

# ECE521 Lecture7

## Logistic Regression



UNIVERSITY OF  
**TORONTO**

# Outline

- **Review of decision theory**
- Logistic regression
- A single neuron
- Multi-class classification

# Outline

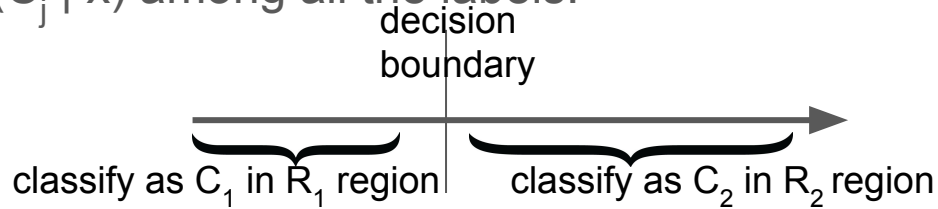
- **Decision theory is conceptually easy and computationally hard**
- Learning objectives:
  - Optimizing misclassification rate -> choosing action according to the largest  $P(C | x)$
  - Develop intuitions about misclassification on the  $P(x, C)$  graph
  - Why expected loss is a good idea? => Capture asymmetry in our decisions

# Decision theory

- We would like to say something formally about why we prefer certain actions over others. We would also like to have a framework to help us decide on the optimal action.
- In classification, the action is choosing the class label given the inputs.
- We again turn to probability theory and assume we have the joint distribution  $P(x, C)$

# Decision theory: misclassification rate

- One possible decision rule (a simple intuition): the best action for the current input  $x$  is to choose the class label  $C_j$  that has the highest conditional probability  $P(C_j | x)$  among all the labels.



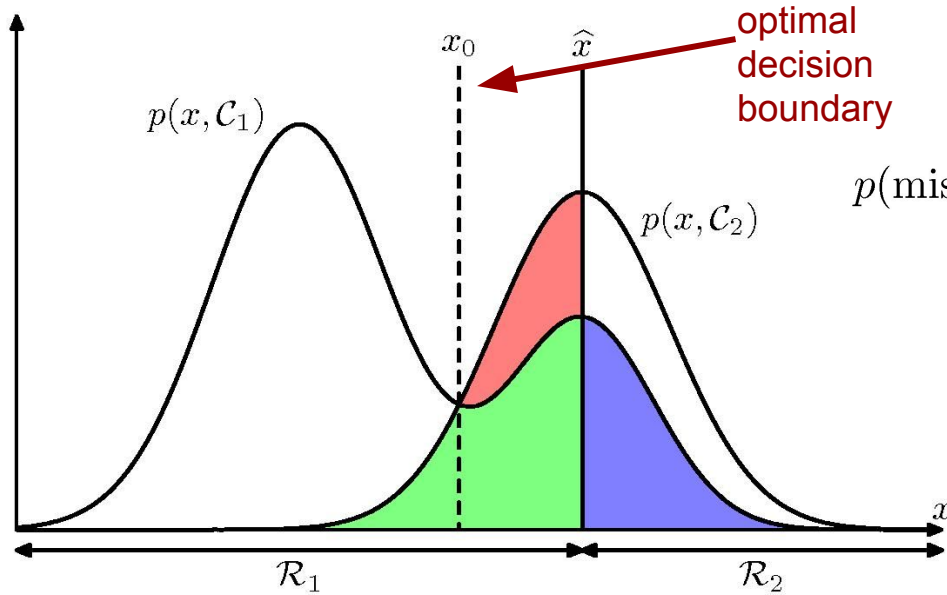
- We can define a framework under which choosing the most probable class  $P(C_j | x)$  is the optimal decision rule:
  - Define misclassification rate =  $P(\text{mistake})$  and consider a binary classification task:

$$p(\text{mistake}) = p(\mathbf{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\mathbf{x} \in \mathcal{R}_2, \mathcal{C}_1)$$

misclassify  $x$  in  $R_1$  as  $C_2$       misclassify  $x$  in  $R_2$  as  $C_1$

