Natural Language Processing

Adam Faulkner

Jan 20, 2026

About Me

- Currently Senior Manager, Data Science at Capital One
- Began teaching a course on Generative AI at Katz in Fall 2024
- Previously at IBM Research, Grammarly, and ETS
- Ph.D. CUNY Graduate Center, 2014

Core NLP Tasks (not exhaustive)

- The "Natural" in NLP:
  - Natural languages, such as English and Mandarin, are used in everyday discourse and have evolved organically. They have fuzzily defined rules and are ambiguity-laden.
  - Formal languages such as first-order-logic and Python have well-defined rules and are unambiguous
- NLP is an engineering discipline: The goal is to get computers to perform useful tasks involving human language—enabling human-machine communication, improving human-human communication, or simply doing useful processing of text.

Why NLP is hard

- To see why this is challenging, let's consider the following user request and the system response from a functioning conversational agent (from the film 2001) *HAL*:

  `User:Open the pod bay doors please HAL.`

  `HAL: I'm sorry Dave, I'm afraid I can't do that`

- What did HAL need to understand about natural language in order to properly respond?

Natural Language Understanding: Syntax

- HAL must use structural knowledge to properly string together the words that constitute its response. For example, HAL must know that the following sequence of words will not make sense to Dave, despite the fact that it contains precisely the same set of words as the original.

  `I'm I do,sorry that afraid Dave I'm can't.`

- The knowledge needed to order and group words together comes under the heading of **syntax.**

Natural Language Understanding: Lexical Semantics

- HAL also needs to understand the meaning of words such as *doors*. Specifically, he must understand that *doors* are things that open and close – this is the word's *lexical frame*.
- Knowledge of word meaning involves knowledge of **lexical semantics**

Natural Language Understanding: Commonsense Reasoning

- HAL also has to somehow understand that a request to open a set of doors implies that the doors are currently closed – nothing in the request itself indicated this.
- World-knowledge of this sort falls under the heading of **commonsense reasoning**.

Natural Language Understanding: Discourse

- The user used “please” when making their request. HAL should reciprocate by also using a polite *register*: "I’m sorry Dave," etc."
- These are the implicit rules of **discourse** governing the interaction

Natural Language Understanding: Pragmatics

- HAL needs to understand that the imperative *Open the pod bay doors please HAL* is a request for action – this knowledge falls under the heading of **pragmatics**
- Additionally, as we learn later in the film, HAL is capable of opening the door but *won't* and could have simply replied *No* or *No, I won't open the door.* Instead, it first embellishes its response with the phrases *I'm sorry* and *I'm afraid*, and then only indirectly signals its refusal by saying *I can't*, rather than the more direct (and truthful) *I won't.*
- Again, this knowledge of the kinds of actions speakers intend by phrasing their sentences one way rather than another is an example of **pragmatic** knowledge.

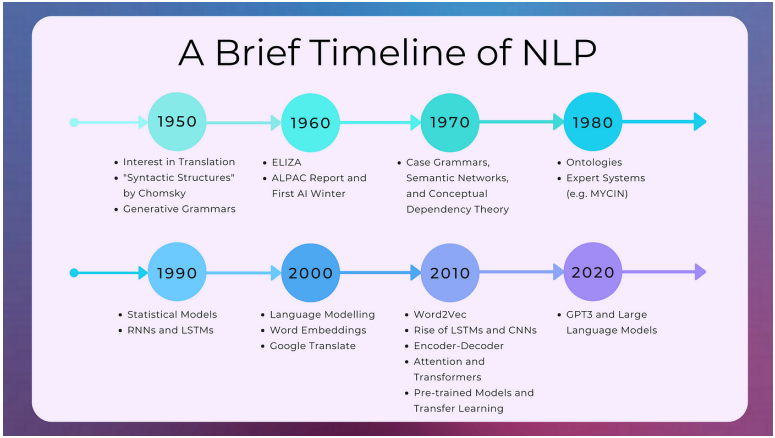
Ambiguity: A core problem in NLP

- A surprising fact about these categories of linguistic knowledge is that most tasks in NLP can be viewed as resolving ambiguity at one of these levels.
- Consider the sentence *I made her duck.* Here are a few possible interpretatons:
  1. I cooked waterfowl for her. (*duck* is ambiguous)
  2. I cooked waterfowl belonging to her. (*her* is ambiguous)
  3. I created the (plaster?) duck she owns. (*make* is ambiguous)
  4. I caused her to quickly lower her head or body. (*make* is ambiguous)

