NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Text Classification

Text Classification — Motivation

- Very common machine learning task: **classification**
  - Focus in the context of NLP: classification of text documents
  - Task: given a text document, assign a document to a class
    (in general, the set of classes are finite and predefined)

- Examples

Task	Classes (examples)
language detection	{english, malay, chinese, tamil, german, …}
spam detection	{spam, not spam}
subject/genre classification	{biology, chemistry, geology, psychology, ...}
authorship attribution	{stephen king, dan brown, jk rowling, …}
sentiment analysis	{positive, negative, neutral, mixed}
…	…

Text Classification — Language Detection

- Identification of the language
  - Relatively straightforward in case of unique alphabets/characters
  - More tricky in case of (closely) related languages

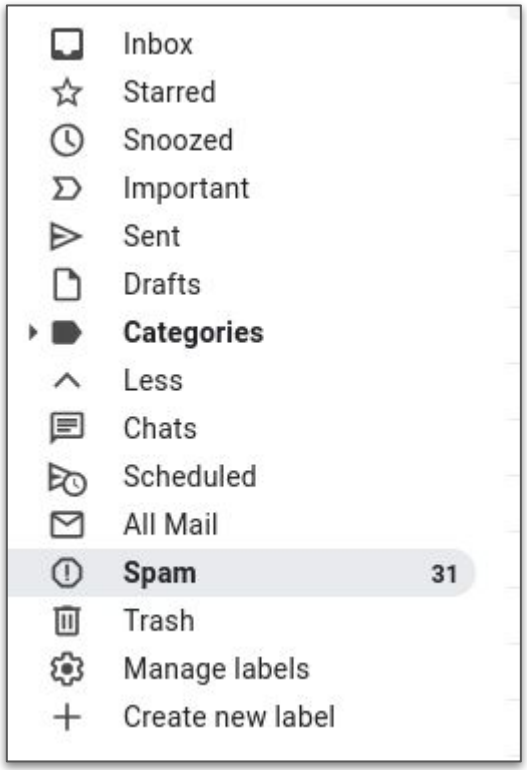
English and German are related.

When text shorted, it is not trivial as you think.

Example: Google Translate

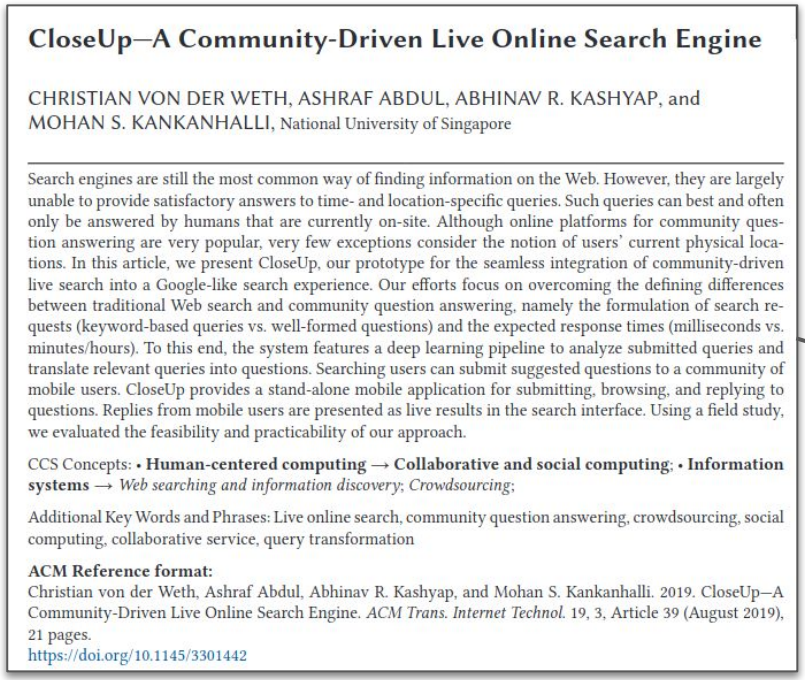
Text Classification — Email Spam Detection

- Email, messenger, SMS spam
  - Mostly annoying (e.g., ads)
  - Security risks (e.g., phishing, malicious attachments)



