Natural Language Processing

Class 2: Early Text Classification

Adam Faulkner

Sept 2, 2025

Is this spam?

Good morning Dan,

Please familiarize yourself with the attached file.
Reply here if you have any questions.

Thank you.

John and Mike,

Appreciate your flexibility this week, as the team navigates the sensitivities surrounding some of the project work taking place at the sites. Please tentatively plan for mobilization on 05/16/2022, in order to begin the final stages of the upgrade.

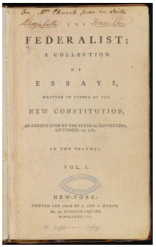
I will follow-up tomorrow with a confirmation if all indications are we will be given the “all-clear” before EOB Wednesday/SOB Thursday.

Appreciate your support.

Regards,

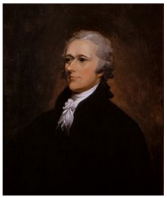
Judy Sewell
Project Manager

Who wrote which Federalist Papers?



- 1787-8: essays anonymously written by:
  - Alexander Hamilton, James Madison, and John Jay to convince New York to ratify U.S Constitution

Who wrote which Federalist Papers?



- Authorship of 12 of the letters unclear between:
  - Alexander Hamilton
  - James Madison
- 1963: solved by Mosteller and Wallace using Bayesian methods

Positive or negative movie review?

- 👎 unbelievably disappointing
- 👍 Full of zany characters and richly applied satire, and some great plot twists
- 👍 this is the greatest screwball comedy ever filmed
- 👎 It was pathetic. The worst part about it was the boxing scenes.

What is the subject of this article?

Brain and Cognition

Syntactic frame and verb bias in aphasia: Plausibility judgments of undergoer-subject sentences

What is the subject of this article?

- MeSH Subject Category Hierarchy
  - Antogonists and Inhibitors
  - Blood Supply
  - Chemistry
  - Drug Therapy
  - Embryology
  - Epidemiology

Text Classification

- Assigning subject categories, topics, or genres
- Spam detection
- Authorship identification (who wrote this?)
- Language Identification (is this Portuguese?)
- Sentiment analysis
- Stance Detection (for or against?)

Text Classification: definition

- Input:
  - a document d
  - a fixed set of classes $C = \{c_1, c_2, ..., c_j\}$
- Output: a predicted class $c \in C$

Basic Classification Method: Hand-coded rules

- Rules based on combinations of words or other features
  - spam: black-list-address OR ("dollars" AND "have been selected")
- Accuracy can be high
  - In very specific domains
  - If rules are carefully refined by experts
- But:
  - building and maintaining rules is expensive
  - they are too literal and specific: "high-precision, low-recall"

Classification Method: Supervised Machine Learning

- Input:
  - a document d
  - a fixed set of classes $C = \{c_1, c_2, ..., c_j\}$
  - A training set of m hand-labeled documents $(d_1, c_1), ..., (d_m, c_m)$
- Output:
  - a learned classifier y:d → C

Classification Methods: Supervised Machine Learning

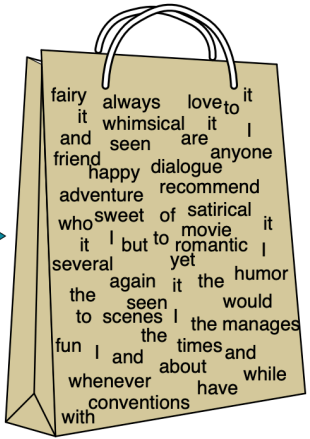
- Early text classification
  - Naive Bayes
  - Logistic regression
  - Neural networks
  - k-nearest neighbors
- These days, LLNs are used for this
  - Fine-tuned as classifiers
  - Prompted to give a classification

The Naive Bayes Classifier

- Simple ("naive") classification method based on Bayes rule
- Relies on very simple representation of document
  - Bag of words