# Decision theory: misclassification rate

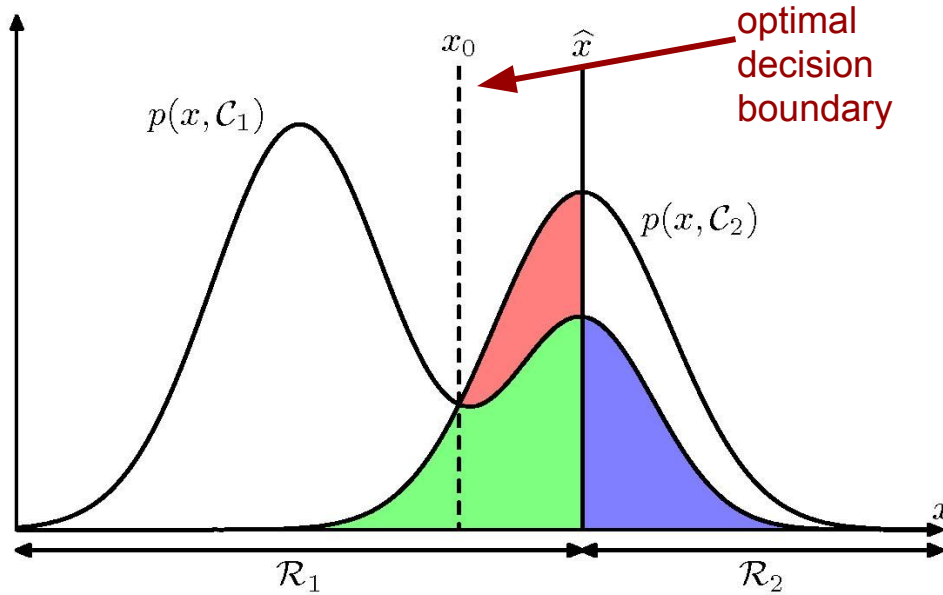
- Given the joint distribution of the inputs and the label,  $P(x, C)$ , we can then plot  $p(x, C_1)$  and  $p(x, C_2)$ . The goal is to “slide”  $\hat{x}$  (decision boundary) and adjust the regions  $R$  such that we minimize the misclassification rate.



$$\begin{aligned} p(\text{mistake}) &= p(\mathbf{x} \in \mathcal{R}_1, C_2) + p(\mathbf{x} \in \mathcal{R}_2, C_1) \\ &= \int_{\mathcal{R}_1} p(\mathbf{x}, C_2) d\mathbf{x} + \int_{\mathcal{R}_2} p(\mathbf{x}, C_1) d\mathbf{x} \\ &= (\text{area of green} + \text{area of red}) + \text{area of blue} \end{aligned}$$

# Decision theory: misclassification rate

- Given the joint distribution of the inputs and the label,  $P(x, C)$ , we can then plot  $p(x, C_1)$  and  $p(x, C_2)$ . The goal is to “slide”  $\hat{x}$  (decision boundary) and adjust the regions  $R$  such that we minimize the misclassification rate.



Intuition: the overlapping area (green + blue) under the joint distribution  $P(x, C)$  is the minimum misclassification rate (the red region vanishes)

This corresponds to picking  $R_1$  such that  $P(x, C_1) > P(x, C_2)$

**So we recovered our intuitive decision rule of picking the most probable class: minimizing misclassification rate is equivalent to picking the most probable class**

# Decision theory: loss matrix

- However, not all mistakes are the same.
  - Misclassifying positive cancer patients as negative (false negative) can be devastating.  
It is not as bad as misclassifying a negative cancer patient as positive (false positive).
  - Misclassifying non-faces as positive (false positive) can be extremely frustrating for smart digital cameras. False negatives are generally more tolerable in this case.
- We need a more general framework to weigh the importance of different mistakes under an application.



# Decision theory: loss matrix

- We might as well just integrate over all regions under the joint distribution  $P(C, \mathbf{x})$  and weight them by a loss matrix  $L$ . We define the expected loss:

$$\mathbb{E}[L] = \sum_{k=1}^2 \sum_{j=1}^2 L_{kj} p(\mathbf{x} \in \mathcal{R}_j, C_k)$$

- In the cancer example,  $C_1 = \text{cancer}$ ,  $C_2 = \text{normal}$ . Because false negatives are really bad, we can weight them highly:

		cancer	normal		
$L =$	cancer	0	1000	$L_{12} = 1000$	false negative
	normal	1	0	$L_{21} = 1$	false positive

# Decision theory: loss matrix

- The misclassification rate is a special case of the expected loss in which all the off-diagonal terms of the loss matrix is 1

$$\mathbb{E}[L] = \sum_{k=1}^2 \sum_{j=1}^2 L_{kj} p(\mathbf{x} \in \mathcal{R}_j, \mathcal{C}_k)$$

under  $L = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \longrightarrow p(\mathbf{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\mathbf{x} \in \mathcal{R}_2, \mathcal{C}_1) = p(\text{mistake})$

- Misclassification rate has equal weightings on the false positives and false negatives. It is often not desirable in many applications.

