

# INTRODUCTION TO MACHINE LEARNING EXERCISE 3

Bauhaus-  
Universität  
Weimar

**Teacher:**

Benno Stein

**Group:**

Group 2

**Submitted by:**

Cesar Fernando Gamba Tiusaba

Abhishek Dilip Patil

Asmae Mehdizadeh

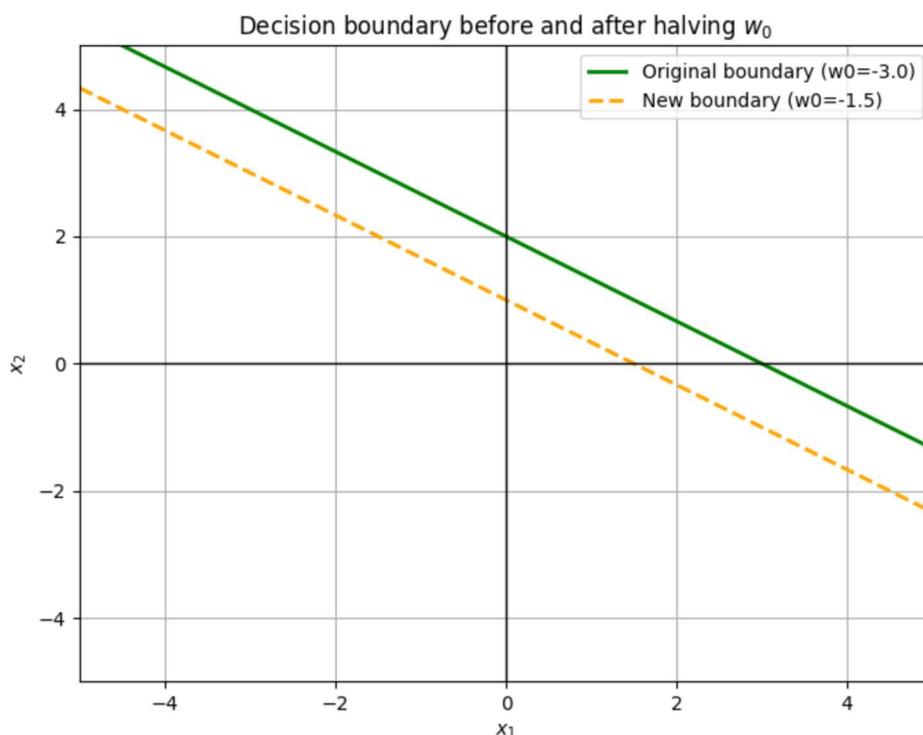
**Exercise 1: Linear Models (0.5+1+1+1+0.5=4 Points)****(a) Name these concepts:  $I(c, y(x))$ ,  $L(w)$ ,  $L(w)$ ,  $w$ ,  $w$**  **$I(c, y(x))$ :** Pointwise/per-example loss (error for a single data point) **$L(w)$ :** Global loss (sum of all pointwise losses) **$L(w)$ :** (global) objective function or cost function or error function **$w$ :** Weight vector (model parameters), entire parameter vector (the actual hypothesis) **$w$  or  $w_{ML}$ :** the direction vector excluding  $w_0$ , Learned / optimal weight vector that minimizes the loss**(b) How would the figure below change if  $w_0$  is halved?**

The decision boundary is the hyperplane defined by

$$w^T x + w_0 = 0.$$

The signed distance from the origin to that hyperplane along the normal  $w$  is
 $d = -\frac{w_0}{\|w\|}$ . If you halve  $w_0$  (replace  $w_0$  by  $\frac{w_0}{2}$ ), the distance becomes  $-\frac{w_0}{2\|w\|}$ .

the boundary moves toward the origin by a factor of 2 along the normal direction (i.e. it shifts parallel to itself halfway toward the origin). The orientation (slope) of the green line does not change because  $w$  is unchanged — only the offset moves.



**(c) What is the difference (if any) between decision boundaries for linear and logistic regression?**

The decision boundaries of linear regression used as a classifier and logistic regression are the same.

Both methods produce linear decision boundaries of the form

$$w^T x = 0.$$

They differ only in how the parameters are learned

- Linear regression minimizes squared error
- Logistic regression minimizes logistic loss / cross-entropy

But the separating hyperplane is still linear in both.

