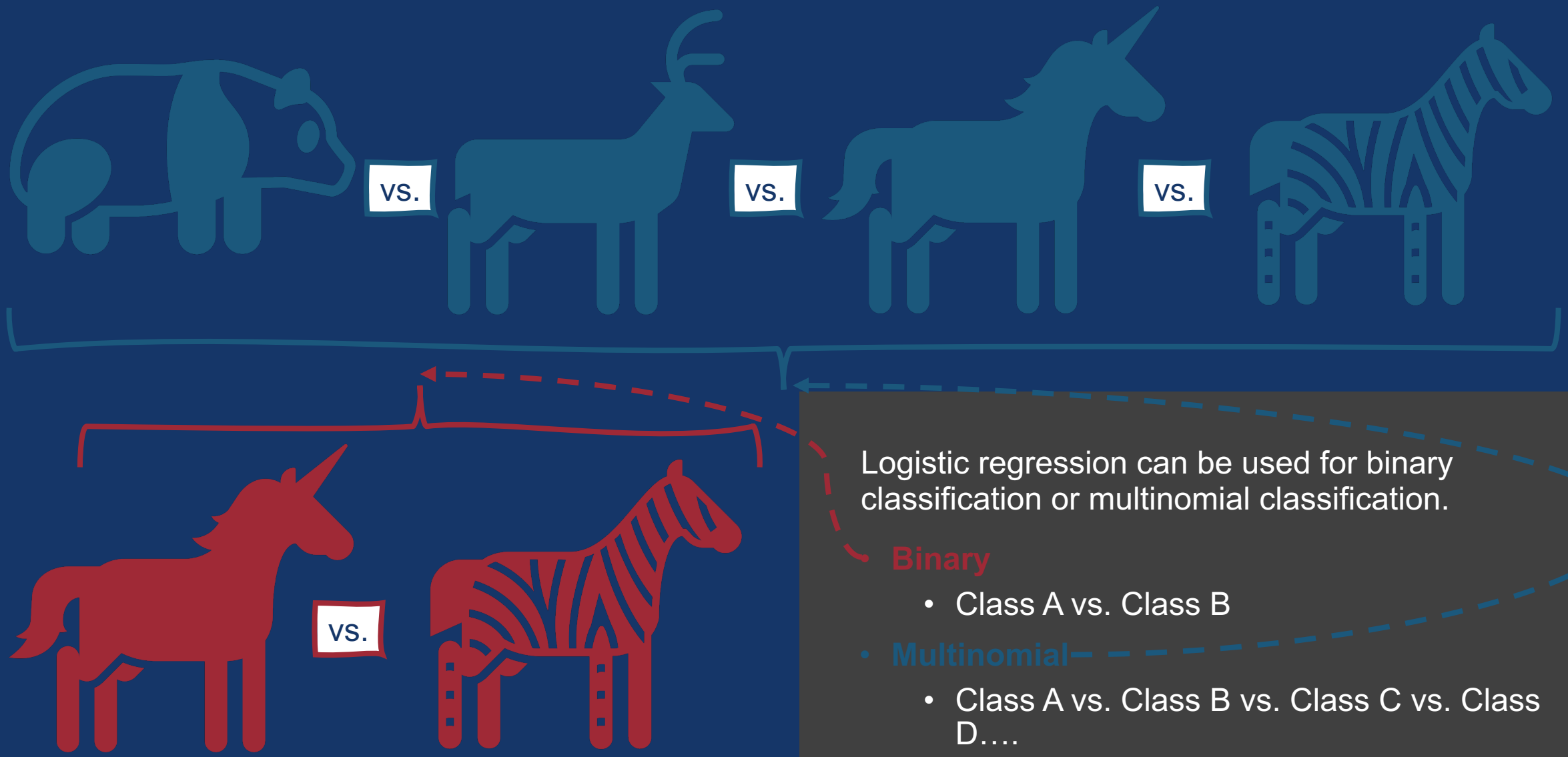
Logistic Regression, Conditional Random Fields, and Vector Semantics

Natalie Parde

UIC CS 421

What is logistic regression?

- Fundamental **supervised machine learning algorithm**
- Used for **text classification**
- Very close relationship with **neural networks!**



Logistic regression can be used for binary classification or multinomial classification.

- **Binary**
  - Class A vs. Class B
- **Multinomial**
  - Class A vs. Class B vs. Class C vs. Class D….

How does logistic regression differ from naïve Bayes?

Naïve Bayes

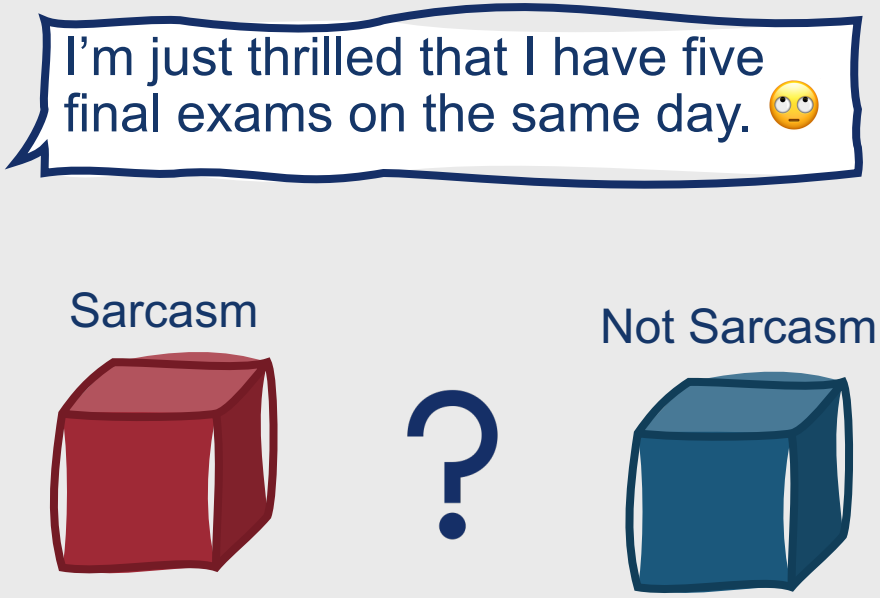
- Generative classifier

Logistic Regression

- Discriminative classifier

Generative Classifiers

- Goal: Understand what each class looks like
  - Should be able to “generate” an instance from each class
- To classify an instance, determines which class model better fits the instance, and chooses that as the label

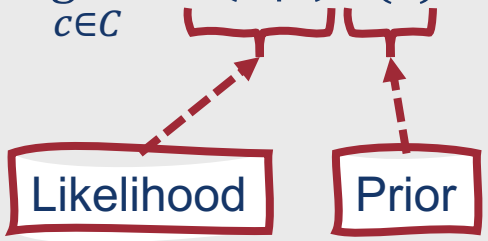


Discriminative Classifiers

- Goal: Learn to distinguish between two classes
  - No need to learn that much about them individually
- To classify an instance, determines whether the distinguishing feature(s) between classes is present

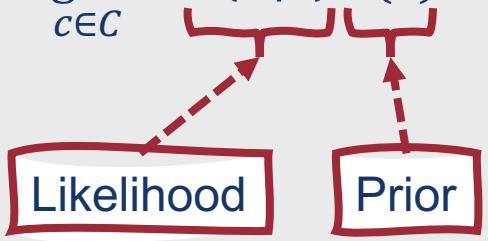
More formally….

- Recall the definition of naïve Bayes:
  - $\hat{c} = \underset{c \in C}{\text{argmax}}\, P(d|c)P(c)$



More formally….

- Recall the definition of naïve Bayes:
  - $\hat{c} = \underset{c \in C}{\text{argmax}}\, P(d|c)P(c)$



A generative model like naïve Bayes makes use of the **likelihood** term

- **Likelihood:** Expresses how to generate an instance *if it knows it is of class c*

More formally….

- Recall the definition of naïve Bayes:
  - $\hat{c} = \underset{c \in C}{\text{argmax}} P(d|c)P(c)$

A discriminative model instead tries to compute P(*c*|*d*) directly!

However, naïve Bayes and logistic regression also have some similarities.

Both are **probabilistic classifiers**

Both perform **supervised machine learning**

- Recall: Supervised machine learning = ML with labeled training and test data
  - Generally formalized as *x*s (instances) and *y*s (labels), where an individual instance is an $x^{(i)}$, $y^{(i)}$ pair

Which is better …naïve Bayes or logistic regression?

- Depends on the task and the dataset
- For larger datasets, logistic regression is usually better
- For smaller datasets, naïve Bayes is sometimes better
- Naïve Bayes is easy to implement and faster to train
- **Best to experiment with multiple classification models to determine which is best for your needs**

In general, supervised machine learning systems for text classification have four main components.

- **Feature representation** of the input
  - Typically, a **vector** of features $[x_1^{(j)}, x_2^{(j)}, \ldots, x_n^{(j)}]$ for a given instance $x^{(j)}$
- **Classification function** that computes the estimated class, $\hat{y}$
  - Sigmoid
  - Softmax
  - Etc.
- **Objective function** or **loss function** that computes error values on training instances
  - Cross-entropy loss function
- **Optimization function** that seeks to minimize the loss function
  - Stochastic gradient descent

To build a logistic regression classifier….

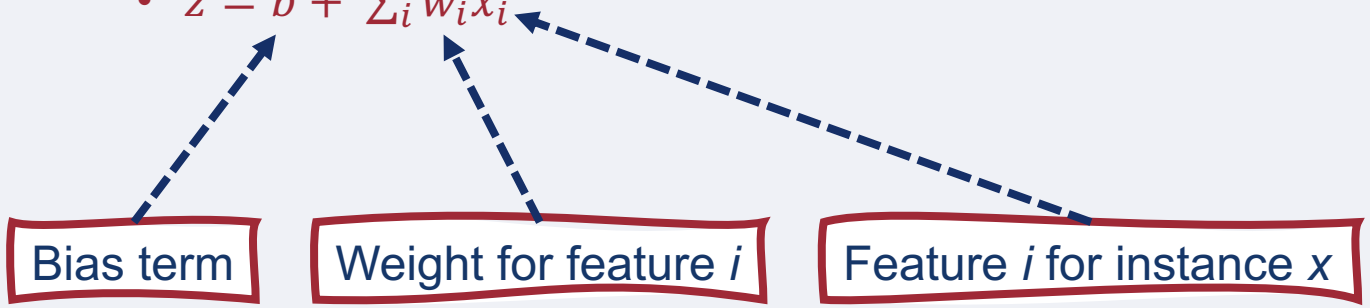
- Train **weights** *w* and a **bias** *b* using **stochastic gradient descent** and **cross-entropy loss**
- Use a **sigmoid classification function**
- Test performance by computing P(*y*|*x*) and returning the **highest-probability label**

Binary Logistic Regression

- Goal:
  - Train a classifier that can decide whether a new input observation belongs to class *a* or class *b*
- To do this, the classifier learns a **vector of weights** (one associated with each input feature) and a **bias term**
- A given **weight indicates how important its corresponding feature is** to the overall classification decision
  - Can be positive or negative
- The **bias term is a real number** that is added to the weighted inputs

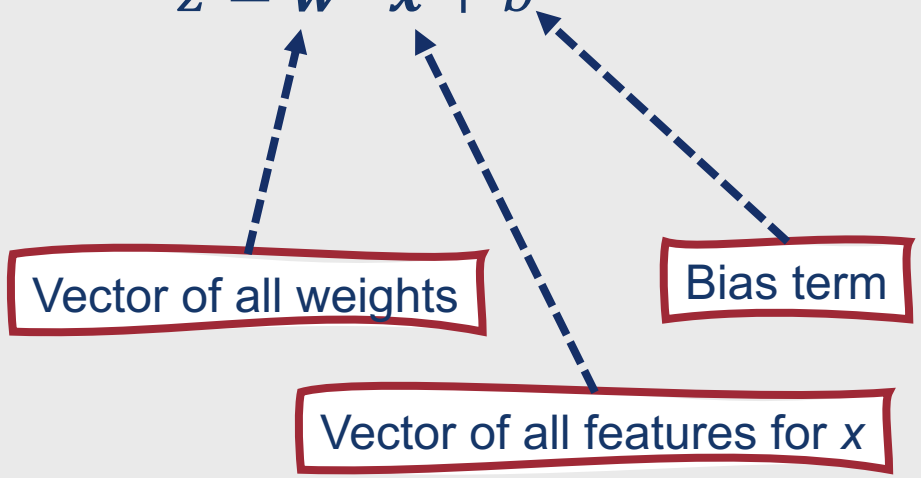
Binary Logistic Regression

- To make a classification decision, the classifier:
  - Multiplies each feature for an input instance *x* by its corresponding weight (learned from the training data)
  - Sums the weighted features
  - Adds the bias term *b*
- This results in a weighted sum of evidence for the class:
  - $z = b + \sum_i w_i x_i$



✳ Vector Notation

- Letting w be the weight vector and x be the input feature vector, we can also represent the weighted sum z using vector notation:
  - $z = \boldsymbol{w} \cdot \boldsymbol{x} + b$



Multiplying feature values by their weights means that *z* is a linear function of *x*

- What we really want is a **probability** ranging from 0 to 1
- To do this, we pass *z* through the sigmoid function, $\sigma(z)$
  - Also called the **logistic function**, hence the name **logistic regression**