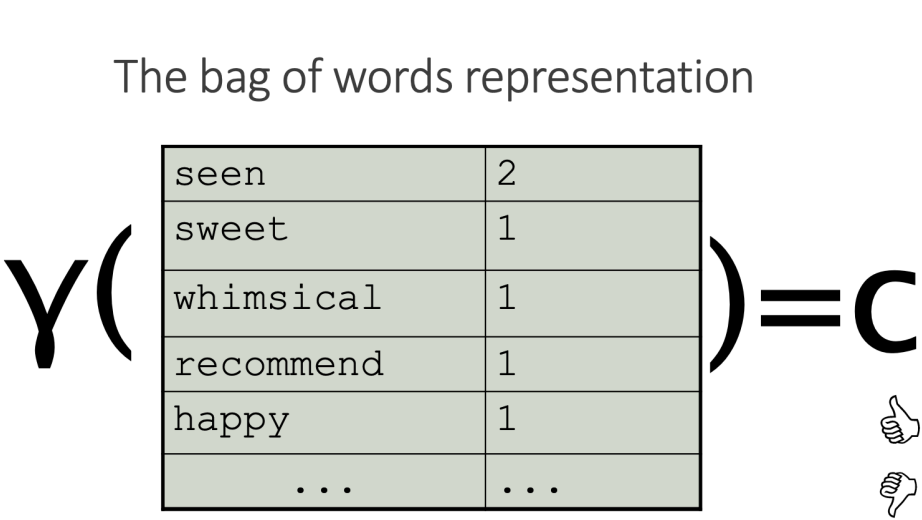
The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

The Bag of Words Representation



The bag of words representation

- Bayes' Rule Applied to Documents and Classes
- For a document d and a class c

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

Naive Bayes Classifier (I)

$$
\begin{aligned}
c_{MAP} &= \underset{c \in C}{\operatorname{argmax}} P(c|d) \\
&= \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)} \\
&= \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)
\end{aligned}
$$

- MAP is "maximum a posteriori" = most likely class
- Bayes Rule
- Dropping the denominator

Naive Bayes Classifier (II)

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$
$$= \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, ..., x_n|c)P(c)$$

- Document d represented as features x1..xn
- "Likelihood"
- "Prior"

Naïve Bayes Classifier (IV)

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, ..., x_n | c) P(c)$$

- How often does this class occur?
- $O(|X|^n \cdot |C|)$ parameters
- We can just count the relative frequencies in a corpus
- Could only be estimated if a very, very large number of training examples was available.

Multinomial Naive Bayes Independence Assumptions

- **Bag of Words assumption:** Assume position doesn't matter
- **Conditional Independence:** Assume the feature probabilities $P(x_i|c_j)$ are independent given the class c.
- $P(x_1, x_2, ..., x_n|c)$
- $P(x_1, ..., x_n|c) = P(x_1|c) \cdot P(x_2|c) \cdot P(x_3|c) \cdot ... \cdot P(x_n|c)$

Multinomial Naive Bayes Classifier

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, ..., x_n | c) P(c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c_j) \prod_{x \in X} P(x | c)$$

Applying Multinomial Naive Bayes Classifiers to Text Classification

positions ← all word positions in test document

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_{i \in \text{positions}} P(x_i | c_j)$$

Problems with multiplying lots of probs

- There's a problem with this:
- Multiplying lots of probabilities can result in floating-point underflow!
- .0006 * .0007 * .0009 * .01 * .5 * .000008….
- Idea: Use logs, because log(ab) = log(a) + log(b)
- We'll sum logs of probabilities instead of multiplying probabilities!

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_{i \in \text{positions}} P(x_i | c_j)$$

We actually do everything in log space

Instead of this:

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_{i \in \text{positions}} P(x_i | c_j)$$

This:

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} \left[\log P(c_j) + \sum_{i \in \text{positions}} \log P(x_i | c_j) \right]$$

- Notes:
  - Taking log doesn't change the ranking of classes!
  - The class with highest probability also has highest log probability!
  - It's a linear model: Just a max of a sum of weights: a linear function of the inputs
  - So Naive Bayes is a linear classifier

Learning the Multinomial Naive Bayes Model

- First attempt: maximum likelihood estimates
  - simply use the frequencies in the data
- $\hat{P}(c_j) = \frac{N_{c_j}}{N_{total}}$
- $\hat{P}(w_i|c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$

Parameter estimation

- $\hat{P}(w_i|c_j) = \frac{count(w_i,c_j)}{\sum_{w \in V} count(w,c_j)}$
- fraction of times word W_i appears among all words in documents of topic c_j
- Create mega-document for topic j by concatenating all docs in this topic
- Use frequency of w in mega-document