**d) The lecture notes slides state that a key difference between ridge ( $R||w^*||_2^2$ ) and lasso ( $R||w^*||_1$ ) regression is that, with lasso regression, parameters can be reduced to zero. Explain why.**

Lasso regression uses the L1-norm  $R||w||_1(w) = \sum i|w_i|$  whose contour lines have sharp corners on the coordinate axes. When minimizing  $L(w) + \lambda R_{||w||_1}(w)$ , the optimum often lies on one of these corners, which correspond to parameter values of exactly zero. Therefore, lasso can eliminate features by driving some weights to zero.

Ridge regression uses the L2-norm  $R||w||_2^2(w) = \sum i w_i^2$ , whose contour lines are smooth ellipses. Because this penalty has no corners, its solution rarely touches the axes, so ridge shrinks all parameters uniformly but typically does not set any of them exactly to zero.

**(e) Why can the gradient descent method not be applied for  $L0/1(w)$ ?**

Gradient descent requires the objective function to be differentiable, because its update rule uses the gradient  $\nabla L(w)$ . It applies to differentiable loss functions only. However, the 0/1-loss is not differentiable. The loss is piecewise constant and changes value only when the sign of  $w^T x$  changes, causing discontinuous jumps. Because of these jumps, the gradient does not exist, and the function is flat elsewhere, giving no direction of steepest descent. Therefore, gradient descent cannot be used to minimize .

**Exercise 2 : Pointwise Loss Functions (2+1=3 Points)** In the lecture notes, slide ML:III-63 on loss computation for logistic regression in detail, the rightmost plot “Loss over hyperplace distance” shows the pointwise logistic and 0/1 loss for a logistic regression model for  $\text{I}\sigma(1, y(x))$ , that means, for examples with  $c = 1$ . In this exercise you will investigate the case of examples with  $c = 0$ . (a) Show that  $\text{I}\sigma(0, y(x)) = \log(1 + e^{w^T x})$ . Hint:  $\sigma(-a) = 1 - \sigma(a)$ .

(a) Show that  $l_\sigma(0, y(x)) = \log(1 + e^{w^T x})$

The **logistic loss** for a single example is:

$$l_\sigma(c, y(x)) = -c \log y(x) - (1 - c) \log(1 - y(x))$$

where

$$y(x) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

and  $c \in \{0,1\}$ .

Insert  $c = 0$

$$l_\sigma(0, y(x)) = -0 \cdot \log y(x) - (1 - 0) \log(1 - y(x))$$

So:

$$l_\sigma(0, y(x)) = -\log(1 - y(x))$$

From the hint and the logistic function:

$$\sigma(-a) = 1 - \sigma(a)$$

Thus:

$$1 - y(x) = 1 - \sigma(w^T x) = \sigma(-w^T x)$$

So:

$$l_\sigma(0, y(x)) = -\log(\sigma(-w^T x))$$

Rewrite using definition of  $\sigma$

$$\sigma(-a) = \frac{1}{1 + e^a}$$

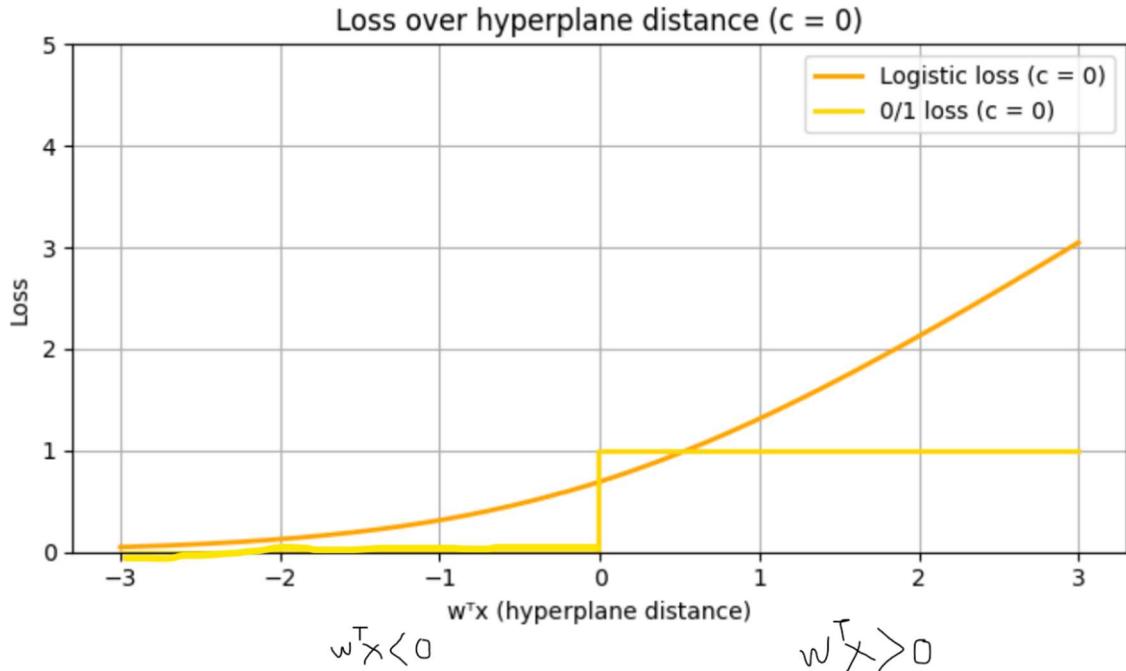
Thus:

$$\begin{aligned} -\log(\sigma(-w^T x)) &= -\log\left(\frac{1}{1 + e^{w^T x}}\right) \\ &= \log(1 + e^{w^T x}) \end{aligned}$$

Final result

$$l_\sigma(0, y(x)) = \log(1 + e^{w^T x})$$

(b) Draw the plot “Loss over hyperplane distance” for examples with  $c = 0$ , showing both logistic loss and 0/1 loss.



### Exercise 3: Gradient Descent (1.5+0.5+0+0.5+0.5+1+0+0=4Points)

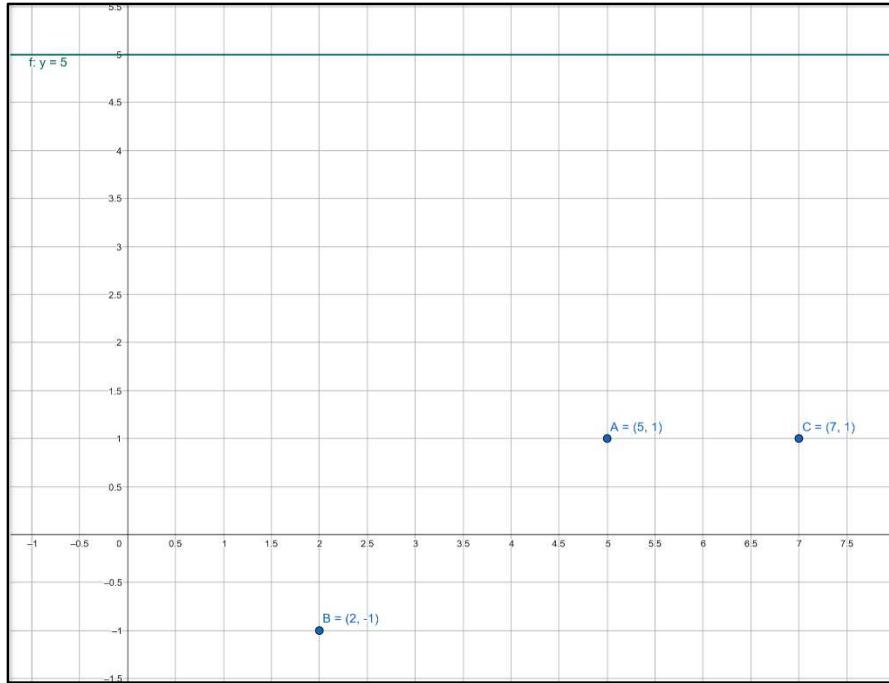
In this exercise you will be calculating one iteration of the LMS algorithm, slide ML:I-42.

The set D contains the following three examples of one-dimensional vectors with the two classes  $\{-1, 1\}$ :

	$x_i$	$c$
$x_1$	5	1
$x_2$	2	-1
$x_3$	7	1

Assume the weight vector  $w$  was randomly initialized to  $w := (0, 1)^T$  and  $x_1$  was randomly selected for the first iteration of the algorithm.

a) Plot the line defined by  $w$  and all examples from  $D$  into one coordinate system.



b) Compute the squared loss w.r.t.  $x_1$  and  $w$ . The squared loss is defined as

$$l_2(c, y(x)) = \frac{1}{2} \cdot (c - y(x))^2 = \frac{1}{2} \cdot (1 - 5)^2 = 8$$

c) Show that the loss gradient,  $\left(\frac{\partial l_2}{\partial w_0} \frac{\partial l_2}{\partial w_1}\right)^T$ , is indeed equal to  $-\delta \cdot x$ .

$$\frac{\partial l_2}{\partial w_i} = \frac{1}{2} \cdot 2(c - w^T x)(-x_i) = -(c - y(x))x_i$$

$$\text{With } \delta = c - y(x), \text{ this is } \left(\frac{\partial l_2}{\partial w_0} \frac{\partial l_2}{\partial w_1}\right)^T = -\delta \cdot x$$

d) Derive the loss gradient for  $x_1$  and  $w$ .

