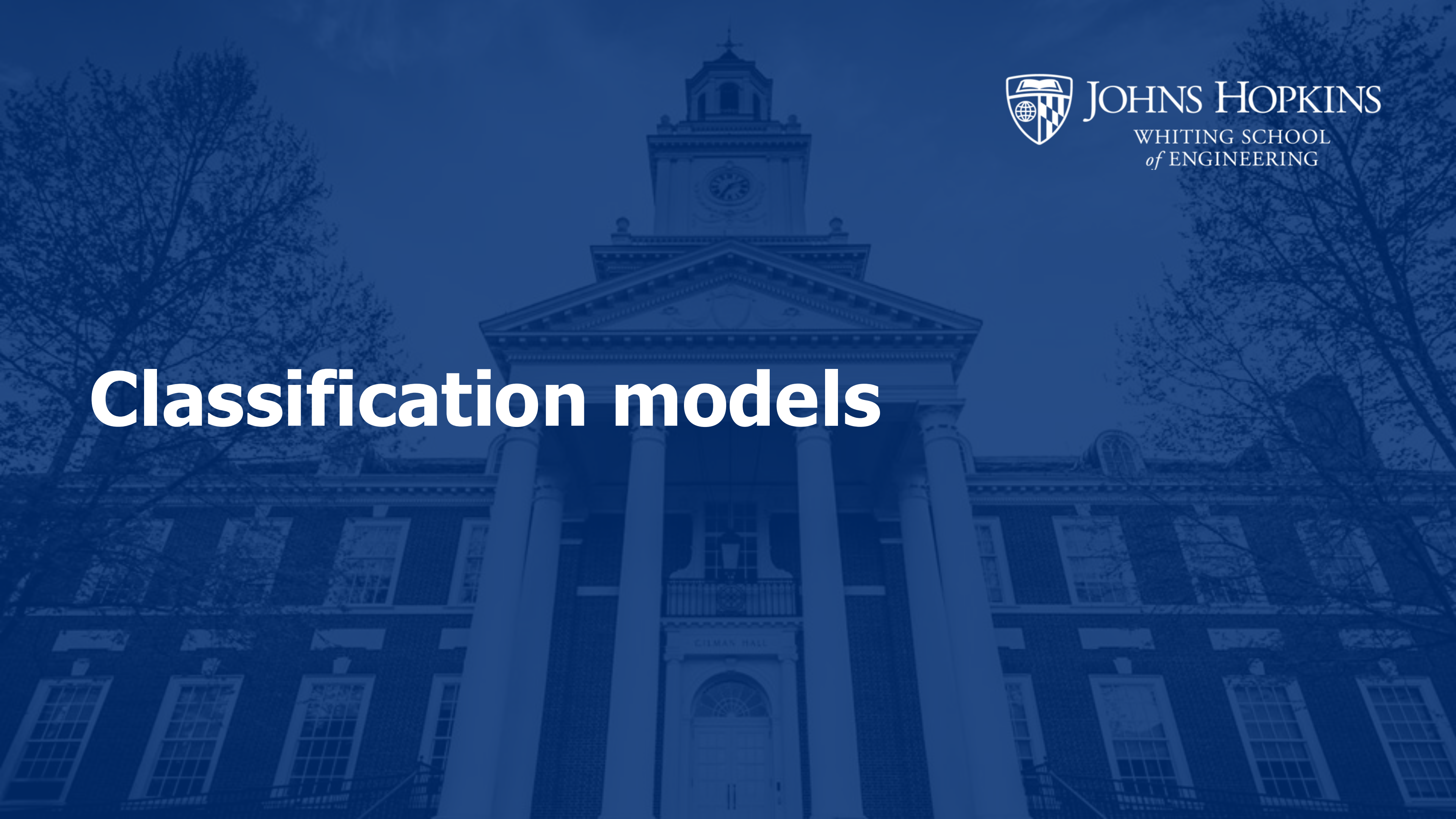


JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Classification models

Overview

- Recap: last class
  - Why annotate data?
  - Tips and tricks for components of annotation process
  - Annotator agreement metrics
  - Ethics of crowdsourcing

This class: What do we do with annotated data?

- Logistic Regression
- Neural networks
- Adjusting for model errors

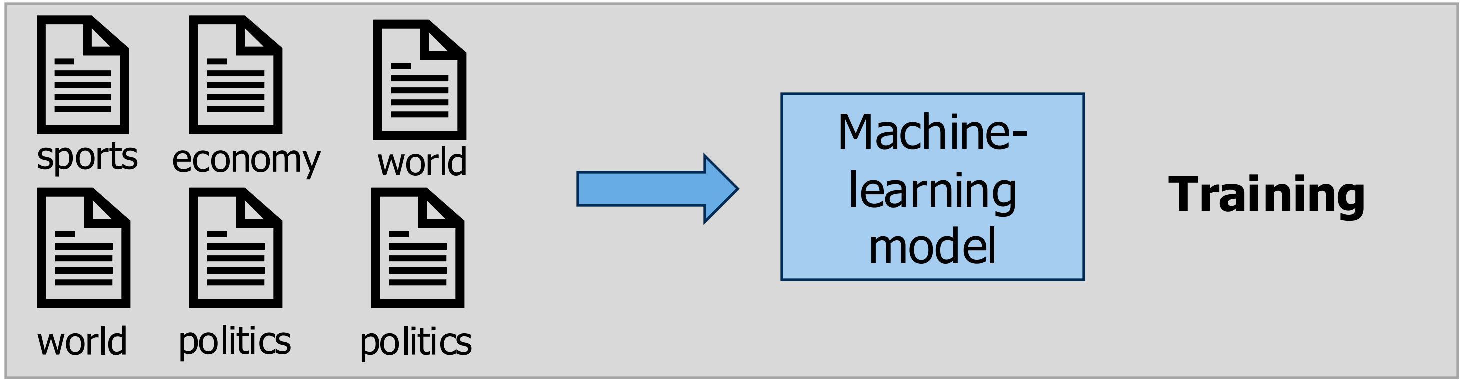
Methods of Data analysis

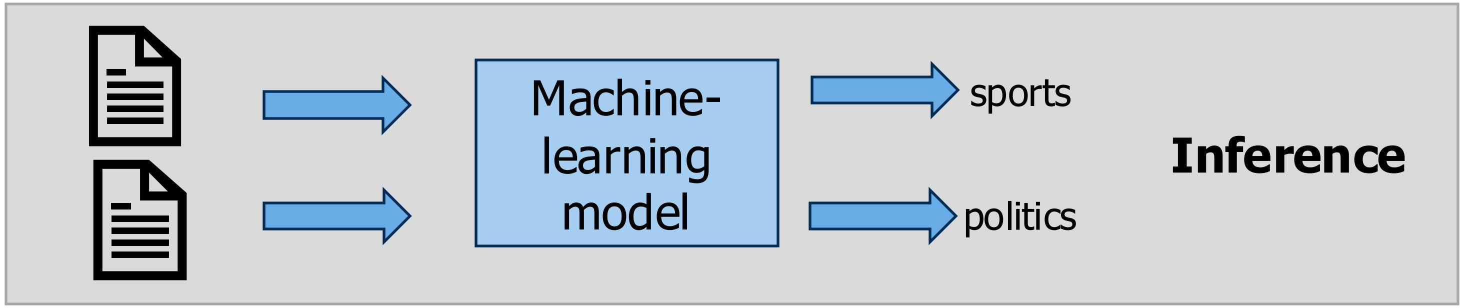
- We want to know if (and when and how) Republicans talk about taxes more than Democrats:
  1. We use word statistics to find if words like "taxes" and "spending" are more common in republican speeches
  2. We can train a topic model, identify the tax-related topics and determine if that topic is more common in Republican vs. Democratic speech (or incorporate party affiliation as co-variate in STM)
  3. **We could go through every speech by hand:**
     - **Label if each speech or sentence or word is related to taxes**
     - **Count if we labeled more Republican speech than Democratic speech**
  4. **We can automate #3 using machine learning**

