

# Introduction to Machine Learning

## Logistic Regression, Multi-class Classification

Ashley Gao

William & Mary

September 25, 2023

# Overview

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete ( $> 2$ )-valued target
- Examples of binary classification:
  - predict whether a patient has a disease, given the presence or absence of various symptoms
  - classify e-mails as spam or non-spam
  - predict whether a financial transaction is fraudulent

# Overview

- Binary linear classification
  - classification: given a D-dimensional input  $\mathbf{x} \in \mathbb{R}^D$  predict a discrete-valued target
  - binary: predict a binary target  $t \in \{0, 1\}$ 
    - Training examples with  $t = 1$  are called positive examples, and training examples with  $t = 0$  are called negative examples.
    - $t \in \{0, 1\}$  or  $t \in \{+1, -1\}$  is for computational convenience.
  - linear: model prediction  $y$  is a linear function of  $\mathbf{x}$ , followed by a threshold  $r$

$$z = \mathbf{w}^\top \mathbf{x} + b \quad (1)$$

$$y = \begin{cases} 1 & z \geq r \\ 0 & z < r \end{cases} \quad (2)$$

# Some Simplification

- Eliminating the threshold

- We can assume without loss of generality (WLOG) that the threshold  $r = 0$ :

$$\mathbf{w}^\top \mathbf{x} + b \geq r \iff \mathbf{w}^\top \mathbf{x} + b - r \geq 0 \quad (3)$$

- Eliminating the bias

- Add a dummy feature  $x_0$  which always takes the value 1. The weight  $w_0 = b$  is equivalent to a bias (same as linear regression)

- Simplified model

- Receive input  $\mathbf{x} \in \mathbb{R}^{D+1}$  with  $x_0 = 1$ :

$$z = \mathbf{w}^\top \mathbf{x} \quad (4)$$

$$y = \begin{cases} 1 & z \geq r \\ 0 & z < r \end{cases} \quad (5)$$

# Some Examples

- Let's consider some simple examples to examine the properties of our model
- Let's focus on minimizing the training set error, and forget about whether our model will generalize to a test set.

# Some Examples

**NOT**

$x_0$	$x_1$	$t$
1	0	1
1	1	0

- Suppose this is our training set, with the dummy feature  $x_0$  included
- Which conditions on  $w_0, w_1$  guarantee perfect classification?
  - When  $x_1 = 0$ , need:  $z = w_0x_0 + w_1x_1 \geq 0 \iff w_0 \geq 0$
  - When  $x_1 = 1$ , need:  $z = w_0x_0 + w_1x_1 < 0 \iff w_0 + w_1 < 0$
- Possible solution:  $w_0 = 1, w_1 = -2$
- Is this the only solution?

# Some Examples

## AND

$x_0$	$x_1$	$x_2$	$t$
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$$z = w_0x_0 + w_1x_1 + w_2x_2$$

$$\text{need: } w_0 < 0$$

$$\text{need: } w_0 + w_2 < 0$$

$$\text{need: } w_0 + w_1 < 0$$

$$\text{need: } w_0 + w_1 + w_2 \geq 0$$

Example solution:  $w_0 = -1.5$ ,  $w_1 = 1$ ,  $w_2 = 1$

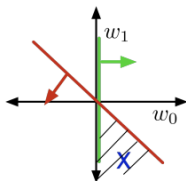
# The Geometric Picture

- Training examples are points
- Weights (hypotheses)  $\mathbf{w}$  can be represented by half-spaces.  
 $H_+ = \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} \geq 0\}$ ,  $H_- = \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} < 0\}$ 
  - The boundaries of these half-spaces pass through the origin (why?)
  - Decision boundary:  $\{\mathbf{x} : \mathbf{w}^\top \mathbf{x} = 0\}$ 
    - In 2-D, it's a line, but in high dimensions it is a hyperplane
- If the training examples can be perfectly separated by a linear decision rule, we say data is linearly separable.