Problem with Maximum Likelihood

- What if we have seen no training documents with the word **fantastic** and classified in the topic **positive**?
- $\hat{P}(\text{"}fantastic\text{"}|positive) = \frac{count(\text{"}fantastic\text{"},positive)}{\sum_{w \in V} count(w,positive)} = 0$
- Zero probabilities cannot be conditioned away, no matter the other evidence!
- $c_{MAP} = \underset{c}{\text{argmax}} \hat{P}(c) \prod_i \hat{P}(x_i|c)$

Laplace (add-1) smoothing for Naïve Bayes

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

Multinomial Naïve Bayes: Learning

- From training corpus, extract Vocabulary
- Calculate $P(c_j)$ terms
  - For each c_j in C do
    - $docs_j \leftarrow$ all docs with class = c_j
    - $P(c_j) \leftarrow \frac{|docs_j|}{|total\#documents|}$
- Calculate $P(w_k|c_j)$ terms
  - $\text{Text}_j \leftarrow$ single doc containing all $docs_j$
  - For each word w_k in Vocabulary
    - $n_k \leftarrow$ # of occurrences of w_k in Text_j
    - $P(w_k|c_j) \leftarrow \frac{n_k+\alpha}{n+\alpha|Vocabulary|}$

Unknown words

- What about unknown words that appear in our test data but not in our training data or vocabulary?
- We ignore them
  - Remove them from the test document!
  - Pretend they weren't there!
  - Don't include any probability for them at all!
- Why don't we build an unknown word model?
  - It doesn't help: knowing which class has more unknown words is not generally helpful!

Stop words

- Some systems ignore stop words
- Stop words: very frequent words like **the** and **a**.
- Sort the vocabulary by word frequency in training set
- Call the top 10 or 50 words the stopword list.
- Remove all stop words from both training and test sets
- But removing stop words doesn't usually help
- So in practice most NB algorithms use all words and don't use stopword lists

A worked sentiment example!

- **4.3 Worked example** Let's walk through an example of training and testing naive Bayes with add-one smoothing. We'll use a sentiment analysis domain with the two classes positive (+) and negative (-).
- **Training Documents:**
  - - just plain boring
  - - entirely predictable and lacks energy
  - - no surprises and very few laughs
  - + very powerful
  - + the most fun film of the summer
- **Test Document:**
  - ? predictable with no fun

A worked sentiment example with add-1 smoothing

- **Prior from training:**
  - $P(-) = \frac{3}{5}$
  - $P(+) = \frac{2}{5}$
- Drop "with"
- **Likelihoods from training:**
  - $P(\text{“}predictable\text{"}|-) = \frac{1+1}{14+20}$
  - $P(\text{“}predictable\text{"}|+) = \frac{0+1}{9+20}$
  - $P(\text{“}no\text{"}|-) = \frac{1+1}{14+20}$
  - $P(\text{“}no\text{"}|+) = \frac{0+1}{9+20}$
  - $P(\text{“}fun\text{"}|-) = \frac{0+1}{14+20}$
  - $P(\text{“}fun\text{"}|+) = \frac{1+1}{9+20}$

A worked sentiment example with add-1 smoothing

- **Scoring the test set:**
  - $P(-)P(S|-) = \frac{3}{5} \times \frac{2}{34} \times \frac{2}{34} \times \frac{1}{34} = 6.1 \times 10^{-5}$
  - $P(+)P(S|+) = \frac{2}{5} \times \frac{1}{29} \times \frac{1}{29} \times \frac{2}{29} = 3.2 \times 10^{-5}$

The model predicts the class **negative** for the test sentence.

TF-IDF-based vectorization: An alternative to Count-based vectorization

- A simple word count (or Bag-of-Words) can be misleading.
- Words like ”the,” ”a,” and ”is” appear frequently in all documents.
- They don’t provide much information about a document’s topic.
- We need a method that weights words based on their importance.

TF-IDF (Term Frequency-Inverse Document Frequency) is a powerful method for this.

Term Frequency (TF)

- Measures how frequently a term appears in a document.
- The more a word appears, the more relevant it might be to that document.