Core NLP Tasks (not exhaustive)

- Commonsense Reasoning
- Constituency parsing
- Coreference resolution
- Dialogue
- Grammatical error correction
- Information extraction
- Intent Detection and Slot Filling
- Language modeling
- Machine translation
- Named entity recognition
- Part-of-speech tagging
- Paraphrase Generation
- Question answering
- Relationship extraction
- Semantic textual similarity
- Semantic parsing
- Semantic role labeling
- Sentiment analysis
- Text Simplification
- Summarization
- Taxonomy learning
- Text classification

History of NLP



History of NLP: 1940's - 1950's

- Post-WWII, funded by the US military, focused on developing Machine Translation systems for the automated translation of Soviet scientific articles
- **McCulloch-Pitts neuron** : A simplified model of the neuron as a kind of computing element that could be described in terms of propositional logic
- Shannon applied probabilistic models of discrete Markov processes to **automata for language**.
- Drawing on the idea of a finite-state Markov process from Shannons work, Chomsky first considered finite-state machines as a way to characterize a grammar, and defined a finite-state language as a language generated by a finite-state grammar. These early models led to the field of **formal language theory**

History of NLP: 1957 - 1970

- Symbolic, rules-based AI
- **Chomsky's generative syntax** and other parsing algorithms – top-down and bottom-up and then parsing via dynamic programming
- First use of the stochastic paradigm: Bledsoe and Browning (1959) built a **Bayesian system for text-recognition** that used a large dictionary and computed the likelihood of each observed letter sequence given each word in the dictionary by multiplying the likelihoods for each letter.
- Mosteller and Wallace (1964) applied Bayesian methods to the problem of **authorship attribution** of The Federalist papers.
- 1963-64: **Brown corpus of American English,** a 1 million word collection of samples from 500 written texts from different genres (newspaper, novels, non-fiction, academic, etc.)

History of NLP: 1970 - 1983

- Three paradigms: *stochastic, logic-based, natural language understanding-based*
- **Stochastic**: Hidden Markov Model and the metaphors of the noisy channel and decoding, developed by Jelinek, Bahl, Mercer, and colleagues at IBM
- **Logic-based**: Logical programming languages such as Prolog
- **Natural Language Understanding**: Terry Winograds SHRDLU system, which simulated a robot embedded in a world of toy blocks. The program was able to accept natural language text commands ("Move the red block on top of the smaller green one") of a hitherto unseen complexity and sophistication.

History of NLP: 1983 - 1999

- Rise of probabilistic models throughout speech and language processing, influenced strongly by the **work at the IBM on probabilistic models of speech recognition and machine translation**
- Algorithms for parsing, part-of-speech tagging, reference resolution, and discourse processing all began to incorporate probabilities, and employ evaluation methodologies borrowed from speech recognition and information retrieval
- **Increases in the speed and memory of computers** had allowed commercial exploitation of a number of subareas of speech and language processing, in particular speech recognition and spelling and grammar checking.

History of NLP: 1999 - 2013

- Dominance of ML-based approaches: **Maximum Entropy Estimation, Naive Bayes, Logistic Regression, SVMs**
- Datasets: **Penn Treebank, PropBank, Penn Discourse Treebank.** These datasets layered standard text sources with various forms of syntactic, semantic and pragmatic annotations. The existence of these resources promoted the trend of casting more complex traditional problems, such as parsing and semantic analysis, as problems in supervised machine learning.
- Widespread availability of high-performance computing systems facilitated the training and deployment of NLP models
- **Rise of unsupervised approaches:** Statistical approaches to machine translation and topic modeling demonstrated that effective applications could be constructed from systems trained on unannotated data alone. This directly led to the shift toward language model-based NLP.