For  $x_1 = 5, \delta = c - y = 1 - 5 = -4$ .

$$\text{loss gradient, } \left(\frac{\partial l_2}{\partial w_0} \frac{\partial l_2}{\partial w_1}\right)^T = -\delta x = -(-4) \begin{pmatrix} 1 \\ 5 \end{pmatrix} = \begin{pmatrix} 4 \\ 20 \end{pmatrix}$$

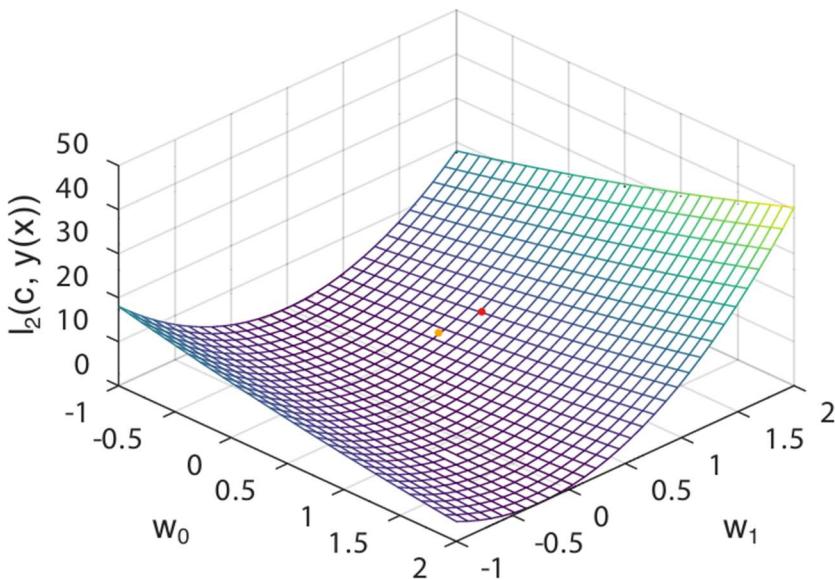
e) Calculate  $\Delta w$  with a learning rate  $\eta = 0.01$  for  $x_1$  and  $w$ .

$$\Delta w = \eta \delta x = 0.01 \cdot (-4) \begin{pmatrix} 1 \\ 5 \end{pmatrix} = \begin{pmatrix} -0.04 \\ -0.2 \end{pmatrix}$$

$$\text{New weights, } w' = w + \Delta w = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} -0.04 \\ -0.2 \end{pmatrix} = \begin{pmatrix} -0.04 \\ 0.8 \end{pmatrix}$$

f) The following plot shows the loss landscape defined by  $l_2$  for  $x_1$ . Mark the location of the model for  $w$  and for its update  $w + \Delta w$ .

colour code:  $w$  ,  $w + \Delta w$



g) Compute the squared loss w.r.t.  $x_1$  and the updated  $w$ . Could it be possible that this loss is now larger than it was before the update?

$$\begin{aligned} y' &= w^T x = -0.04 + 0.8 \cdot 5 = 3.96, c - y' = 1 - 3.96 = -2.96, l'_2 = \frac{1}{2}(-2.96)^2 \\ &= 4.3808 \end{aligned}$$

So the loss decreased from 8 to 4.3808 although with a much larger  $\eta$  it is possible to overshoot the minima and make the loss larger.

h) Repeat for  $x_2$ , and  $x_3$ .

For  $x_2 = 2, c_2 = -1$

$$x = (1,2)^T, y = 2, \delta = -1 - 2 = -3$$

$$l_2 = \frac{1}{2} \cdot (-3)^2 = 4.5$$

$$\left( \frac{\partial l_2}{\partial w_0} \frac{\partial l_2}{\partial w_1} \right)^T = 0.01 \cdot -3 \cdot (1,2)^T = (-0.03, -0.06)^T$$

$$w' = (-0.03, 0.94)^T$$

$$l'_2 = 4.06125$$

For  $x_3 = 7, c_3 = 1$

$$x = (1,7)^T, y = 7, \delta = 1 - 7 = -6$$

$$l_2 = \frac{1}{2} \cdot (-6)^2 = 18$$

$$\left( \frac{\partial l_2}{\partial w_0} \frac{\partial l_2}{\partial w_1} \right)^T = 0.01 \cdot -6 \cdot (1,7)^T = (-0.06, -0.42)^T$$

$$w' = (-0.06, 0.58)^T$$

$$l'_2 = 4.5$$

#### **Exercise 4: Overfitting and train-test leakage (1+1+0=2 Points)**

a) What is the experimental setup of choice when trying to detect overfitting?

