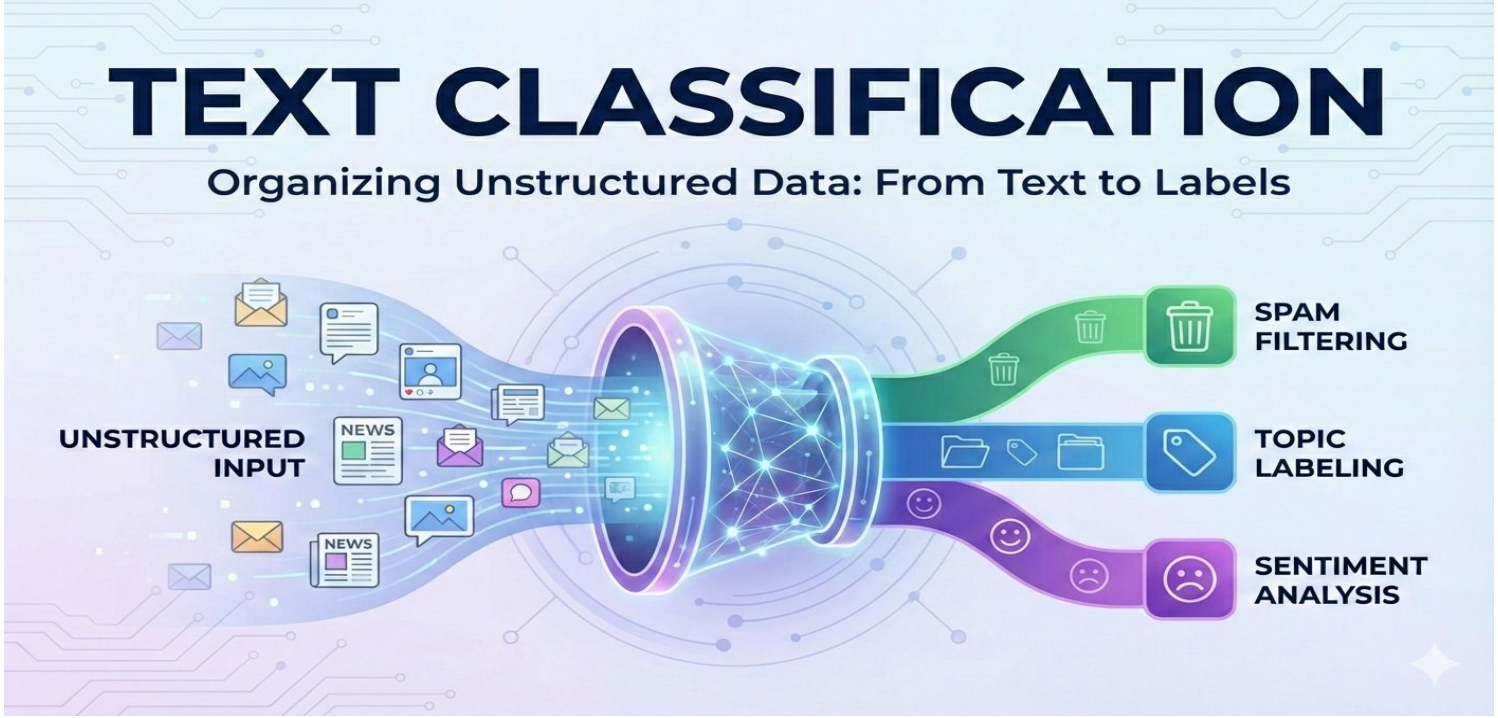


Natural Language Understanding

CSCI 4907/6515

Lecture 3: January 27, 2026

Aya Zirikly

Today

- Review of Naive Bayes
- Logistic Regression
- Discriminative vs. Generative Models
- Evaluation

Text Classification via Supervised Machine Learning

You want to be able to take a document and assign it to one of a fixed set of classes

A document (text)

“It was pathetic. The worst part about it was the boxing scenes.”

A fixed set of classes

c_0: Negative sentiment
c_1: Positive sentiment

Text Classification via Supervised Machine Learning

You want to be able to take a document and assign it to one of a fixed set of classes

A document (text)

A fixed set of classes

“It was pathetic. The worst part about it was the boxing scenes.”

c_0: Negative sentiment

c_1: Positive sentiment

1. In supervised machine learning, collect (or access) a training set of documents with labels

...zany characters and richly applied satire, and some great plot twists

It was pathetic. The worst part about it was the boxing scenes...

...awesome caramel sauce and sweet toasty almonds. I love this place!

...awful pizza and ridiculously overpriced...

1. In supervised machine learning, collect (or access) a training set of documents with labels

\+ *...zany characters and richly applied satire, and some great plot twists*

\– *It was pathetic. The worst part about it was the boxing scenes...*

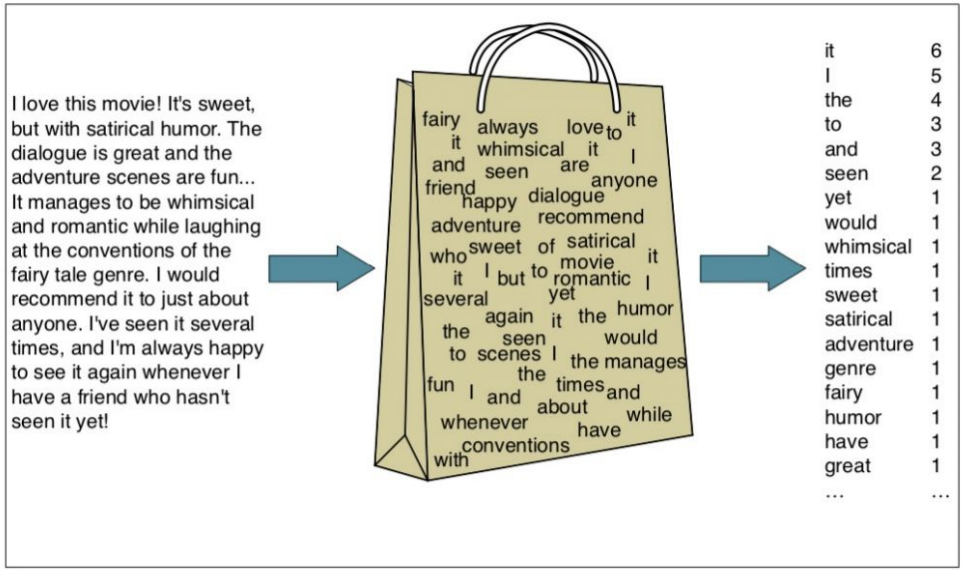
\+ *...awesome caramel sauce and sweet toasty almonds. I love this place!*

\– *...awful pizza and ridiculously overpriced...*

2. Represent text documents using features

2. Represent text documents using features

Bag-of-words document representation



Features

Example: "Enough thinking"

Features: [enough, thinking] (after lower casing)
Vectors: [1,1]

Now we see
Thinking is thinking

Feature (Word)	Value (Count)	Meaning
enough	**0**	The word "enough" appears 0 times.
thinking	**2**	The word "thinking" appears 2 times.
is	**1**	The word "is" appears 1 time.

Bag of words representation

A simplified representation of text that represents a document as an unordered set of words (a "bag"), disregarding grammar and word order but keeping track of **frequency** (counts).

Corpus:

- *Doc A:* "the dog bit the man."
- *Doc B:* "the man bit the dog."

Vocabulary (Unique Types): `["bit", "dog", "man", "the"]`
Vector Representation:

- *Doc A Vector:* `[1, 1, 1, 2]` (Contains 2 'the', 1 'dog', 1 'bit', 1 'man')
- *Doc B Vector:* `[1, 1, 1, 2]`
- *Note:* The vectors are identical, even though the meanings are opposite!

Bag of words representation

A simplified representation of text that represents a document as an unordered set of words (a "bag"), disregarding grammar and word order but keeping track of **frequen**

Corpus:

- *Doc A:* "the dog bit the man."
- *Doc B:* "the man bit the dog."

Vocabulary (Unique Types): `["bit", "dog", "man", "the"`
Vector Representation:

- *Doc A Vector:* `[1, 1, 1, 2]` (Contains 2 'the', 1 'dog', 1 'bi
- *Doc B Vector:* `[1, 1, 1, 2]`
- *Note:* The vectors are identical, even though the meanings are opposite!

Loss of Context: "Not bad" and "Bad not" look the same.

Dimensionality: The vector length equals the size of your entire vocabulary (often very large).

Sparsity: Most documents only contain a tiny fraction of the total vocabulary, resulting in vectors filled mostly with zeros.

3. Use labeled training set to learn a function that maps from documents (represented using features) to classes

Last time: Naive Bayes

- Probabilistic supervised model: Calculate P(c|d) to classify document as +/-

Document: “It was pathetic. The worst part about it was the boxing scenes.”

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

Training Naive Bayes

1. Extract vocabulary, V, from the training set. Drop any unknown words in test.
2. Calculate class priors

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

1. Calculate probability of each word in the vocabulary occurring in a document from each class type

$$\hat{P}(w_i|c) = \frac{count(w_i,c)+1}{\sum_{w\in V}(count(w,c)+1)} = \frac{count(w_i,c)+1}{\left(\sum_{w\in V} count(w,c)\right)+|V|}$$

In practice, use logs instead

$$\text{positions} \leftarrow \text{all word positions in test document}$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in positions} P(w_i|c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

Let's do a worked sentiment example!