Sigmoid Function

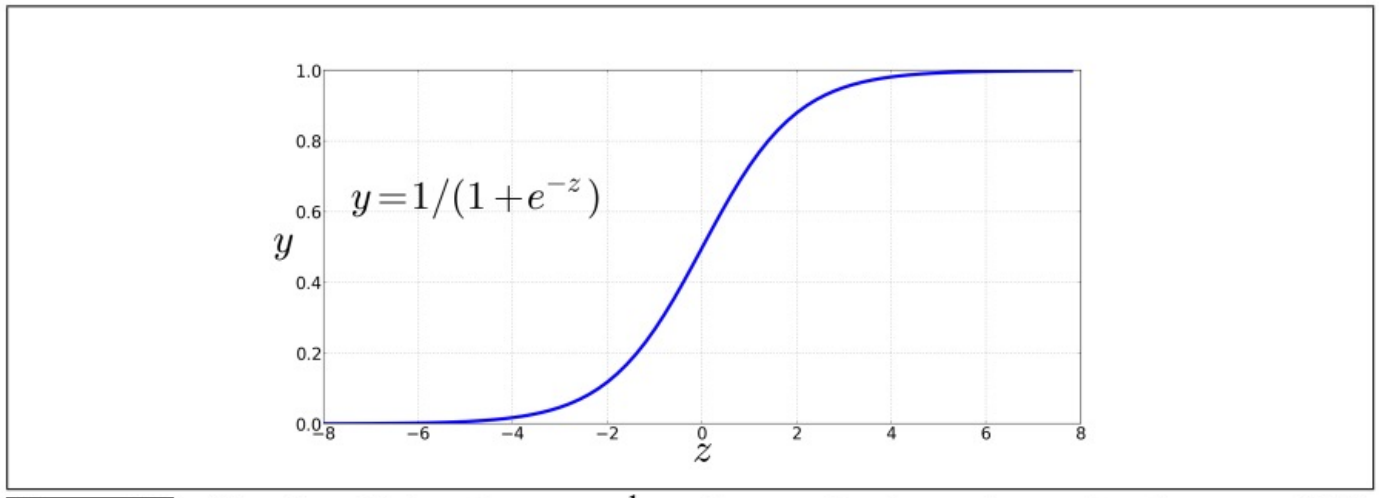


Figure 5.1 The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0, 1]$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

Source: https://web.stanford.edu/~jurafsky/slp3/5.pdf

- Sigmoid Function:
  - $\sigma(x) = \frac{1}{1+e^{-x}}$
- Given its name because when plotted, it looks like an *s*
- Results in a value *y* ranging from 0 to 1
  - $y = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-w \cdot x+b}}$

There are many useful properties of the sigmoid function!

- Maps a real-valued number to a 0 to 1 range
  - Just what we need for a probability….
- Squashes outlier values towards 0 or 1
- Differentiable
  - Necessary for learning….

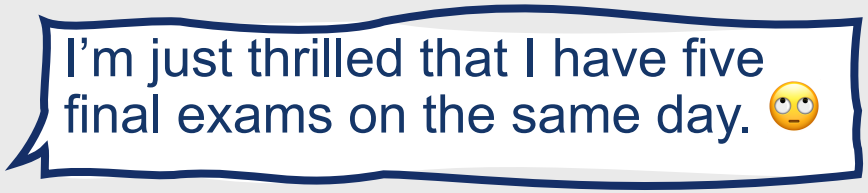
In binary logistic regression, to make the probability for all classes sum to one….

- $P(y = 1) = \sigma(z)$
- $P(y = 0) = 1 - \sigma(z)$

How do we make a classification decision?

- Choose a **decision boundary**
  - For binary classification, often 0.5
- For a test instance *x*, assign a label *c* if $P(y = c|x)$ is greater than the decision boundary
  - If performing binary classification, assign the other label if $P(y = c|x)$ is lower than or equal to the decision boundary

Example: Sigmoid Classification



Sarcastic or not sarcastic?

Example: Sigmoid Classification

I'm just thrilled that I have five final exams on the same day. 🙄

Sarcastic or not sarcastic?

Feature
Contains 🙄
Contains 😊
Contains "I'm"

Example: Sigmoid Classification

I'm just thrilled that I have five final exams on the same day. 🙄

Sarcastic or not sarcastic?

Feature	Weight
Contains 🙄	2.5
Contains 😊	-3.0
Contains "I'm"	0.5

Example: Sigmoid Classification

I'm just thrilled that I have five final exams on the same day. 🙄

Sarcastic or not sarcastic?

Feature	Weight
Contains 🙄	2.5
Contains 😊	-3.0
Contains "I'm"	0.5

Positively associated with sarcasm

Negatively associated with sarcasm

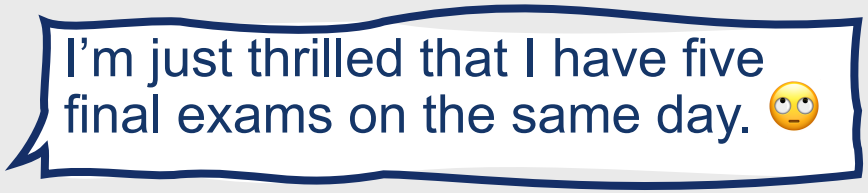
Example: Sigmoid Classification

I'm just thrilled that I have five final exams on the same day. 🙄

Sarcastic or not sarcastic?

Feature	Weight	Value
Contains 🙄	2.5	1
Contains 😊	-3.0	0
Contains "I'm"	0.5	1

Example: Sigmoid Classification



Sarcastic or not sarcastic?

Feature	Weight	Value
Contains 🙄	2.5	1
Contains 😊	-3.0	0
Contains "I'm"	0.5	1

Bias = 0.1

Example: Sigmoid Classification

I'm just thrilled that I have five final exams on the same day. 🙄

Sarcastic or not sarcastic?

Feature	Weight	Value
Contains 🙄	2.5	1
Contains 😊	-3.0	0
Contains "I'm"	0.5	1

Bias = 0.1

$$z = b + \sum_i w_i x_i$$

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

Example: Sigmoid Classification

I'm just thrilled that I have five final exams on the same day. 🙄

Sarcastic or not sarcastic?

Feature	Weight	Value
Contains 🙄	2.5	1
Contains 😊	-3.0	0
Contains "I'm"	0.5	1

Bias = 0.1

$$z = b + \sum_i w_i x_i$$

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

$$P(\text{sarcasm}|x) = \sigma\big(0.1 + (2.5 * 1 + (-3.0) * 0 + 0.5 * 1)\big) = \sigma(0.1 + 3.0) = \sigma(3.1) = \frac{1}{1+e^{-3.1}} = 0.96$$

Example: Sigmoid Classification

I'm just thrilled that I have five final exams on the same day. 🙄

Sarcastic or not sarcastic?

Feature	Weight	Value
Contains 🙄	2.5	1
Contains 😊	-3.0	0
Contains "I'm"	0.5	1

Bias = 0.1

$$z = b + \sum_i w_i x_i$$

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

$$P(\text{sarcasm}|x) = \sigma\big(0.1 + (2.5 * 1 + (-3.0) * 0 + 0.5 * 1)\big) = \sigma(0.1 + 3.0) = \sigma(3.1) = \frac{1}{1+e^{-3.1}} = 0.96$$

$$P(\text{not sarcasm}|x) = 1 - \sigma\big(0.1 + (2.5 * 1 + (-3.0) * 0 + 0.5 * 1)\big) = 1 - \sigma(0.1 + 3.0) = 1 - \sigma(3.1) = 1 - \frac{1}{1+e^{-3.1}} = 1 - 0.96 = 0.04$$

Example: Sigmoid Classification

I'm just thrilled that I have five final exams on the same day. 🙄

Sarcastic or not sarcastic?

Feature	Weight	Value
Contains 🙄	2.5	1
Contains 😊	-3.0	0
Contains "I'm"	0.5	1

Bias = 0.1

$$z = b + \sum_i w_i x_i$$

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

$$P(\text{sarcasm}|x) = \sigma\big(0.1 + (2.5 * 1 + (-3.0) * 0 + 0.5 * 1)\big) = \sigma(0.1 + 3.0) = \sigma(3.1) = \frac{1}{1+e^{-3.1}} = 0.96$$

$$P(\text{not sarcasm}|x) = 1 - \sigma\big(0.1 + (2.5 * 1 + (-3.0) * 0 + 0.5 * 1)\big) = 1 - \sigma(0.1 + 3.0) = 1 - \sigma(3.1) = 1 - \frac{1}{1+e^{-3.1}} = 1 - 0.96 = 0.04$$

Any useful (or not useful) property of the language sample can be a feature!

- For example….
  - Specific words or n-grams
  - Information from external lexicons
  - Grammatical elements
  - Part-of-speech tags
- In neural classification models, the feature vector often includes word embeddings
  - More about these soon!

Learning in Logistic Regression

- How are the parameters of a logistic regression model, *w* and *b*, learned?
  - **Loss function**
  - **Optimization function**
- Goal: Learn parameters that make $\hat{y}$ for each training observation as close as possible to the true y

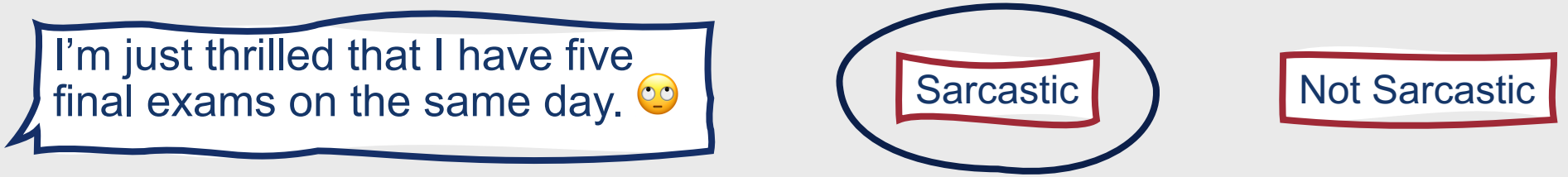
Loss Function

- We need to determine the distance between the predicted and true output value
  - How much does $\hat{y}$ differ from y?
- We do this using a **conditional maximum likelihood estimation**
  - Select *w* and *b* such that they maximize the log probability of the true *y* values in the training data, given their observations *x*
- This results in a **negative log likelihood loss**
  - More commonly referred to as **cross-entropy loss**

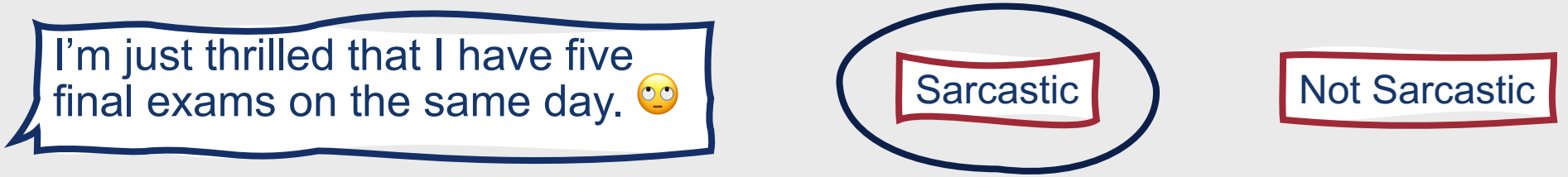
Cross-Entropy Loss

- Most common loss function for many classification tasks
- Measures the distance between the probability distributions of predicted and actual values
  - $loss(y_i, \widehat{y_i}) = -\sum_{c=1}^{|C|} p_{i,c} \log \widehat{p_{i,c}}$
    - *C* is the set of all possible classes
    - $p_{i,c}$ is the actual probability that instance *i* should be labeled with class *c*
    - $\widehat{p_{i,c}}$ is the predicted probability that instance *i* should be labeled with class *c*
- Observations with a big distance between the predicted and actual values have much higher cross-entropy loss than observations with only a small distance between the two values

Example: Cross-Entropy Loss

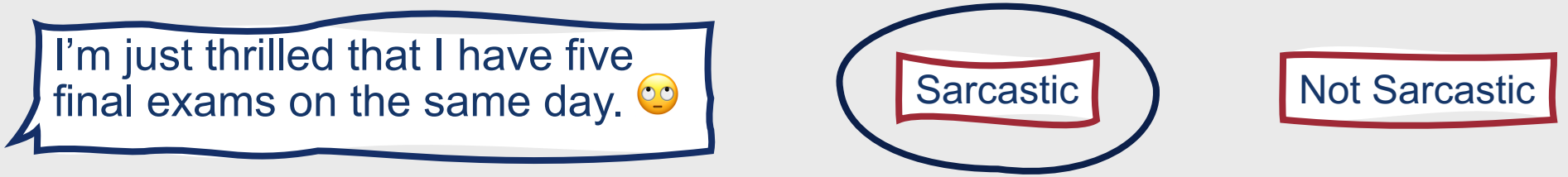


Example: Cross-Entropy Loss



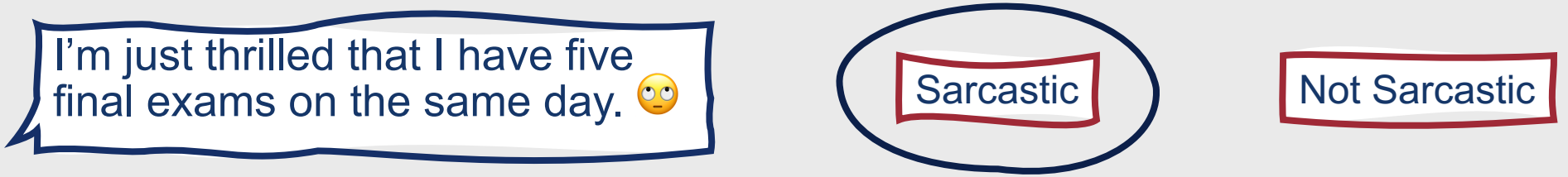
Instance	Predicted Probability: Sarcastic	Predicted Probability: Not Sarcastic	Actual Probability: Sarcastic	Actual Probability: Not Sarcastic
I'm just thrilled that I have five final exams on the same day. 🙄			1	0