Overfitting can be detected by

- Visual inspection by applying projection or embedding for dimensionalities  $p > 3$ .
- Validating by checking how large is the difference between the training error and the test error.

b) What are methods to mitigate overfitting?

How to mitigate Overfitting

- Increase the quantity and/or the quality of the training data.
- Early stopping of a model optimization/refinement process
- Regularization: Increase model bias by constraining the hypothesis space

c) What must be paid attention to when performing a train-validation split on the following datasets in the given problems?

- Detecting pneumonia from chest x-rays. Data includes 112,120 unique images from 30,805 unique patients.

Making sure splitting by patient ID rather than image to not have the same x-ray in training and validation.

- Given 1000 voice recordings (single sentences) of 100 people in total from 5 German cities. The model should be able to classify the dialects of arbitrary people into one of these cities.

Splitting by speaker and not by recording. Also keeping the city class distribution even in validation and training.

- Given 1000 voice recordings (single sentences) of 100 people in total from 5 German cities. The model should be able to rate the dialects of arbitrary people from all over Germany by intelligibility.

Again, split by speaker and not by recording. Also, keeping the range of intelligibility scores and cities in both splits.

- Given 1000 voice recordings (single sentences) of 100 people in total from 5 German cities. The model should be able to classify the person that said a given sentence.

The important thing is to ensure that individual recordings don't appear in both sets and per-person counts are reasonable.

### **Exercise 5: Regularization (1+1=2 Points)**

Suppose we are estimating the regression coefficients in a linear regression model by minimizing the objective function  $\zeta$ .

$$\mathcal{L}(\mathbf{w}) = \text{RSS}_{tr}(\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}$$

The term  $\text{RSS}_{tr}(w) = \sum_{(x_i, y_i) \in D_{tr}} (y_i - W^T X_i)^2$  refers to the residual sum of squares computed on the set  $D_{tr}$  that is used for parameter estimation. Assume that we can also compute an  $\text{RSS}_{test}$  on a separate set  $D_{test}$  that we don't use during training.

When we vary the hyperparameter  $\lambda$ , starting from 0 and gradually increase it, what will happen to the following quantities? Explain your answers.

- a) The value of RSStr(w) will . . .
- remain constant.
  - steadily increase.
  - steadily decrease.
  - increase initially, then eventually start decreasing in an inverted U shape.
  - decrease initially, then eventually start increasing in a U shape.

This is ridge regularization. When we increase  $\lambda$ , we are forcing the model to keep weights smaller, which **restricts** its capacity to fit the training data perfectly. As a result, RSS\_tr(w) will **steadily increase** because the model cannot adjust as freely to minimize the training error.

- b) The value of RSS test(w) will . . .
- remain constant.
  - steadily increase.
  - steadily decrease.
  - increase initially, then eventually start decreasing in an inverted U shape.
  - decrease initially, then eventually start increasing in a U shape.

When the training RSS increases due to stronger regularization (or reduced model complexity), the model becomes more generalized. This initially causes the test RSS to decrease, as the model is less prone to overfitting and captures broader patterns.

However, this trend has a limit. If regularization becomes too strong, the model becomes excessively simple, leading to underfitting. At this point, the model fails to capture the essential patterns in the data, and consequently, the test RSS begins to increase again.

### Exercise 6: Decision Boundaries (1.5+1.5=3 Points)

- a) Linear Boundary: Consider a 2D feature space ( $p = 2$ ) with an extended weight vector  $w = (-3, 1, 1)^T$  (where  $w_0 = -3$  corresponds to the bias term  $x_0 = 1$ ).
- (a1) Determine the equation of the decision boundary.