Text Classification — Subject Classification

- Typical application:
  - Automated organization of huge volumes of documents



CloseUp—A Community-Driven Live Online Search Engine

CHRISTIAN VON DER WETH, ASHRAF ABDUL, ABHINAV R. KASHYAP, and MOHAN S. KANKANHALLI, National University of Singapore

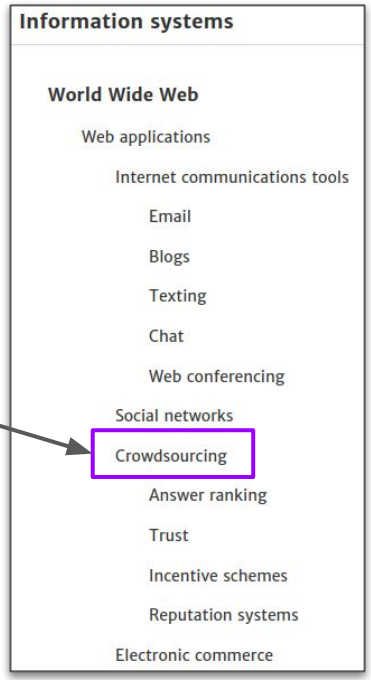
Search engines are still the most common way of finding information on the Web. However, they are largely unable to provide satisfactory answers to time- and location-specific queries. Such queries can best and often only be answered by humans that are currently on-site. Although online platforms for community question answering are very popular, very few exceptions consider the notion of users' current physical locations. In this article, we present CloseUp, our prototype for the seamless integration of community-driven live search into a Google-like search experience. Our efforts focus on overcoming the defining differences between traditional Web search and community question answering, namely the formulation of search requests (keyword-based queries vs. well-formed questions) and the expected response times (milliseconds vs. minutes/hours). To this end, the system features a deep learning pipeline to analyze submitted queries and translate relevant queries into questions. Searching users can submit suggested questions to a community of mobile users. CloseUp provides a stand-alone mobile application for submitting, browsing, and replying to questions. Replies from mobile users are presented as live results in the search interface. Using a field study, we evaluated the feasibility and practicability of our approach.

CCS Concepts: • **Human-centered computing** → **Collaborative and social computing**; • **Information systems** → *Web searching and information discovery*; *Crowdsourcing*;

Additional Key Words and Phrases: Live online search, community question answering, crowdsourcing, social computing, collaborative service, query transformation

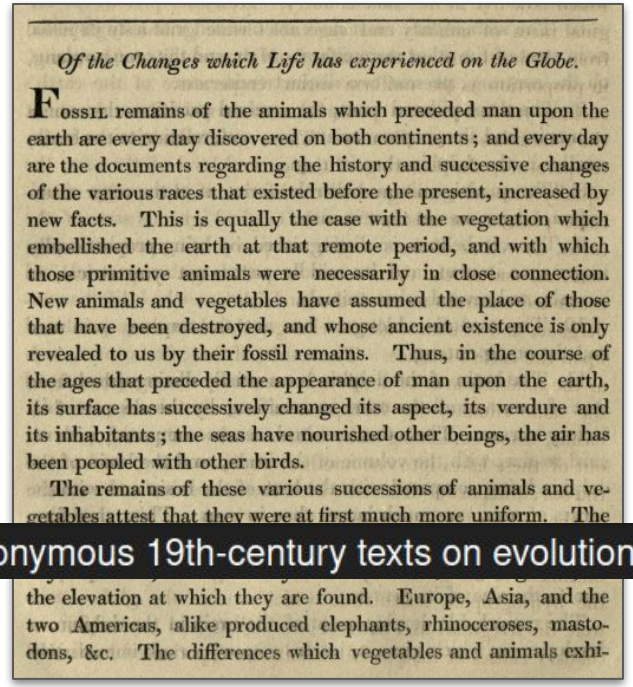
ACM Reference format:
Christian von der Weth, Ashraf Abdul, Abhinav R. Kashyap, and Mohan S. Kankanhalli. 2019. CloseUp—A Community-Driven Live Online Search Engine. *ACM Trans. Internet Technol.* 19, 3, Article 39 (August 2019), 21 pages.
https://doi.org/10.1145/3301442

ACM Computing Classification System (very small snippet)



Text Classification — Authorship Attribution

- NLP/AI meets Linguistic Forensics
  - Anonymously written documents
  - Documents written under a pseudonym
- Observation — underlying assumption:
  - People have unique writing styles
  - Vocabulary, frequent phrases, sentence lengths, typos, etc.