- Simple TF:
  TF(t,d)=count of t in d
- Normalized TF (more common):
  TF(t,d)= total words in d count of t in d

Example: In a 10-word document, if the word "cat" appears twice:
TF("cat",doc)= $\frac{2}{10} = 0.2$

Inverse Document Frequency (IDF)

- Measures the importance of a term across the entire corpus.
- It penalizes common words (like "the") and gives higher weight to rare, specific words.

- The IDF formula:
  $IDF(t, D) = log(\frac{N}{(df(t)+1)})$
  where:
  - N is the total number of documents in the corpus.
  - DF(t) is the number of documents containing the term t.

Intuition:

- If a word is rare (DF(t) is small), IDF is large.
- If a word is common (DF(t) is large), IDF is small.
- The logarithm helps to dampen the effect of very large term counts.

Putting it Together: TF-IDF

- The TF-IDF score is the product of TF and IDF.
  $\text{TF-IDF(t,d,D)}=\text{TF(t,d)}\times\text{IDF(t,D)}$
- A high TF-IDF score for a word in a document means:
  - The word appears frequently in that document (high TF).
  - The word is rare across the entire set of documents (high IDF).
- This combination effectively highlights words that are unique and specific to a document.

TF-IDF as a Feature for Naive Bayes

- Naive Bayes relies on word counts to estimate probabilities (P(w C)).
- However, as we saw, simple counts can be misleading.
- Instead of using raw word counts, we can use the TF-IDF score for each word.
- TF-IDF values provide a much more informative numerical representation of the text.

The TF-IDF transformation:

- Transforms the text (a sequence of words) into a feature vector.
- Each dimension in the vector corresponds to a unique word in the vocabulary.
- The value in each dimension is the TF-IDF score of that word for the document.
- This feature vector is then used as input to the Naive Bayes classifier.

Notebook example

- Let's walkthrough an example using `sklearn` in `Class_2_Early_text_classification_NLP_Fall_25.ipynb`

Logistic Regression

- Important analytic tool in natural and social sciences
- Baseline supervised machine learning tool for classification
- Is also the foundation of 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 $[x_1, x_2, ..., x_n]$. 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.

The two phases of logistic regression

- **Training**: we learn weights w and b using stochastic gradient descent and cross-entropy loss.
- **Test** (also called inference): Given a test example x we compute $p(y|x)$ using learned weights w and b, and return whichever label ($y=1$ or $y=0$) has higher probability

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 $[x_1, x_2, ..., x_n]$
- We compute an output: a predicted class $\hat{y}^{(i)} \in \{0, 1\}$

Features for Logistic Regression

Given a series of input/output pairs: Features in logistic regression

- 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'": $wj = -10$
- x_k = "review contains 'mediocre'": $w_k = -2$

Logistic Regression for one observation x

- Input observation: vector $x = [x_1, x_2, \ldots, x_n]$
- Weights: one per feature: $W = [w_1, w_2, \ldots, w_n]$ (Sometimes we call the weights $[\theta_1, \theta_2, \ldots, \theta_n]$
- Output: a predicted class$\hat{y}^{(i)} \in \{0, 1\}$ (Multinomial Logistic regression: $\hat{y}^{(i)} \in \{0, 1, 2, 3, 4\}$

How to do classification

- For each feature x_i, weight w_i tells us the importance of x_i
- We'll sum the weighted features and the bias

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

$$z = wx + b$$

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

How to do classification

- But we want a probabilistic classifier
- We need to formalize "sum is high".
- We'd like a principled classifier that gives us a probability of the kind we saw with Naive Bayes
- We want a model that can tell us:

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

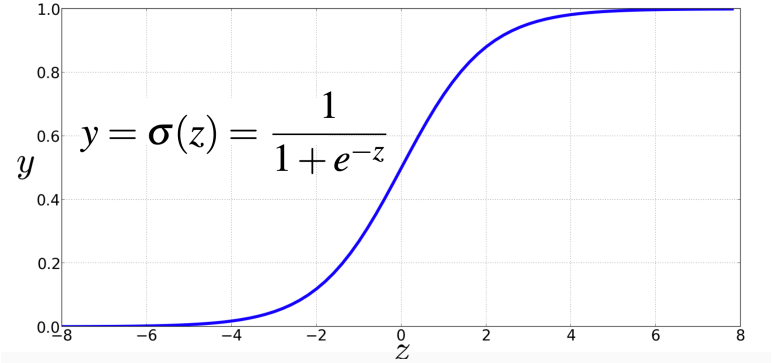
$$p(y = 0|x;\theta)$$

How to do classification

- Problem: z isn't a probability, it's just a number
- Solution: use a function of z that goes from 0 to 1