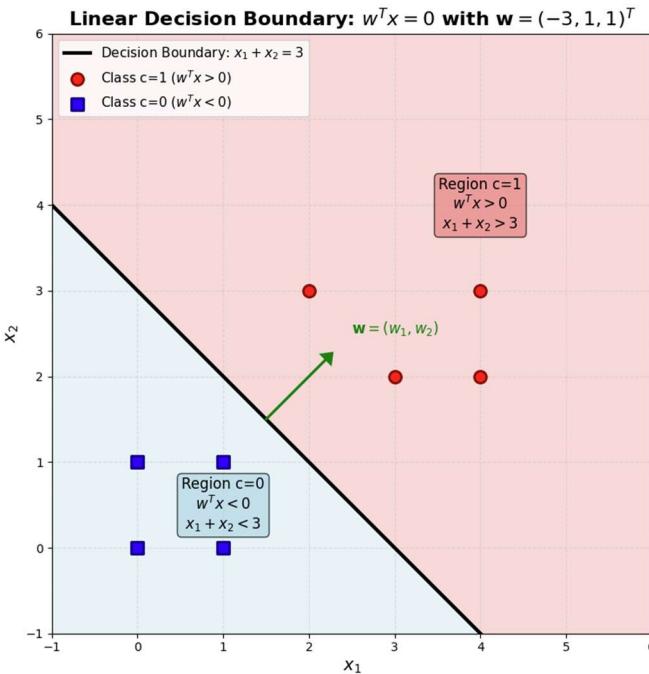
$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} * (x_0, x_1, x_2) = 0$$

$$x_0w_0 + x_1w_1 + x_2w_2 = 0$$

$$-3 + x_1 + x_2 = 0$$

$$x_1 + x_2 = 3$$

(a2) Draw the decision boundary in a coordinate system with axes  $x_1$  and  $x_2$ . Mark the region classified as  $c = 1$ .

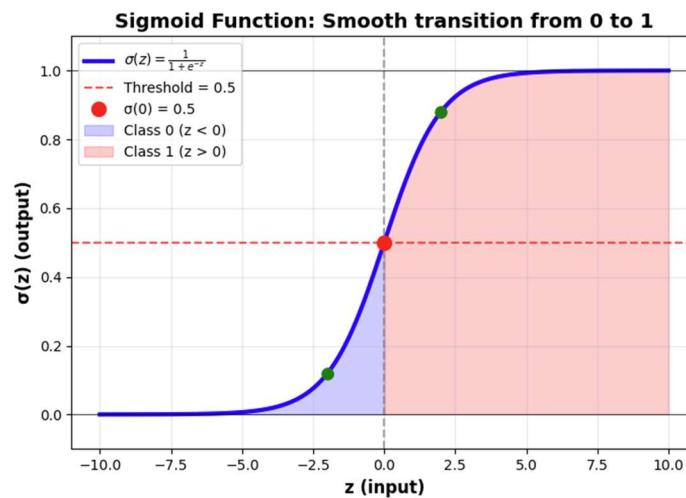


b) Non-Linear Boundary: Given the model:

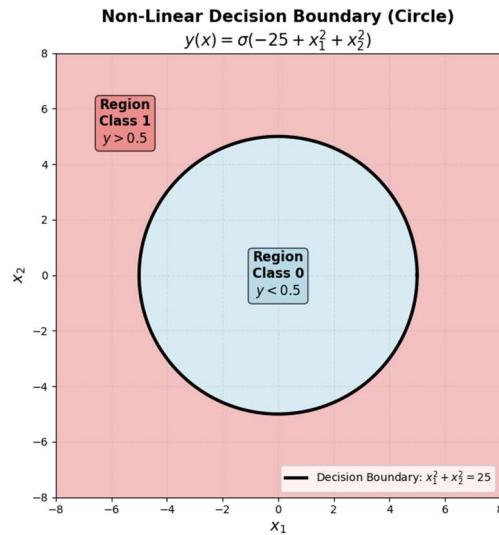
$$y(x) = \sigma(-25 + x_1^2 + x_2^2)$$

(b1) Find the equation of the decision boundary and plot it in a coordinate axis.

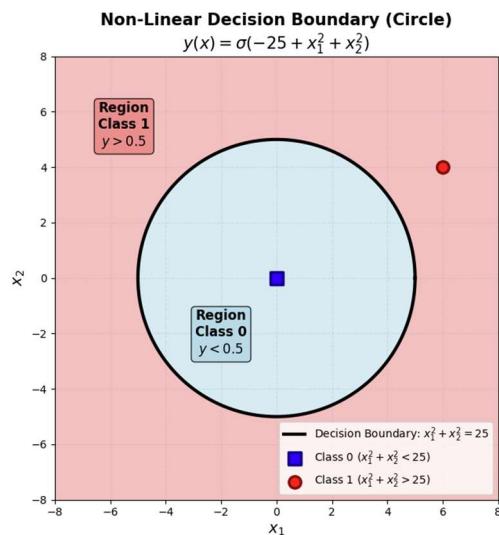
We use the sigmoid function. In order to have the plane that divides the dataset, we need to see where is the limit where we classify values from one class to another. In the sigmoid function, **the value we are looking for is where  $z = 0$**  (the argument inside sigmoid), where  $\sigma(z) = 0.5$  (see the image below), which is the limit between one class and the other.