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

A worked sentiment example with add-1 smoothing

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable ~~with~~ no fun

1. Prior from training:

$$\hat{P}(c_j) = \frac{N_{c_j}}{N_{total}}$$

P(-) = 3/5

P(+) = 2/5

2. Drop "with"

3. Likelihoods from training:

$$p(w_i|c) = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|}$$

$$P(\text{“predictable”}|-) = \frac{1+1}{14+20} \qquad P(\text{“predictable”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“no”}|-) = \frac{1+1}{14+20} \qquad P(\text{“no”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“fun”}|-) = \frac{0+1}{14+20} \qquad P(\text{“fun”}|+) = \frac{1+1}{9+20}$$

4. Scoring the test set:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

Key concepts from what we've seen so far

- Corpus
- Text classification
- Classifier
- Naive Bayes
- Maximum likelihood estimate: parameters are chosen to maximize the likelihood that the assume model results in the observed data
- Smoothing: what's the goal? Add-1 smoothing

Today: Logistic Regression

- Like Naive Bayes, this is a probabilistic, supervised classification model.
- Trying to calculate: P(y = 1 | x)
  - E.g., P(y = ‘positive sentiment’ | document represented as features)

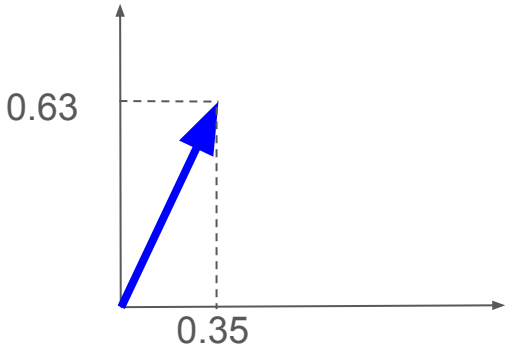
Represent text documents using features

Represent text documents using features

- For each document, a vector **x** of input features
- Vector = object with direction and magnitude, given by values (coordinates in different dimensions); list/array of numbers
  - E.g., [0.35, 0.63], [0.35, 0.63, -0.21, 0.08]

Represent text documents using features

- For each document, a vector **x** of input features
- Vector = object with direction and magnitude, given by values (coordinates in different dimensions); list/array of numbers
  - E.g., [0.35, 0.63], [0.35, 0.63, -0.21, 0.08]



Represent text documents using features

- For each document, a vector **x** of input features
- Vector = object with direction and magnitude, given by values (coordinates in different dimensions); list/array of numbers
  - E.g., [0.35, 0.63], [0.35, 0.63, -0.21, 0.08]
- Feature: types of information that the classifier can learn from
  - E.g., # of occurrences of word ‘awesome’, length of document, etc.

Represent text documents using features

- For each document, a vector **x** of input features
- Vector = object with direction and magnitude, given by values (coordinates in different dimensions); list/array of numbers
  - E.g., [0.35, 0.63], [0.35, 0.63, -0.21, 0.08]

In Sparse Models (e.g., Bag of Words, TF-IDF)

- The dimension is controlled by the Vocabulary Size (|V|)
  - If your text corpus has 20,000 unique words, every document vector will have a length of 20,000.

Features

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ 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	ln(word count of doc)	$\ln(66) = 4.19$

Let's assume for the moment that we've already learned a real-valued weight for

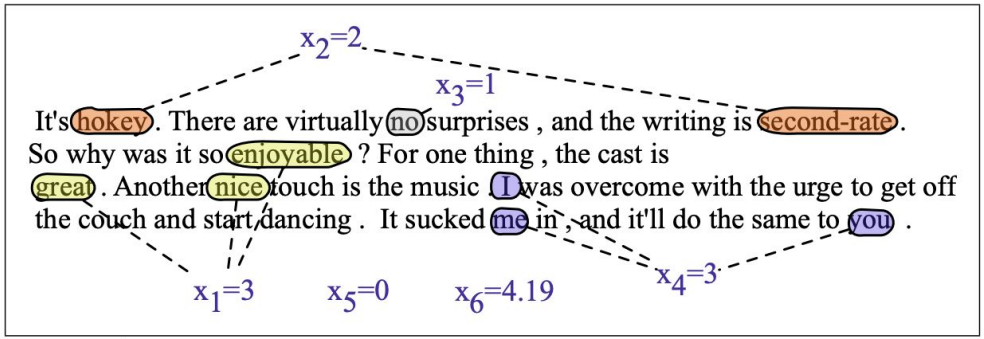


Figure 5.2 A sample mini test document showing the extracted features in the vector x.

Features

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ 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	ln(word count of doc)	$\ln(66) = 4.19$

Let's assume for the moment that we've already learned a real-valued weight for

Feature Scaling: Raw word counts can vary wildly

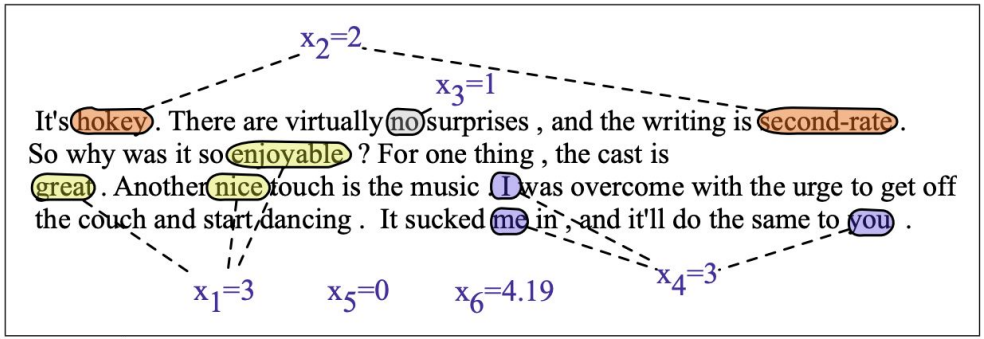


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Features

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ 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	ln(word count of doc)	ln(66) = 4.19

x = [3, 2, 1, 3, 0, 4.19]

Let's assume for the moment that we've already learned a real-valued weight for

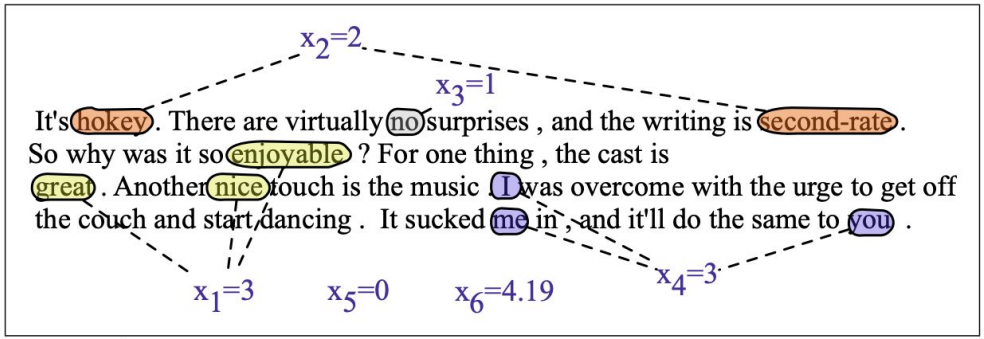


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

3. Use labeled training set to learn a function that maps from documents (represented using features) to classes

3. Use labeled training set to learn a function that maps from documents (represented using features) to classes

Let’s start by considering the form of the function and how to determine the class of a new document once we have learned the function

3. Use labeled training set to learn a function that maps from documents (represented using features) to classes

- During training, logistic regression learns a vector of real-valued *weights* and a bias term, to transform the input and map to an output

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

3. Use labeled training set to learn a function that maps from documents (represented using features) to classes

- During training, logistic regression learns a vector of real-valued *weights* and a bias term, to transform the input and map to an output

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

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

3. Use labeled training set to learn a function that maps from documents (represented using features) to classes

- During training, logistic regression learns a vector of real-valued *weights* and a bias term, to transform the input and map to an output

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

If *z* is high then we'll say y=1; if *z* is low then we'll say y=0
How should we formalize this?

Then, sigmoid function takes z and maps it to a probability

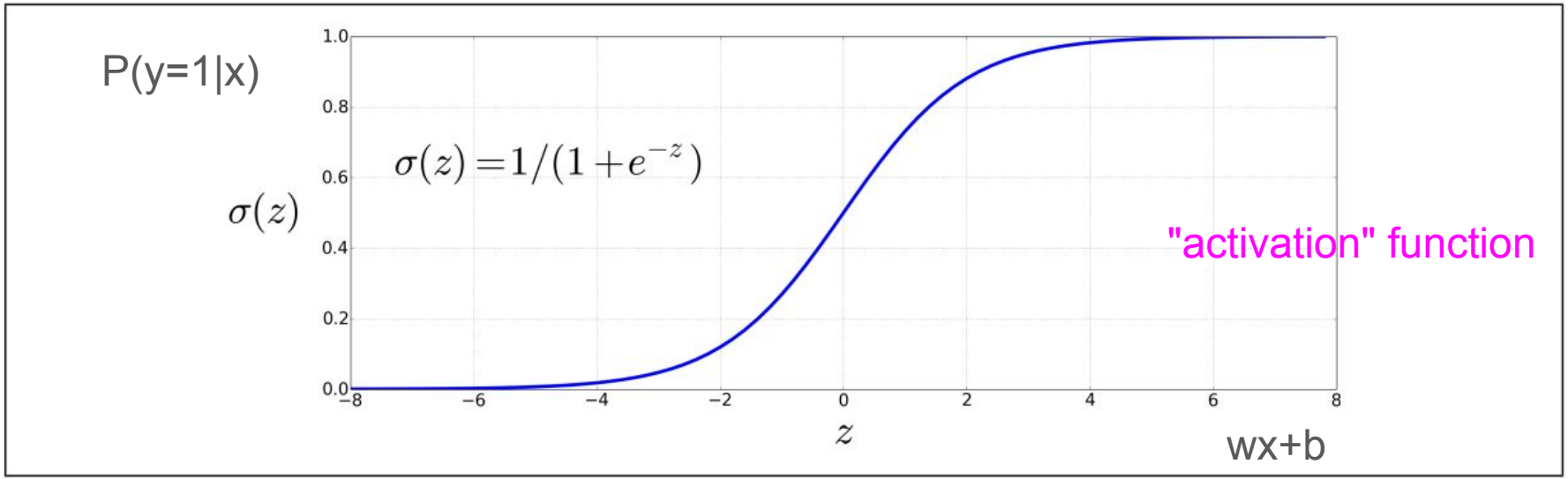


Figure 5.1 The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $(0, 1)$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

Sigmoid function

How it behaves:

- **Large positive numbers** become close to **1**. (e.g., Sigmoid(10) ~ 0.9999)
- **Large negative numbers** become close to **0**. (e.g., Sigmoid(-10) ~ 0.00004)
- **Zero** becomes exactly **0.5**

Why sigmoid

A standard linear model produces a raw score (called a **logit**): 5, -120, or 4000.