# The Geometric Picture

- Weight space



$$w_0 \geq 0$$

$$w_0 + w_1 < 0$$

- Weights (hypotheses)  $\mathbf{w}$  are points
- Each training example  $\mathbf{x}$  specifies a half-space  $\mathbf{w}$  must lie in to be correctly classified:  $\mathbf{w}^\top \mathbf{x} \geq 0$  if  $t = 1$ .
- For NOT example:
  - $x_0 = 1, x_1 = 0, t = 1 \implies (w_0, w_1) \in \mathcal{W} : w_0 \geq 0$
  - $x_0 = 1, x_1 = 1, t = 0 \implies (w_0, w_1) \in \mathcal{W} : w_0 + w_1 < 0$
- The region satisfying all the constraints is the feasible region; if this region is nonempty, the problem is feasible, otherwise it is infeasible.

# Summary — Binary Linear Classifiers

- Summary: Targets  $t \in \{0, 1\}$ , inputs  $\mathbf{x} \in \mathbb{R}^{D+1}$  with  $x_0 = 1$ , and model is defined by weights  $\mathbf{w}$  and

$$z = \mathbf{w}^\top \mathbf{x} \tag{6}$$

$$y = \begin{cases} 1 & z \geq r \\ 0 & z < r \end{cases} \tag{7}$$

- How can we find good values for  $\mathbf{w}$ ?
- If the training set is linearly separable, we could solve for  $\mathbf{w}$  using linear programming
  - We could also apply an iterative procedure known as the perceptron algorithm (but this is primarily of historical interest).
- If it's not linearly separable, the problem is harder
  - Data is almost never linearly separable in real life.

# Towards Logistic Regression

# Loss Function

- Instead: define loss function then try to minimize the resulting cost function
  - Recall: cost is loss averaged (or summed) over the training set
- Seemingly obvious loss function: 0-1 loss

$$\mathcal{L}_{0,1}(y, t) = \begin{cases} 1 & y = t \\ 0 & y \neq t \end{cases} \quad (8)$$

$$\mathcal{L}_{0,1}(y, t) = \mathbb{I}(y \neq t) \quad (9)$$

# Attempt 1: 0-1 loss

- Usually, the cost  $\mathcal{J}$  is the averaged loss over training examples; for 0-1 loss, this is the misclassification rate:

$$\mathcal{J} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(y^{(i)} \neq t^{(i)}) \quad (10)$$

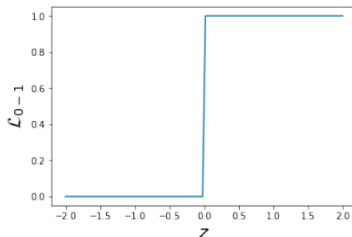
- Problem: how to optimize? In general, a hard problem (can be NP-hard)
- This is due to the step function (0-1 loss) not being nice (continuous/smooth/convex etc)

# Attempt 1: 0-1 loss

- Minimum of a function will be at its critical points.
- Let's try to find the critical point of 0-1 loss
- Chain rule:

$$\frac{\partial \mathcal{L}_{0,1}}{\partial w_j} = \frac{\partial \mathcal{L}_{0,1}}{\partial z} \frac{\partial z}{\partial w_j} \quad (11)$$

- But  $\frac{\partial \mathcal{L}_{0,1}}{\partial z}$  is zero everywhere it's defined!



- $\frac{\partial \mathcal{L}_{0,1}}{\partial w_j} = 0$  means that changing the weights by a very small amount probably has no effect on the loss  $\implies$  Almost any point has 0 gradient!