The decision boundary for a nonlinear classifier looks like the image below.



(b2) Classify the points  $(6, 4)$  and  $(0, 0)$ . Calculate the linear scores ( $z$ ) explicitly. See the image below.



### Exercise 7: Implementing Logistic Regression Classifier (1+1+2+2+1+1=8 Points).

In this exercise, you will implement a logistic regression model for predicting whether a given text was written by a human or generated by a language model. Download and use these files from Moodle:

- features-train.tsv: Feature vectors for each example in the training set.
- features-test.tsv: Feature vectors for each example in the test set.
- Labels-train.tsv: Labels for each example in the training set indicating the class is\_human (C = {True, False})

- `programming_exercise_logistic_model.py`: Template program for writing your implementation. It contains function stubs for each function mentioned below. Use the following command to run the program:

```
python3 programming_exercise_logistic_model.py features-train.tsv  
labels-train.tsv features-test.tsv predictions-test.tsv
```

- `requirements.txt`: Requirements file for the template; can be used to install dependencies.

- 
- a. Implement two functions to load the dataset: `load_feature_vectors` reads feature vectors from a `features-*.tsv` and returns the contained multiset of feature vectors  $X$  as an  $n$ -by- $(p+1)$  matrix.  
`load_class_values` reads the `is_human` labels from the `labels-train.tsv` as one list of 1s if the example is human and 0s if it is machine-generated. How many examples of each class are in the data set?
    - 1087 human-written examples, 1087 AI-generated examples (balanced dataset)
  - b. Implement a function `misclassification_rate` to measure the misclassification rate of the model's predictions. What is the misclassification rate of a random classifier on the training set? Support your answer with code.
    - Random classifier performance: ~52.53% misclassification rate (as expected for balanced binary classification)

```
def misclassification_rate(cs: np.array, ys: np.array) -> float:  
    if len(cs) == 0:  
        return float('nan')  
    else:  
        misclassified = np.sum(cs != ys)  
        return misclassified / len(cs)
```

```

# Single trial
random_predictions = np.random.randint(0, 2, len(cs))
random_misclass = misclassification_rate(cs, random_predictions)
# Result: 0.5253 (52.53%)

# 1000 trials for statistical validation
n_trials = 1000
random_misclass_rates = []
for _ in range(n_trials):
    random_preds = np.random.randint(0, 2, len(cs))
    random_misclass_rates.append(misclassification_rate(cs, random_preds))

avg_random_misclass = np.mean(random_misclass_rates)
# Average: 0.4998 (49.98%) ± 0.0109 (1.09%)

```

The `misclassification_rate()` function calculates the proportion of incorrect predictions. A random classifier on the balanced training set (1087 human, 1087 AI examples) achieves approximately **50% misclassification rate**, confirmed by averaging 1000 trials ( $49.98\% \pm 1.09\%$ ). This is expected since the classifier randomly assigns labels with equal probability.

- c. Implement a function `logistic_function` to calculate the output of the logistic (sigmoid) function w.r.t. input  $x$  parameterized by  $w$  and a function `logistic_prediction` to predict the class of  $x$  accordingly.
  - See the `programming_exercise_logistic_model.py` file
- d. Implement a function `train_logistic_regression_with_bgd` that fits a logistic regression model using the Batch Gradient Descent (BGD) algorithm. A parameter of the function specifies the fraction of training examples to not use for training but for validation. The function returns the trained weights as  $p + 1$ -vector and three lists containing the training loss, misclassification rate on the training examples, and the misclassification rate on the validation examples after each iteration.
  - See the `programming_exercise_logistic_model.py` file
- e. Plot the training loss, misclassification rate on the training examples, and the misclassification rate on the validation examples after each iteration. Are loss and misclassification rate correlated?