History of NLP:2013 - present

- **Deep Learning and GPUs**: RNNs and LSTMs show massive performance gains relative to traditional ML approaches leading to the Deep Learning revolution (all made possible by the commercial availability of GPU chips)
- **word2vec**, **GloVE**
- **Transformers**, small LMs, **BERT**, the finetuning paradigm
- **GPT2,** NLP tasks recast as word prediction tasks
- **GPT3**, first LLMs, post-training regimes: RLHF, DPO
- **ChatGPT, RAG, Agents**

Course structure: Part 1

- Review of classic approaches to NLP tasks using supervised ML approaches such as Logistic Regression–this roughly covers the techniques in the slide *History of NLP: 1999 - 2013*. (These techniques aren't simply historical curiosities: Logistic Regression, softmax, and a general understanding of neural networks are all relevant to an understanding of the Deep Learning architecture that currently dominates AI, the Transformer)
- N-gram language modeling, sparse vector-based representations of text such as bag-of-words
- Deep Learning-based modeling techniques and vector representations such as LSTMs and dense-vector representations.
- The Transformer and language-model-based solutions to NLP tasks via frameworks such as finetuning.

Course structure: Part 2

- Review of traditional NLP tasks such as classification (Sentiment Analysis, Intent Detection, etc.), sequence labeling (Named Entity Recognition, syntactic and semantic parsing, etc.), and text generation (Machine Translation, Question-Answering, Summarization, etc.).
- This section ends with a review of text generation tasks. This is intentional since, in contemporary NLP, all tasks, including classification and sequence labeling, are now framed as text generation tasks

Course structure: Part 3

- Large Language Models (LLMs), the dominant paradigm of contemporary NLP
- Reframing of NLP tasks described in section 2, which were traditionally solved using the ML techniques described in section 1, as word-prediction tasks.
- Autoregressive language modelling objective, post-training regimes such as Reinforcement Learning from Human Feedback
- Retrieval Augmented Generation (RAG), Agents, LLM-as-a-Judge, LLM interpretability, and hallucination detection
- Additional topics: Automated detection of LLM-generated text, What do LLMs really understand?

Course content: Github repo

- In addition to Canvas, all course content can be accessed in the course Github repo.

Course content: Slides and readings

- The bulk of the course content consists of the lecture slides and the week's reading material (textbook chapters and representative technical papers)

Course content: Juptyer notebooks

- Each lecture has an accompanying notebook that will be run live (with instructor commentary) during the class. These notebooks can also be run before or after class to get a better understanding of the code and the lecture material

Assignments

- Students will complete 5 assignments. These are a combination of math and programming. Math assignments must be printed, completed by hand, and submitted online (via scan or photo) or handed in to me directly. The programming assignments can be completed as Jupyter notebooks and uploaded to Canvas.
- Note that most of the math assignments contain calculations that we'll walk through together during the class – if you take good notes/screenshots during these sessions you should be able to complete these assignments quickly
- Group assignment. The final group project can be completed singly or as a group (max 5 members). The project should consist of an NLP application that implements one or more of the concepts described in class. The application will be demoed on the last day of class as a 5-10 minute presentation

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.

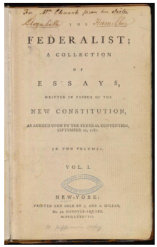
I will follow-up tomorrow with a confirmation if all indications are we will be given the “all-clear” before EOB Wednesday/SOB Thursday.

Appreciate your support.

Regards,

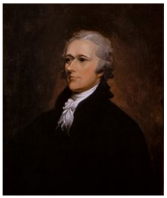
Judy Sewell
Project Manager

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?