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Logistic Regression

Supervised learning



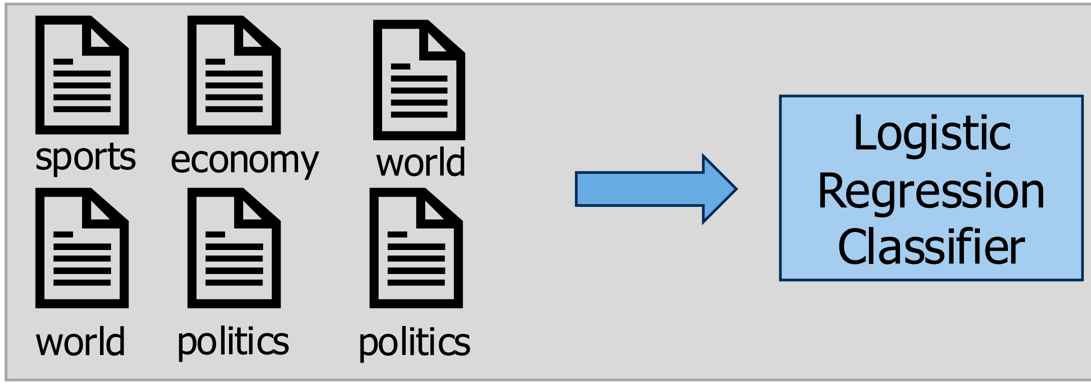


Components of a probabilistic machine learning classifier

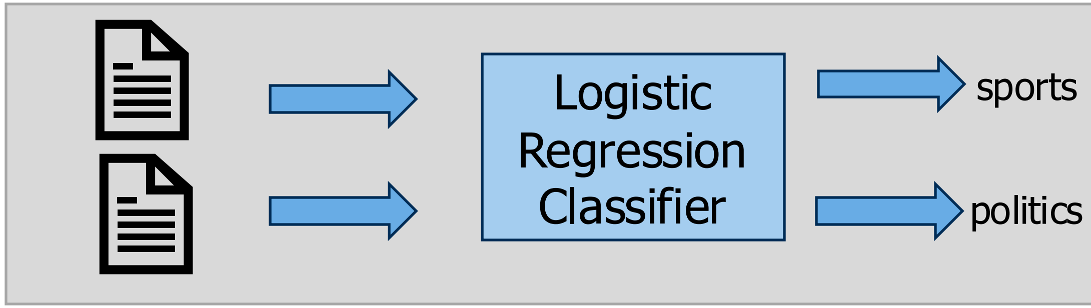
- Given m input/output pairs $(x^{(i)},y^{(i)})$:

1. A **feature representation** of the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \ldots , x_n]$. Feature j for input $x^{(i)}$ is x_j, more completely $x_j^{(i)}$, or sometimes $f_j(x)$.
2. A **classification function** that computes $\hat{y}$, the estimated class, via $p(y|x)$, like the **sigmoid** or **softmax** functions.
3. An objective function for learning, like **cross-entropy loss**.
4. An algorithm for optimizing the objective function: **stochastic gradient descent**.

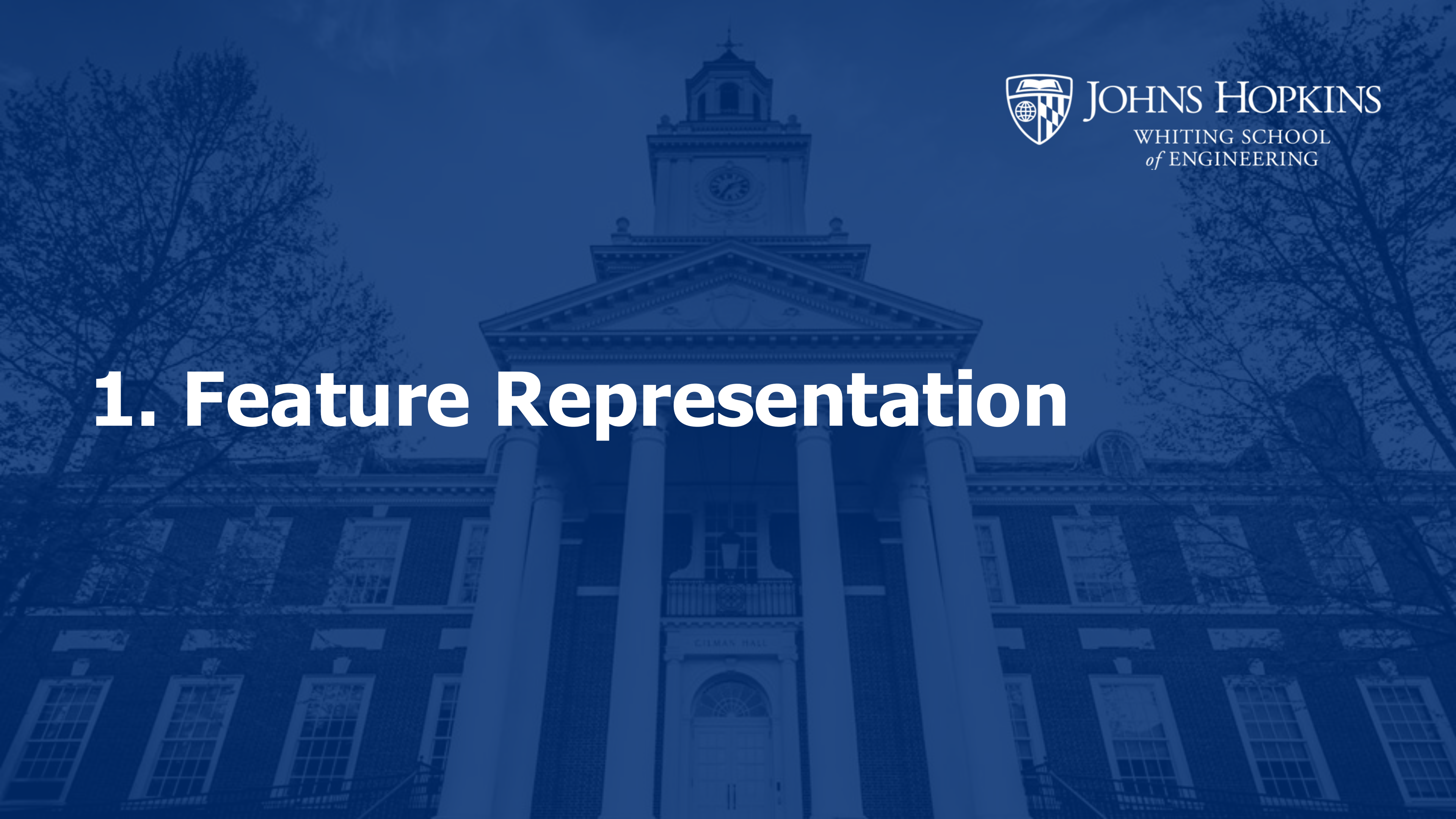
Supervised learning



Training: learn weights w and b using **stochastic gradient descent** and **cross-entropy loss**.