Hard to interpret

- **Problem:** "This email has a spam score of 42." (Is that high? Low?)
- **Sigmoid Solution:** "This email has a 0.99 probability of being spam."
  Because the output is strictly between 0 and 1, we can treat it as a probability percentage.

Creates a smooth Decision Boundary

In binary classification, we usually set a threshold at **0.5**

- If Sigmoid(score) > 0.5 → Class 1 (Positive)
- If Sigmoid(score) < 0.5 → Class 0 (Negative)

Differentiable (Crucial for Training)

Unlike a hard "step function" (which jumps instantly from 0 to 1), the Sigmoid curve is smooth. This allows algorithms like **Gradient Descent** to calculate the slope (derivative) at any point.

- This tells the model ***how much*** to adjust its weights during training. If the model is "sort of wrong" (output 0.6 instead of 1), the slope tells it which direction to nudge the parameters.

Turning a probability into a classification

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

0.5 here is called the **decision boundary**

Turning a probability into a classification

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

0.5 here is called the **decision boundary**

Decision boundary other than 0.5?

Decision boundary

Move the threshold to minimize the **expensive** mistake

- High threshold
  - Use cases?
- Low threshold
  - Use cases?

Decision boundary

Move the threshold to minimize the **expensive** mistake

- High threshold
  - Biometric Security (FaceID)
- Low threshold
  - Cancer screening

Example: Which class will be predicted?

	Feature 1 count(pos)	Feature 2 count(neg)	Feature 3 Has “no”	Feature 4 count(pron)	Feature 5 Has “!”	Feature 6 ln(word count)
x	3	2	1	3	0	4.19
weights	2.5	-5	-1.2	0.5	2.0	0.7

b = 0.4

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ 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	ln(word count of doc)	$\ln(66) = 4.19$

Let’s assume for the moment that we’ve already learned a real-valued weight for

Example: Which class will be predicted?

	Feature 1 count(pos)	Feature 2 count(neg)	Feature 3 Has “no”	Feature 4 count(pron)	Feature 5 Has “!”	Feature 6 ln(word count)
x	3	2	1	3	0	4.19
weights	2.5	-5	-1.2	0.5	2.0	0.7

b = 0.4

$$z = \text{???}$$

Example: Which class will be predicted?

	Feature 1 count(pos)	Feature 2 count(neg)	Feature 3 Has “no”	Feature 4 count(pron)	Feature 5 Has “!”	Feature 6 ln(word count)
x	3	2	1	3	0	4.19
weights	2.5	-5	-1.2	0.5	2.0	0.7

b = 0.4

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

Example: Which class will be predicted?

	Feature 1 count(pos)	Feature 2 count(neg)	Feature 3 Has “no”	Feature 4 count(pron)	Feature 5 Has “!”	Feature 6 ln(word count)
x	3	2	1	3	0	4.19
weights	2.5	-5	-1.2	0.5	2.0	0.7

b = 0.4

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

$z = 3(2.5) + 2(-5) + 1(-1.2) + 3(0.5) + 0(2.0) + 4.19(0.7) + 0.4$

$z = 1.133$

Example: Which class will be predicted?

$P(y=1|x) = \sigma(z) = 1/(1+e^{-1.133}) = 0.756$

$P(y=0|x) = 1-0.756 = 0.244$

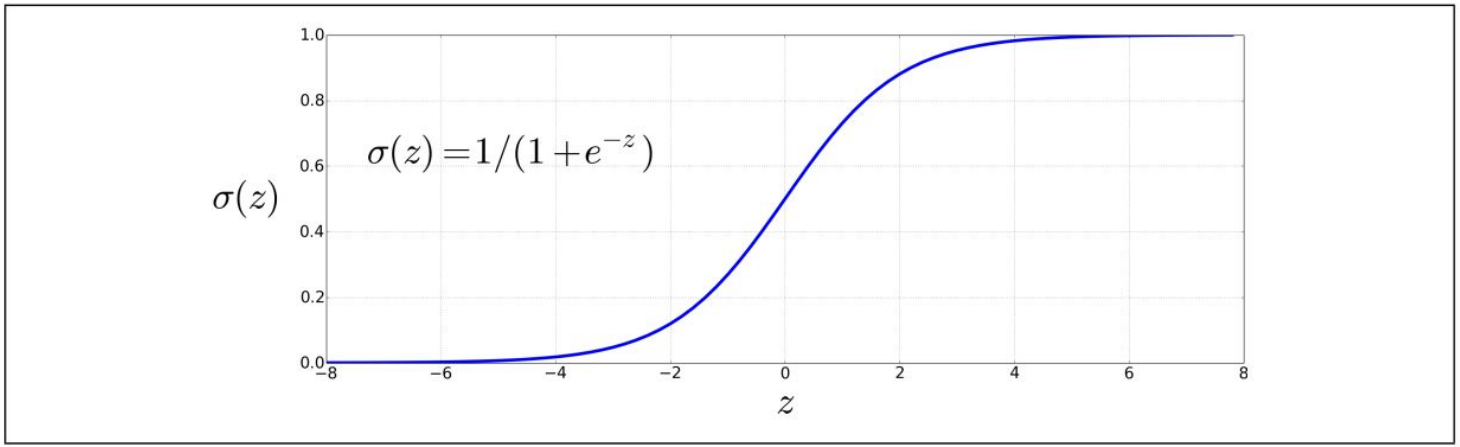


Figure 5.1 The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $(0, 1)$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

Example: Which class will be predicted?

$P(y=1|x) = \sigma(z) = 1/(1+e^{-1.133}) = 0.756$

$P(y=0|x) = 1-0.756 = 0.244$ Predict class 1

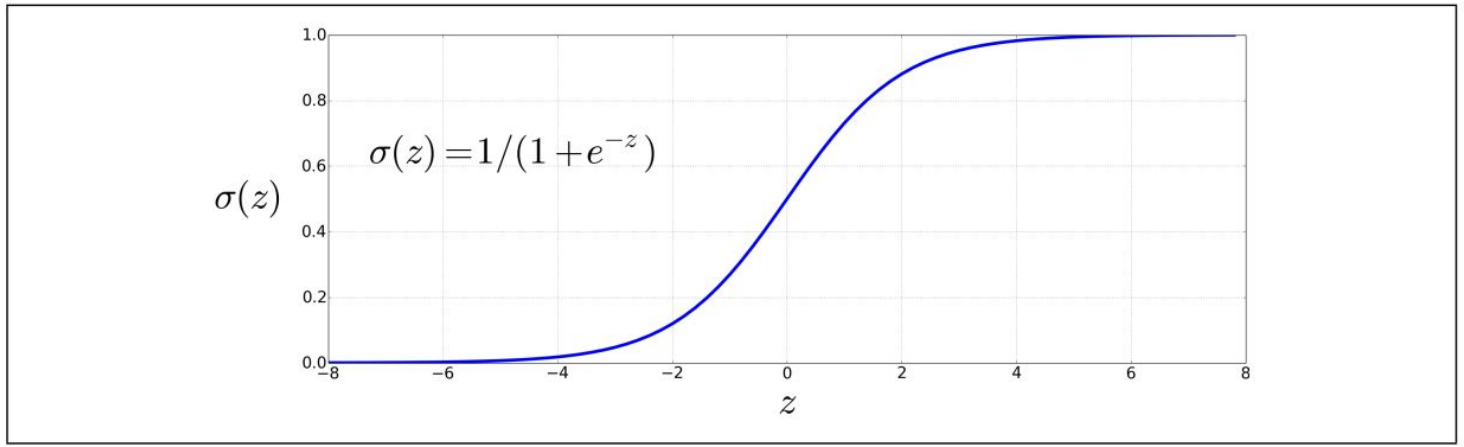


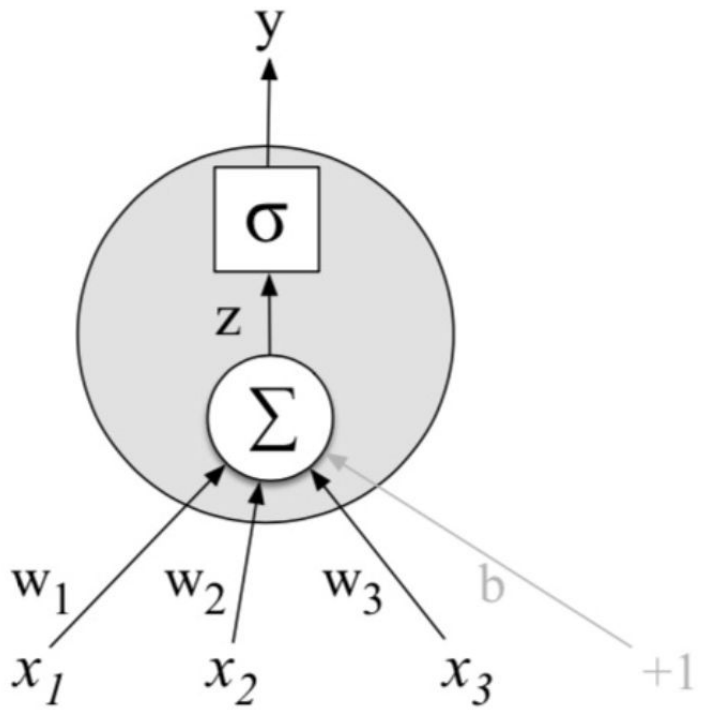
Figure 5.1 The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $(0, 1)$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