Of the Changes which Life has experienced on the Globe.

Fossil remains of the animals which preceded man upon the earth are every day discovered on both continents; and every day are the documents regarding the history and successive changes of the various races that existed before the present, increased by new facts. This is equally the case with the vegetation which embellished the earth at that remote period, and with which those primitive animals were necessarily in close connection. New animals and vegetables have assumed the place of those that have been destroyed, and whose ancient existence is only revealed to us by their fossil remains. Thus, in the course of the ages that preceded the appearance of man upon the earth, its surface has successively changed its aspect, its verdure and its inhabitants; the seas have nourished other beings, the air has been peopled with other birds.

The remains of these various successions of animals and vegetables attest that they were at first much more uniform. The

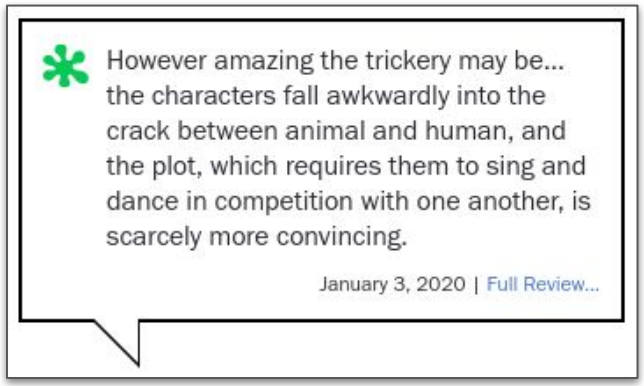
the elevation at which they are found. Europe, Asia, and the two Americas, alike produced elephants, rhinoceroses, mastodons, &c. The differences which vegetables and animals exhi-

AI reveals authors of anonymous 19th-century texts on evolution

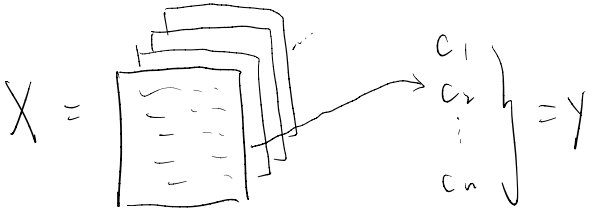
Source: https://www.intelligentcio.com/eu/2018/08/28/ai-reveals-authors-of-anonymous-19th-century-texts-on-evolution/

Text Classification — Sentiment Analysis

- Sentiment Analysis:
  - An author's subjective or emotional attitude towards the central topic of the text
  - Very commonly applied to assess online users opinions about product and services
    (e.g., product reviews, hotel/restaurant reviews, movie/song/book reviews)
  - Also: consumer feedback, brand monitoring, political views, trend analysis, etc.



Text Classification



- Formal setup
  - X — set of all documents; $x \in X$ — a single document
  - Y — set of all classes (or class labels); $y \in Y$ — a single class (or class label)

- Classification task
  - Mapping h from input space X to output space Y $\quad h : X \to Y$

$$h(x) = y$$

"True" mapping which is unknown in practice

e.g., $h(\text{"The movie is great."}) = \text{"positive"}$

sentiment analysis example.

Note: A document might be assigned to more than one class ➜ **multilabel classification**

Text Classification

- Goal of a classification task
  - Find the best $\hat{h}(x)$ to approximate the true mapping $h(x)$ ➜ **But how?**

    ↳ estimate.

- Two main approaches

  (1) **Rule-based** (decision rules)

  ← sentiment analysis

$$IF \ "good" \in x \ OR \ "great" \in x \ OR \ "nice" \in x \ OR \ ...$$
$$h(x) = "positive"$$
$$ELSE \ IF \ "bad" \in x \ OR \ "boring" \in x \ OR \ "dumb" \in x \ OR \ ...$$
$$h(x) = "negative"$$

  (2) Supervised Learning (machine learning classifiers)

  - Automatically learn $\hat{h}(x)$ based on a dataset of $\langle x, y \rangle$ pairs

  ① Naïve Bayes

  ② Logistic Regression.