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

The sigmoid, or logistic function



The sigmoid, or logistic function

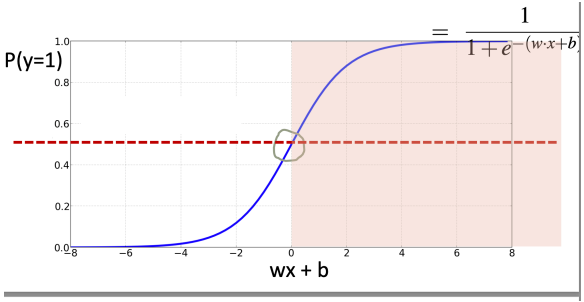
- We'll compute $wx+b$ And then we'll pass it through the sigmoid function:

$$\sigma(wx+b)$$

And we'll just treat it as a probability

Turning a probability into a classifier

- $\hat{y} = \begin{cases} 1 & \text{if } p(y=1|x) > 0.5 \\ 0 & otherwise \end{cases}$
- 0.5 here is called the **decision boundary**



Beyond BoW: Classifying sentiment with basic feature engineering

- Before the advent of LLMs, most of an NLP researcher's time was spent engaged in feature engineering.
- BoW is simply a baseline – to get better performance we need features that capture nuances in the text.

Beyond BoW: Classifying sentiment with basic feature engineering

- Negation is a classic example: A BoW representation of `There were virtually no surprises` doesn't capture the fact that `surprises` in this context signals negative sentiment. How do we turn this into a feature? Suppose we have a **lexicon** of positive and negative words, we could write a feature that captures the polarity flip that occurs when a positive, such as *surprises*, or a negative word, such as *terrible*, is negated.
- **Sentiment polarity flip feature:** For a token x_i,
$$\begin{cases} 0 & \text{if } x-1 \in \text{positive sentiment lexicon and } \textit{no}/\textit{not} \text{ precedes } x_i \\ 1 & \text{otherwise} \end{cases}$$
- We'd write something similar for the *negative* to *positive* flip in *not bad*

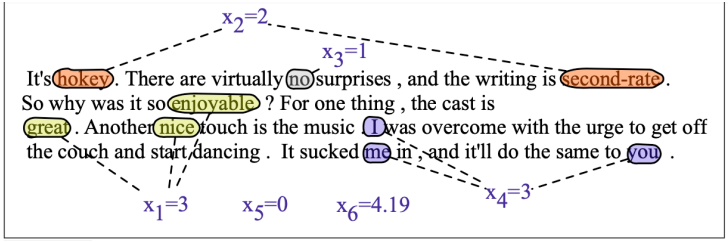
Beyond BoW: Classifying sentiment with basic feature engineering

```
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.
```

Beyond BoW: Classifying sentiment with basic feature engineering

- Other potential features:
  - Counts of words found in positive an negative lexicons
  - Counts of 1st (*I, we*) and second person pronouns (*you*) – the intuition here might be that positive reviews contain more of these
  - Review length – longer reviews tend to be more positive

Beyond BoW: Classifying sentiment with basic feature engineering



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)	$\ln(66) = 4.19$

Beyond BoW: Classifying sentiment with basic feature engineering

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

$$w = [2.5, 5.0, 1.2, 0.5, 2.0, 0.7], b = 0.1$$

- Given these 6 features and the input review x, $P(+|x)$ and $P(-|x)$ can be computed as

$$\begin{aligned} P(+|x) = P(y=1|x) &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, 5.0, 1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= \sigma(.833) \\ &= 0.70 \end{aligned}$$

Beyond BoW: Classifying sentiment with basic feature engineering

- And $P(|x)$ can be computed as
$$P(-|x) = P(y=0|x) = 1-\sigma(w\cdot x+b)$$
$$= 0.30$$

$\cdots$ so the review is classified as *positive* which aligns with our interpretation of the review

Learning: Cross-Entropy Loss

But where did that vector of weights w come from?