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, LLMs are used for this
  - Fine-tuned as classifiers
  - Prompted to give a classification

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
- We want a model that can tell us:

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

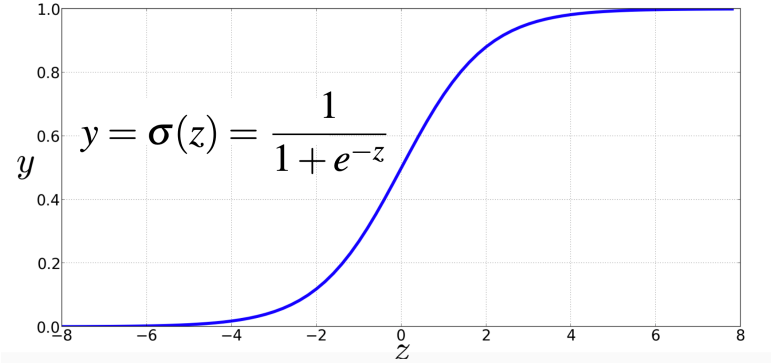
$$p(y = 0|x;\theta)$$

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

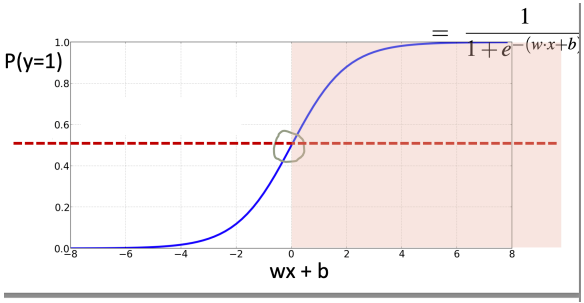
- Well compute $wx+b$ And then well pass it through the sigmoid function:

$$\sigma(wx+b)$$

And we'll just treat it as a probability

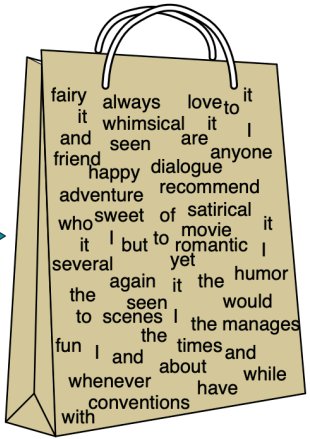
Turning a probability into a classifier

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



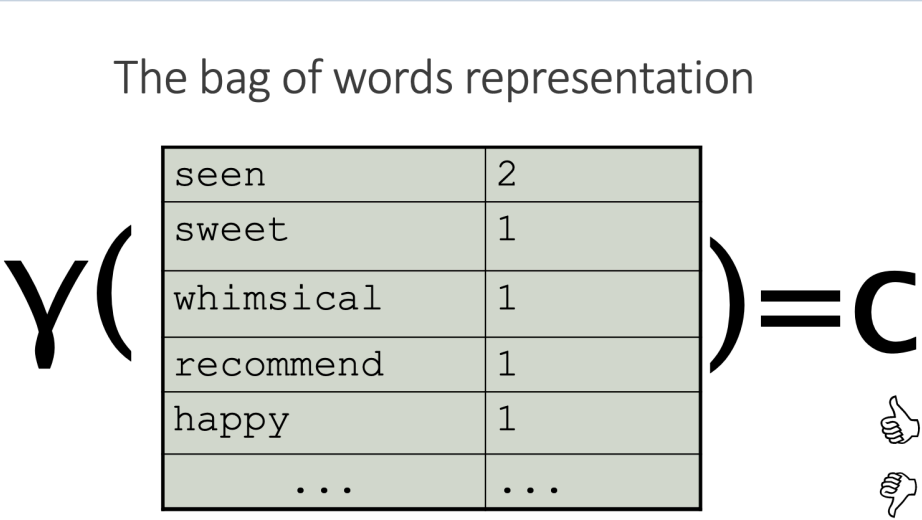
The Bag of Words Representation

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



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,

$$\begin{cases} 0 & \text{if } x-1 \in \text{positive sentiment lexicon and } no|not \text{ precedes } x_i \\ 1 & \text{otherwise} \end{cases}$$