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Naive Bayes Classifier I (Intuition & Definition)

Naive Bayes Classifier — Intuition

- Simple ("naive") probabilistic classifier based on Bayes Rule
  - Given a document x, for each class y_i compute $P(y_i|x)$
  - Assign document to class y with the highest probability $P(y_i|x)$ ➜ $y_{NB} = \underset{y_i \in Y}{\operatorname{argmax}} P(y_i|x)$
  - Calculate $P(y_i|x)$ using Bayes Rule ➜ $P(y_i|x) = \frac{P(x|y_i)P(y_i)}{P(x)}$

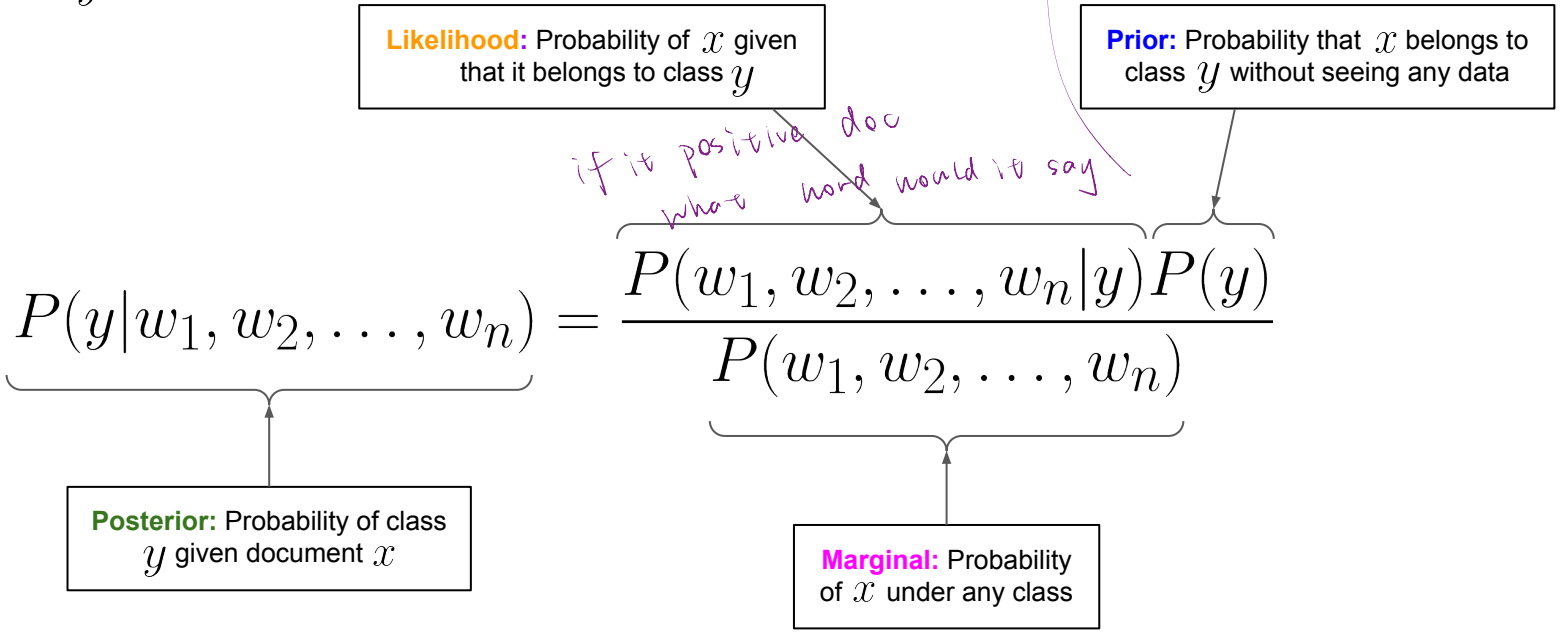
- Example (sentiment analysis)

hopefully :)

$$P(\mathbf{pos} \mid \text{"}The\ movie\ is\ really\ funny\text{"}) > P(\mathbf{neg} \mid \text{"}The\ movie\ is\ really\ funny\text{"})$$