Supervised Classification

- We know the true label y (either 0 or 1) for each input x.
- The system produces an estimate, $\hat{y}$.
- Our goal is to set parameters w and b to minimize the distance between our estimate $\hat{y}^{(i)}$ and the true label $y^{(i)}$.

Key Components

- **Loss Function:** A distance estimator to measure the difference between $\hat{y}$ and y. We'll use cross-entropy loss.
- **Optimization Algorithm:** A method to update w and b to minimize the loss, such as stochastic gradient descent.

The Distance Between $\hat{y}$ and y

Classifier Output

- Our classifier output is $\hat{y} = \sigma(w \cdot x + b)$.
- The true output is y, which is either 0 or 1.

Loss Function

We define a loss function $L(\hat{y}, y)$ to quantify the difference between our estimated output and the true output.

$$L(\hat{y}, y) = \text{how much } \hat{y} \text{ differs from the true } y$$

Intuition of Negative Log Likelihood Loss = cross-entropy loss

Conditional Maximum Likelihood Estimation

- We choose the parameters w, b that **maximize the log probability** of the true y labels in the training data, given the observations x.
- This is equivalent to minimizing the negative log likelihood.

Deriving Cross-Entropy Loss (for a single observation x)

Goal

Maximize the probability of the correct label, $p(y|x)$.

Probability Expression

Since there are only two discrete outcomes (0 or 1), we can express the probability from our classifier as:

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

- If $y = 1$, this simplifies to $p(1|x) = \hat{y}^1(1-\hat{y})^0 = \hat{y}$.
- If $y = 0$, this simplifies to $p(0|x) = \hat{y}^0(1-\hat{y})^1 = 1-\hat{y}$.

Example: Case $y = 1$ (Positive Sentiment)

Goal

Loss should be smaller if the model estimate is close to correct.

- Suppose the true label is $y = 1$.
- Our model predicts $\hat{y} = 0.70$. This is a pretty good prediction.
- The loss calculation is:

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \\ &= -[1 \cdot \log(0.70) + (1-1) \log(1-0.70)] \\ &= -\log(0.70) \approx 0.36 \end{aligned}$$

Example: Case $y = 0$ (Negative Sentiment)

Goal

Loss should be bigger if the model is confused.

- Suppose the true label is $y = 0$.
- Our model still predicts $\hat{y} = 0.70$. This is now a very bad prediction.
- The loss calculation is:

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \\ &= -[0 \cdot \log(0.70) + (1-0) \log(1-0.70)] \\ &= -\log(0.30) \approx 1.2 \end{aligned}$$

Why does this loss function work?

- When the model is right ($y = 1, \hat{y} = 0.70$), the loss is low (≈ 0.36).
- When the model is wrong ($y = 0, \hat{y} = 0.70$), the loss is high (≈ 1.2).
- A perfect classifier would assign a probability of 1 to the correct outcome, resulting in a loss of 0 (since $-\log(1) = 0$).
- As the predicted probability of the correct answer approaches 0, the loss approaches infinity.

The Training Phase

Goal of Training

- Given a dataset of m input/output pairs, $(x^{(i)}, y^{(i)})$, we want to find the optimal set of weights w and bias b.
- Optimal means minimizing our loss function, averaged over all examples.

Loss Function

The total loss is the average of the cross-entropy loss over all m examples:

$$L(w, b) = \frac{1}{m} \sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

Notebook example

- Let's walkthrough an example using `sklearn` in `Class_2_Early_text_classification_NLP_Fall_25.ipynb`

Gradient Descent: The Intuition

Navigating the Loss Surface

- Imagine the loss function as a surface, where the height represents the loss. Our goal is to find the lowest point on this surface.
- The **gradient** of a function is a vector that points in the direction of the steepest increase.
- **Gradient Descent** works by taking small steps in the **opposite direction** of the gradient. This is like "walking downhill."