## Attempt 2: Linear Regression

- Sometimes we can replace the loss function we care about with one which is easier to optimize. This is known as relaxation with a smooth surrogate loss function.
- One problem with  $\mathcal{L}_{0,1}$ : defined in terms of final prediction, which inherently involves a discontinuity
- Instead, define loss in terms of  $\mathbf{w}^\top \mathbf{x}$  directly
  - Redo notation for convenience:  $z = \mathbf{w}^\top \mathbf{x}$

## Attempt 2: Linear Regression

- We already know how to fit a linear regression model. Can we use this instead?

$$z = \mathbf{w}^\top \mathbf{x} \quad (12)$$

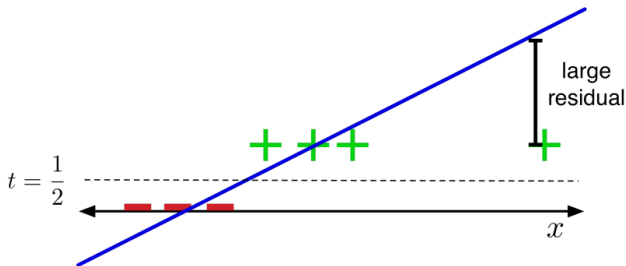
$$\mathcal{L}_{SE} = \frac{1}{2}(z - t)^2 \quad (13)$$

- Doesn't matter that the targets are actually binary. Treat them as continuous values.
- For this loss function, it makes sense to make final predictions by thresholding  $z$  at 0.5



## Attempt 2: Linear Regression

- The problem:



- The loss function hates when you make correct predictions with high confidence!
- If  $t = 1$ , it's more unhappy about  $z = 10$  than  $z = 0$ .

## Attempt 3: Logistic Activation Function

- There's obviously no reason to predict values outside  $[0, 1]$ . Let's squash  $y$  into this interval.
- The logistic function is a kind of sigmoid, or S-shaped function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (14)$$

- A linear model with a logistic nonlinearity is known as log-linear:

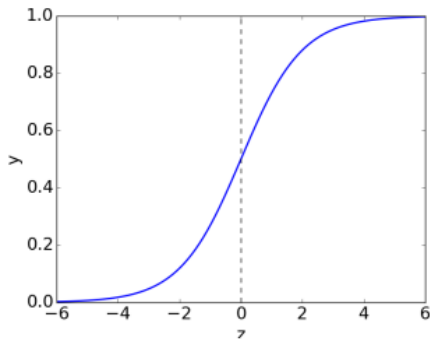
$$z = \mathbf{w}^\top \mathbf{x} \quad (15)$$

$$y = \sigma(z) \quad (16)$$

$$\mathcal{L}_{SE} = \frac{1}{2}(y - t)^2 \quad (17)$$

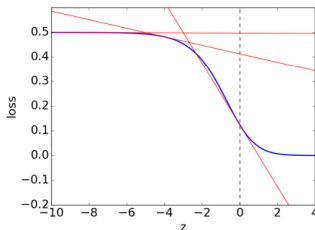
- Used in this way,  $\sigma$  is called an activation function.

## Attempt 3: Logistic Activation Function



## Attempt 3: Logistic Activation Function

- The problem: (plot of  $\mathcal{L}_{SE}$  as a function of  $z$ , assuming  $t = 1$ )



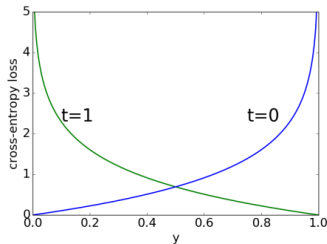
$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w_j}$$

- For  $z \ll 0$ , we have  $\sigma(z) \approx 0$ .
- $\frac{\partial \mathcal{L}}{\partial z} \approx 0$  (check!)  $\implies \frac{\partial \mathcal{L}}{\partial w_j} \approx 0 \implies$  derivative w.r.t.  $w_j$  is small  $\implies w_j$  is like a critical point
- If the prediction is really wrong, you should be far from a critical point (which is your candidate solution).

