x_i)}\right)$$

$$\frac{\partial f(w)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n} x_i \left(\frac{1}{1+\exp(-w^T x_i)} - y_i)\right)$$

- Let's derive that now.
- We'll denote:

$$z = -w^T x_i$$

$$\sigma(z) = \frac{1}{1+\exp(z)}$$

- Such that:

$$f(w) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log\sigma(z) + (1-y_i)\log(1-\sigma(z))$$

- Okay let's do it:

$$\begin{aligned}
\frac{\partial f(w)}{\partial w} &= -\frac{1}{n}\sum_{i=1}^{n} y_i \times \frac{1}{\sigma(z)} \times \frac{\partial \sigma(z)}{\partial w} + (1-y_i) \times \frac{1}{1-\sigma(z)} \times -\frac{\partial \sigma(z)}{\partial w} \\
&= -\frac{1}{n}\sum_{i=1}^{n} \left(\frac{y_i}{\sigma(z)} - \frac{1-y_i}{1-\sigma(z)}\right)\frac{\partial \sigma(z)}{\partial w} \\
&= \frac{1}{n}\sum_{i=1}^{n} \frac{\sigma(z)-y_i}{\sigma(z)(1-\sigma(z))}\frac{\partial \sigma(z)}{\partial w}
\end{aligned}$$

- Now we just need to work out $\frac{\partial \sigma(z)}{\partial w}$, I'll mostly skip this part but there's an intuitive derivation here, it's just about using the chain rule:

$$\begin{aligned}\frac{\partial \sigma(z)}{\partial w} &= \frac{\partial \sigma(z)}{\partial z} \times \frac{\partial z}{\partial w} \\ &= \sigma(z)(1-\sigma(z))x_i\end{aligned}$$

- So finally:

$$\begin{aligned}\frac{\partial f(w)}{\partial w} &= \frac{1}{n}\sum_{i=1}^{n} \frac{\sigma(z)-y_i}{\sigma(z)(1-\sigma(z))} \times \sigma(z)(1-\sigma(z))x_i \\ &= \frac{1}{n}\sum_{i=1}^{n} x_i(\sigma(z)-y_i) \\ &= \frac{1}{n}\sum_{i=1}^{n} x_i\left(\frac{1}{1+\exp(-w^T x_i)} - y_i\right)\end{aligned}$$

```
1/(1+np.exp(-3))
```

```
np.float64(0.9525741268224334)
```

```
1/(1+np.exp(3))
```

```
np.float64(0.04742587317756678)
```

```
-np.log(0.001)
```

```
np.float64(6.907755278982137)
```