Inference Test: Given a test example x, compute p(y|x) using learned weights w and b, and return whichever label (y = 1 or y = 0) is higher probability

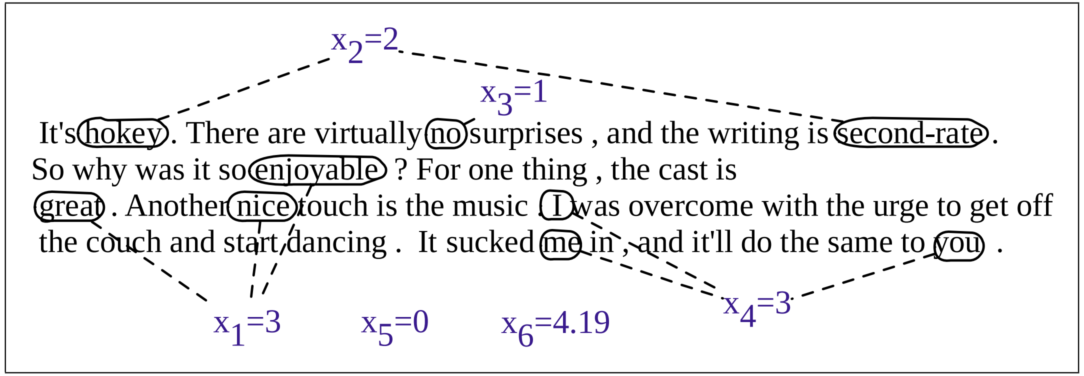


JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

1. Feature Representation

Feature representation

- We can craft specific features:



$x_2=2$
$x_3=1$
It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .
$x_1=3$
$x_5=0$
$x_6=4.19$
$x_4=3$

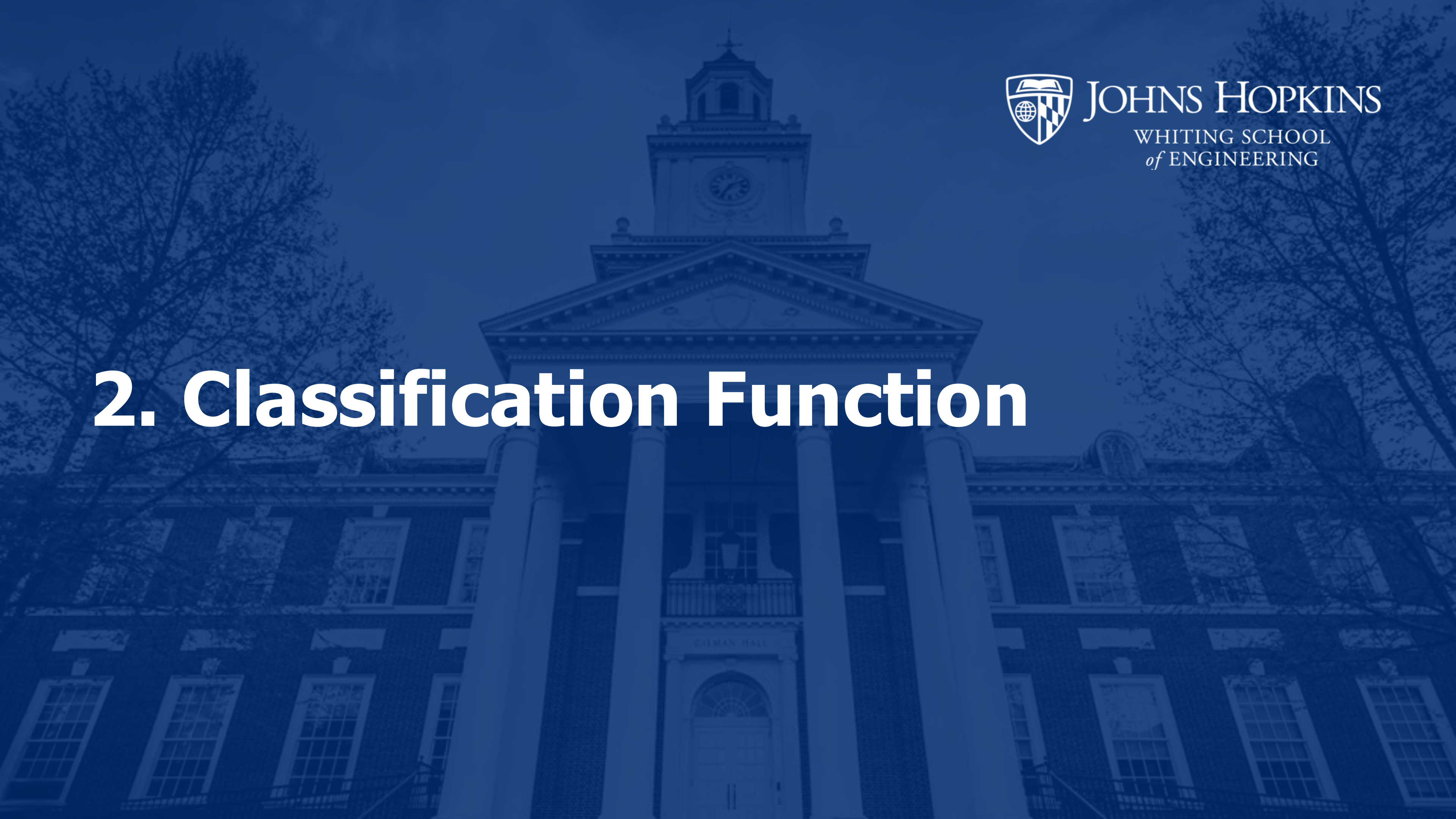
Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Feature representation

- Common choice for document-level tasks:
  - BOW representation (with TF-IDF weighting)

	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

Bag-of-words document representation



JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

2. Classification Function

Binary Classification in Logistic Regression

- Given a series of input/output pairs:
  - $(x^{(i)}, y^{(i)})$

- For each observation $x^{(i)}$
  - We represent $x^{(i)}$ by a **feature vector** $[x_1, x_2, \ldots, x_n]$
  - We compute an output: a predicted class $\hat{y}^{(i)} \in \{0,1\}$

  - (multinomial logistic regression: $\hat{y} \in \{0, 1, 2, 3, 4\}$)

Introducing feature weights

- For feature x_i, weight w_i tells is how important is x_i
  - x_i ="review contains 'awesome'": w_i = +10
  - x_j ="review contains 'abysmal'": w_j = -10
  - x_k ="review contains 'mediocre'": w_k = -2

- Feature weights are useful for learning an accurate classifier, but they are also useful for analyzing feature importance
  - Example: we want to learn what words people perceive as more polite and respectful
    - We have annotators rate if a text is polite/respectful or not
    - We train a classifier and examine which features are weighted the highest

How to do classification

- For each feature x_i, introduce weight w_i which tells us importance of x_i
  - (Plus we'll have a bias b)
- We'll sum up all the weighted features and the bias

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$z = w \cdot x + b$$

- If this sum is high, we say y=1; if low, then y=0

We want a probabilistic classifier

We need to formalize "sum is high".