# Logistic Regression

- Because  $y \in [0, 1]$ , we can interpret it as the estimated probability that  $t = 1$ . If  $t = 0$ , then we want to heavily penalize  $y \approx 1$ .
- The people who were 99% confident a certain presidential candidate would win were much more wrong than the ones who were only 90% confident, given that the person didn't win.
- Cross-entropy loss (aka log loss) captures this intuition:

$$\begin{aligned}\mathcal{L}_{\text{CE}}(y, t) &= \begin{cases} -\log y & \text{if } t = 1 \\ -\log(1 - y) & \text{if } t = 0 \end{cases} \\ &= -t \log y - (1 - t) \log(1 - y)\end{aligned}$$

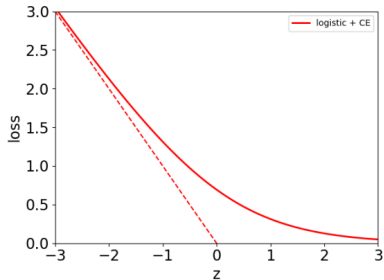


# Logistic Regression

- Logistic regression:

$$\begin{aligned}z &= \mathbf{w}^\top \mathbf{x} \\y &= \sigma(z) \\&= \frac{1}{1 + e^{-z}} \\\mathcal{L}_{\text{CE}} &= -t \log y - (1 - t) \log(1 - y)\end{aligned}$$

Plot is for target  $t = 1$ .



# Logistic Regression - Numerical Instabilities

- If we implement logistic regression naively, we can end up with numerical instabilities.
- Consider:  $t = 1$  but you're really confident that  $z \ll 0$
- If  $y$  is small enough, it may be numerically zero. This can cause very subtle and hard-to-find bugs.

$$y = \sigma(z) \Rightarrow y \approx 0 \quad (18)$$

$$\mathcal{L}_{CE} = -t \log y - (1 - t) \log(1 - y) \quad (19)$$

# Logistic Regression - Numerical Stable Version

- Instead, we combine the activation function and the loss into a single logistic-cross-entropy function

$$\mathcal{L}_{LCE} = \mathcal{L}_{CE}(\sigma(z), t) = -t \log\left(\frac{1}{1 + e^{-z}}\right) - (1 - t) \log\left(1 - \frac{1}{1 + e^{-z}}\right) \quad (20)$$

$$\mathcal{L}_{LCE} = t \log(1 + e^{-z}) - (1 - t)(-z - \log(1 + e^{-z})) \quad (21)$$

$$\mathcal{L}_{LCE} = z - zt + \log(1 + e^{-z}) \quad (22)$$



# Gradient Descent for Logistic Regression

- How do we minimize the cost  $\mathcal{J}$  for logistic regression? No direct solution.
  - Taking derivatives of  $\mathcal{J}$  w.r.t.  $\mathbf{w}$  and setting them to 0 doesn't have an explicit solution.
- However, the logistic loss is a convex function in  $\mathbf{w}$ , so let's consider the gradient descent method/
  - Recall: we initialize the weights to something reasonable and repeatedly adjust them in the direction of the steepest descent.
  - A standard initialization is  $\mathbf{w} = \mathbf{0}$ .

# Gradient of Logistic Loss

- Back to logistic regression:

$$\mathcal{L}_{CE}(y, t) = -t \log y - (1 - t) \log(1 - y) \quad (23)$$

$$y = \frac{1}{1 + e^{(-z)}}, z = \mathbf{w}^\top \mathbf{x} \quad (24)$$

$$\frac{\partial \mathcal{L}_{CE}}{\partial w_j} = \frac{\partial \mathcal{L}_{CE}}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_j} \quad (25)$$

$$\frac{\partial \mathcal{L}_{CE}}{\partial w_j} = \left( -\frac{t}{y} + \frac{1-t}{1-y} \right) \cdot y(1-y) \cdot x_j = (y-t)x_j \quad (26)$$

- Gradient descent update to find the weights of logistic regression:

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j} = w_j - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) x_j^{(i)} \quad (27)$$

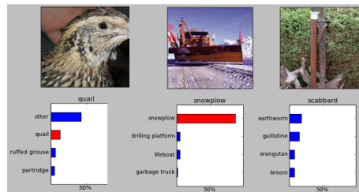
# Multiclass Classification and Softmax Regression