Naive Bayes Classifier — Annotated

- Basic setup
  - Document $x \in X$ with $x = w_1, w_2, \ldots, w_n$
  - Class label $y \in Y$



Naive Bayes Classifier

- Observation
  - We are not really interested in the exact values of $P(y_i|x)$
  - We only care about the difference between $P(y_i|x)$ and $P(y_j|x)$

$$\frac{P(w_1, w_2, \ldots, w_n|y_i)P(y_i)}{P(w_1, w_2, \ldots, w_n)} \overset{?}{\gtrless} \frac{P(w_1, w_2, \ldots, w_n|y_j)P(y_j)}{P(w_1, w_2, \ldots, w_n)}$$

The **marginal** does not affect the result of the comparison!

$$P(y|w_1, w_2, \ldots, w_n) \propto P(w_1, w_2, \ldots, w_n|y)P(y)$$

Naive Bayes Classifier — The "Naive" Part

- Simplifying assumption
  - All words $w_1, w_2, \ldots, w_n$ are independent from each other
  - Obviously does not hold, but still good results in practice

funny happy

simle.

independent each other

$$P(y|w_1, w_2, \ldots, w_n) \propto P(w_1, w_2, \ldots, w_n|y)P(y)$$

"Naive" assumption

$$P(y|w_1, w_2, \ldots, w_n) \propto P(w_1|y)P(w_2|y)\ldots P(w_n|y)P(y) = P(y)\prod_{i=1}^{n} P(w_i|y)$$

unigram likelihood 의 확률.

How to calculate these probabilities?

Naive Bayes Classifier — Maximum Likelihood Estimates

- **Prior** $P(y)$

$$\hat{P}(y) = \frac{N_y}{N}$$

N_y: #documents of class y

N: #documents (total)

- **Likelihood** $P(w_i|y)$

$$\hat{P}(w_i|y) = \frac{Count(w_i, y)}{\sum_{w \in V} Count(w, y)}$$

$Count(w_i, y)$: #occurrences of w_i in documents of class y

$\sum_{w \in V} Count(w, y)$: #words (total) in documents of class y

Does this look familiar? ← LM

Naive Bayes Classifier — Practical Considerations

- Risk of arithmetic underflow ➜ Calculate log probabilities

$$P(y|w_1, w_2, \ldots, w_n) \propto P(y) \prod_{i=1}^{n} P(w_i|y) \quad \rightarrow \quad \log P(y|w_1, w_2, \ldots, w_n) \propto \log P(y) + \sum_{i=1}^{n} \log P(w_i|y)$$

加法比乘法快。

- Out-of-vocabulary (OOV) words + unrepresented classes
  - Unseen words w_i during test/prediction time ➜ $Count(w_i, y) = 0$ ➜ $P(w_i|y) = 0$
  - No documents of class y ➜ $P(y) = 0$

e.g.: Add-k Smoothing:

$$\hat{P}(w_i|y) = \frac{Count(w_i, y) + k}{\sum_{w \in V} Count(w, y) + k|V|}$$

$$\hat{P}(y) = \frac{N_y + k}{N + k|Y|}$$

k example

Laplacian

In the context of text classification, we generally only use the term Naive Bayes. However, the Naive Bayes classifier represents a family of classification algorithms. The implementation we introduce in this course is the so-called Multinomial Naive Bayes (MNB) classifier. The MNB classifier is typically used for text classification tasks where the features (i.e., words) are discrete and countable. It assumes that the probability distribution of the features is a multinomial distribution.

In statistics and machine learning, nominal features (here: words) are categorical features that do not have a natural ordering or ranking. Nominal features represent discrete values or categories that are not inherently comparable or measurable in any meaningful way. Note that a natural ordering does not refer to a lexicographical ordering. While we can say, for example, that "bread" should appear before "milk" in a dictionary, a statement such as "bread < milk" is in general semantically not meaningful.

In fact, the MNB classifier (or just the Naive Bayes classifier in our context) is one of the very few text classification algorithms that directly work on texts. Many to most other classification algorithms assume numerical (e.g., non-nominal) inputs. These algorithms therefore require the transformation or encoding of word and/or text documents into a numerical representation. We will cover one of the most fundamental transformation methods later in this section: the Vector Space Model.