$p(y=1|x;\theta)$
$p(y=0|x;\theta)$

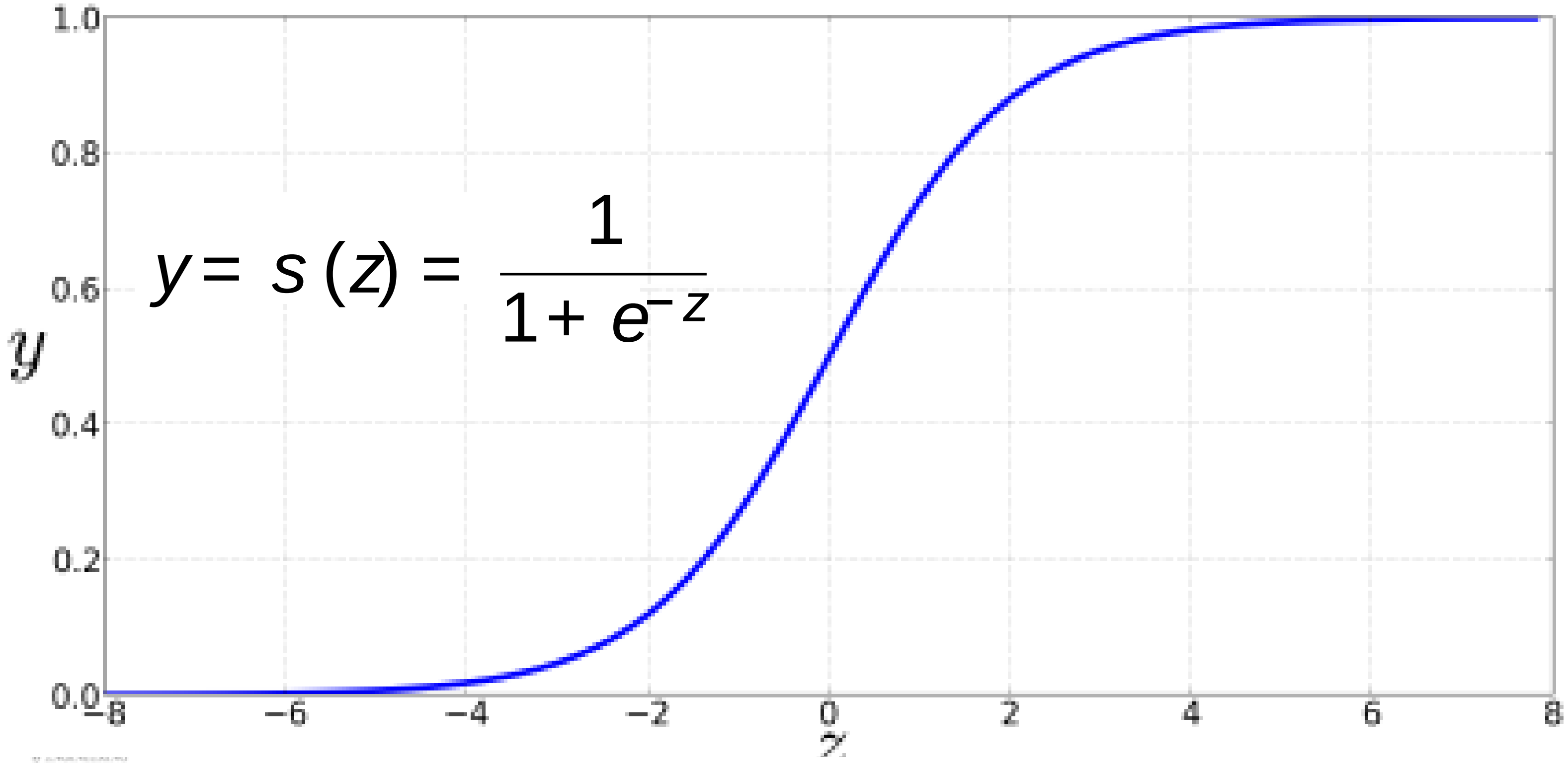
The problem: z isn't a probability, it's just a number!

$$z = w \cdot x + b$$

- Solution: use a function of z that goes from 0 to 1

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

The very useful sigmoid or logistic function



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

Idea of logistic regression

- We’ll compute $w \cdot x + b$
- And then we’ll pass it through the sigmoid function:
- $\sigma(w \cdot x + b)$
- And we'll just treat it as a probability

Making probabilities with sigmoids

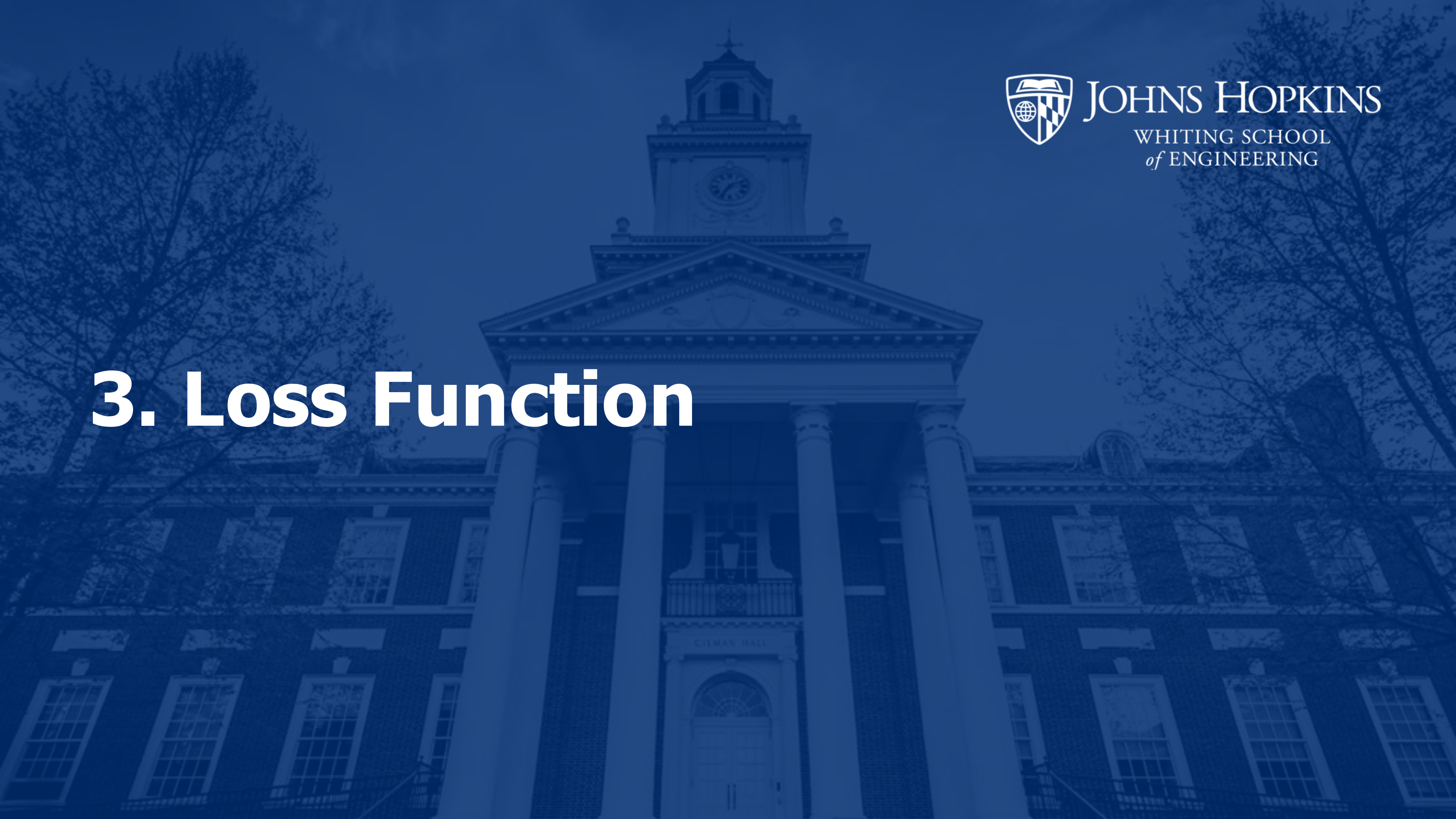
$$\begin{aligned}
P(y=1) &= \sigma(w\cdot x+b) \\
&= \frac{1}{1+\exp\left(-(w\cdot x+b)\right)}
\end{aligned}$$

$$\begin{aligned}
P(y=0) &= 1-\sigma(w\cdot x+b) \\
&= 1-\frac{1}{1+\exp\left(-(w\cdot x+b)\right)} \\
&= \frac{\exp\left(-(w\cdot x+b)\right)}{1+\exp\left(-(w\cdot x+b)\right)} \\
&= \sigma\left(-(w\cdot x+b)\right)
\end{aligned}$$

Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

0.5 here is called the **decision boundary**



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

3. Loss Function

Loss function

- Supervised classification:
  - We know the correct label y (either 0 or 1) for each x.
  - But what the system produces is an estimate, $\hat{y}$
- We want to set w and b to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.
  - We need a distance estimator: a **loss function** or a **cost function** (#3)
  - We need an optimization algorithm to update w and b to minimize the loss (#4)

Loss Function

- We want to know how far is the classifier output:
- $\hat{y} = \sigma(w \cdot x + b)$
- from the true output:
- y [= either 0 or 1]
- We'll call this difference:
- $L(\hat{y}, y)$ = how much $\hat{y}$ differs from the true y

Deriving cross-entropy loss for a single observation x

- **Goal**: maximize probability of the correct label p(y|x)
- Since there are only 2 discrete outcomes (0 or 1) we can express the probability p(y|x) from our classifier (the thing we want to maximize) as

$$p(y|x) = \hat{y}^{y}\,(1-\hat{y})^{1-y}$$

- noting:
    - if y=1, this simplifies to $\hat{y}$
    - if y=0, this simplifies to 1− $\hat{y}$

Deriving cross-entropy loss for a single observation x

- **Goal**: maximize probability of the correct label p(y|x)
- Since there are only 2 discrete outcomes (0 or 1) we can express the probability p(y|x) from our classifier (the thing we want to maximize) as

$$p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

- Take the log of both sides

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^{y}(1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y})\end{aligned}$$

JOHNS HOPKINS WHITING SCHOOL of ENGINEERING

Deriving cross-entropy loss for a single observation x

- **Goal**: maximize probability of the correct label p(y|x)

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^{y}\,(1-\hat{y})^{1-y}\right]\\ &= y\log\hat{y}+(1-y)\log(1-\hat{y})\end{aligned}$$

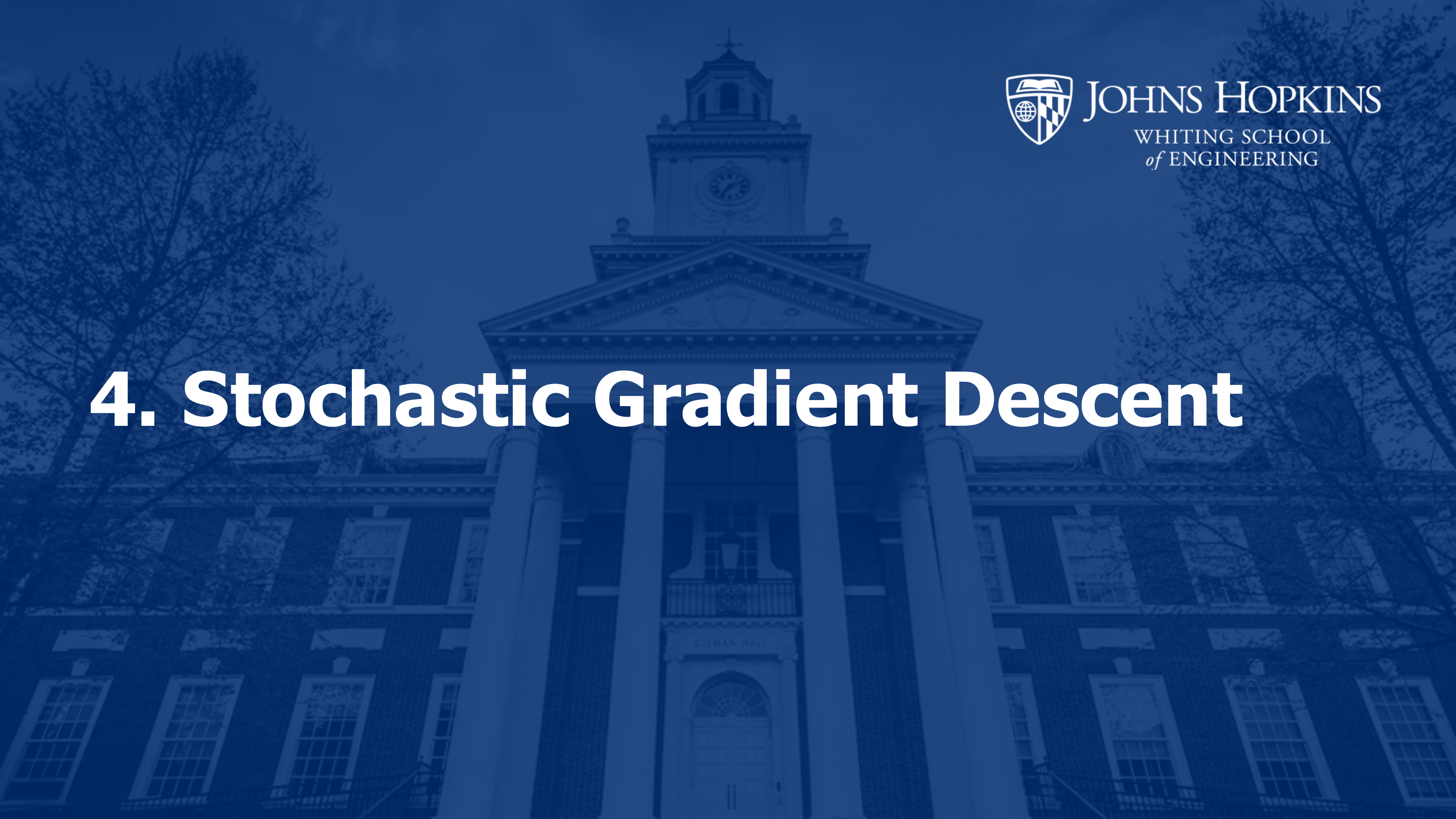
- Now flip sign to turn this into a loss: something to minimize
- **Cross-entropy loss** (because is formula for cross-entropy(y, $\hat{y}$))

$$L_{\mathrm{CE}}(\hat{y},y)=-\log p(y|x) \;=\; -[y\log\hat{y}+(1-y)\log(1-\hat{y})]$$

- Or, plugging in definition of $\hat{y}$:

$$L_{\mathrm{CE}}(\hat{y},y) \;=\; -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))]$$