# Overview

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete  $v(> 2)$ -valued target
- Examples of multi-class classification
  - predict the value of a handwritten digit
  - classify e-mails as spam, travel, work, personal

# Multiclass Classification

- Classification tasks with more than two categories:



# Multiclass Classification

- Targets form a discrete set  $\{1, \dots, K\}$ .
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:
  - Entry  $k$  is 1, the other entries are all 0's.
  - $k$  is not to be confused with  $K$ .

$$\mathbf{t} = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^K \quad (28)$$

# Multiclass Linear Classification

- We can start with a linear function of the inputs.
- Now there are  $D$  input dimensions and  $K$  output dimensions, so we need  $K \times D$  weights, which we arrange as a weight matrix  $W$ .
- Also, we have a  $K$ -dimensional vector  $b$  of biases.
- A linear function of the inputs:

$$z_k = \sum_{j=1}^D w_{kj}x_j + b_k \text{ for } k = 1, 2, \dots, K \quad (29)$$

- We can eliminate the bias  $b$  by taking  $W \in \mathbb{R}^{K \times (D+1)}$  adding a dummy variable  $x_0 = 1$ . So, vectorized:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}, \text{ or with dummy } x_0 = 1, \mathbf{z} = \mathbf{W}\mathbf{x} \quad (30)$$

# Multiclass Linear Classification

- How can we turn this linear prediction into a one-hot prediction?
- We can interpret the magnitude of  $z_k$  as a measure of how much the model prefers  $k$  as its prediction.
- If we do this, we should set

$$y_i = \begin{cases} 1 & i = \underset{k}{\operatorname{argmax}} z_k \\ 0 & \text{otherwise} \end{cases} \quad (31)$$

- Exercise: how does the case of  $K = 2$  relate to the prediction rule in binary linear classifiers?



# Softmax Regression

- We need to soften our predictions for the sake of optimization.
- We want soft predictions that are like probabilities, i.e.,  $0 \leq y_k \leq 1$  and  $\sum_k y_k = 1$ .
- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \text{softmax}(z_1, \dots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}} \quad (32)$$

- Outputs can be interpreted as probabilities (positive and sum to 1)
- If  $z_k$  is larger than the others, then  $\text{softmax}(z)_k \approx 1$  and it behaves like  $\text{argmax}$ .
- The inputs  $z_k$  are called the logits.

# Softmax Regression

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function, where the log is applied elementwise.

$$\mathcal{L}_{CE}(\mathbf{y}, \mathbf{t}) = - \sum_{k=1}^K t_k \log y_k = -\mathbf{t}^\top (\log \mathbf{y}) \quad (33)$$

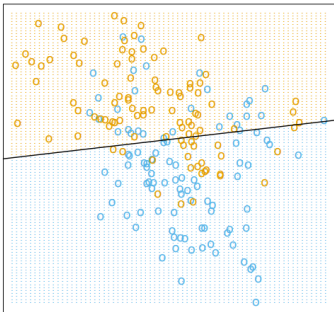
- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function

# Linear Classifiers vs. KNN

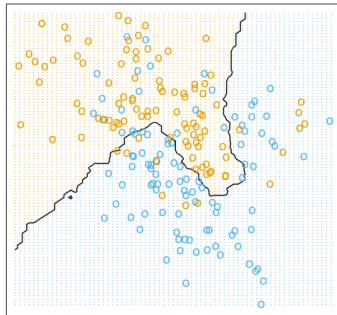
# Linear Classifiers vs. KNN

- Linear classifiers and KNN have very different decision boundaries:

Linear Classifier



K Nearest Neighbours



# Linear Classifiers vs. KNN

- Advantages of linear classifiers over KNN?