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Naive Bayes Classifier II (Worked Example)

Naive Bayes Classifier — Worked Example

- Sentiment Analysis
  - Documents: movie reviews
  - Two classes: "pos", "neg"

$$V = \{funny,\ boring,\ movie,\ cast,\ good\}$$

$$|V| = 5$$

Example corpus

(greyed-out words/tokens removed during normalization)

	Review	Class
1	very good and funny movie!	pos
2	what a funny cast!	pos
3	a very boring movie and boring cast	neg
4	very boring cast!	neg
5	such a funny movie!	pos
6	really good cast, really good movie.	pos
7	"boring…such a boring movie!!!	neg

Smooth ✱

$\frac{4}{7}$

P(positive)	P(negative)

w_i	P(w_i|positive)	P(w_i|negative)
funny	$\frac{3}{11}$	0
boring		$\frac{5}{9}$
movie	$\frac{4}{11}$	$\frac{2}{9}$
cast		
good		0

$$\frac{\text{Count}(\text{funny}, \text{pos}) + 1}{\sum \text{Count}(w, \text{pos}) + |V|} = \frac{3}{11}$$

Review	Class
very **good** and **funny movie**!	pos
what a **funny cast**!	pos
a very **boring movie** and **boring cast**	neg
very **boring cast**!	neg
such a **funny movie**!	pos
really **good cast**, really **good movie**.	pos
"**boring**...such a **boring movie**!!!	neg

Naive Bayes Classifier — Worked Example

Smooth FE.

- Calculating **priors** (with Laplace Smoothing)
  - Number of reviews $N = 7$
  - Number of positive reviews $N_{pos} = 4$
  - Number of negative reviews $N_{neg} = 3$

$$P(pos) = \frac{N_{pos} + 1}{N + |Y|} = \frac{4 + 1}{7 + 2} = \frac{5}{9}$$

1 for positive review

$$P(neg) = \frac{N_{neg} + 1}{N + |Y|} = \frac{3 + 1}{7 + 2} = \frac{4}{9}$$

1 for negative review.

P(pos)	P(neg)
5/9	4/9

Naive Bayes Classifier — Worked Example

- Calculating **likelihoods** (with Laplace Smoothing)

$$\hat{P}(funny|pos) = \frac{Count(funny, pos) + 1}{\sum_{w \in V} Count(w, pos) + |V|} = \frac{3 + 1}{11 + 5} = \frac{4}{16}$$

$$\hat{P}(funny|neg) = \frac{Count(funny, neg) + 1}{\sum_{w \in V} Count(w, neg) + |V|} = \frac{0 + 1}{9 + 5} = \frac{1}{14}$$

…

w_i	$P(w_i\|pos)$	$P(w_i\|neg)$
funny	**4/16**	**1/14**
boring	1/16	6/14
movie	4/16	3/14
cast	3/16	3/14
good	4/16	1/14

We have the **priors** and **likelihoods** ➜ Naive Bayes Classifier is done training

Naive Bayes Classifier — Worked Example

P(pos)	P(neg)
5/9	4/9

- Predict the class for a new review

Review	Class
a funny movie and cast	**???**

w_i	$P(w_i\|pos)$	$P(w_i\|neg)$
funny	4/16	1/14
boring	1/16	6/14
movie	4/16	3/14
cast	3/16	3/14
good	4/16	1/14

$$P(pos|funny, movie, cast) \propto P(pos)P(funny|pos)P(movie|pos)P(cast|pos) = \frac{5}{9} \cdot \frac{4}{16} \cdot \frac{4}{16} \cdot \frac{3}{16} = 0.0065$$

$$P(neg|funny, movie, cast) \propto P(neg)P(funny|neg)P(movie|neg)P(cast|neg) = \frac{4}{9} \cdot \frac{1}{14} \cdot \frac{3}{14} \cdot \frac{3}{14} = 0.0015$$

$P(pos|funny, movie, cast) > P(neg|funny, movie, cast)$ ➜ Label review with "pos"