Alternative dot product notation

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b \qquad z = \mathbf{w} \cdot \mathbf{x} + b$$

x	3	2	1	3	0	4.19
w	2.5	-5	-1.2	0.5	2.0	0.7

Diagram of logistic regression calculation



Summary so far

- We now know how to determine the class of a new document **given** the weights, **w**, and the bias term, b.
  - Represent the document as a vector of feature values
  - Calculate σ(*z*) to get a probability
  - Choose the class with the highest probability

- But how do we learn the weight vector, **w**, and the bias term, b, from our training data?
  - We want to set **w** and b to minimize the distance between the true label, y, and our predicted label $\hat{y}$

Training in Logistic Regression

Training in Logistic Regression (High-level idea)

1. Randomly set **w** and b (“initialize” **w** and b)
2. Calculate $\hat{y}$ = P(y=1|**x**) using those initialized weight values for a document in our training set (remember: we have the doc’s true label, y, which is 0 or 1)
3. Compare y and $\hat{y}$.
4. Update **w** and b to make it more likely to predict y correctly for that instance
5. Repeat with another training item in our document.
6. …

Training in Logistic Regression (High-level idea)

1. Randomly set **w** and b ("initialize" **w** and b)
2. Calculate $\hat{y}$ = P(y=1|**x**) using those initialized weight values for a document in our training set (remember: we have the doc's true label, y, which is 0 or 1)
3. Compare y and $\hat{y}$.
4. Update **w** and b to make it more likely to predict y correctly for that instance
5. Repeat with another training item in our document.
6. …

Training in Logistic Regression

We need:

- A way of comparing *y* and $\hat{y}$ (true vs. predicted label): i.e., for a given **x**, how far is $\hat{y} = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$ from *y* (which is 0 or 1)

- A way of updating **w** and b to minimize the distance between y and $\hat{y}$

Training in Logistic Regression

We need:

- A way of comparing *y* and $\hat{y}$ (true vs. predicted label): i.e., for a given **x**, how far is $\hat{y} = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$ from *y* (which is 0 or 1)

  Loss function: Cross-entropy loss

- A way of updating **w** and b to minimize the distance between y and ŷ

Training in Logistic Regression

We need:

- A way of comparing *y* and $\hat{y}$ (true vs. predicted label): i.e., for a given **x**, how far is $\hat{y} = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$ from *y* (which is 0 or 1)

  Loss function: Cross-entropy loss

- A way of updating **w** and b to minimize the distance between y and ŷ

  Learning algorithm: Stochastic gradient descent

Loss function

The distance between $\hat{y}$ and y

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

Loss function: cross-entropy loss function (negative log-likelihood loss)

- Choose the parameters (**w**, b) that maximize the (log) probability of the true labels, **y**, in the training data given the input features, **x**
- Measures how well the predicted probability distribution matches the actual true labels

Measure of "Surprise"

Bernoulli distribution and likelihood

In logistic regression, you assume that each target value follows a Bernoulli distribution - takes on value 1 with some probability p and 0 with probability $1-p$

$$p = \sigma(w \cdot x + b)$$

The statistical assumption

$$y \mid x \sim \text{Bernoulli}(\sigma(w \cdot x + b))$$

Bernoulli distribution and likelihood

The simplest probability distribution in statistics.

It models a **single experiment** with exactly **two possible outcomes**.

- **Success (k=1):** "Yes", "Heads", "Spam", "Positive".
- **Failure (k=0):** "No", "Tails", "Not Spam", "Negative".

2. The Single Parameter (p)

- The distribution is controlled by one number: **p**
- p = The probability of Success (k=1)
- Consequently, the probability of Failure (k=0) is always **1 - p**

3. Probability Mass Function $$P(y|\hat{y}) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

Bernoulli distribution and likelihood

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

if **(y=1)**

$$\hat{y}^{1}(1-\hat{y})^{0} \rightarrow \hat{y} \cdot 1 \rightarrow \hat{\mathbf{y}}$$

if **(y=0):**

$$\hat{y}^{0}(1-\hat{y})^{1} \rightarrow 1 \cdot (1-\hat{y}) \rightarrow \mathbf{1} - \hat{\mathbf{y}}$$

Loss function: cross-entropy loss function

P(y=1|x) = $\hat{y}$ and P(y=0|x) = 1 - $\hat{y}$

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

Take the log of both sides:

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

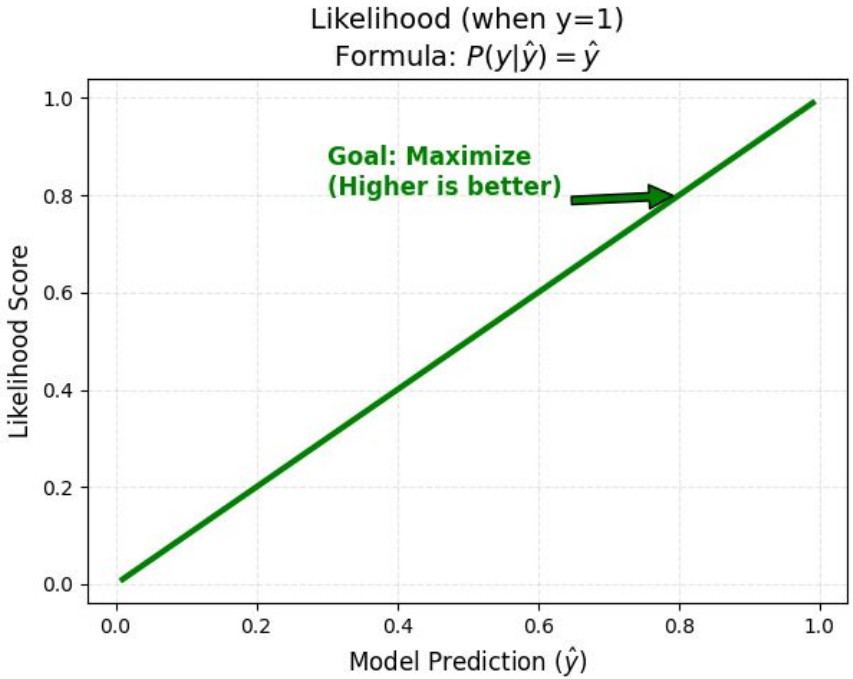
We want a function to *minimize*, so we flip the sign:

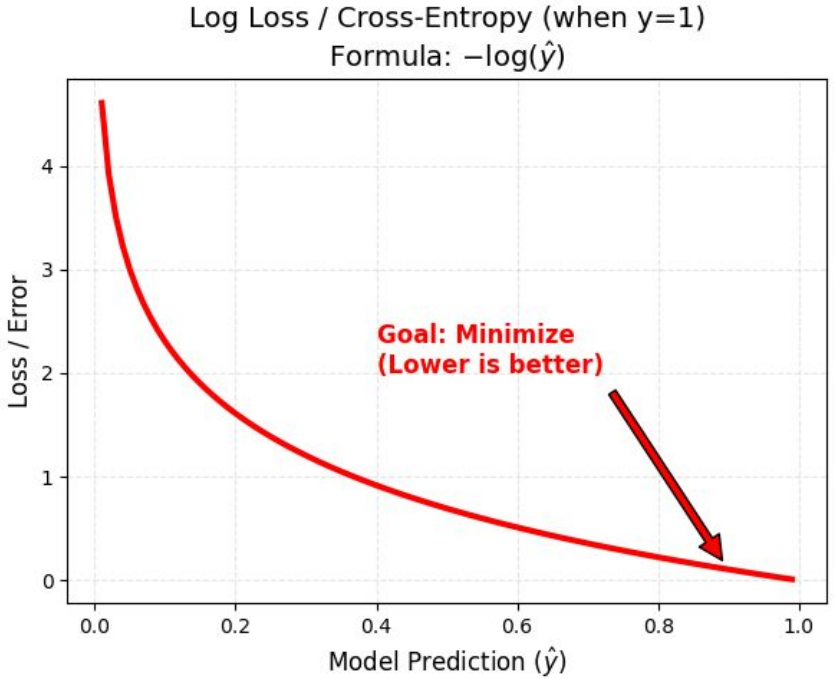
$$L_{\text{CE}}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

Plug in $\hat{y} = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$

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

Likelihood vs. loss





Example

"Boring movie"

- **Input (x):** The word **"boring"** appears x_boring = 1
- **True Label (y): 0** (Negative Sentiment).
- **Model Prediction y^: 0.80**
  - *Situation:* The model is currently **wrong**. It thinks "boring" is positive (80% confidence).

Calculate Cross-Entropy Loss

$$L = - [y \cdot \ln(\hat{y}) + (1 - y) \cdot \ln(1 - \hat{y})]$$

$$L = - [0 + (1) \cdot \ln(1 - 0.80)]$$

$$L = - \ln(0.20)$$

$$L \approx 1.61$$

High loss, threshold -ln(0.5) = 0.69
random guess

Result: The model is penalized heavily for being confident (0.8) about the wrong answer.