- Robustness to irrelevant features
  - Linear classifiers are generally robust to irrelevant or redundant features.
- Scalability
  - Linear classifiers can handle high-dimensional feature spaces efficiently and are more scalable as the number of features increases.
  - The curse of dimensionality!
- Easy updates of the model

- Advantages of KNN over linear classifiers?

- No assumption of data distribution
  - It is a non-parametric method, which means it does not assume any specific functional form for the decision boundaries.
- Non-linearity
  - KNN can capture complex, non-linear decision boundaries
- Robustness to imbalanced data
  - It relies on the local neighborhood and not global statistics.

# Limitations of Linear Classification

# A Few Basic Concepts

- A hypothesis is a function  $f : \mathcal{X} \rightarrow \mathcal{T}$  that we might use to make predictions (recall  $\mathcal{X}$  is the input space and  $\mathcal{T}$  is the target space).
- The hypothesis space  $\mathcal{H}$  for a particular machine learning model or algorithm is a set of hypotheses that it can represent.
  - E.g., in linear regression,  $\mathcal{H}$  is the set of functions that are linear in the data features
  - The job of a machine learning algorithm is to find a good hypothesis  $f \in \mathcal{H}$
- The members of  $\mathcal{H}$ , together with an algorithm's preference for some hypotheses of  $\mathcal{H}$  over others, determine an algorithm's inductive bias.
  - Inductive biases can be understood as general natural patterns or domain knowledge that helps our algorithms to generalize;
    - E.g., linearity, continuity, simplicity ( $L_2$  regularization) ...
  - The so-called No Free Lunch (NFL) theorems assert that if datasets/problems were not naturally biased, no ML algorithm would be better than another

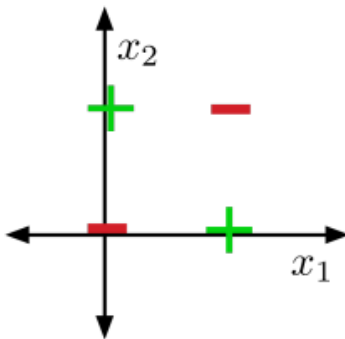
# A Few Basic Concepts

- If an algorithm's hypothesis space  $\mathcal{H}$  can be defined using a finite set of parameters, denoted  $\theta$ , we say the algorithm is parametric.
  - In linear regression,  $\theta = (\mathbf{w}, b)$
  - Other examples: logistic regression, neural networks, k-means and Gaussian mixture models
- If the members of  $\mathcal{H}$  are defined in terms of the data, we say that the algorithm is non-parametric.
  - In  $k$ -nearest neighbors, the learned hypothesis is defined in terms of the training data
  - Other examples: Gaussian processes, decision trees, support vector machines, kernel density estimation
  - These models can sometimes be understood as having an infinite number of parameters



# Limits of Linear Classification

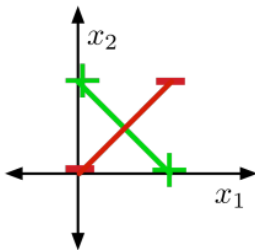
- Some datasets are not linearly separable, e.g. XOR,



- Visually obvious, but how to show this?

# Showing that XOR is not linearly separable (proof by contradiction)

- If two points lie in a half-space, the line segment connecting them also lies in the same half-space.
- Suppose there were some feasible weights (hypothesis). If the positive examples are in the positive half-space, then the green line segment must be as well.
- Similarly, the red line segment must lie within the negative half-space



- But the intersection can't lie in both half-spaces. **Contradiction!**

# Limits of Linear Classification

- Sometimes we can overcome this limitation using feature maps, just like for linear regression. E.g., for XOR:

$$\psi(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

$x_1$	$x_2$	$\psi_1(\mathbf{x})$	$\psi_2(\mathbf{x})$	$\psi_3(\mathbf{x})$	$t$
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	1	1	1	0

- This is linearly separable.

# In the Future

- Feature maps are hard to design well, so next time we'll see how to learn nonlinear feature maps directly using neural networks...
- The basics of NN will be covered in this class.

...