# Decision theory: loss matrix

- Interesting facts:
- The loss matrix is also called the risk or utility function.
- One of the challenges of decision theory is figuring out what the loss matrix or utility function should be. It is known in economics that the utility of money is notoriously nonlinear and even inconsistent: e.g. below, most people choose A over B but at the same time prefer D over C:

- A. \$1 million guaranteed
- B. 89% chance \$1 million  
10% chance \$2.5 million  
1% chance nothing

- C. 89% chance nothing  
11% chance \$1 million
- D. 90% chance nothing  
10% \$2.5 million

# Outline

- **Decision theory is conceptually easy and computationally hard**
- Learning objectives:
  - Optimizing misclassification rate -> choosing action according to the largest  $P(C | x)$
  - Develop intuitions about misclassification on the  $P(x, C)$  graph
  - Why expected loss is a good idea? => Capture asymmetry in our decisions

# Outline

- Review of decision theory
- **Logistic regression**
- A single neuron
- Multi-class classification

# Outline

- **Logistic regression**
- Prerequisite: linear regression, MLE
- Learning objectives:
  - Logistic/sigmoid function and its derivatives
  - Cross-entropy loss function and its derivatives
  - Probabilistic interpretation (assignment 2)

# Linear regression

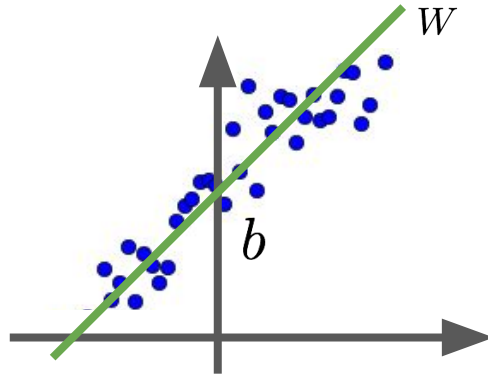
- Given a set of training data and their *real-valued* output, linear regression models the output as a weighted sum of the inputs:

$$y = \sum_n w_n x_n + b = \mathbf{W}^T \mathbf{x} + b$$

- This is also known as linear filtering, in signal processing

# Linear regression

- The weight vector learns the linear trend/slope of the training data



- If the input  $x$  has zero mean, the bias unit learns the averaged output value



# Linear regression as a classifier

- Given a set of training data and their target class labels (positive or negative class) encoded as 1 and 0, you can train a model that produces a score, and then threshold the score. Problems:
  - It is not clear how to compute the confidence of prediction in this setup
  - What should be the threshold? Use decision theory to set it?

# Linear regression as a classifier

- Alternatively, have a model directly regress the binary class labels
- We may decide to use a linear regression model to directly predict 1 and 0.

Problems:

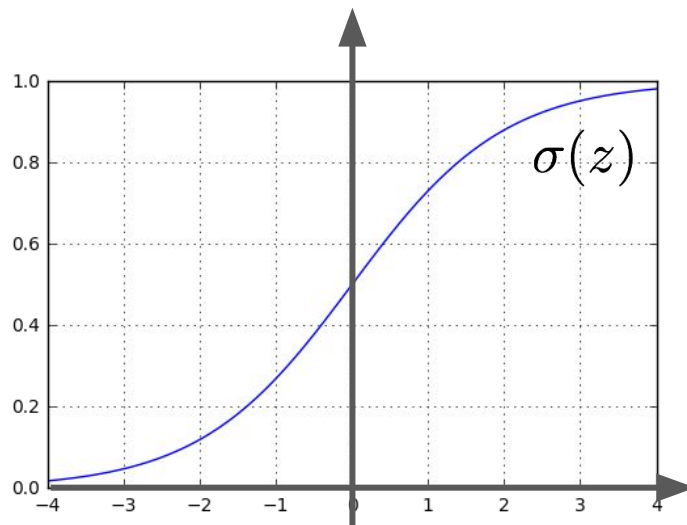
- What does it mean for the model to predict -10 on a test data?
- The range of the linear regression's prediction is unrestricted:  $-\infty$  to  $\infty$

# Logistic regression

- A simple solution is to use a logistic/sigmoid function to “squash” the linear regression output to be between 0 and 1:

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

- Some nice properties:
  - $\sigma : \mathbb{R} \rightarrow (0, 1)$
  - $\sigma(-z) = 1 - \sigma(z)$
  - $\sigma(0) = 0.5$



This is perfect to encode  $P(t=1 | x)$  !