Learning Algorithm: Our goal is to 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:

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

Learning Algorithm: Stochastic Gradient Descent

High-level idea: Find the parameters (**w**, b) that minimize the loss function by calculating the **gradient** (the direction in which the function's slope is rising most steeply), and moving in the opposite direction

Gradients

The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

Gradient Descent: Find the gradient of the loss function at the current point and move in the **opposite** direction.

The intuition - The blind hiker

1. The Problem

- Imagine you are standing on top of a mountain at night (blindfolded).
- **The Mountain:** Represents your **Loss Function** (High Error).
- **The Bottom of the Valley:** Represents the **Optimal Model** (Lowest Error).
- **Your Goal:** Get to the bottom as quickly as possible without falling off a cliff.

2. The Solution (Gradient Descent)

- You cannot see the bottom, so you feel the ground with your feet.
- **The Gradient:** The "slope" or steepness of the ground right where you are standing.
- **The Strategy:** If the slope goes *up* to the right, you take a step to the *left*. You always step in the **opposite direction** of the slope.

The math

$$w_{new} = w_{old} - \text{Learning Rate} \times \text{Gradient}$$

w_old: Your current weight (e.g., 2.5).

Gradient: The slope. Tells you which direction is "uphill."

- *Positive Slope:* We subtract (go left).
- *Negative Slope:* We add (go right).

Minus Sign (−): This ensures we always go **down** (against the slope).

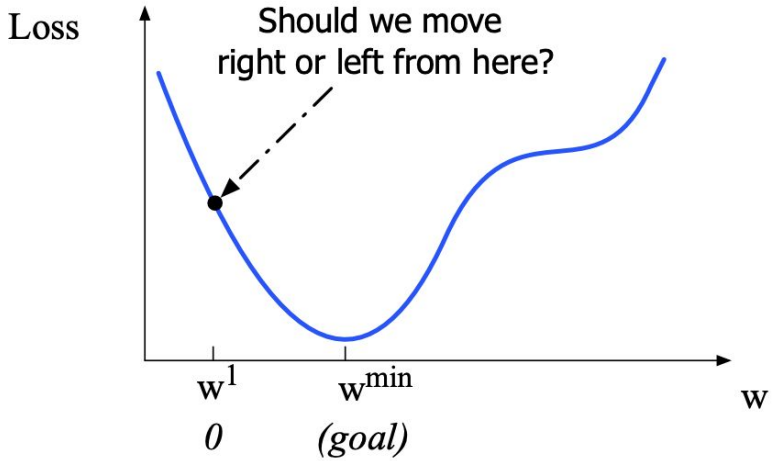
Learning Rate (η): The "Step Size."

  - *Too small:* Takes forever to reach the bottom.
  - *Too big:* You might overshoot the bottom and jump to the other side.

The math -cont'd

Repeat until gradient is close to 0

1. **Predict:** The model takes a guess using current weights (w).
2. **Calculate Loss:** Compare the guess (ŷ) to the truth (y).
3. **Compute Gradient:** Calculate how much the Loss changes if we wiggle w. Gradient = $(\hat{y} - y) \cdot x$
4. **Update Weights:** update w slightly in the opposite direction.



Learning Algorithm: Stochastic Gradient Descent

What about with two parameters:

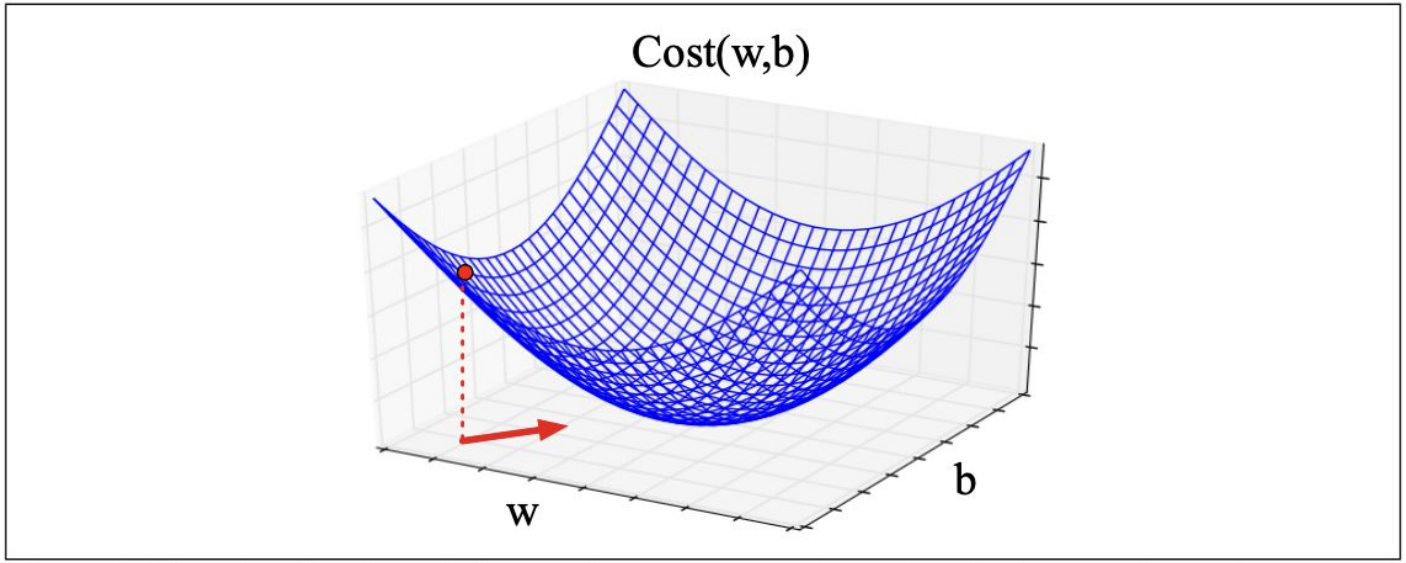


Figure 5.5 Visualization of the gradient vector at the red point in two dimensions *w* and *b*, showing a red arrow in the x-y plane pointing in the direction we will go to look for the minimum: the opposite direction of the gradient (recall that the gradient points in the direction of increase not decrease).

Stochastic Gradient Descent - with two parameters

Think of: marble rolling down the inside of a salad bowl. The marble moves along both the X-axis (w_1) and Y-axis (w_2) simultaneously to find the lowest point in the center.

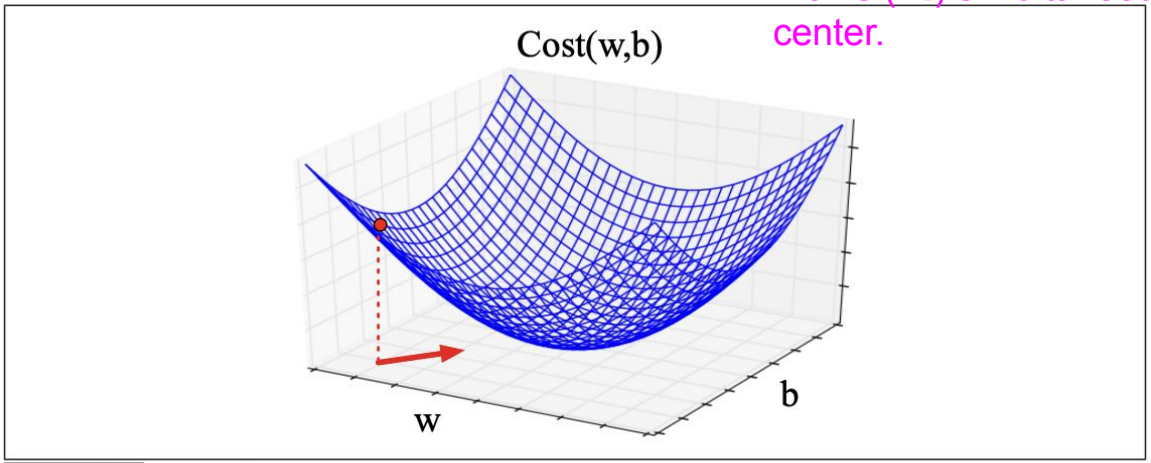


Figure 5.5 Visualization of the gradient vector at the red point in two dimensions *w* and *b*, showing a red arrow in the x-y plane pointing in the direction we will go to look for the minimum: the opposite direction of the gradient (recall that the gradient points in the direction of increase not decrease).

Stochastic Gradient Descent - with two parameters

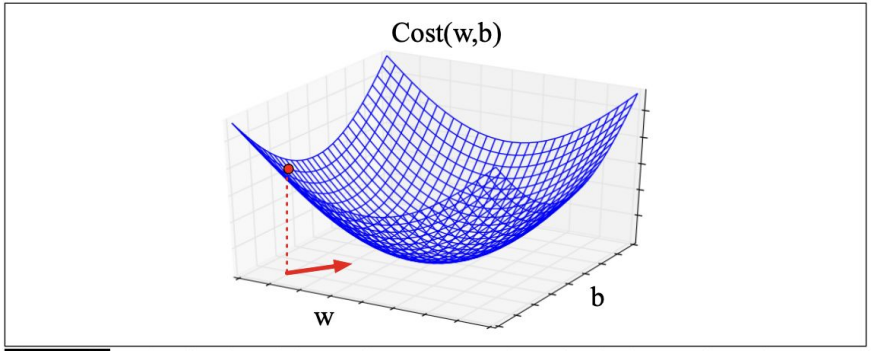


Figure 5.5 Visualization of the gradient vector at the red point in two dimensions *w* and *b*, showing a red arrow in the x-y plane pointing in the direction we will go to look for the minimum: the opposite direction of the gradient (recall that the gradient points in the direction of increase not decrease).