- We'd write something similar for the *negative* to *positive* flip in *not bad*

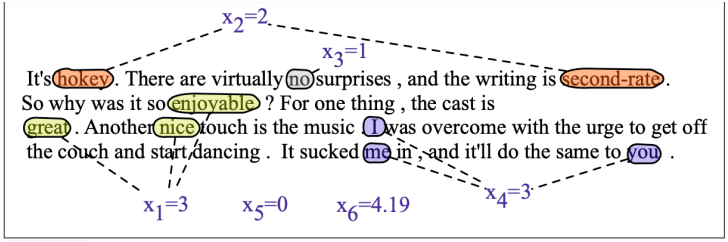
Beyond BoW: Classifying sentiment with basic feature engineering

```
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 weve 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:

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

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:

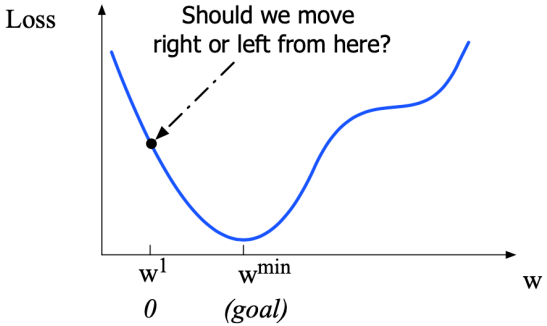
$$w_j \leftarrow w_j - \alpha \frac{\partial L}{\partial w_j}$$

- The update for the bias b is:

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

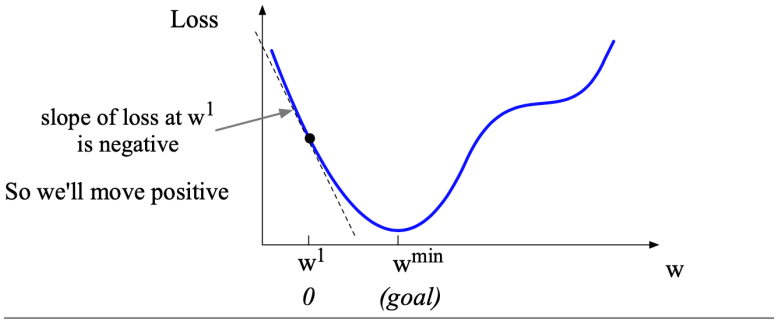
Gradient Descent: The Intuition

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



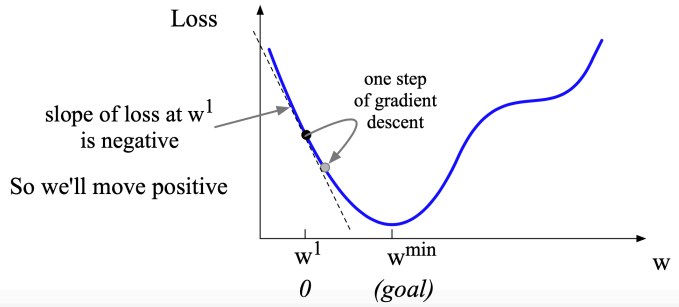
Gradient Descent: The Intuition

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.

Next class: Jan 27

Early Language Modeling

- N-gram-models
- Neural architectures
- Embeddings
- Applications of embeddings

Libraries:

- *sklearn, gensim*

Reading

- Jurafsky & Martin Chapter 3: Ngram Language Models
- Jurafsky & Martin Chapter 5: Embeddings
- Efficient Estimation of Word Representations in Vector Space

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**: $\frac{8}{8+10+1} = \frac{8}{19}$