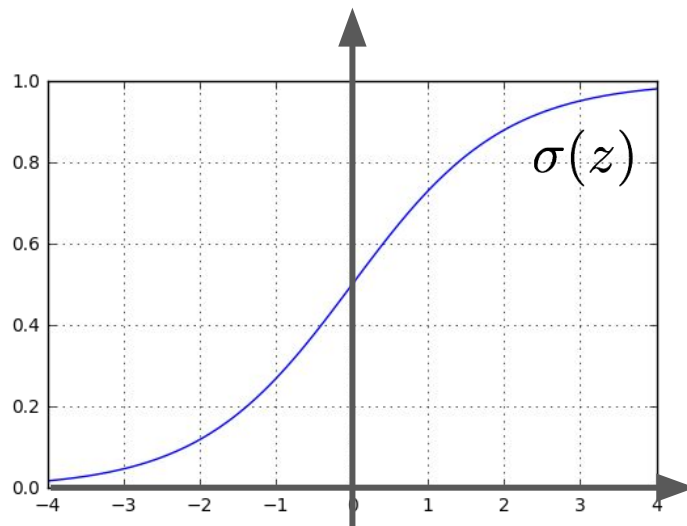
# Logistic regression

- The sigmoid function can be derived from the ratio of log probabilities of two classes, which is also known as log-odds:

$$z = \log \frac{P(t = 1|\mathbf{x})}{P(t = 0|\mathbf{x})} = \log \frac{P(t = 1|\mathbf{x})}{1 - P(t = 1|\mathbf{x})}$$

- If we rearrange the terms and solve for  $P(t=1 | x)$ , we will get again the sigmoid function:

$$P(t = 1|\mathbf{x}) = \frac{1}{1 + e^{-z}} = \sigma(z)$$



# Logistic regression

- We now have a new model whose output is a logistic function of the weighted sum of the inputs, i.e. the logistic regression model. The classification prediction is computed as:

$$\hat{y}^{(m)} = \sigma(z^{(m)}) = \sigma(W^T \mathbf{x}^{(m)} + b)$$

- Define the training loss function using the squared L2 norm:

$$\mathcal{L} = \frac{1}{2} \sum_m \|\hat{y}^{(m)} - t^{(m)}\|_2^2$$

# Logistic regression

- We now have a new model whose output is a logistic function of the weighted sum of the inputs, i.e. the logistic regression model. The classification prediction is computed as:

$$\hat{y}^{(m)} = \sigma(z^{(m)}) = \sigma(W^T \mathbf{x}^{(m)} + b)$$

- Define the training loss function using the squared L2 norm:

$$\mathcal{L} = \frac{1}{2} \sum_m \|\hat{y}^{(m)} - t^{(m)}\|_2^2$$

- Take the gradient w.r.t.  $W$

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_m (\hat{y}^{(m)} - t^{(m)}) \hat{y}^{(m)} (1 - \hat{y}^{(m)}) \mathbf{x}^{(m)}$$