Example: Cross-Entropy Loss



Instance	Predicted Probability: Sarcastic	Predicted Probability: Not Sarcastic	Actual Probability: Sarcastic	Actual Probability: Not Sarcastic
I'm just thrilled that I have five final exams on the same day. 🙄	0.96	0.04	1	0

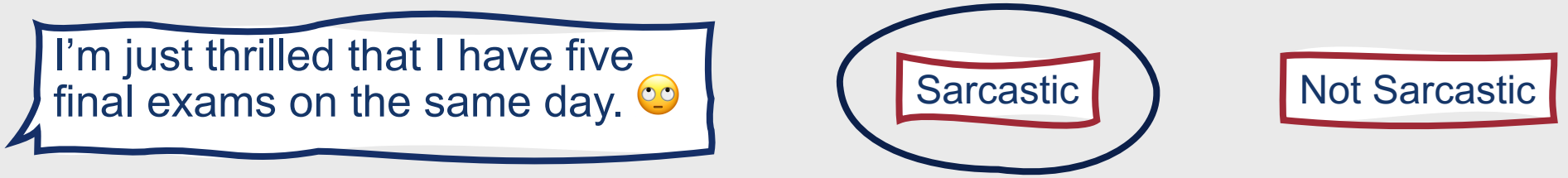
Example: Cross-Entropy Loss



Instance	Predicted Probability: Sarcastic	Predicted Probability: Not Sarcastic	Actual Probability: Sarcastic	Actual Probability: Not Sarcastic
I'm just thrilled that I have five final exams on the same day. 🙄	0.96	0.04	1	0

$$loss(y_i, y_i') = -\sum_{c=1}^{|C|} p_{i,c} \log \widehat{p_{i,c}} = -p_{i,sarcastic} \log \widehat{p_{i,sarcastic}} - p_{i,not\ sarcastic} \log \widehat{p_{i,not\ sarcastic}}$$

Example: Cross-Entropy Loss

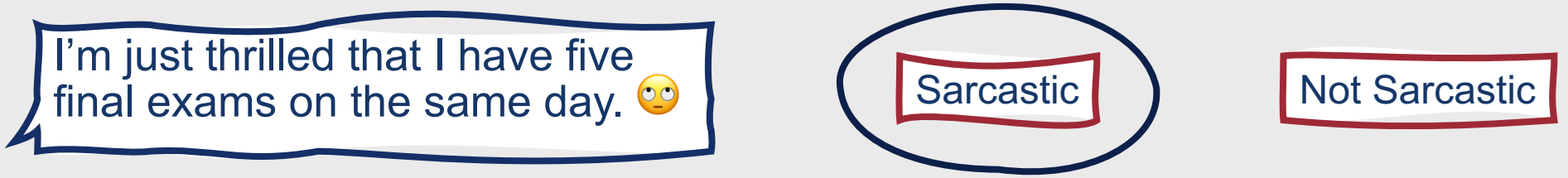


Instance	Predicted Probability: Sarcastic	Predicted Probability: Not Sarcastic	Actual Probability: Sarcastic	Actual Probability: Not Sarcastic
I'm just thrilled that I have five final exams on the same day. 🙄	0.96	0.04	1	0

$$loss(y_i, y_i') = -\sum_{c=1}^{|C|} p_{i,c} \log \widehat{p_{i,c}} = -p_{i,sarcastic} \log \widehat{p_{i,sarcastic}} - p_{i,not\ sarcastic} \log \widehat{p_{i,not\ sarcastic}}$$

$$loss(y_i, y_i') = -1 * \log 0.96 - 0 * \log 0.04$$

Example: Cross-Entropy Loss

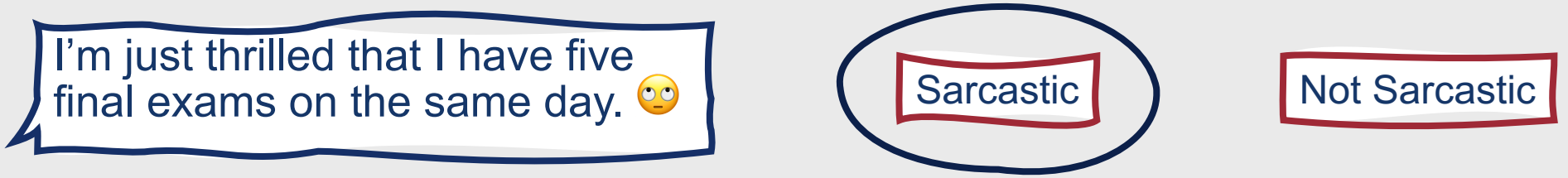


Instance	Predicted Probability: Sarcastic	Predicted Probability: Not Sarcastic	Actual Probability: Sarcastic	Actual Probability: Not Sarcastic
I'm just thrilled that I have five final exams on the same day. 🙄	0.96	0.04	1	0

$$loss(y_i, y_i') = -\sum_{c=1}^{|C|} p_{i,c} \log \widehat{p_{i,c}} = -p_{i,sarcastic} \log \widehat{p_{i,sarcastic}} - p_{i,not\ sarcastic} \log \widehat{p_{i,not\ sarcastic}}$$

$$loss(y_i, y_i') = -1 * \log 0.96 - 0 * \log 0.04 = -\log 0.96 = 0.02$$

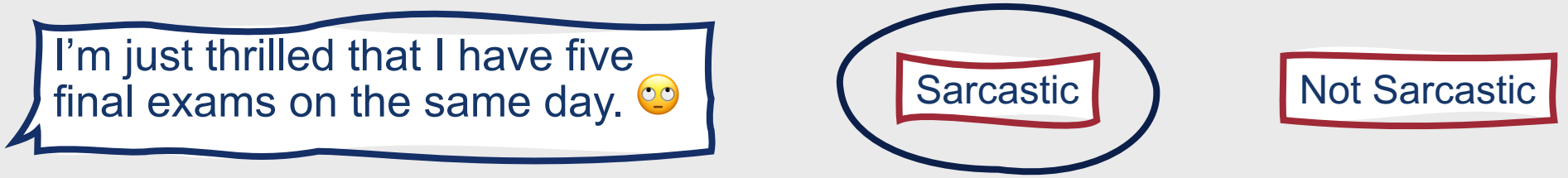
Example: Cross-Entropy Loss



Instance	Predicted Probability: Sarcastic	Predicted Probability: Not Sarcastic	Actual Probability: Sarcastic	Actual Probability: Not Sarcastic
I'm just thrilled that I have five final exams on the same day. 🙄			1	0

What if our predicted values were switched?

Example: Cross-Entropy Loss



Instance	Predicted Probability: Sarcastic	Predicted Probability: Not Sarcastic	Actual Probability: Sarcastic	Actual Probability: Not Sarcastic
I'm just thrilled that I have five final exams on the same day. 🙄	0.04	0.96	1	0

$$loss(y_i, y_i') = -\sum_{c=1}^{|C|} p_{i,c} \log \widehat{p_{i,c}} = -p_{i,sarcastic} \log \widehat{p_{i,sarcastic}} - p_{i,not\ sarcastic} \log \widehat{p_{i,not\ sarcastic}}$$

$$loss(y_i, y_i') = -1 * \log 0.04 - 0 * \log 0.96 = -\log 0.04 = 1.40$$

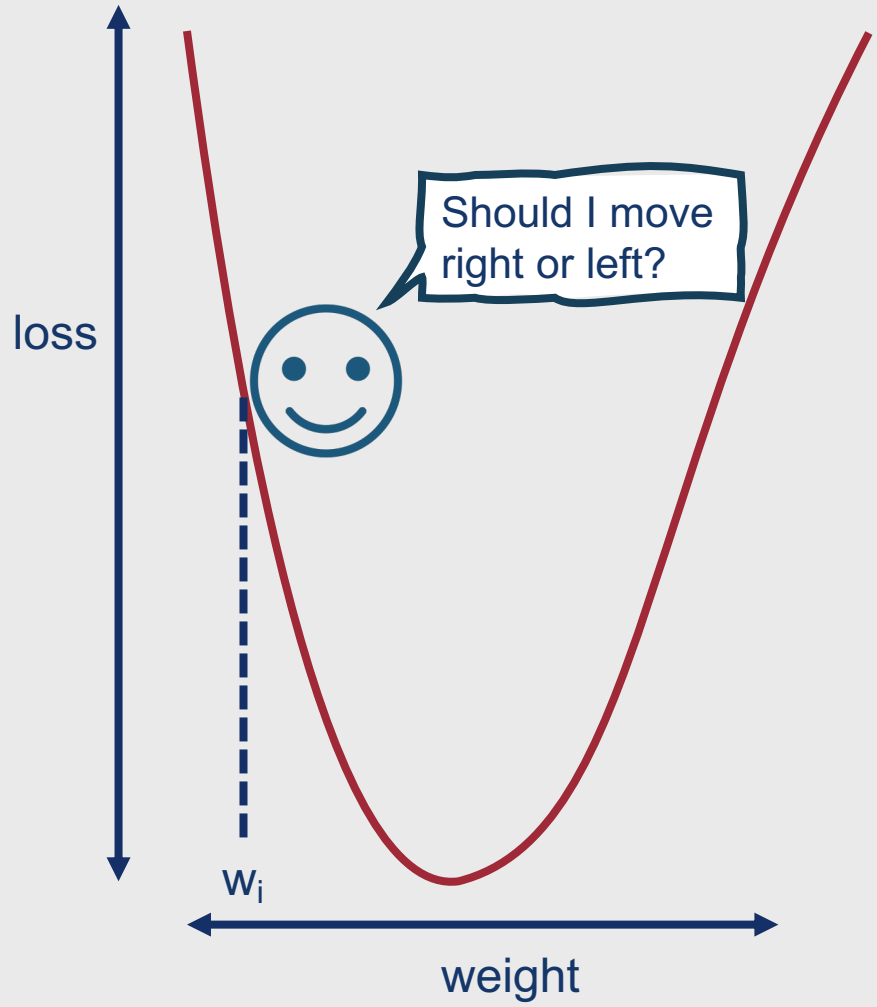
Greater loss value!

Finding Optimal Weights

- Goal: Minimize the loss function defined for the model
  - $\hat{\theta} = \underset{\theta}{\text{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y^{(i)}, x^{(i)}; \theta)$
- For logistic regression, $\theta = w, b$
- One way to do this is by using **gradient descent**

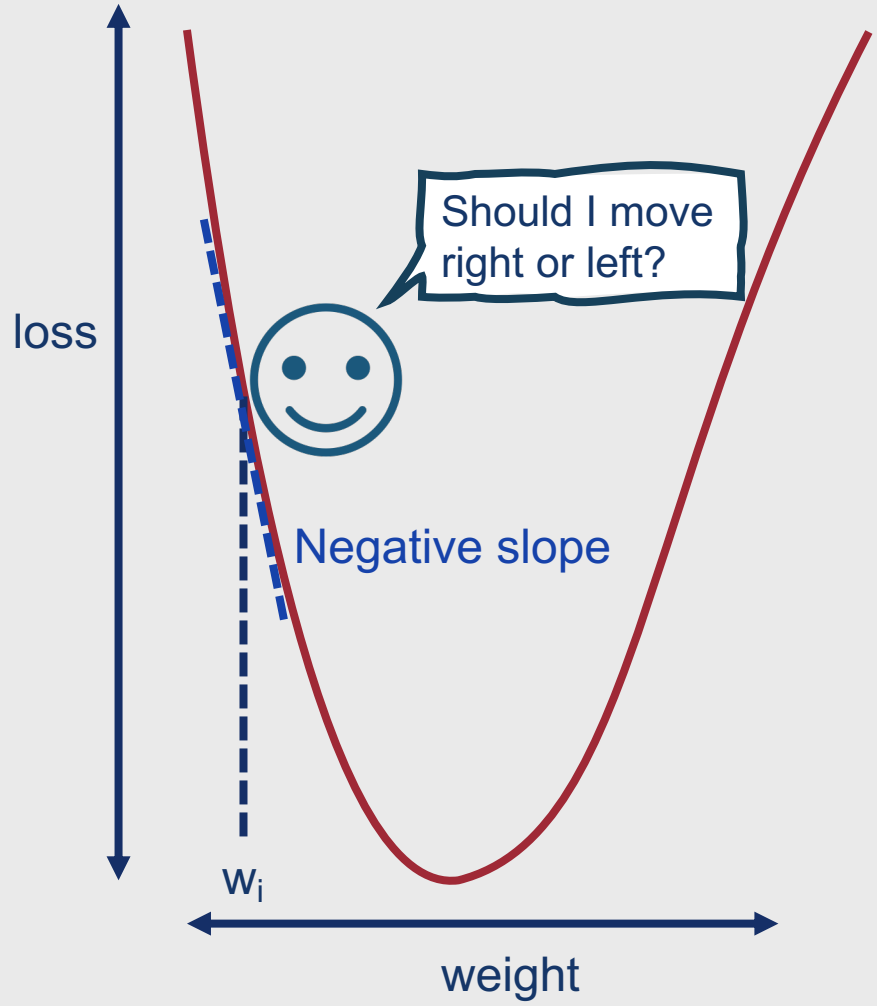
Gradient Descent

- Finds the minimum of a function by:
  - Figuring out the direction (in the space of θ) the function's slope
  - Moving in the opposite direction
- For logistic regression, loss functions are **convex**
  - Only one minimum
  - Gradient descent starting at any point is guaranteed to find it