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING



Johns Hopkins
Whiting School of Engineering

4. Stochastic Gradient Descent

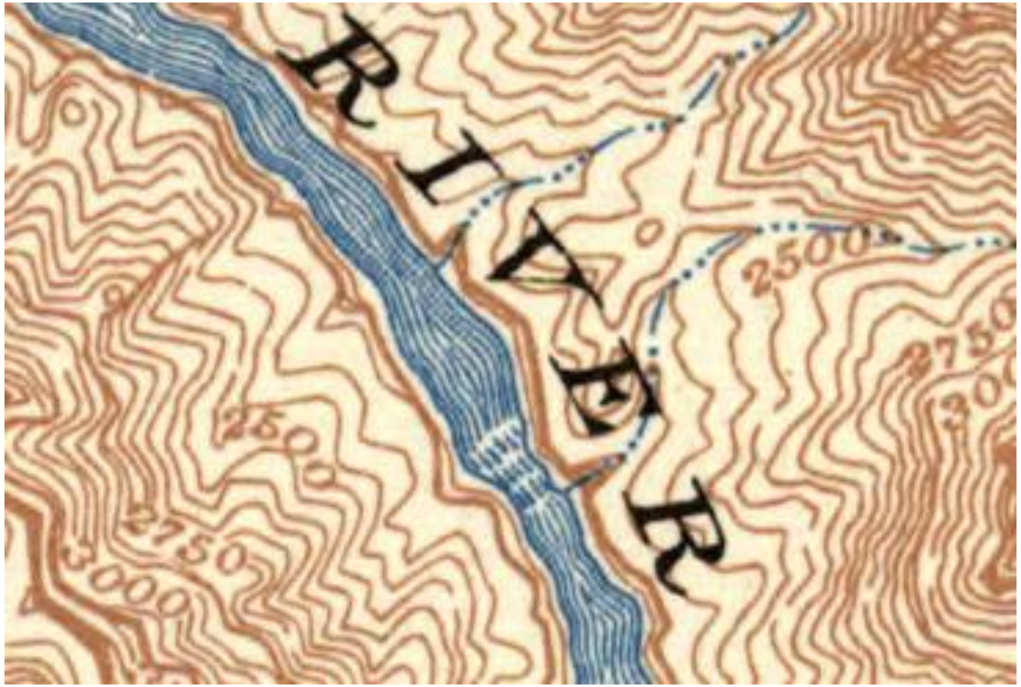
Our goal: minimize the loss

- Let's make explicit that the loss function is parameterized by weights θ=(w,b)
- And we'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious
- We want the weights that minimize the loss, averaged over all examples:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)}; \theta), y^{(i)})$$

Intuition of gradient descent

- How do I get to the bottom of this river canyon?



Look around me 360°

Find the direction of steepest slope down

Go that way

Gradient Descent

- The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.
- For each dimension w_i the gradient component i tells us the slope with respect to that variable.
  - "How much would a small change in w_i influence the total loss function L?"
  - We express each element as a partial derivative ∂ of the loss ∂w_i
  - The gradient is then defined as a vector of these partials.
- **Gradient Descent**: Find the gradient of the loss function at the current point and move in the **opposite** direction.

$$w^{t+1} = w^{t} - \boxed{h} \frac{d}{dw} L(f(x;w), y)$$

"learning rate" hyperparameter determines how far we move in the direction specified by the gradient

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Neural Networks

Components of a probabilistic machine learning classifier

- Given m input/output pairs $(x^{(i)}, y^{(i)})$:

1. A **feature representation** of the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \ldots, x_n]$. Feature j for input $x^{(i)}$ is x_j, more completely $x_j^{(i)}$, or sometimes $f_j(x)$.
2. A **classification function** that computes $\hat{y}$, the estimated class, via $p(y|x)$, like the **sigmoid** or **softmax** functions.
3. An objective function for learning, like **cross-entropy loss**.
4. An algorithm for optimizing the objective function: **stochastic gradient descent**.