# Logistic regression

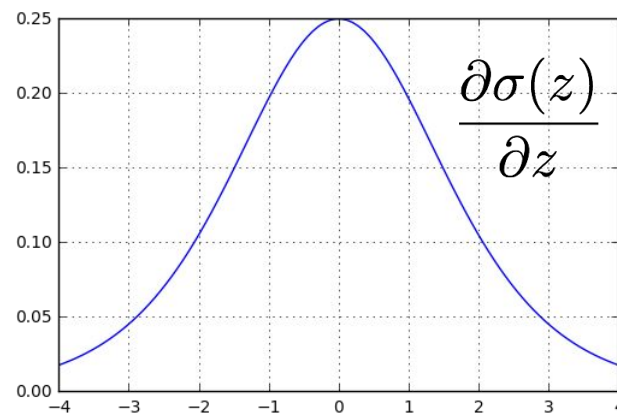
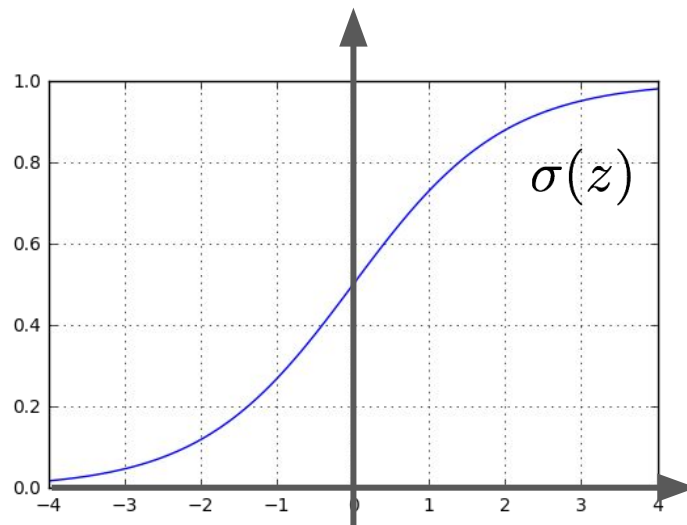
- Recap: we started out first by designing a reasonable linear model. We chose an L2 loss function to measure the discrepancy between predicted label and the target class.
- So take a step back: why did we choose squared L2 in the first place?
  - Analysis is easy. It is differentiable and has a simple gradient.
  - It has a nice probabilistic interpretation with Gaussian error
    - i. In the linear regression case, maximize the likelihood of data  $P(\text{data}) = \text{minimizing sum of the squared L2 loss}$

# Logistic regression

- Squared L2 loss has a problem for classification under the sigmoid function
- Consider gradient of the sigmoid function:

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$$

- The partial derivative is very close to zero away from the origin. i.e. there is very little learning signal as the model gets better





# Cross-entropy loss function

- Cross-entropy is a better loss function.


$$\mathcal{L} = \sum_m -t^{(m)} \log \sigma(z^{(m)}) - (1 - t^{(m)}) \log(1 - \sigma(z^{(m)}))$$

- Take the gradient w.r.t.  $W$ . The gradient under the cross-entropy loss is the same as the gradient for the linear regression model!

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_m (\hat{y}^{(m)} - t^{(m)}) \mathbf{x}^{(m)} \quad \text{where, } \hat{y}^{(m)} = \sigma(z^{(m)}) \\ = \sigma(W^T \mathbf{x}^{(m)} + b)$$

# Learning logistic regression

- What is happening during learning?

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_m (\hat{y}^{(m)} - t^{(m)}) \mathbf{x}^{(m)}$$


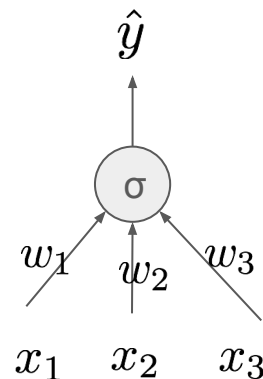
The gradient reflects the correlation between **the error** and **the inputs**

- In what circumstances is the gradient zero? i.e. the model stops learning any new information. Either:
  - All the individual gradients are zero (perfectly separable case), or
  - The gradients from different training examples cancel out (most likely scenario)

# Learning logistic regression

- The gradient reflects the correlation between the mistakes and the input features
  - After learning, the values of the individual weights indicate the importance of its input to the final prediction
  - If an input feature  $x_n$  is positively correlated with the target label, its weight will be a large positive value

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_m (\hat{y}^{(m)} - t^{(m)}) \mathbf{x}^{(m)}$$



# Concepts in the course so far

- **Problem formulations:**

- i.i.d.; choose a loss function (distance function), i.e. squared L2 loss, cross-entropy loss; MLE, MAP (choose a likelihood function and prior, then optimize it to obtain the model parameters); weight-decay regularizer

- **Learning algorithms:**

- gradient descent, stochastic gradient descent, momentum,

- **Models:**

- linear regression; logistic regression; k-NN (no learning required)

- **Some theoretical results**

- Provide some additional intuition: how to pick the optimal regressor, optimal decision rules (how to set the threshold/decision boundary); expected loss

# Outline

- **Logistic regression**
- Learning objectives:
  - Logistic/sigmoid function and its derivatives
  - Cross-entropy loss function and its derivatives
  - Probabilistic interpretation (assignment 2)