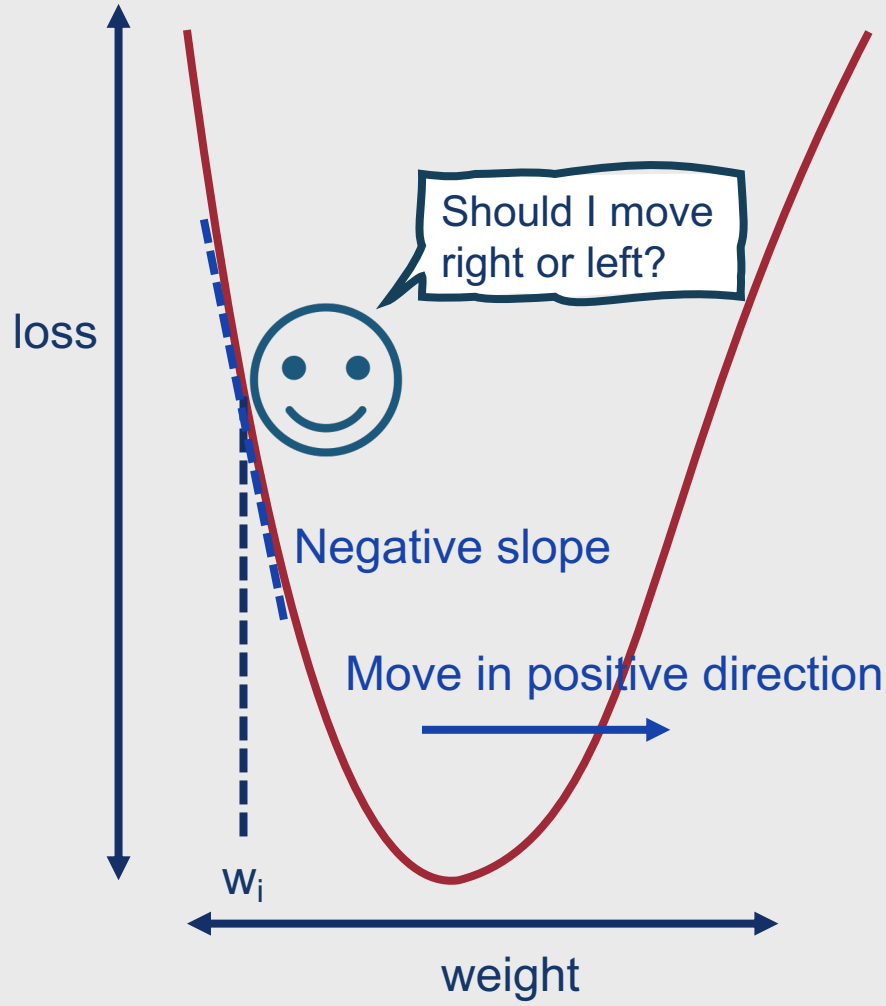
Gradient Descent

- Finds the minimum of a function by:
  - Figuring out the direction (in the space of θ) the function's slope
  - Moving in the opposite direction
- For logistic regression, loss functions are **convex**
  - Only one minimum
  - Gradient descent starting at any point is guaranteed to find it



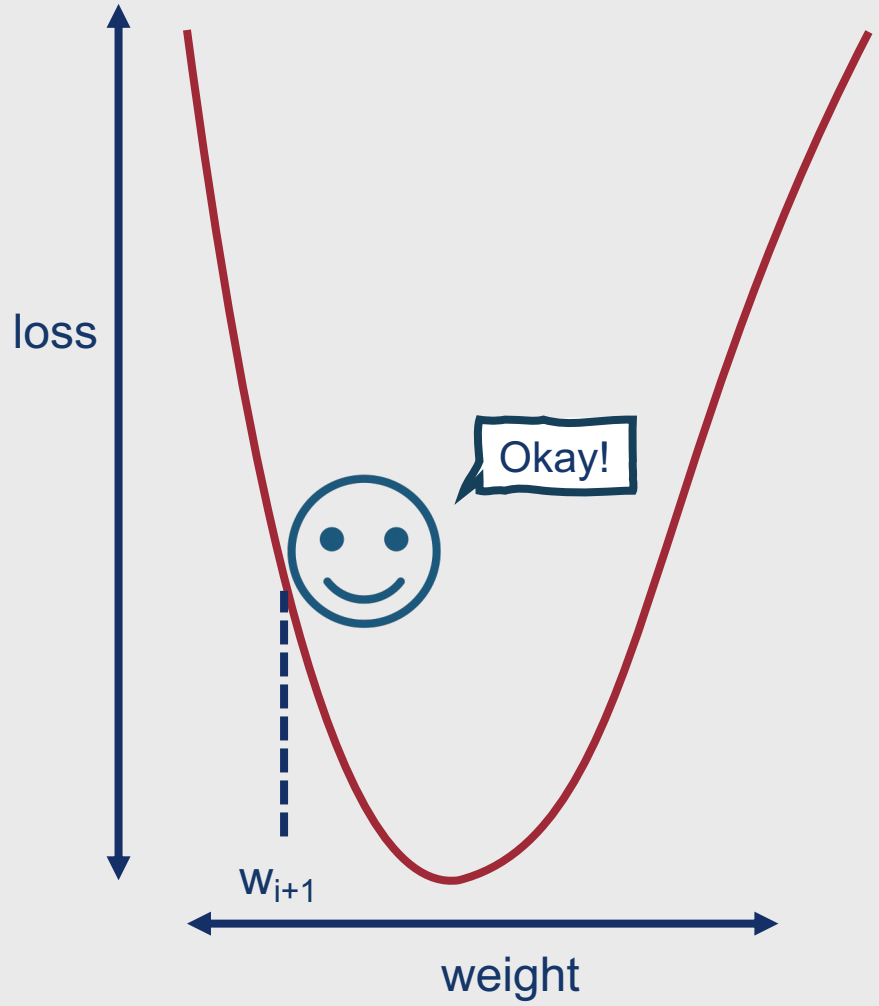
Gradient Descent

- Finds the minimum of a function by:
  - Figuring out the direction (in the space of θ) the function's slope
  - Moving in the opposite direction
- For logistic regression, loss functions are **convex**
  - Only one minimum
  - Gradient descent starting at any point is guaranteed to find it



Gradient Descent

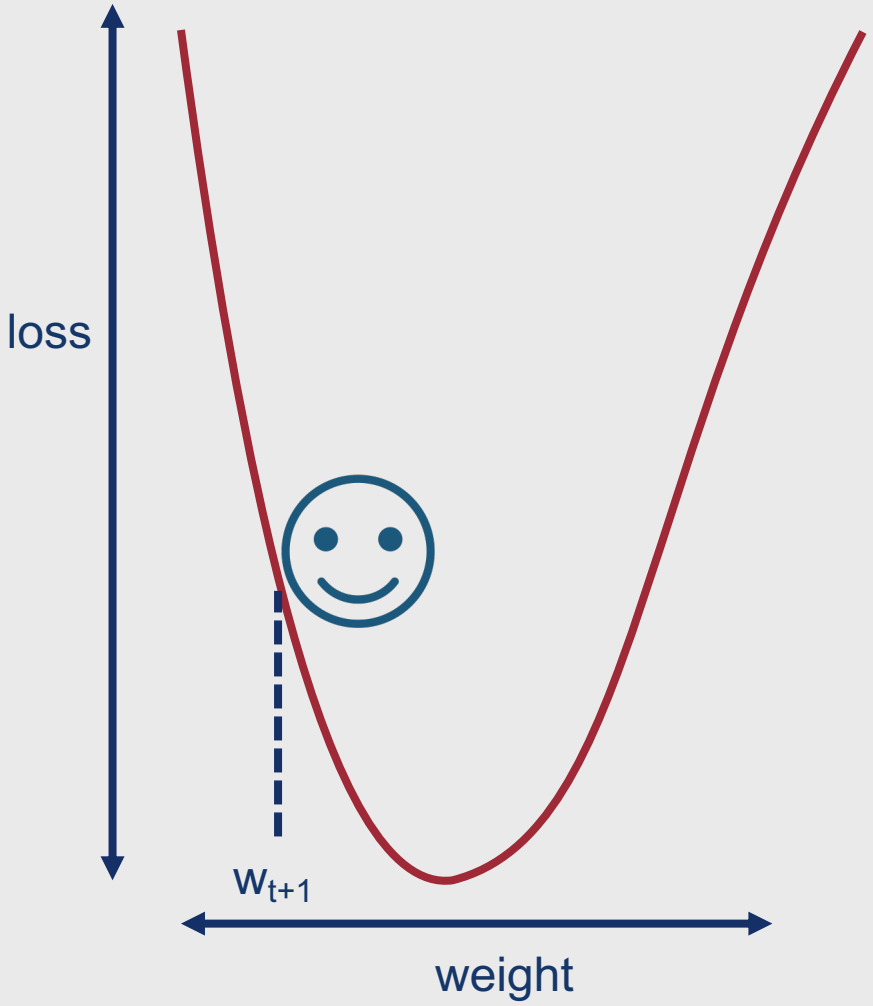
- Finds the minimum of a function by:
  - Figuring out the direction (in the space of θ) the function's slope
  - Moving in the opposite direction
- For logistic regression, loss functions are **convex**
  - Only one minimum
  - Gradient descent starting at any point is guaranteed to find it



Gradient Descent

- How much do we move?
  - Value of the slope
    - $\frac{d}{dw} f(x; w)$
  - Weighted by a learning rate η
- Faster learning rate → move *w* more on each step
- So, the change to a weight at time *t* is actually:
  - $w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$

Derivative of loss function curve with respect to a given weight



Remember, there are weights for each feature.

- The gradient is then a vector of the slopes of each dimension:
  - $\nabla_\theta L(f(x;\theta),y) = \begin{bmatrix} \frac{d}{dw_1} L(f(x;\theta),y) \\ \dots \\ \frac{d}{dw_n} L(f(x;\theta),y) \end{bmatrix}$
- This in turn means that the final equation for updating θ is:
  - $\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$

The Gradient for Logistic Regression

- Recall our cross-entropy loss function:
  - $loss(y_i, \hat{y}_i) = -\sum_{c=1}^{|C|} y \log \hat{y} = -\sum_{c=1}^{|C|} y \log \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$
- The derivative for this function is:
  - $\frac{dL_{CE}(w,b)}{dw_j} = [\sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b) - y]x_j$

Difference between true and estimated *y*

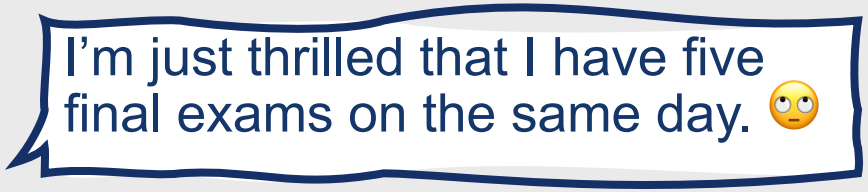
Corresponding input observation

Stochastic Gradient Descent Algorithm

$\theta \leftarrow 0$ # initialize weights to 0
repeat until convergence:
    For each training instance $(x^{(i)}, y^{(i)})$ in random order:
        # What is our gradient, given our current parameters?
        $g \leftarrow \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)$

        $\theta \leftarrow \theta - \eta g$ # What are our updated parameters?
    return θ

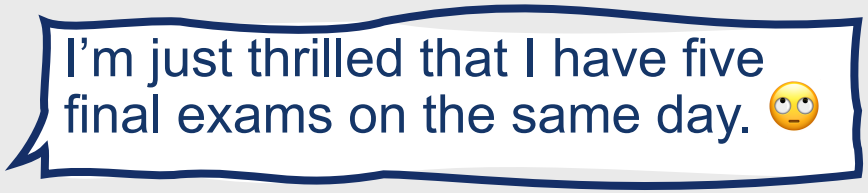
Example: Gradient Descent (First Step)



Sarcastic

Feature	Weight	Value
Contains 🙄	0	1
Contains 😊	0	0
Contains "I'm"	0	1

Example: Gradient Descent (First Step)



Sarcastic

Feature	Weight	Value
Contains 🙄	0	1
Contains 😊	0	0
Contains "I'm"	0	1

Bias (b) = 0
Learning rate (η) = 0.1

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)$$

Example: Gradient Descent (First Step)

I'm just thrilled that I have five final exams on the same day. 🙄

← Sarcastic

Feature	Weight	Value
Contains 🙄	0	1
Contains 😊	0	0
Contains "I'm"	0	1

Bias (b) = 0
Learning rate (η) = 0.1

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right)$$

$$\nabla_\theta L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right) = \begin{bmatrix} \frac{dL_{CE}(w,b)}{dw_1} \\ \frac{dL_{CE}(w,b)}{dw_2} \\ \frac{dL_{CE}(w,b)}{dw_3} \\ \frac{dL_{CE}(w,b)}{db} \end{bmatrix} = \begin{bmatrix} (\sigma(w\cdot x+b)-y)x_1 \\ (\sigma(w\cdot x+b)-y)x_2 \\ (\sigma(w\cdot x+b)-y)x_3 \\ \sigma(w\cdot x+b)-y \end{bmatrix} = \begin{bmatrix} (\sigma(0)-1)x_1 \\ (\sigma(0)-1)x_2 \\ (\sigma(0)-1)x_3 \\ \sigma(0)-1 \end{bmatrix} = \begin{bmatrix} (0.5-1)x_1 \\ (0.5-1)x_2 \\ (0.5-1)x_3 \\ (0.5-1) \end{bmatrix} = \begin{bmatrix} -0.5*1 \\ -0.5*0 \\ -0.5*1 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0 \\ -0.5 \\ -0.5 \end{bmatrix}$$