The Gradient Vector: $\nabla L = [\, \partial L/\partial w_1 \,,\, \partial L/\partial w_2 \,]$

The Update Rule: • $w_1 = w_1 - \eta \cdot (\hat{y} - y) \cdot x_1$ • $w_2 = w_2 - \eta \cdot (\hat{y} - y) \cdot x_2$

The gradient

We'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious:

$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

The final equation for updating θ based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$$

What are these partial derivatives for logistic regression?

The loss function

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

The elegant derivative of this function (see textbook 5.8 for derivation)

$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

What are these partial derivatives for logistic regression?

$$\nabla_{w,b} L = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial \mathbf{w}_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial \mathbf{w}_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(\mathbf{w}\cdot\mathbf{x}+b)-y)\mathbf{x}_1 \\ (\sigma(\mathbf{w}\cdot\mathbf{x}+b)-y)\mathbf{x}_2 \\ \sigma(\mathbf{w}\cdot\mathbf{x}+b)-y \end{bmatrix}$$

Working through an example

One step of gradient descent

A mini-sentiment example, where the true y=1 (positive)

Two features:

$x_1 = 3$ (count of positive lexicon words)

$x_2 = 2$ (count of negative lexicon words)

Assume 3 parameters (2 weights and 1 bias) in Θ^0 are zero:

$w_1 = w_2 = b = 0$

$\eta = 0.1$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y) \qquad \eta = 0.1;$$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Note that enough negative examples would eventually make w_2 negative

Questions?

Summary

- We saw two methods for text classification
- Both are probabilistic, supervised algorithms
- Naive Bayes
  - $P(label|doc) \propto P(f_i|label)*P(label)$
- Logistic Regression
  - $P(label|doc) = 1/(1+e^{(\mathbf{wx} + b)})$

Generative vs. discriminative classification models

Naive Bayes is a **generative** classifier

by contrast:

Logistic regression is a **discriminative** classifier

Suppose we're distinguishing cat from dog images

imagenet

imagenet

Generative Classifier:

- Build a model of what's in a cat image
  - Knows about whiskers, ears, eyes
  - Assigns a probability to any image:
    - how cat-y is this image?

Also build a model for dog images

Now given a new image:

Run both models and see which one fits better

Discriminative Classifier

Just try to distinguish dogs from cats

Oh look, dogs have collars!

Let's ignore everything else

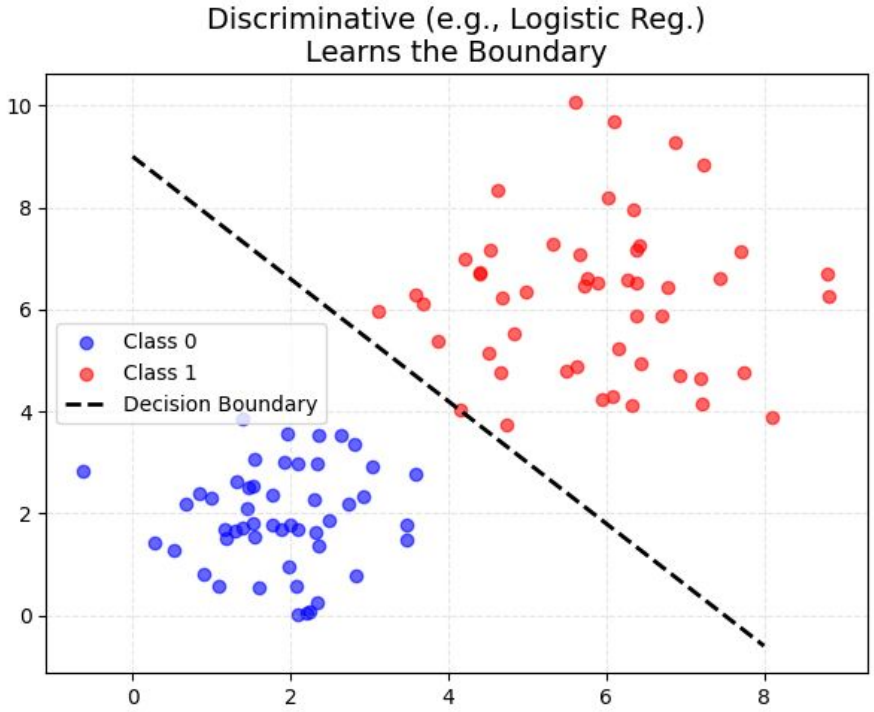
The core concepts

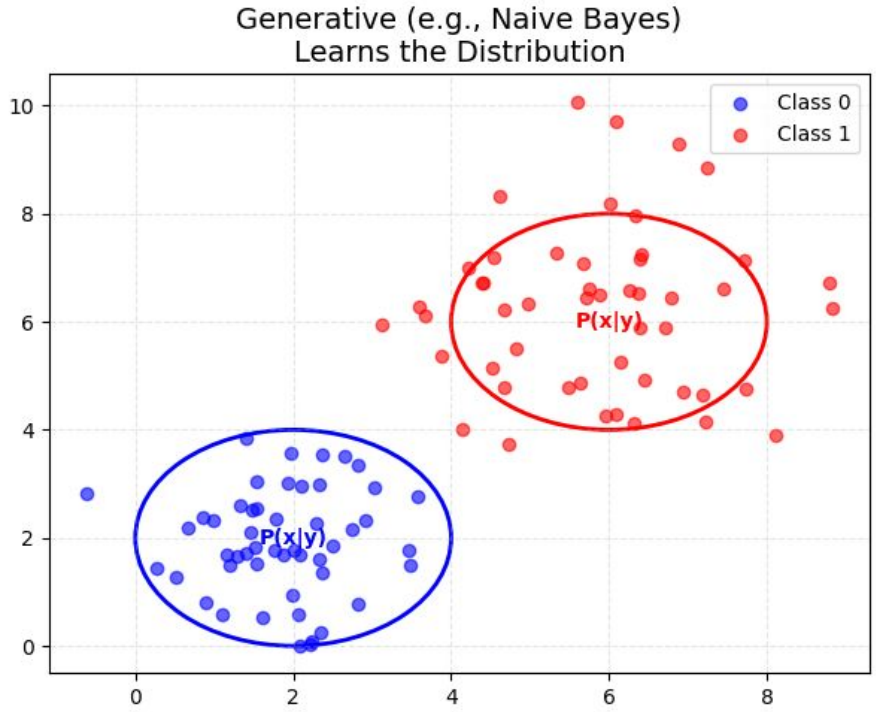
Generative Models

- **Focus:** Learns the **distribution** of each class individually.
- **Question:** "What does a 'Cat' look like? What does a 'Dog' look like?"
- **Math:** Models P(x|y) and P(y).
- **Analogy:** To distinguish French from Spanish, a generative model learns to **speak** both languages entirely. Then, when it sees a sentence, it asks: "Which language likely generated this?"

Discriminative Models

- **Focus:** Learns the **boundary** between classes.
- **Question:** "Given this input x, which side of the line does it fall on?"
- **Math:** Models P(y|x) directly.
- **Analogy:** To distinguish French from Spanish, a discriminative model looks for specific differences (e.g., "Spanish uses 'ñ', French uses 'ç'"). It doesn't care about the grammar rules of either language, only what separates them.


Discriminative (e.g., Logistic Reg.)
Learns the Boundary
Class 0
Class 1
Decision Boundary


Generative (e.g., Naive Bayes)
Learns the Distribution
Class 0
Class 1
P(x|y)
P(x|y)

When to use what

Feature	Generative (Naive Bayes)	Discriminative (LR, SVM, Neural Nets)
Data Size	**Better for Small Data.** Converges faster because it makes stronger assumptions (interprets the data distribution).	**Better for Big Data.** As data grows, these usually beat generative models in accuracy (focuses purely on minimizing error).
Missing Data	**Excellent.** Can handle missing features by simply ignoring them and focusing on the ones present: $P(x_i, y)$	**Breaks**
Outliers	**Robust.** Because it models the "center" of the class, outliers don't shift the model much.	**Sensitive.** Outliers can pull the decision boundary significantly (especially in SVM/LR).
New Classes	**Easy.** You can add a new class (e.g., "Bird") without retraining the "Cat" and "Dog" models.	**Hard.** You must retrain the entire model from scratch to find new boundaries for everyone.

Why Discriminative Models Crash on Missing Data

1. The Rigid Structure A Discriminative model (like Logistic Regression) is a fixed equation.

- It expects an input vector of exactly size N.
- Score = $w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b$
- **Training:** The model spent hours learning that w_2 (the weight for "Age") is exactly 0.5.

2. The Failure Mechanism Imagine we are predicting heart disease, but the patient's "Age" (x_2) is missing.

• **Generative Model (The Doctor):** "I don't know the age? That's fine. I will just look at blood pressure (x_1) and cholesterol (x_3) to make my best guess based on what I *do* see."

- It simply drops the $P(x_2|y)$ term from the multiplication.

• **Discriminative Model:** "I *must* multiply 0.5 by 'Age' to function. If 'Age' is empty, I cannot calculate a sum."

Score = $(2.1 \cdot 120) + (0.5 \cdot \text{NaN}) + (1.3 \cdot 200)$ Score = NaN (System Error)

Can I fix it

To make the Discriminative model work, you are forced to **lie** to it.

- You must fill in the blank with the *average* age.
- **The Risk:** If this patient is actually very young or very old, plugging in the "Average" introduces noise and makes the prediction less accurate.

Questions?