2. Neural Networks: Made up of units

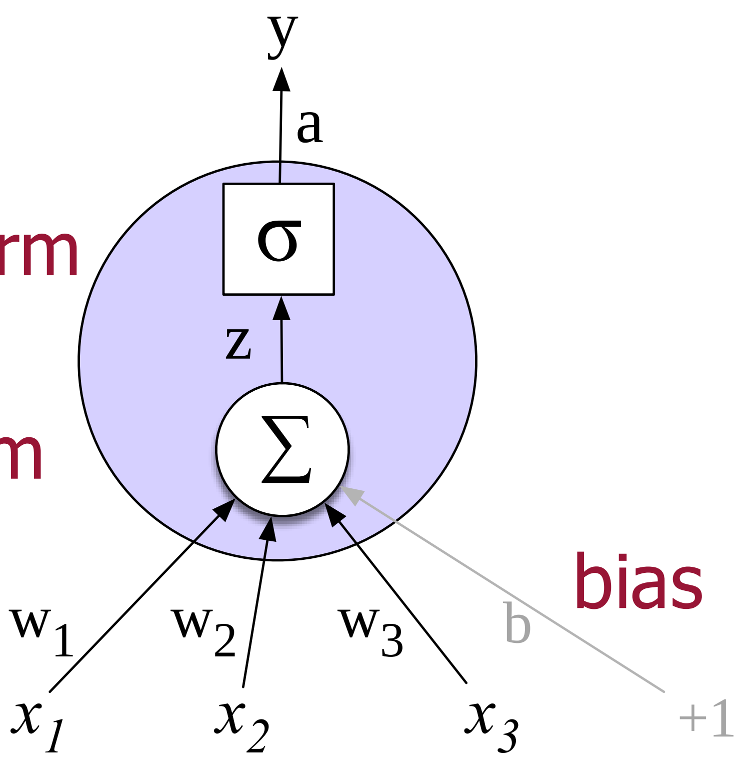
Output value

Non-linear transform

Weighted sum

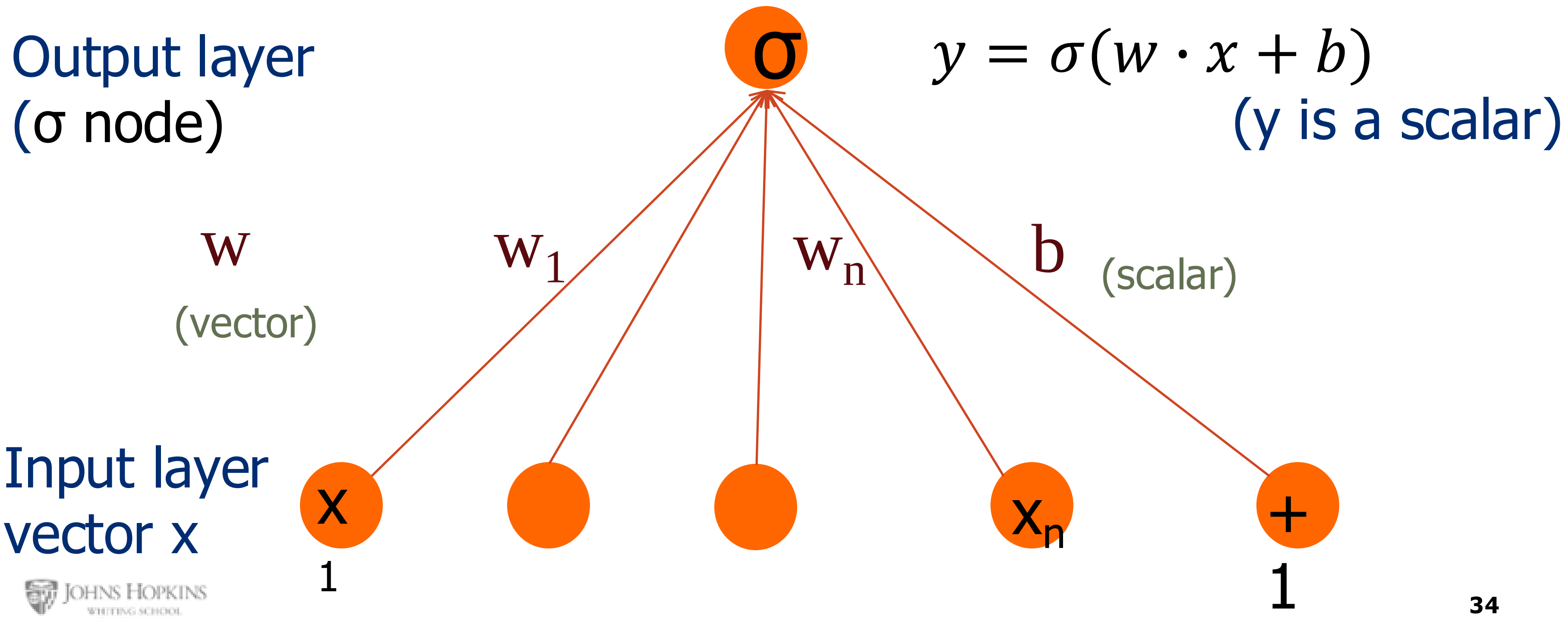
Weights

Input layer

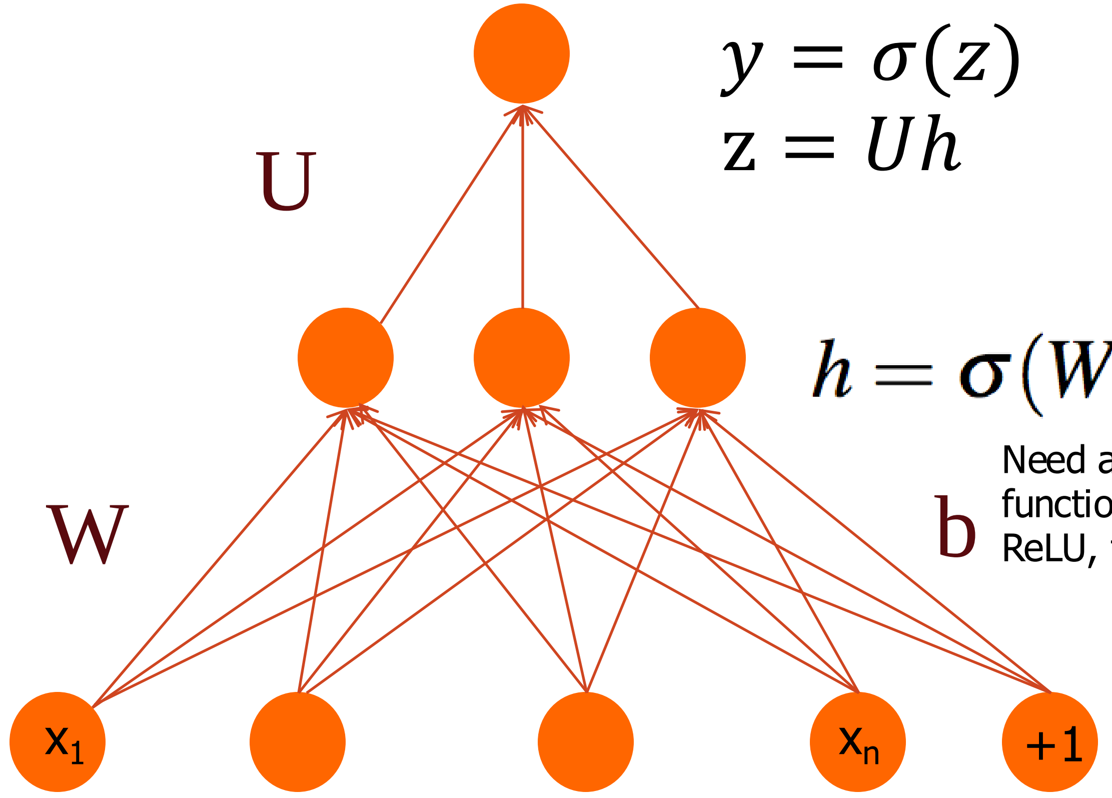


2. Binary Logistic Regression as a 1-layer Network

(we don't count the input layer in counting layers!)



Two-layer Neural Network with scalar output



$$y = \sigma(z)$$
$$z = Uh$$

$$h = \sigma(Wx + b)$$

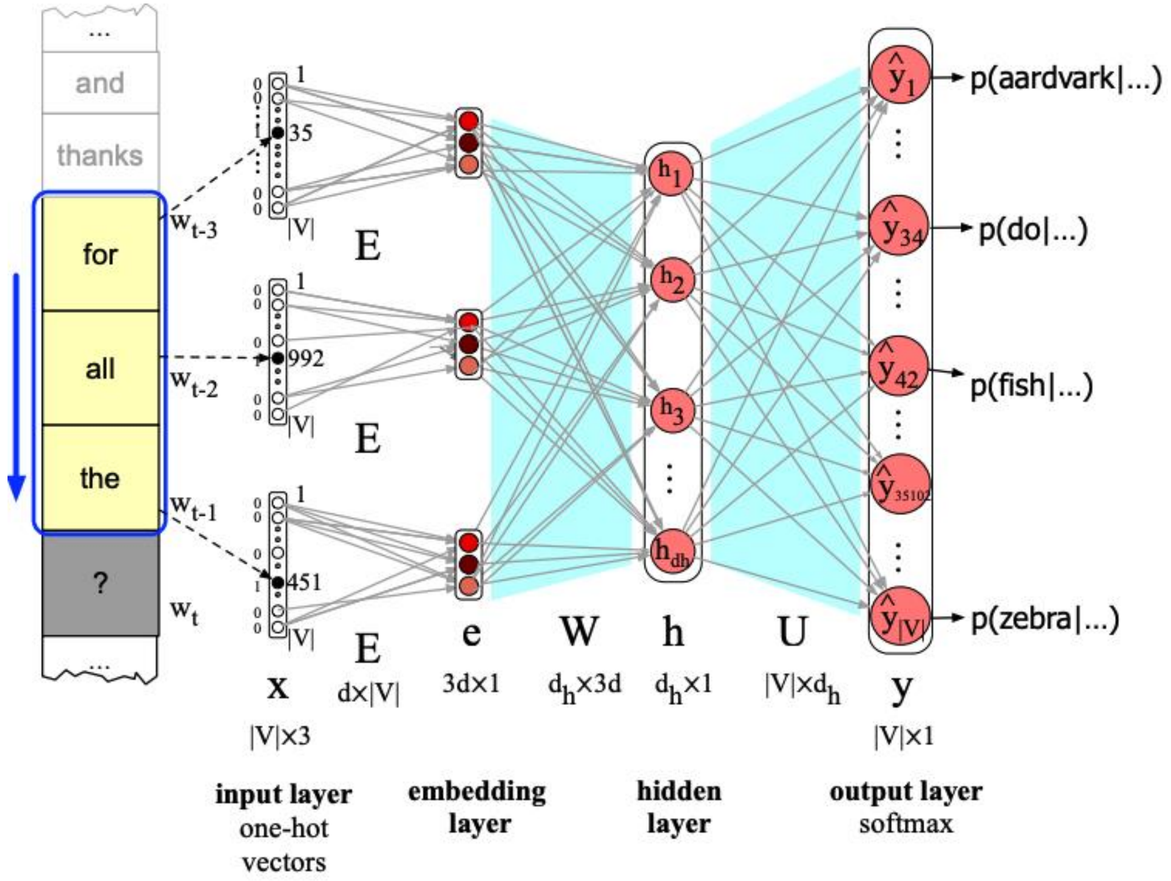
Need a non-linear function, e.g. sigmoid, ReLU, tanh

4. Backpropogation for Gradient Estimation

- We can train the model in a similar way, but we need the derivative of the loss with respect to each weight in every layer of the network
  - But the loss is computed only at the very end of the network!