Example: Gradient Descent (First Step)

I'm just thrilled that I have five final exams on the same day. 🙄

← Sarcastic

Feature	Weight	Value
Contains 🙄	0	1
Contains 😊	0	0
Contains "I'm"	0	1

Bias (b) = 0
Learning rate (η) = 0.1

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)$$

$$\nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right) = \begin{bmatrix} -0.5 \\ 0 \\ -0.5 \\ -0.5 \end{bmatrix}$$

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \eta \begin{bmatrix} -0.5 \\ 0 \\ -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - 0.1 \begin{bmatrix} -0.5 \\ 0 \\ -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} -0.05 \\ 0 \\ -0.05 \\ -0.05 \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0 \\ 0.05 \\ 0.05 \end{bmatrix}$$

Example: Gradient Descent (First Step)

I'm just thrilled that I have five final exams on the same day. 🙄

← Sarcastic

Feature	Weight	Value
Contains 🙄	0	1
Contains 😊	0	0
Contains "I'm"	0	1

Bias (b) = 0
Learning rate (η) = 0.1

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)$$

$$\nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right) = \begin{bmatrix} -0.5 \\ 0 \\ -0.5 \\ -0.5 \end{bmatrix}$$

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \eta \begin{bmatrix} -0.5 \\ 0 \\ -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - 0.1 \begin{bmatrix} -0.5 \\ 0 \\ -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} -0.05 \\ 0 \\ -0.05 \\ -0.05 \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0 \\ 0.05 \\ 0.05 \end{bmatrix}$$

Example: Gradient Descent (Second Step)

I'm just thrilled that I have five final exams on the same day. 🙄

← Sarcastic

Feature	Weight	Value
Contains 🙄	0.05	1
Contains 😊	0	0
Contains "I'm"	0.05	1

Bias (b) = 0.05
Learning rate (η) = 0.1

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right)$$

$$\nabla_\theta L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right) = \begin{bmatrix} \frac{dL_{CE}(w,b)}{dw_1} \\ \frac{dL_{CE}(w,b)}{dw_2} \\ \frac{dL_{CE}(w,b)}{dw_3} \\ \frac{dL_{CE}(w,b)}{db} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ (\sigma(w \cdot x + b) - y)x_3 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0.05 * 1 + 0 * 0 + 0.05 * 1 + .05) - 1)x_1 \\ (\sigma(0.05 * 1 + 0 * 0 + 0.05 * 1 + .05) - 1)x_2 \\ (\sigma(0.05 * 1 + 0 * 0 + 0.05 * 1 + .05) - 1)x_3 \\ \sigma(0.05 * 1 + 0 * 0 + 0.05 * 1 + .05) - 1 \end{bmatrix} = \begin{bmatrix} (\sigma(0.15) - 1)x_1 \\ (\sigma(0.15) - 1)x_2 \\ (\sigma(0.15) - 1)x_3 \\ \sigma(0.15) - 1 \end{bmatrix} = \begin{bmatrix} (0.54 - 1)x_1 \\ (0.54 - 1)x_2 \\ (0.54 - 1)x_3 \\ (0.54 - 1) \end{bmatrix} = \begin{bmatrix} -0.46 * 1 \\ -0.46 * 0 \\ -0.46 * 1 \\ -0.46 \end{bmatrix} = \begin{bmatrix} -0.46 \\ 0 \\ -0.46 \\ -0.46 \end{bmatrix}$$

Example: Gradient Descent (Second Step)

I'm just thrilled that I have five final exams on the same day. 🙄

← Sarcastic

Feature	Weight	Value
Contains 🙄	0.05	1
Contains 😊	0	0
Contains "I'm"	0.05	1

Bias (b) = 0.05
Learning rate (η) = 0.1

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)$$

$$\nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right) = \begin{bmatrix} -0.46 \\ 0 \\ -0.46 \\ -0.46 \end{bmatrix}$$

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right) = \begin{bmatrix} 0.05 \\ 0 \\ 0.05 \\ 0.05 \end{bmatrix} - \eta \begin{bmatrix} -0.46 \\ 0 \\ -0.46 \\ -0.46 \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0 \\ 0.05 \\ 0.05 \end{bmatrix} - 0.1 \begin{bmatrix} -0.46 \\ 0 \\ -0.46 \\ -0.46 \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0 \\ 0.05 \\ 0.05 \end{bmatrix} - \begin{bmatrix} -0.046 \\ 0 \\ -0.046 \\ -0.046 \end{bmatrix} = \begin{bmatrix} 0.096 \\ 0 \\ 0.096 \\ 0.096 \end{bmatrix}$$

Mini-Batch Training

- Stochastic gradient descent chooses a single random example at a time …this can result in choppy movements!
- Often, the gradient will be computed over batches of training instances rather than a single instance
- **Batch training:** Gradient is computed over the entire dataset
  - Perfect direction, but very computationally expensive
- **Mini-batch training:** Gradient is computed over a group of *m* examples

Mini-Batch Versions of Cross-Entropy Loss and Gradient

- Cross-Entropy Loss:
  - $L_{CE}(\text{training samples}) = -\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$
- Gradient:
  - $\frac{d\theta}{dw_j} = \frac{1}{m}\sum_{i=1}^{m}\left[\sigma\left(w \cdot x^{(i)} + b\right) - y^{(i)}\right]x_j^{(i)}$

m training samples

Training sample *i*

Regularization

- To avoid **overfitting**, regularization terms ($R(\theta)$) are usually added to the loss function
- These terms are used to penalize large weights (which can hinder a model's ability to generalize)
- Two common **regularization terms**:
  - L2 regularization
  - L1 regularization

L2 Regularization

- Quadratic function of the weight values
- Square of the L2 norm (Euclidean distance of θ from the origin)
  - $R(\theta) = \|\theta\|_2^2 = \sum_{j=1}^{n} \theta_j^2$

L1 Regularization

- Linear function of the weight values
- Sum of the absolute values of the weights (Manhattan distance from the origin)
  - $R(\theta) = \|\theta\|_1 = \sum_{i=1}^{n} |\theta_i|$

Which regularization term is better?

- L2 regularization is easier to optimize (simpler derivative)
- L2 regularization → weight vectors with many small weights
- L1 regularization → sparse weight vectors with some larger weights

Multinomial Logistic Regression

- Other names:
  - Softmax regression
  - Maxent classification (short for maximum entropy classification)
- Uses a **softmax** function rather than a sigmoid function
- Softmax takes a vector **z** of arbitrary values (same as the sigmoid function) and maps them to a probability distribution summing to 1
  - $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{|\mathbf{Z}|} e^{z_j}}$

Interpreting Models

- What if we want to know more than just the correct classification?
  - Why did the classifier make the decision it made?
- In these cases, we can say we want our model to be **interpretable**
- We can interpret logistic regression models by determining how much weight is associated with a given feature

This is a key advantage of logistic regression over neural models.

- Manually-defined features facilitate **interpretability**
- **Implicitly-learned** features can be very difficult to interpret!
- Because of this, some researchers may choose to use logistic regression rather than neural models if they are particularly interested in which factors are influencing the model's decisions
  - Common example: Healthcare applications
- This allows logistic regression to function not only as a simple classification model, but as a powerful analytic tool

Summary: Logistic Regression

- **Logistic regression** is a **discriminative** classification model used for **supervised machine learning**
- It is characterized by four key components:
  - **Feature representation**
  - **Classification function**
  - **Loss function**
  - **Optimization function**
- Classification decisions are made using a **sigmoid** function for binary logistic regression, or a **softmax** function for multinomial logistic regression
- Loss is typically computed using a **cross-entropy** function
- Weights are usually optimized using **stochastic gradient descent**
- A **regularization** term may be added to the loss function to avoid overfitting
- In addition to serving as a simple **classifier** and a useful **foundation for neural networks**, logistic regression can function as a powerful **analytic tool**

We've learned about a variety of text classification techniques….

- **Hidden Markov Models**
- **Naïve Bayes**
- **Logistic Regression**

Types of Classification Techniques

Label Type

- **Individual Labels**
  - Naïve Bayes
  - Logistic Regression
- **Sequences of Labels**
  - Hidden Markov Models
  - Conditional Random Fields

Model Type

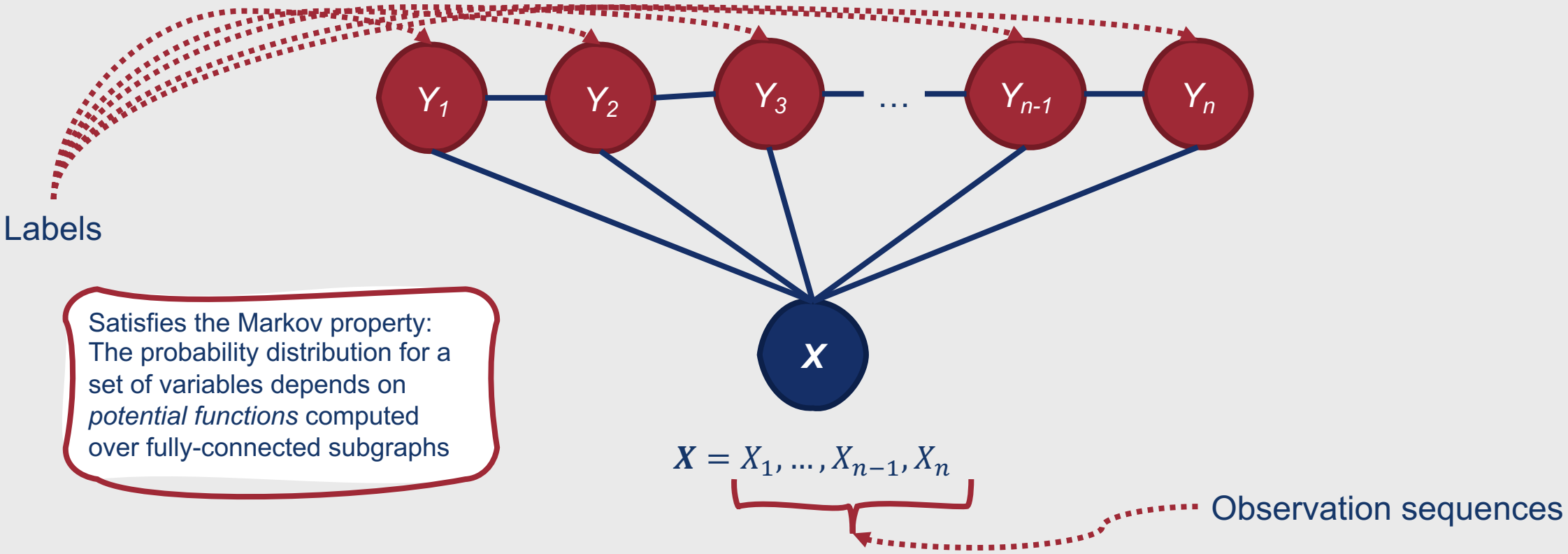
- **Generative**
  - Naïve Bayes
  - Hidden Markov Models
- **Discriminative**
  - Logistic Regression
  - Conditional Random Fields

Conditional Random Fields (CRFs)

- **Generalized multi-class logistic regression**
- Increased flexibility for sequence labeling
  - **HMMs: Joint probability** ranging over observations and corresponding labels
    - Can lead to rigid (and inaccurate) independence assumptions
  - **CRFs: Conditional probability** over label sequences given specific sequence of observations
    - Relaxes independence assumptions (model may more easily capture arbitrary or long-range dependencies)