Gradient Descent: The Intuition

- For logistic regression, loss function is convex
- A convex function has just one minimum
- Gradient descent starting from any point is guaranteed to find the minimum
- (Loss for neural networks is non-convex)

The Gradient: How to Move

Finding the Slope

To find the direction of steepest descent, we take the partial derivatives of the loss function with respect to each parameter (w_j and b).

Partial Derivatives for One Example

- For a single example (x, y), the gradients of the cross-entropy loss are:

$$\frac{\partial L_{CE}}{\partial w_j} = (\hat{y} - y)x_j$$
$$\frac{\partial L_{CE}}{\partial b} = (\hat{y} - y)$$

- These derivatives tell us how much a change in w_j or b affects the loss.

The Update Rule

Updating Parameters

We update our weights and bias by moving them a small amount in the direction that decreases the loss. This is governed by the learning rate, α.

- The update for each weight w_j is:

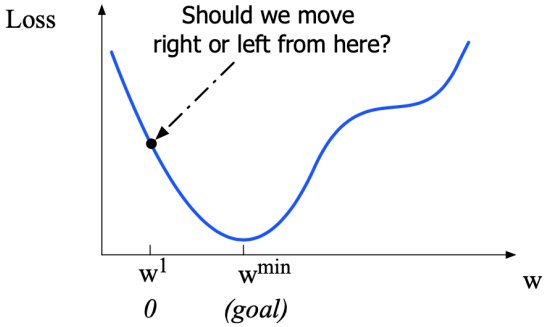
$$w_j \leftarrow w_j - \alpha \frac{\partial L}{\partial w_j}$$

- The update for the bias b is:

$$b \leftarrow b - \alpha \frac{\partial L}{\partial b}$$

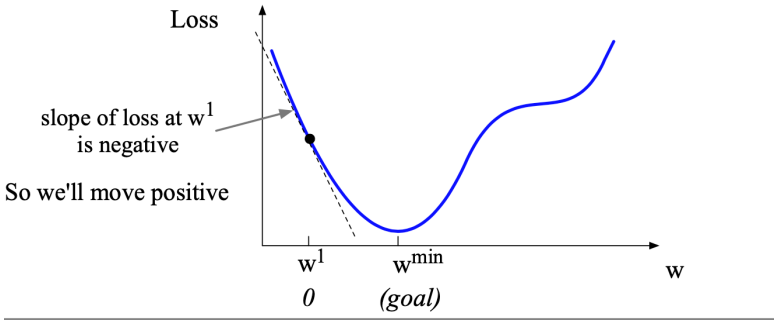
Gradient Descent: The Intuition

Q: Given current w, should we make it bigger or smaller? A: Move w in the reverse direction from the slope of the function



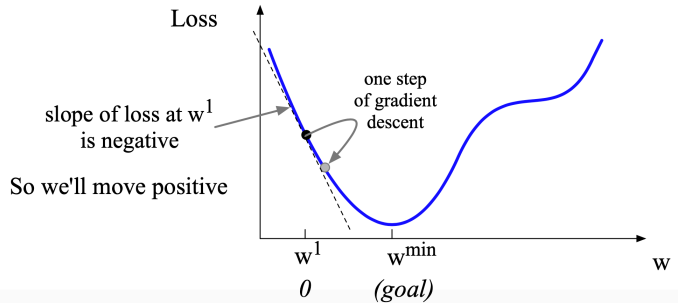
Gradient Descent: The Intuition

Q: Given current w, should we make it bigger or smaller? A: Move w in the reverse direction from the slope of the function



Gradient Descent: The Intuition

Q: Given current w, should we make it bigger or smaller? A: Move w in the reverse direction from the slope of the function



Types of Gradient Descent

Batch Gradient Descent

- Calculates the gradient over the **entire dataset** before updating the parameters.
- Provides a very accurate direction, but is computationally expensive and slow for large datasets.

Stochastic Gradient Descent (SGD)

- Calculates the gradient and updates the parameters for a **single, randomly selected example**.
- Much faster, but the updates can be ”choppy” and noisy.

Types of Gradient Descent

Mini-batch Gradient Descent

- A compromise between the two. Calculates the gradient and updates the parameters for a small **mini-batch** of examples (e.g., 64, 256, 512).
- This is the most common approach in practice.