- Solution: **error backpropagation** (Rumelhart, Hinton, Williams, 1986)
  - Algorithm for gradient estimation

1. Learned word embeddings instead of crafted features



JOHNS HOPKINS
WHITING SCHOOL *of* ENGINEERING

Evaluation and Prevalence Estimation

Evaluation Metrics

- How can we tell if model is correct?
  - Performance on held-out test set

- Data splits:
  - **Training set**: used to learn model parameters
  - **Validation/development set**: used to learn hyperparameters, debug, choose best model instance
  - **Test set**: used to evaluate model performance

Evaluation

		Gold Labels		
		Not Offensive	**Offensive**	**Sum**
Model Prediction	**Not Offensive**	147	50	197
	Offensive	10	15	25
	Sum	157	65	222

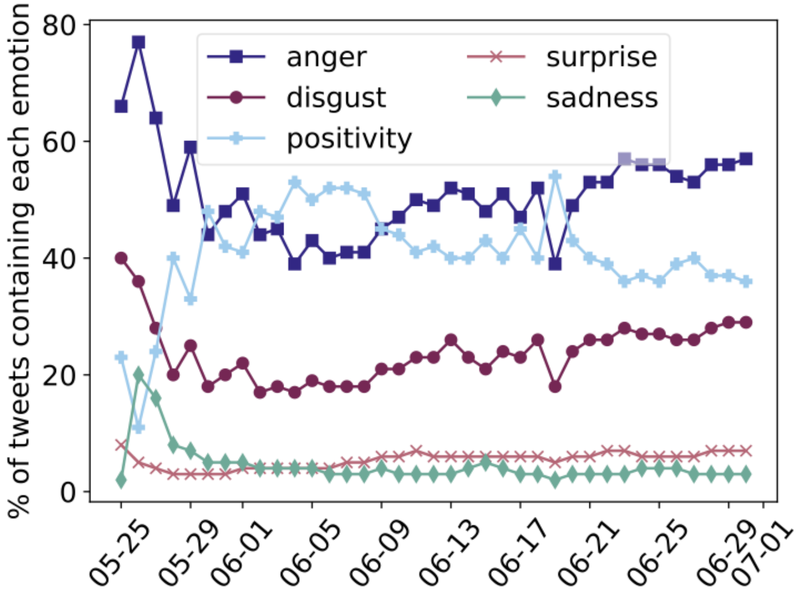
Accuracy: $\frac{Number\ correct}{Total} = \frac{147+15}{222} = 73\%$

Precision: $\frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{15}{15+10} = 60\%$

Recall: $\frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{15}{15+50} = 23\%$

Prevalence Estimates

- We often want to use the model for **prevalence estimates**
  - Did prevalence of positive emotions increase over time?



Simple Approach: Classify and Count (CC)

$$\hat{\theta}^{CC} = \frac{1}{n}\sum_i 1\{p_i > 0.5\}$$

- Convert classifier output p_i to binary decision and compute average over all n data points (model estimates that x% of tweets express anger)

- What if our held-out test accuracy is 75%? Should we still count all outputs predicted by the model?

George Forman. 2005. Counting positives accurately despite inaccurate classification. In European Conference on Machine Learning.
Keith, Katherine, and Brendan O'Connor. "Uncertainty-aware generative models for inferring document class prevalence." *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018

Adjusted Classify and Count (ACC)

$$\hat{\theta}^{ACC} = \frac{\hat{\theta}^{CC} - \mathrm{FPR}}{\mathrm{TPR} - \mathrm{FPR}}$$

- Dependent on the correctness of TPR and FPR

George Forman. 2005. Counting positives accurately despite inaccurate classification. In European Conference on Machine Learning.

Probablistic Classify and Count (PCC)

$$\hat{\theta}^{PCC} = \frac{1}{n} \sum_i p_i$$

- Is typically effective *if model is well-calibrated*
  - For all test samples where p=0.9, ~90% should be true positives
  - For all test samples where p=0.7, ~70% should be true positives
  - For all test samples where p=0.1, ~10% should be true positives

Design-based Supervised Learning

- Scenario:
  - We have a classification model that outputs predicted values $\widehat{Y}_i$
  - Our model probably has non-random errors
  - We're also willing to hand-code some data, so we have "gold" data Y_i
- This set-up is generalizable to lots of settings where we have some data we trust more than others
  - Some data is hand-coded and some labels are predicted
  - Some data is coded by researchers and some by crowd-workers

Note: If classification errors are totally random, we can ignore them, they won't change our prevalence estimates

Egami, Naoki, et al. "Using Large Language Model Annotations for the Social Sciences: A General Framework of Using Predicted Variables in Downstream Analyses."." (2024).

Design-based Supervised Learning

$$\widetilde{Y}_i = \underbrace{\widehat{Y}_i}_{\text{Predicted Outcome}} - \underbrace{\frac{R_i}{\pi_i}(\widehat{Y}_i - Y_i)}_{\text{Bias-Correction Term}},$$

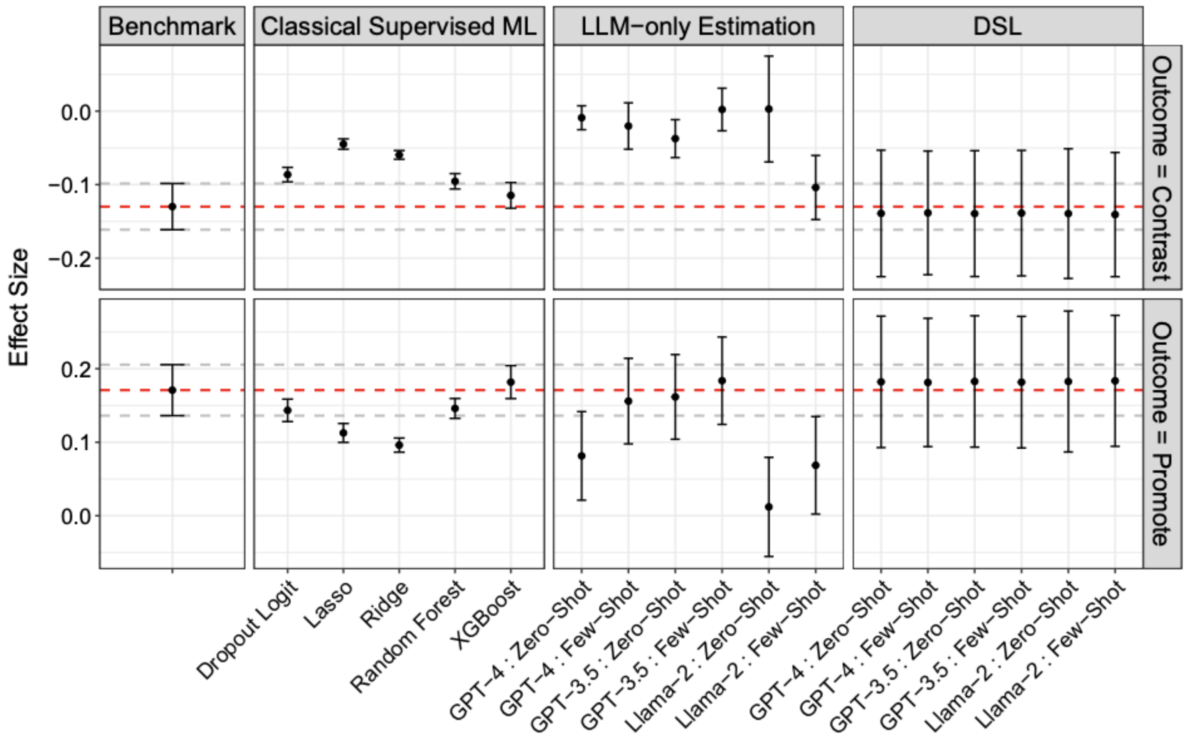
- R_i is a binary variable taking 1 if document i is expert-coded and 0 otherwise
- π_i is the probability of sampling document i for expert coding

Example: adjustment term will be large if we coded a small random sample of the data

Egami, Naoki, et al. "Using Large Language Model Annotations for the Social Sciences: A General Framework of Using Predicted Variables in Downstream Analyses.”." (2024).

Validation through simulations

Estimates and Standard Errors



References and Acknowledgements

- Slide thanks to Jurafasky & Martin: https://web.stanford.edu/~jurafsky/slp3/
- Jurafsky & Martin Chapter 5
- Jurafsky & Martin Chapter 7
- Keith, Katherine, and Brendan O'Connor. "Uncertainty-aware generative models for inferring document class prevalence." *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018.