Special Case of Markov Random Fields

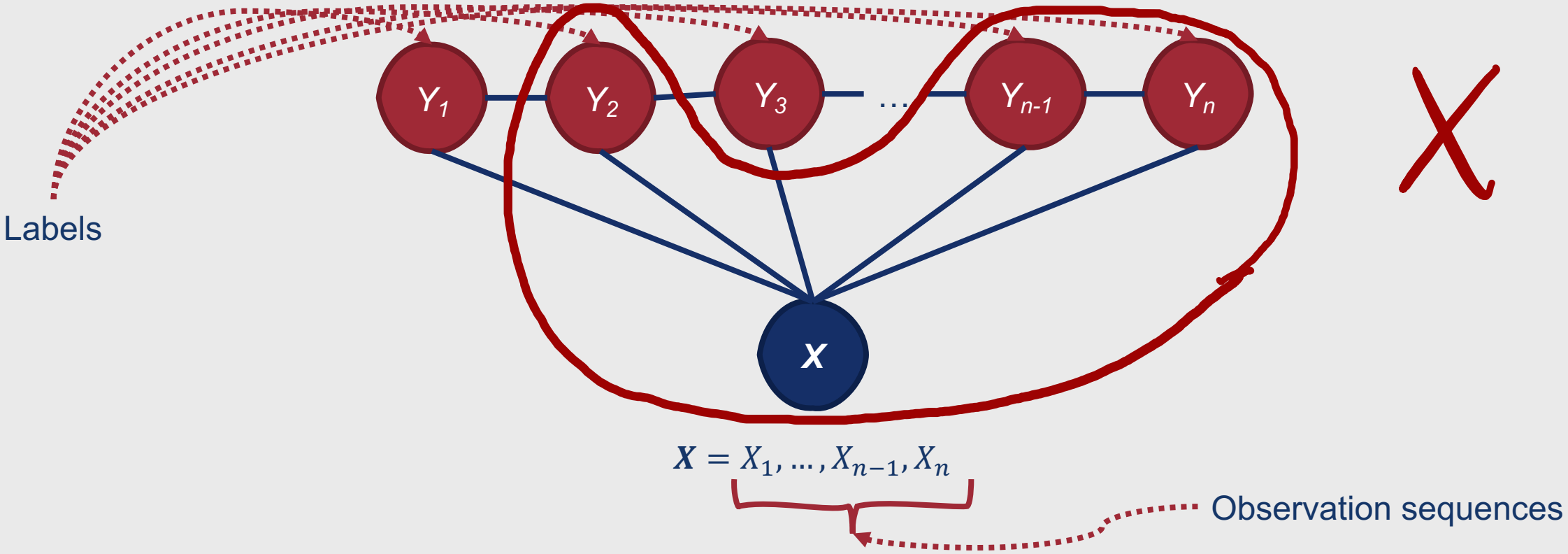
- Undirected graphical model



Special Case of Markov Random Fields

- Undirected graphical model

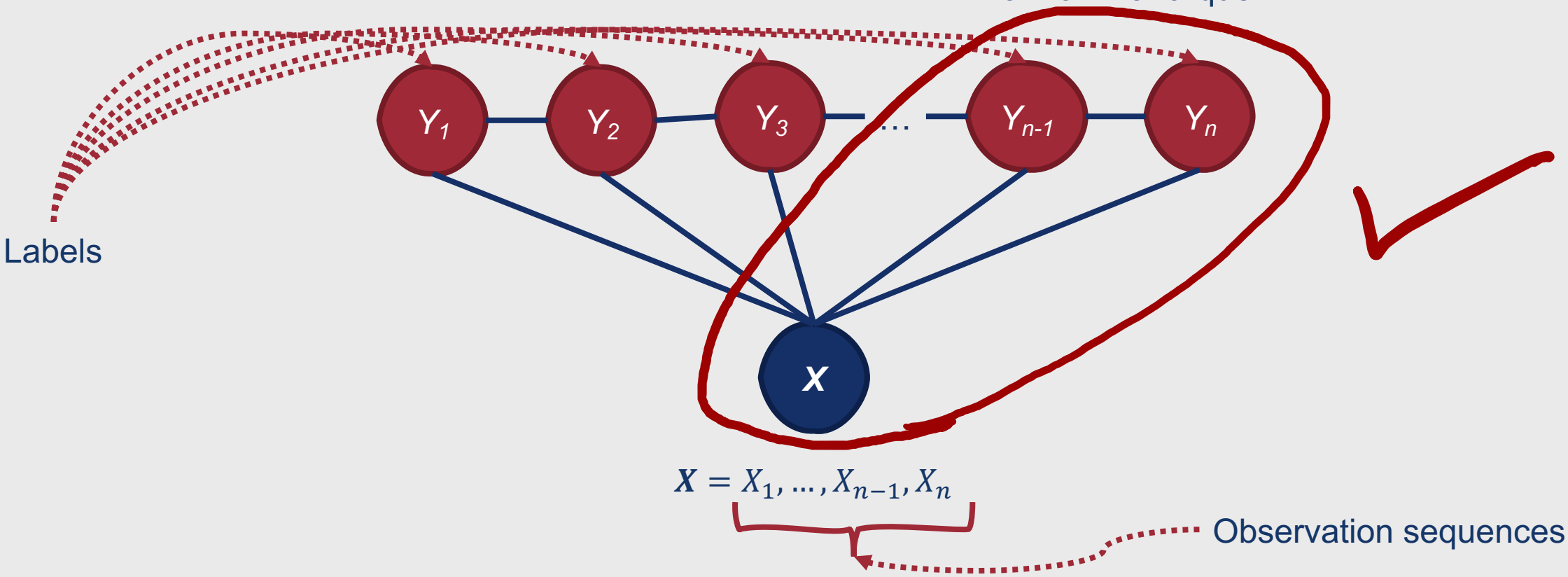
Conditionally independent labels cannot appear in the same potential function



Special Case of Markov Random Fields

- Undirected graphical model

Instead, require potential functions to operate only on random variables forming a maximal clique



Conditional Random Fields

- Probability of label sequence *Y* given observation sequence *X* is then a normalized product of feature functions
  - $p(Y|X) = \frac{1}{Z(x)} e^{\sum_{k=1}^{K} w_k F_k(X,Y)}$

Normalization factor

Global feature function (a property of the entire input sequence *X* and output sequence *Y*)

Computed by decomposing into a sum of local features for each position *i* in *Y*

Training CRFs

- Seek to find the model distribution with maximum entropy
- Thus, parameters can be optimized by minimizing cross-entropy loss using stochastic gradient descent

How do we find the best tag sequence for a given input?

- We can turn to an old favorite …dynamic programming!
  - Viterbi-like algorithm where prior probabilities and likelihoods are replaced with CRF feature functions
    - $v_t(j) = \max_{i \in \{1,\ldots,n\}} v_{t-1}(i) \sum_{k=1}^{K} w_k f_k(y_{t-1}, y_t, X, t)$
  - Fill in a table with the appropriate values, keeping track of backpointers as you go
  - When the table is filled, follow the backpointers from the maximum value back to the beginning of the table to get the best sequence of labels

Additional Details about CRFs

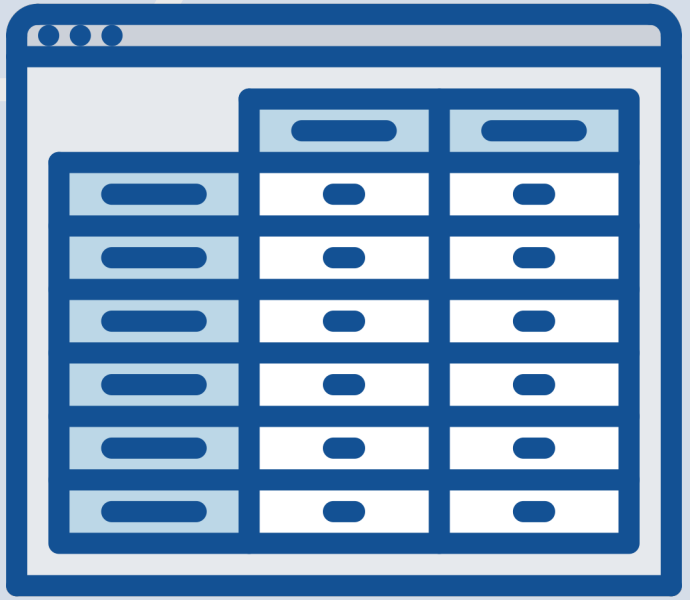
- http://dirichlet.net/pdf/wallach04conditional.pdf
- http://pages.cs.wisc.edu/~jerryzhu/cs769/CRF.pdf
- https://people.cs.umass.edu/~mccallum/papers/crf-tutorial.pdf

Overview of Vector Semantics

- **Vector semantics** facilitates a form of **representation learning** based on the notion that similar words tend to occur in similar environments
- Representations that encode these contextual similarities are often called **word embeddings**

Representation Learning

- The process of learning useful representations of input text
- Modern representation learning is **self-supervised**
  - Doesn't require manually-defined features or labels



DISTRIBUTIONAL STRUCTURE

ZELLIG S. HARRIS

1. Does language have a distributional structure? For the purposes of the present discussion, the term structure will be used in the following non-rigorous sense: A set of phonemes or a set of data is structured in respect to some feature, to the extent that we can form in terms of that feature some organized system of statements which describes the members of the set and their interrelations (at least up to some limit of complexity). In this sense, language can be structured in respect to various independent features. And whether it is structured (to more than a trivial extent) in respect to, say, regular historical change, social intercourse, meaning, or distribution—or to what extent it is structured in any of these respects—is a matter decidable by investigation. Here we will discuss how each language can be described in terms of a distributional structure, i.e. in terms of the occurrence of parts (ultimately sounds) relative to other parts, and how this description is complete without intrusion of other features such as history or meaning. It goes without saying that other studies of language—historical, psychological, etc.—are also possible, both in relation to distributional structure and independently of it.

The distribution of an element will be understood as the sum of all its environments. An environment of an element A is an existing array of its co-occurrents, i.e. the other elements, each in a particular position, with which A occurs to yield an utterance. A's co-occurrents in a particular position are called its selection for that position.

1.1. Possibilities of structure for the distributional facts.

To see that there can be a distributional structure we note the following: First, the parts of a language do not occur arbitrarily relative to each other: each element occurs in certain positions relative to certain other elements. The perennial man in the street believes that when he speaks he freely puts together whatever elements have the meanings he intends; but he does so only by choosing members of those classes that regularly occur together, and in the order in which these classes occur.

Second, the restricted distribution of classes persists for all their occurrences; the restrictions are not disregarded arbitrarily, e.g. for semantic needs. Some logicians, for example, have considered that an exact distributional description of natural languages is impossible because of their inherent vagueness. This is not quite the case. All elements in a language can be grouped into classes whose relative occurrence can be stated exactly. However, for the occurrence of a particular member of one class relative to a particular member of another class it would be necessary to speak in terms of probability, based on the frequency of that occurrence in a sample.

Third, it is possible to state the occurrence of any element relative to any other element, to the degree of exactness indicated above, so that distributional state-

The corresponding notion of encoding words based on their distribution is referred to as the distributional hypothesis.

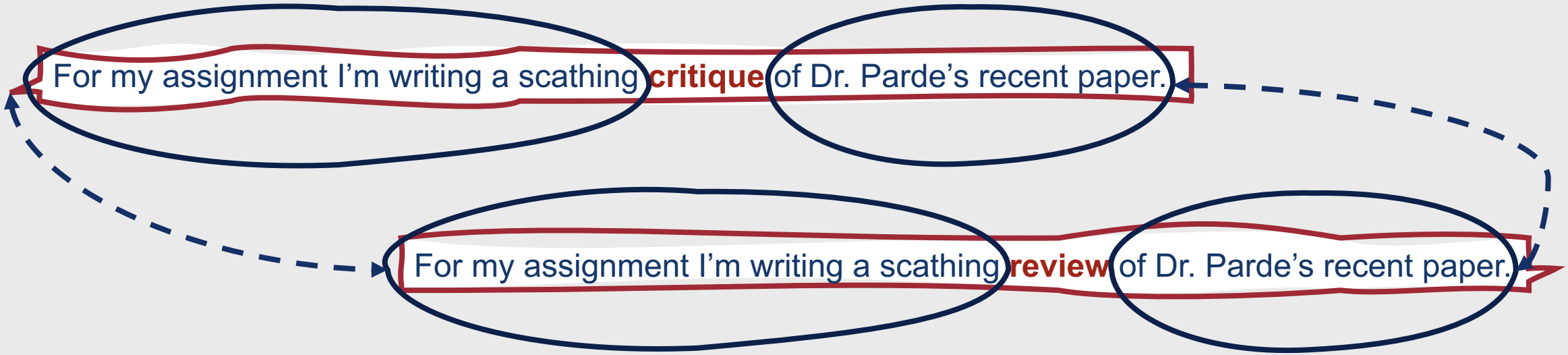
- First formulated by linguists in the 1950s
  - Joos (1950)
  - Harris (1954)
  - Firth (1957)

Vector Semantics

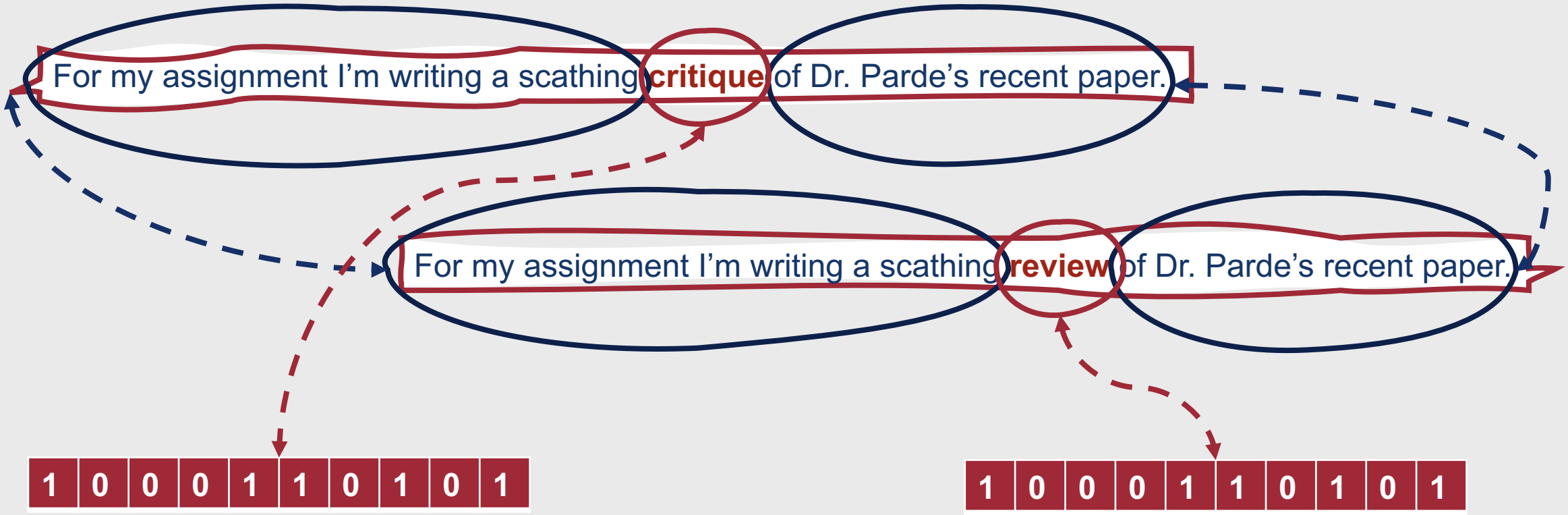
For my assignment I'm writing a scathing **critique** of Dr. Parde's recent paper.