The `plot_loss_and_misclassification_rates()` function creates two plots:

Left: Normalized training loss over 100 iterations

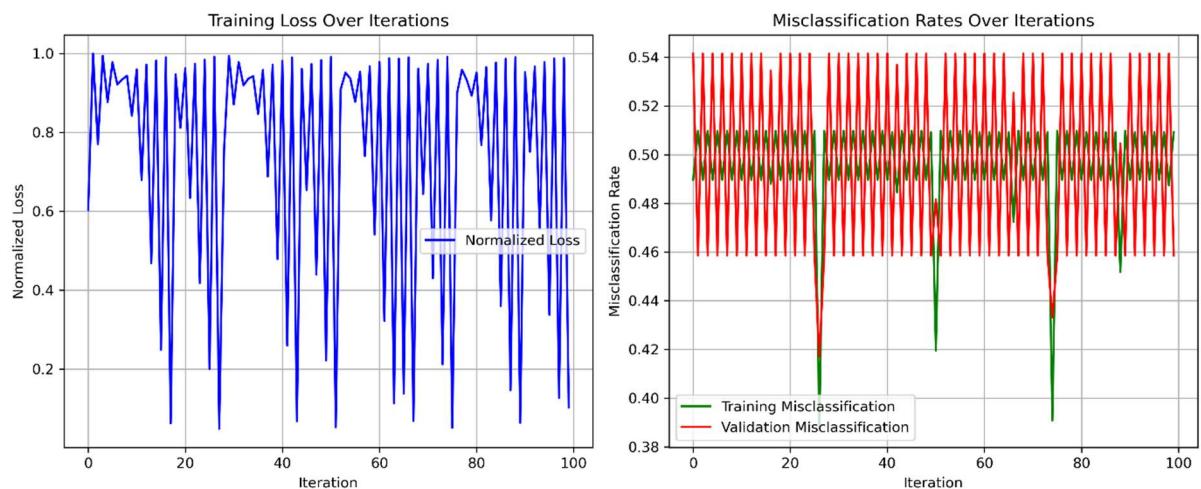
Right: Training (green) and validation (red) misclassification rates

Are loss and misclassification rate correlated?

Yes, they show positive correlation. The plots demonstrate that loss and misclassification rates move together - both oscillate with similar patterns throughout training. When loss spikes up, error rates tend to increase; when loss drops, error rates decrease.

However, the high volatility in both metrics indicates that the model has not converged after 100 iterations. The oscillations suggest the training process is still unstable, likely due to the small learning rate ( $\eta=1e-8$ ). Despite this, the correlation between loss and misclassification is evident from the synchronized fluctuation patterns.

The correlation is not perfect because loss is continuous while misclassification is discrete (correct/incorrect).



- Use the trained model to predict the labels for each example in the test set (features-test.tsv). Write the prediction as one column to a file predictions-test.tsv and submit that along with your other solutions.

See predictions-test.tsv file.

If you would like to improve your model, here are some hints of what you could try:

- The features you have implemented may have vastly different value scales, which can harm the performance of linear models. For details and a rationale on how that influences the training process, you can watch this video by Andrew Ng. Try to implement one of the feature scaling methods explained in the video.

Currently, features have vastly different scales (e.g., num\_characters ranges in thousands while stop\_word\_ratio is between 0-1). Implementing standardization (z-score normalization) would normalize all features to have

mean=0 and std=1, which would significantly improve convergence speed and model stability. This addresses the oscillation seen in the current training plots.

- Experiment with further hyperparameters and model variants to get the best results, e.g., tweaking the learning rate and number of iterations, selecting subsets of the features, computing additional features as (non-)linear combinations of the existing ones, adding a regularization term, etc.

The current settings show high oscillation without convergence:

- Learning rate: The current value of 1e-8 is extremely small, causing very slow learning. Increasing to 1e-5, 1e-4, or 1e-3 would allow faster convergence.
- Iterations: Increasing from 100 to 500-2000 iterations would give the model more time to converge.
- Feature selection: Testing with different feature subsets could identify which features are most predictive.
- Regularization: Adding L2 regularization would prevent overfitting and improve generalization.
- Consider implementing k-fold cross-validation instead of using a single hold-out in order to get a more robust estimate of your model's performance on the unseen data.

Instead of a single 80/20 train-validation split, using 5-fold or 10-fold cross-validation would provide more robust performance estimates by training and validating on different data subsets. This reduces the variance in performance metrics and gives a better estimate of how the model will perform on unseen data.

Priority: Feature scaling and learning rate adjustment would likely have the most immediate impact on the oscillating behavior observed in the current plots, leading to smoother convergence and better performance.