Evaluation

Evaluating a classifier -Data split

Training Set (The "Textbook")

- **Purpose:** Used to teach the model. The model sees these answers and adjusts its weights.
- **Dev (Validation) Set (The "Practice Exam")**

Purpose: Used to tune hyperparameters (e.g., Learning Rate, Model Size).

- **Crucial:** The model *never* learns from this data directly, but we use it to measure progress and stop training early if needed.

Test Set (The "Final Exam")

- **Purpose:** Used **once** at the very end to estimate real-world performance. • **Crucial:** You must never look at this during development. If you tune your model based on Test results, you are cheating (overfitting to the test).

How much per data split

Small Data

If you have < 100,000 examples, use standard ratios to ensure the Dev/Test sets are statistically significant

- **Split:** 60% Train / 20% Dev / 20% Test
- **Split:** 70% Train / 30% Test

"Big Data"

If you have millions of examples, you don't need 20% for testing. 1% is enough to get a precise accuracy score.

Total Data: 10,000,000 images

- **Split:** 98% Train / 1% Dev / 1% Test
- **Why?** We want to give the model as much training data as possible.

How to split

Shuffle Before Splitting

- If your data is sorted by date, you might train on "2019 data" and test on "2020 data."
- Randomize the order to ensure all sets look similar.

Stratification (for Imbalanced Data)

- If you have rare classes (e.g., only 100 Fraud cases), a random split might put *all* of them in the Test set.
- Stratified Splitting ensures the ratio of classes (e.g., 99% Safe, 1% Fraud) is identical across Train, Dev, and Test.

Distribution Mismatch

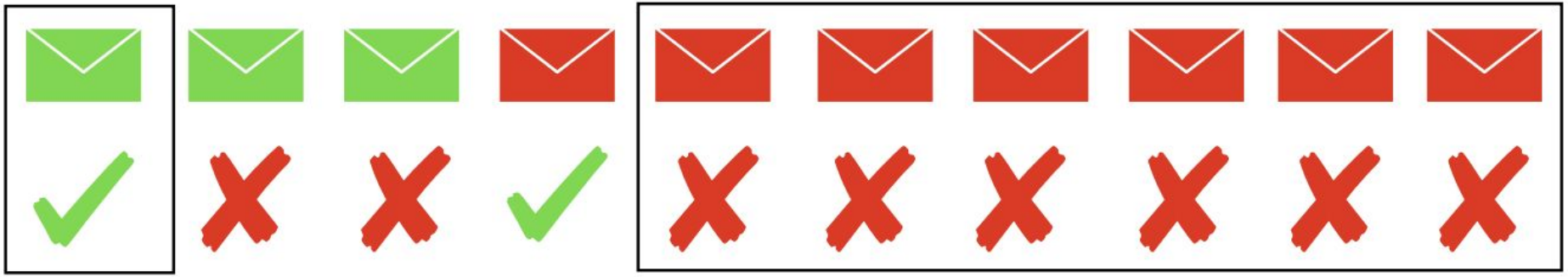
- Dev and Test sets **must** come from the same distribution.
- Bad Example: Training on high-quality professional photos, but testing on blurry mobile phone photos.
- Result: Your model will fail in production because the "Practice Exam" (Dev) was easier than the "Final Exam" (Test).

Evaluating a classifier

- Accuracy: How often is the label correct?
- Precision: Probability that it is spam, given that the model says it is.
- Recall: Probability that the model says its spam, given that it actually is spam.
- Summarizing Precision and Recall:
  - F1: Harmonic Mean of Precision and Recall

Classification Metrics

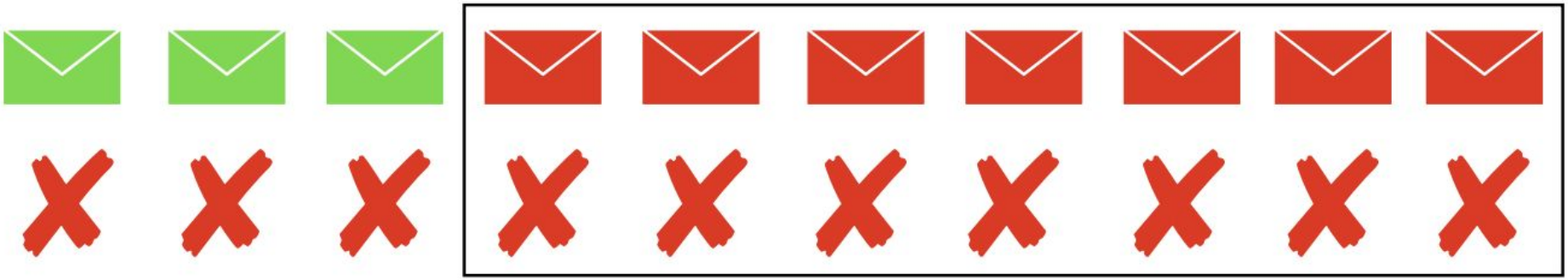
Example: Spam FIlter



Accuracy = 7/10 = 70%

Classification metrics

Example: Spam FIlter



Accuracy = 7/10 = 70%

The trap of accuracy

Imbalanced Data Accuracy is dangerous when the classes are not equal (e.g., Fraud Detection, where only 1% of transactions are fraud).

- **Scenario:** You have 100 emails. 99 are Safe, 1 is Spam.
- **The "Lazy" Model:** It just predicts "Safe" for everything.
- **99% Accuracy.**
- The model is useless. It failed to catch the *one* thing it was supposed to find.

2. The Solution We split "success" into two separate questions to expose this failure:

1. **Precision:** When you *did* predict Spam, were you right?
2. **Recall:** Out of *all* the Spam that exists, how much did you find?

Precision vs. recall (quality vs. quantity)

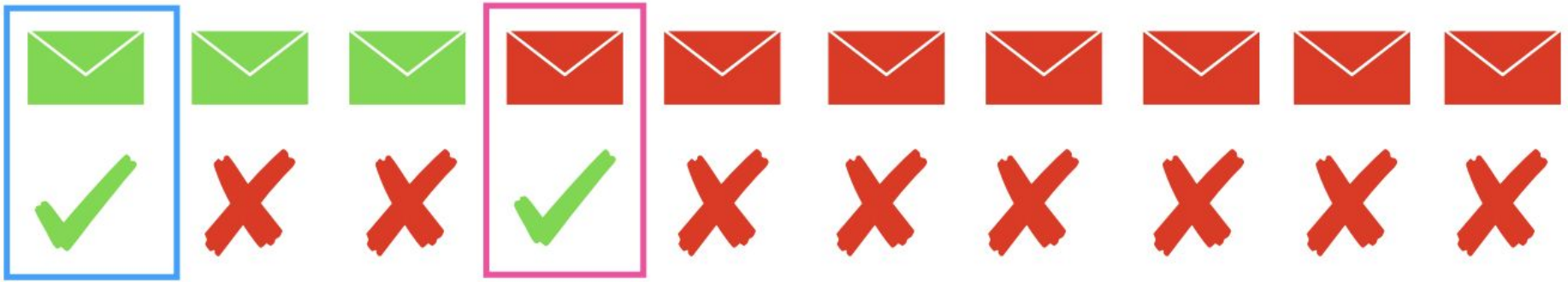
Precision (Trust Metric)

- **Question:** "Of all the emails you labeled as Spam, how many were actually Spam?"
- **Focus:** Minimizing False Positives.
- **When to prioritize:** When a false alarm is expensive.
    - *Example:* **Spam Filter.** You don't want to accidentally delete your boss's email. You would rather miss a few spam emails than lose one real one.

2. Recall (Coverage Metric)

- **Question:** "Of all the Spam that actually exists, how many did you find?"
- **Focus:** Minimizing False Negatives.
- **When to prioritize:** When missing a target is dangerous.
    - *Example:* **Cancer Diagnosis.** It is better to flag a healthy person for a checkup (False Positive) than to tell a sick person they are fine (False Negative).