For my assignment I'm writing a scathing **review** of Dr. Parde's recent paper.

Vector Semantics



Vector Semantics



There are many ways to make use of the distributional hypothesis!

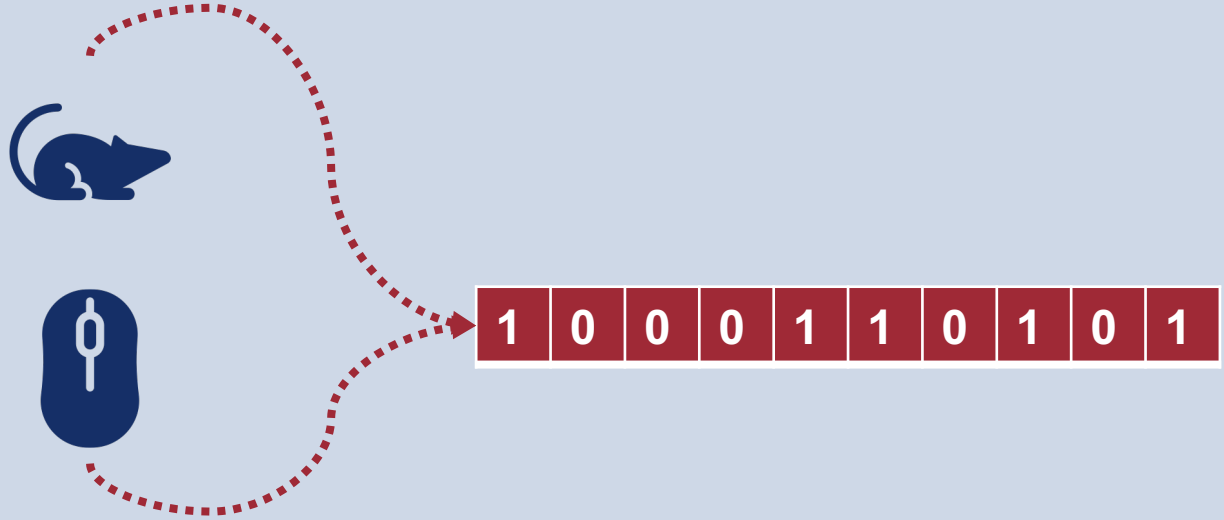
- **Classical word vectors**
  - Bag of words representations and their variations
- **Implicitly learned word vectors**
  - Word2Vec
  - GloVe

Vector semantics seeks to encode the same types of linguistic structures and phenomena that are often associated with lexical semantics.

- Key linguistics concepts and terminology (and useful properties of words):
  - Lemmas and senses
  - Synonymy
  - Word similarity
  - Word relatedness
  - Frames and roles
  - Connotation

Lemmas and Senses

- **Lemma:** The base form of a word
  - Papers → paper
  - Mice → mouse
- **Word Sense:** Different aspects of meaning for a word
  - Mouse (1): A small rodent
  - Mouse (2): A device to control a computer cursor
- Words with the same lemma should (hopefully!) reside near one another in vector space
- All of the word representations we'll cover this semester encode different senses of the same word in the same way
  - **Contextual embeddings** represent different word senses in different ways



Synonymy

- When a word sense for one word is (nearly) identical to the word sense for another word
- **Synonymy:** Two words are synonymous if they are substitutable for one another in any sentence without changing the situations in which the sentence would be true
  - This means that the words have the same **propositional meaning**

For my assignment I'm writing a scathing **critique** of Dr. Parde's recent paper.

For my assignment I'm writing a scathing **review** of Dr. Parde's recent paper.

Word Similarity

- Words don't often have that many synonyms, but they do have a lot of **similar** words
  - Review ≈ summary
- Good way to check if two words are similar: Can word *Y* be commonly used in the same context as word *X*?
    - I'm writing a summary 🙂
    - Did you submit your summary yet? 🙂
    - That is a scathing summary 😒

Word Relatedness

- Sometimes words are **related**, but not similar, to one another
- **Word Relatedness:** An association between words based on their shared participation in an event or **semantic field**
  - **Semantic Field:** A set of words covering a semantic domain, even if they cannot be used in the same context as one another
    - Restaurant: {waiter, menu, plate, food, …, chef}

Semantic Frames

- **Semantic Frame:** A set of words that denote perspectives or participants in a particular type of event
  - Commercial Transaction = {buyer, seller, goods, money}
- **Semantic Role:** A participant's underlying role with respect to the main verb in the sentence

Connotation

- Also referred to as **affective meaning**
- The aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations
- Generally three dimensions:
  - **Valence:** Positivity
    - High: Happy, satisfied
    - Low: Unhappy, annoyed
  - **Arousal:** Intensity of emotion
    - High: Excited, frenzied
    - Low: Relaxed, calm
  - **Dominance:** Degree of control
    - High: Important, controlling
    - Low: Awed, influenced

Connotation (Continued)

- For example, a word could be represented by its value on each of the three affective dimensions

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89

Word vector! (Osgood et al., 1957)

How, then, should we represent the meaning of a word?

- Two classic strategies:
  - **Bag of words representations:** A word is a string of letters, or an index in a vocabulary list
  - **Logical representation:** A word defines its own meaning (“dog” = DOG)

How, then, should we represent the meaning of a word?

- Two classic strategies:
  - **Bag of words representations:** A word is a string of letters, or an index in a vocabulary list
  - **Logical representation:** A word defines its own meaning ("dog" = DOG)

Bag of words features leverage simple, document-level vector semantics.

- Under the distributional hypothesis, we define a word by its **environment** or its **distribution** in language use
- This corresponds to the set of **contexts** in which the word occurs
  - **Context:** Neighboring words or grammatical environments
- **Two words with very similar sets of contexts (i.e., similar distributions) are assumed to have very similar meanings**

We do this to infer meaning in the real world all the time.

- Pretend you don't know what the Cantonese word *ongchoi* means
- However, you read the following sentences:
  - Ongchoi is delicious sautéed with garlic.
  - Ongchoi is superb over rice.
  - …ongchoi leaves with salty sauces…
- You've seen many of the other context words in these sentences previously:
  - …spinach sautéed with garlic over rice…
  - …chard stems and leaves are delicious…
  - …collard greens and other salty leafy greens…
- Your (correct!) conclusion?
  - Ongchoi is probably a leafy green similar to spinach, chard, or collard greens

How can we do this computationally?

- Count the words in the context of *ongchoi*
- See what other words occur in those same contexts

We can represent a word's context using vectors.

- Define a word as a single vector point in an *n*-dimensional space
  - For bag of words representations, *n* = vocabulary size
- Represent the presence or absence of words in its surrounding context using numeric values
  - For bag of words representations, the value stored in a dimension *n* corresponds to the presence of a context word *c* in the same sample as the target word *w*

The goal is for the values in these vector representations to correspond with dimensions of meaning.

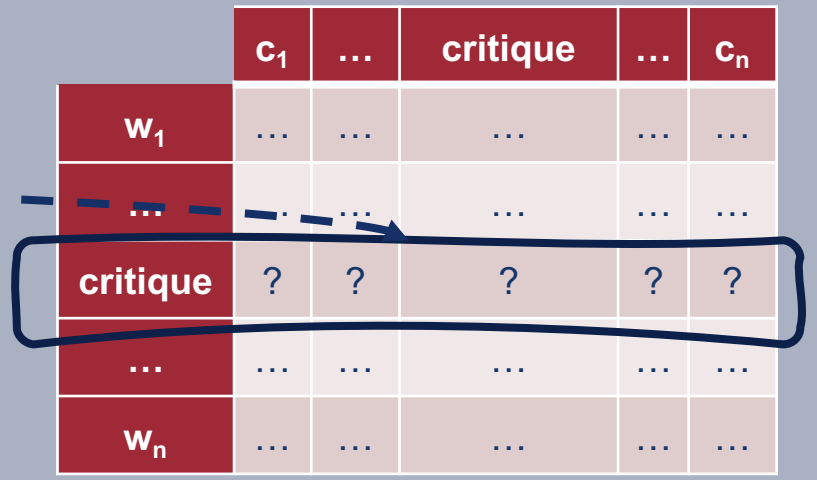
- Assuming this is the case, we should be able to:
  - Cluster vectors into semantic groups
  - Perform operations that are semantically intuitive

The goal is for the values in these vector representations to correspond with dimensions of meaning.

- Assuming this is the case, we should be able to:
  - Cluster vectors into semantic groups
  - Perform operations that are semantically intuitive

What other ways can we build vector representations for words?

critique

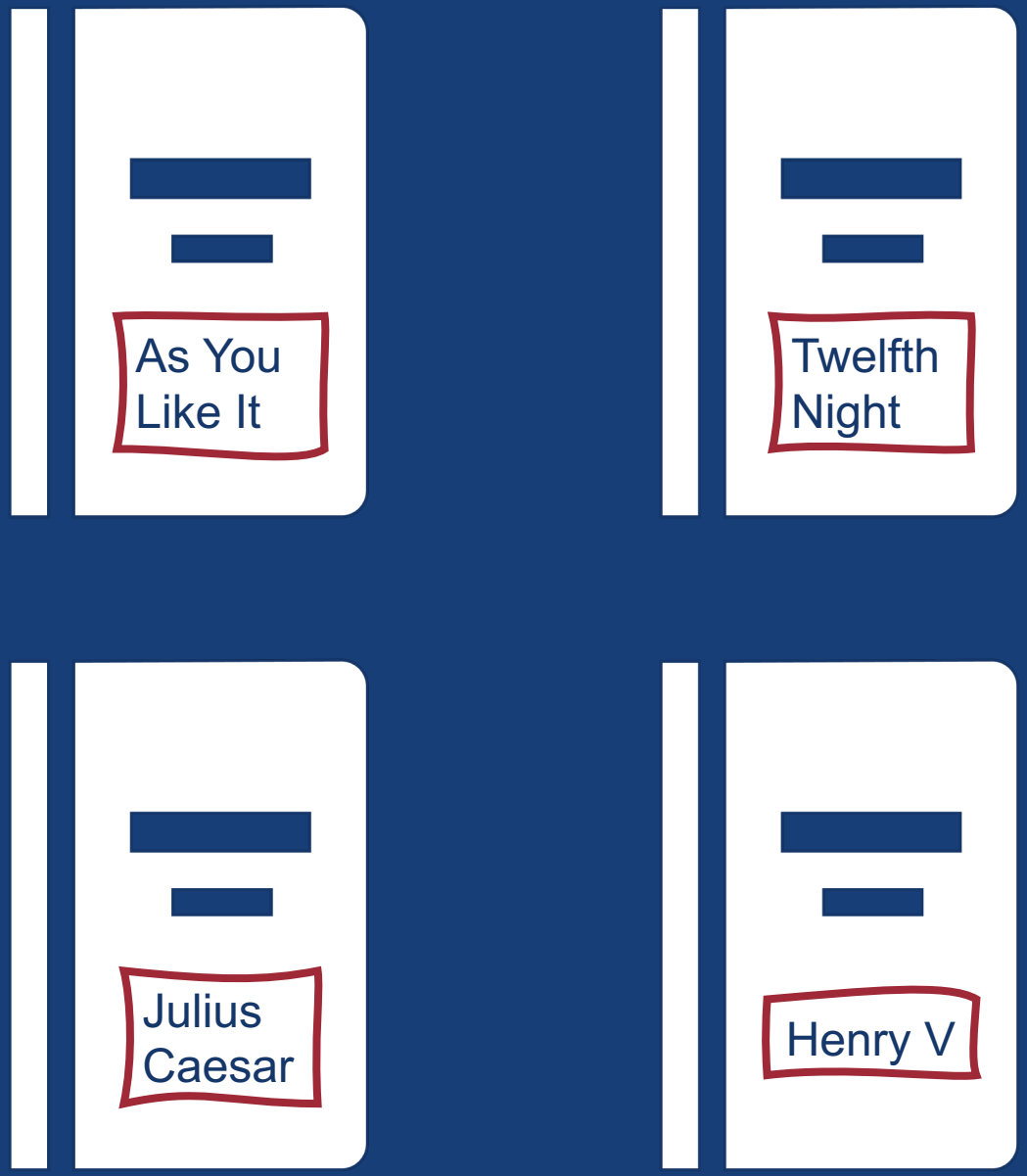


One Approach: TF-IDF

- Term Frequency * Inverse Document Frequency
- Meaning of a word is defined by the counts of words in the same document, as well as overall
- To do this, a **co-occurrence matrix** is needed

TF-IDF originated as a tool for information retrieval.