The SGD Algorithm

Putting it All Together

1. **Initialize** weights w and bias b to small random values (or zero).
2. **Loop** for a number of epochs (passes over the training data).
3. In each epoch, **shuffle** the training data.
4. For each mini-batch of size m in the training data:
   - Compute $\hat{y}$ for each example in the batch: $\hat{y} = \sigma(w \cdot x + b)$.
   - Compute the average gradient for the batch.
   - Update weights and bias using the learning rate α.
5. **Repeat** until the loss is minimized or the maximum number of epochs is reached.

The Learning Rate, α

A Hyperparameter

- The learning rate is a special parameter that controls how big of a step we take in the direction of the gradient.
- It is a **hyperparameter** that is set by the user, not learned by the algorithm.

Impact of α

- If α is too high, the updates will be too large, and the algorithm may "overshoot" the minimum and fail to converge.
- If α is too low, the algorithm will take tiny steps, making training very slow.

Evaluating Classifiers

The Task

- We need to determine how well our classifier performs on new data.
- Let's first address binary classifiers, such as "Is this email spam?".

What we need to know

- What did our classifier say about each example?
- What was the correct answer (the "gold label")?

First Step: The Confusion Matrix

A 2x2 Table

The confusion matrix is a table that summarizes the performance of a classifier.

System Output Labels	Gold Standard Labels	
	Positive	Negative
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

- **True Positive (TP)**: Correctly classified positive.
- **False Positive (FP)**: Incorrectly classified as positive.
- **False Negative (FN)**: Incorrectly classified as negative.
- **True Negative (TN)**: Correctly classified negative.

Accuracy on the Confusion Matrix

Definition

Accuracy is the percentage of correct predictions out of all predictions made.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

Why Don't We Use Accuracy?

The Problem with Imbalanced Classes

Accuracy doesn't work well when one class is much more common than another.

- Consider a dataset of 1,000,000 social media posts.
- Only 100 posts are about "Delicious Pie".
- 999,900 posts are about other things.
- A simple classifier that says "not about pie" for every post would have an accuracy of $\frac{999{,}900}{1{,}000{,}000} = 99.99\%$, which is misleadingly high.

Precision

Definition

Precision is a measure of correctness on positive predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- "Of all the emails the classifier labeled as spam, what percentage were actually spam?"
- For our "not about pie" classifier, the precision is $\frac{0}{0+0}$ = undefined or 0, which correctly reflects its uselessness for finding pie posts.

Recall

Definition

Recall is a measure of completeness on positive predictions.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- "Of all the emails that were actually spam, what percentage did the classifier find?"
- For our "not about pie" classifier, the recall is $\frac{0}{0+100} = 0$, also reflecting its poor performance.

The Precision-Recall Trade-off

How it Works

- Increasing precision often decreases recall, and vice versa.
- A spam classifier that is very cautious (high precision) may miss some spam emails (low recall).
- A spam classifier that flags many emails as spam (high recall) may also misclassify some legitimate emails (low precision).

F-Measure: Combining Precision and Recall

The Harmonic Mean

The F-measure is the harmonic mean of precision and recall. It is a more robust metric for imbalanced classes.

The general formula is:

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

or, with $\beta^2 = \frac{1-\alpha}{\alpha}$:

$$F = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

- **F1 Score**: A special case where $\beta = 1$ and $\alpha = \frac{1}{2}$. It gives equal weight to precision and recall.

$$F_1 = \frac{2PR}{P+R}$$

Evaluation with More Than Two Classes

Example: 3-way Email Task

We can extend precision and recall to multi-class problems by calculating them for each class individually.

	Gold Labels		
System Output	**Urgent**	**Normal**	**Spam**
Urgent	8	5	3
Normal	10	60	50
Spam	1	30	200

- **Recall for Urgent**: $\frac{8}{8+5+3} = \frac{8}{16}$
- **Precision for Urgent**: $\frac{8}{8+10+1} = \frac{8}{19}$

Next Class 3 Sept 9

Topics

- Early Language Modeling
- N-gram-models
- Neural architectures • Embeddings
- Applications of embeddings

Reading

- Jurafsky & Martin Chapter 3: Ngram Lanuage Models
- Jurafsky & Martin Chapter 6: Vector Semantics Embeddings
- Efficient Estimation of Word Representations in Vector Space