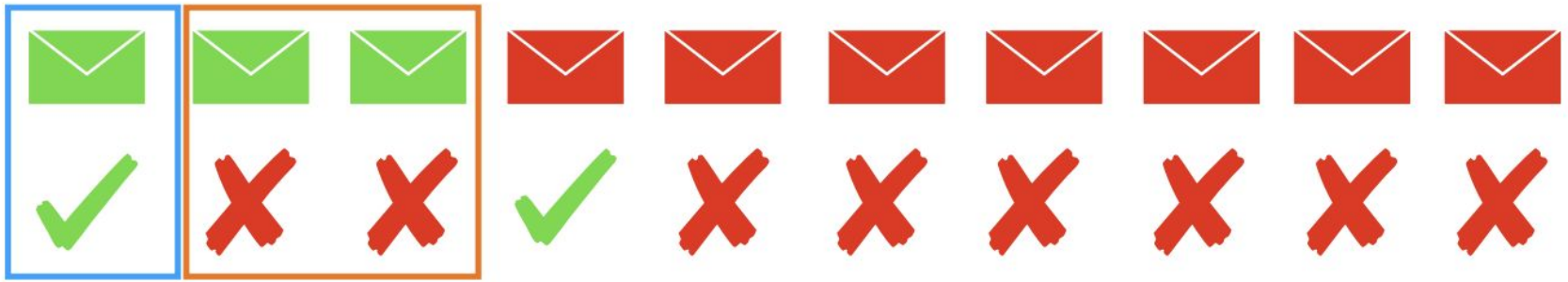
Classification metrics



$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} = \frac{1}{2}$$

Classification Metrics

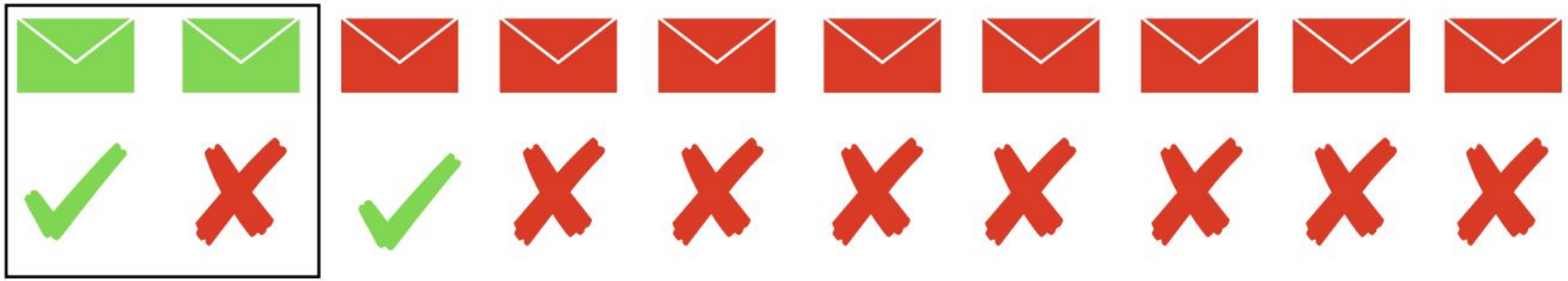
Example: Spam FIlter



$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} = \frac{1}{3}$$

Classification Metrics

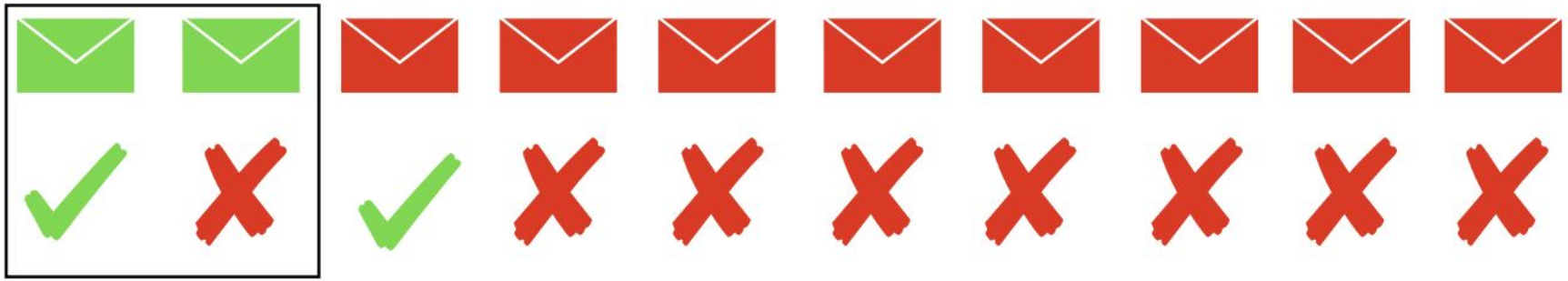
Example: Spam FIlter



$$F1 = \frac{2 * P * R}{P + R}$$

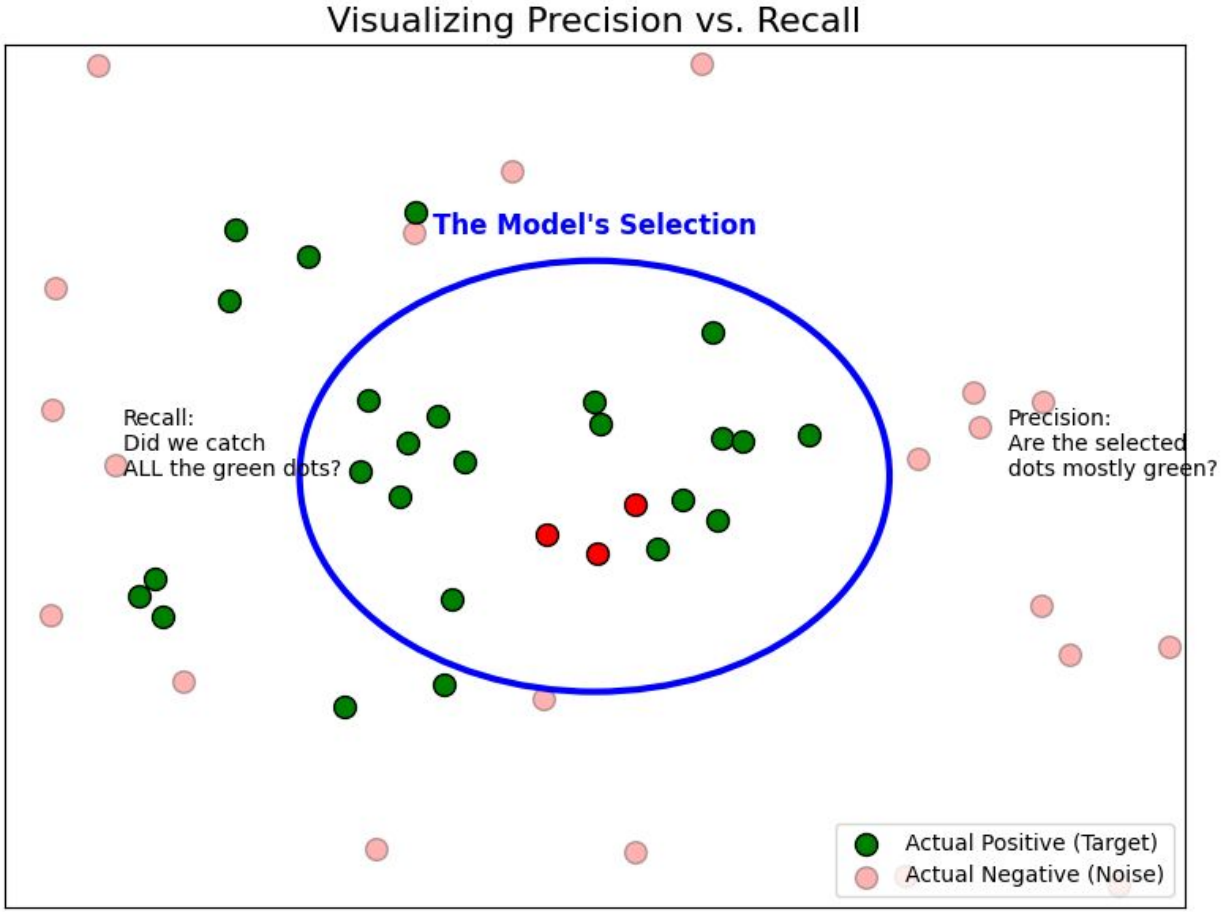
Classification Metrics

Example: Spam FIlter



$$F1 = \frac{2 * P * R}{P + R}$$

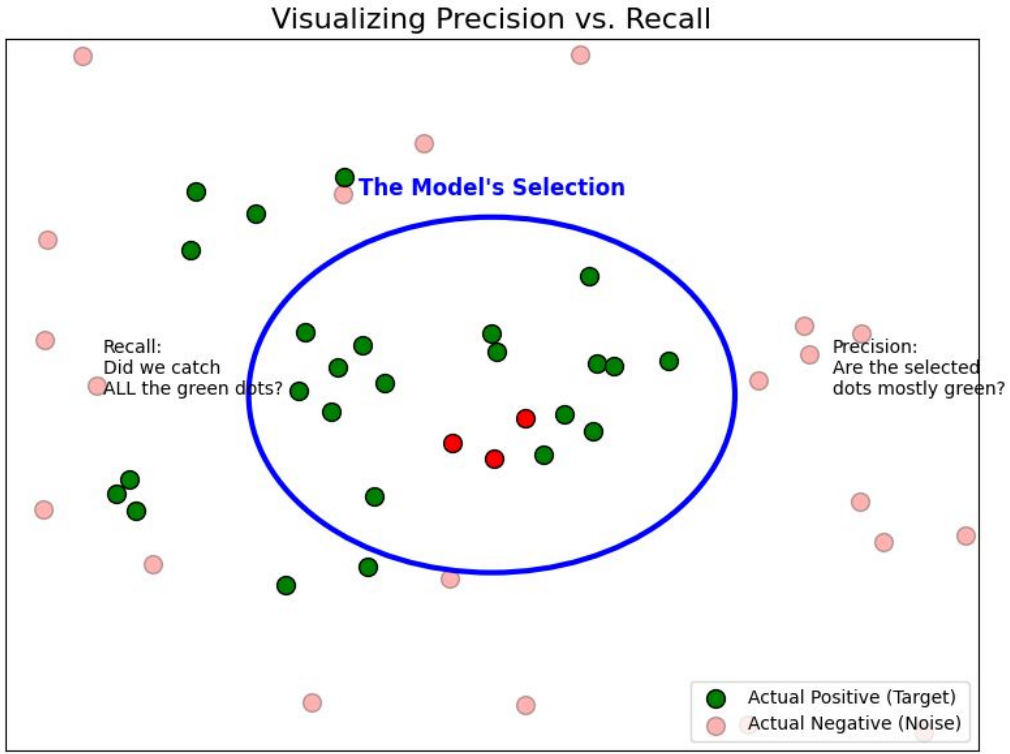
$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$


Visualizing Precision vs. Recall
The Model's Selection
Recall:
Did we catch
ALL the green dots?
Precision:
Are the selected
dots mostly green?
Actual Positive (Target)
Actual Negative (Noise)

Green Dots: Actual Positive cases.
Red Dots: Actual Negative cases.

Precision = ?

Recall = ?



Green Dots: Actual Positive cases.
Red Dots: Actual Negative cases.

Precision =
(Green dots in circle) / (Total dots in circle).

Recall =
(Green dots in circle) / (Total Green dots in the whole image).