- Rows: Words in a vocabulary
- Columns: Documents in a selection

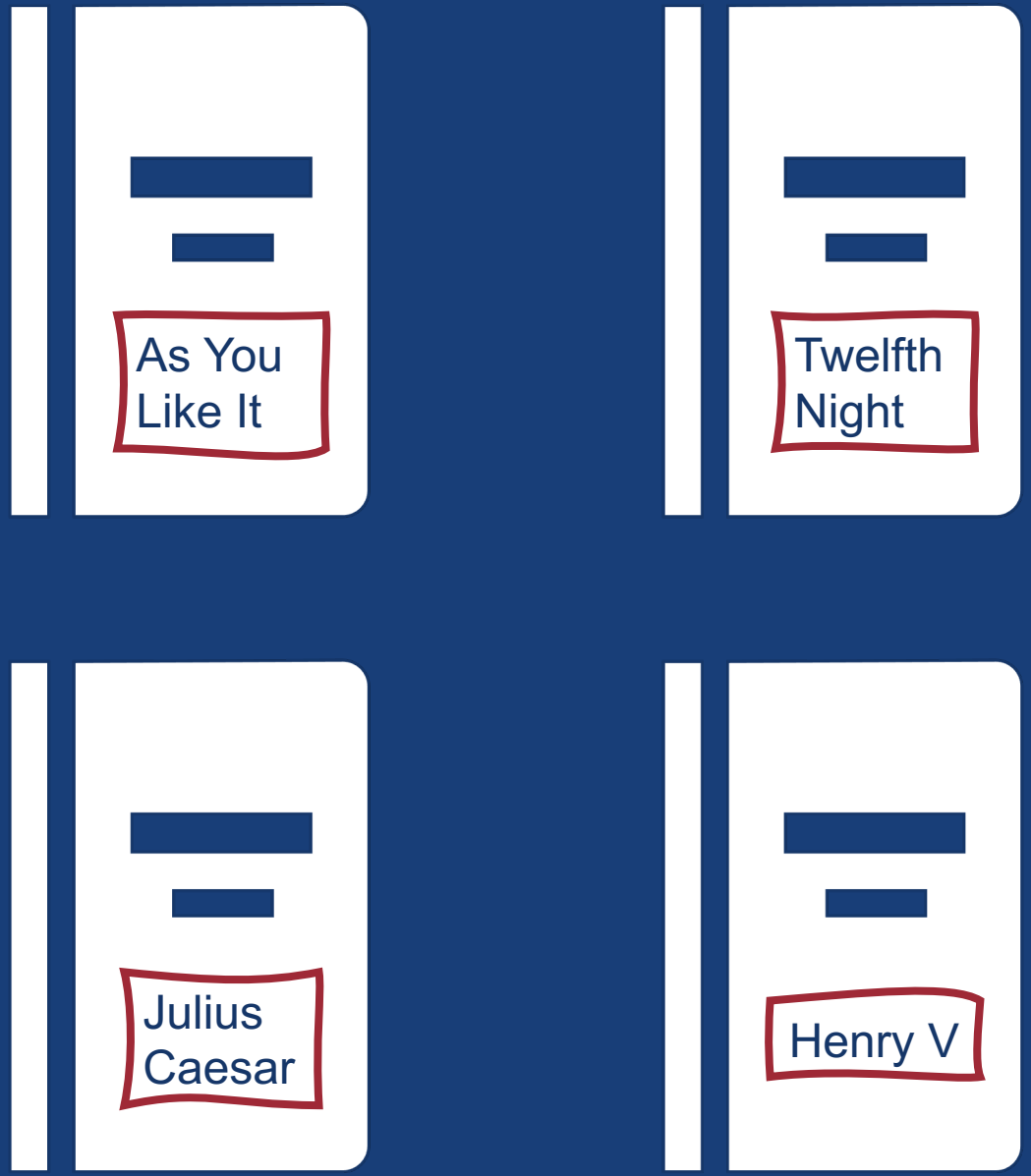


TF-IDF originated as a tool for information retrieval.

- Rows: Words in a vocabulary
- Columns: Documents in a selection

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

"wit" appears 3 times in Henry V

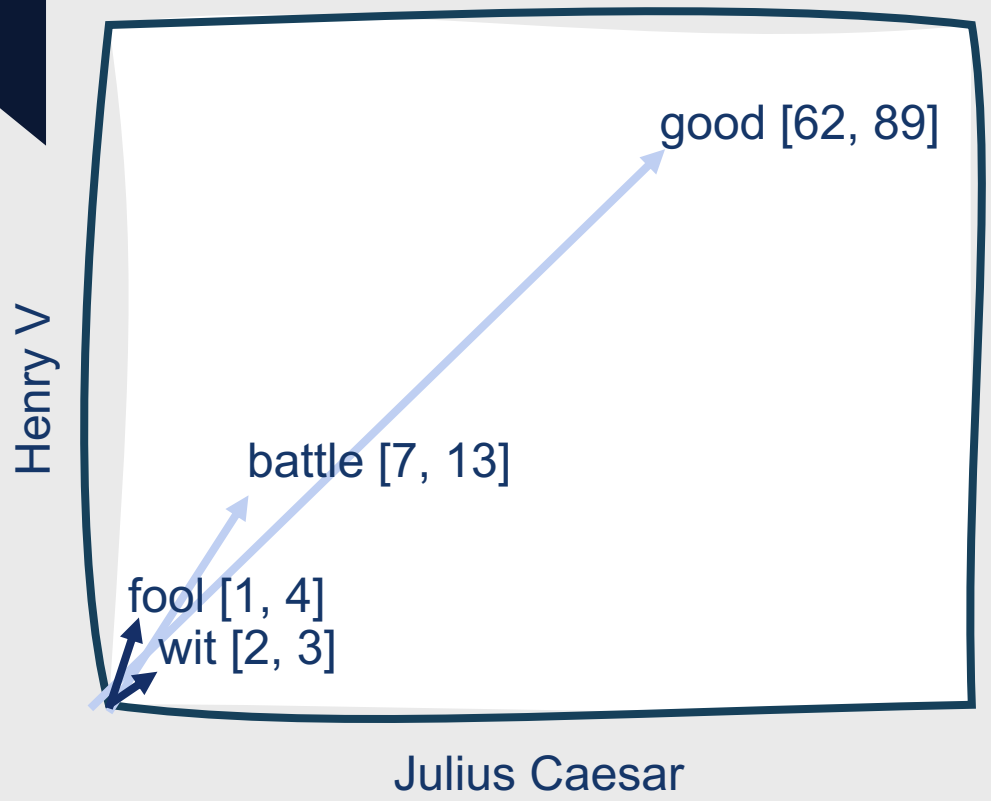


In a term-document matrix, rows can be viewed as word vectors.

- Each dimension corresponds to a document
- Words with **similar vectors** occur in **similar documents**

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

In a term-document matrix, rows can be viewed as word vectors.



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Different Types of Context

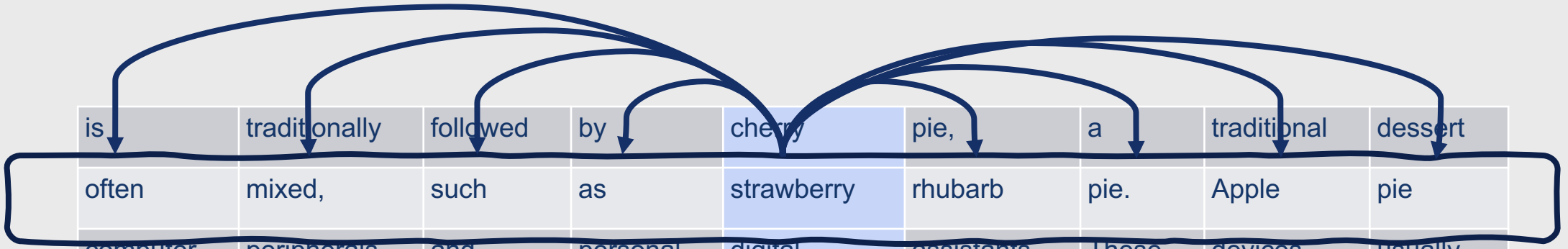
- We can also use **word context** for vector representations
  - Referred to as a term-term matrix, word-word matrix, or term-context matrix
- In a **word-word matrix**, the columns are also labeled by words
  - Thus, dimensionality is |V| x |V|
  - Each cell records the number of times the row (target) word and the column (context) word co-occur in some context in a training corpus

How can you decide if two words occur in the same context?

- Common **context windows**:
  - Entire document
    - Cell value = # times the words co-occur in the same document
  - Predetermined span surrounding the target
    - Cell value = # times the words co-occur in this span of words

Example Context Window (Size = 4)

- Take each occurrence of a word (e.g., strawberry)
- Count the context words in the four-word spans before and after it to get a word-word co-occurrence matrix



is	traditionally	followed	by	cherry	pie,	a	traditional	dessert
often	mixed,	such	as	strawberry	rhubarb	pie.	Apple	pie
computer	peripherals	and	personal	digital	assistants.	These	devices	usually
a	computer.	This	includes	information	available	on	the	internet

Example Context Window (Size = 4)

- A simplified subset of a word-word co-occurrence matrix could appear as follows, given a sufficient corpus

is	traditionally	followed	by	cherry	pie,	a	traditional	dessert
often	mixed,	such	as	strawberry	rhubarb	pie.	Apple	pie
computer	peripherals	and	personal	digital	assistants.	These	devices	usually
a	computer.	This	includes	information	available	on	the	internet

	aardvark	…	computer	data	result	pie	sugar	…
cherry	0	…	2	8	9	442	25	…
strawberry	0	…	0	0	1	60	19	…
digital	0	…	1670	1683	85	5	4	…
information	0	…	3325	3982	378	5	13	…

Vector for "strawberry"

Co-occurrence matrices provide raw frequency counts of word co-occurrences.

- However, this isn't the best measure of association between words
  - Some words co-occur frequently with many words, so won't be very informative
    - *the*, *it*, *they*
- We want to know about **words that co-occur frequently with one another, but less frequently across all texts**

TF-IDF is here to save the day!

- **Term Frequency:** The frequency of the word t in the document d
  - $tf_{t,d} = \text{count}(t, d)$
- **Document Frequency:** The number of documents in which the word t occurs
  - Different from collection frequency (the number of times the word occurs in the entire collection of documents)

Computing TF-IDF

- **Inverse Document Frequency:** The inverse of document frequency, where *N* is the total number of documents in the collection
  - $idf_t = \frac{N}{df_t}$
- IDF is higher when the term occurs in fewer documents
  - Document = Whatever is considered an instance or context in your dataset
- It is often useful to perform these computations in log space
  - TF: $\log_{10}(tf_{t,d}+1)$
  - IDF: $\log_{10} idf_t$

Computing TF*IDF

- TF-IDF combines TF and IDF
  - $tfidf_{t,d} = tf_{t,d} \times idf_t$

Assume we're looking at a subset of a 37-document corpus of Shakespearean plays….

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- **TF(battle, d_1) = 1**

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- TF(battle, d_1) = 1
- **IDF(battle) = N/DF(battle) = 37/21 = 1.76**

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58		
wit	20	15		

Overall document frequencies from our 37 plays

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- TF(battle, d_1) = 1
- IDF(battle) = N/DF(battle) = 37/21 = 1.76
- **TF-IDF(battle, d_1) = 1 * 1.76 = 1.76**

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- TF(battle, d_1) = 1
- IDF(battle) = N/DF(battle) = 37/21 = 1.76
- TF-IDF(battle, d_1) = 1 * 1.76 = 1.76
- **Alternately, TF-IDF(battle, d_1) = $log_{10}(1+1) * log_{10} 1.76 =$ 0.074**

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- TF(battle, d_1) = 1
- IDF(battle) = N/DF(battle) = 37/21 = 1.76
- TF-IDF(battle, d_1) = 1 * 1.76 = 1.76
- Alternately, TF-IDF(battle, d_1) = $log_{10}(1+1) * log_{10} 1.76 = 0.074$

	d_1	d_2	d_3	d_4
battle	0.074	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

To convert our entire word co-occurrence matrix to a TF-IDF matrix, we need to repeat this calculation for each element.

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

How does the TF-IDF matrix compare to the original term frequency matrix?

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

How does the TF-IDF matrix compare to the original term frequency matrix?

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

Occurs in every document …not important in the overall scheme of things!

How does the TF-IDF matrix compare to the original term frequency matrix?

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

Increases the importance of rarer words like “battle”

Note that the TF-IDF model produces a sparse vector.

- **Sparse:** Many (usually most) cells have values of 0

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

Note that the TF-IDF model produces a sparse vector.

- **Sparse:** Many (usually most) cells have values of 0

	d_1	d_2	d_3	d_4	d_5	d_6	d_7
battle	0.1	0.0	0.0	0.0	0.2	0.0	0.3
good	0.0	0.0	0.0	0.0	0.0	0.0	0.0
fool	0.0	0.0	0.0	0.0	0.0	0.0	0.0
wit	0.0	0.0	0.0	0.0	0.0	0.0	0.0

We'll learn about more sophisticated word representation techniques soon….

- However, TF-IDF remains a useful starting point for vector space models
- Generally combined with standard machine learning algorithms
  - Logistic Regression
  - Naïve Bayes

Summary: CRFs and Introduction to Vector Semantics

- **Conditional random fields** are a generalized case of multi-class logistic regression in which conditional probabilities are computed over label sequences given specific sequences of observations
- Like logistic regression, **CRF model parameters can be optimized by minimizing cross-entropy loss**, using a dynamic programming method to compute expectations
- **Representation learning** is the act of building or learning **word vectors** based on **the distributional hypothesis**
- This process seeks to encode the same linguistic phenomena observed in studies of **lexical semantics**
- Bag-of-words representations are one form of word vector, and **TF-IDF representations** are another
- TF-IDF representations combine simple term frequency with **inverse document frequency** to minimize the impact of words that occur more frequently in general