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Naive Bayes Classifier III (Discussion & Limitations)

Naive Bayes Classifier — Discussion

- Naive Bayes vs. Language Models
  - Naive Bayes makes a non-contextual decision (unigram model; but can be extended to larger n-grams)
  - Naive Bayes treats each class like a separate language model

- Biggest pro: simplicity
  - Easy to understand & implement, fast, not very data hungry, interpretable results

- Biggest con: assumption of conditional independence
  - For most types of data, the features are typically not independent
  - For text classification (features = words), it actually often works well in practice
    (particularly with some additional "tweaking" of the data)

Naive Bayes Classifier — Limitations

- Example: Sentiment Analysis
  - Naive Bayes incapable to handle some relevant linguistic phenomena
  - Most prominently: **negation** (typically flips the sentiment)

Particularly a problem if "not" is removed as a stopword

$$P(pos|"the\ movie\ is\ very\ funny.") \approx P(pos|"the\ movie\ is\ not\ very\ funny.")$$

- Possible countermeasure (to handle negation)
  - Add prefix "NOT" to every word between negation word and next punctuation mark
    (**Note:** this is a common heuristic which is neither trivial nor perfect — but if often works well)

$$"the\ movie\ is\ not\ very\ funny." \rightarrow "the\ movie\ is\ not\ NOT\_very\ NOT\_funny."$$

Naive Bayes Classifier — Limitations

- Example: Sentiment Analysis
  - Sentiment is often expressed/conveyed in phrases or idioms (not just individual words)
  - Other challenges: modals (e.g., *may*, *might*), conditionals (e.g., *if*), questions, literary devices (e.g., sarcasm)
  - Often requires deep world and contextual knowledge

★★★★★ Dec 07, 2021

If you don't love this movie you're the problem

★★★★☆ 1d ago

Not my cup of tea. Good cast. A decent movie experience overall

★☆☆☆☆ 4d ago

Finally saw this yesterday. I have watched screen savers with more tension

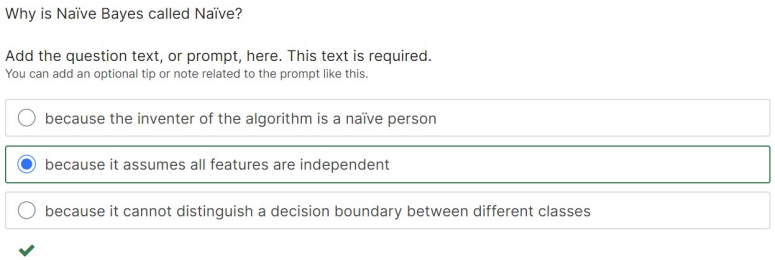
★★★★½ 4d ago

Only thing wrong with this movie is that it ended too soon. Oh, and don't get too attached to any of the characters.

Note: These challenges are not limited to the Naive Bayes classifier.

Naive Bayes Classifier — Summary

- Naive Bayes = class-specific language model
  - Probabilistic classifier based on Bayes Rule
- Good baseline
  - Robust, fast to train, low storage requirements
  - Works actually pretty well for many text classification tasks
    (e.g., sentiment analysis over reviews which often contain very indicative words)
- Strong assumption: conditional independence
  - Requires careful assessment if this assumption holds (at least somewhat)
  - Can engineer tweaks to address this issue (e.g., negation handling)



NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Logistic Regression I (Basic Setup: Linear Models)

Logistic Regression is a statistical machine learning algorithm used for binary classification, which means that it is used to predict one of two possible outcomes – although the extensions to multiclass classification for more than two outcomes is rather straightforward. Logistic Regression is important and popular for several reasons:

- **Simplicity:** Logistic Regression is a simple and easy-to-understand algorithm that can be implemented quickly. It does not require much computational power, and it is not computationally expensive.
- **Interpretability:** Logistic Regression provides interpretable results, meaning that the coefficients and the odds ratios can be easily interpreted to understand the relationships between the input variables and the target variable.
- **Efficiency:** Logistic Regression is efficient in processing large datasets, which makes it a suitable algorithm for large-scale applications.
- **Flexibility:** Logistic Regression can handle both binary and multi-class classification problems.
- **Robustness:** Logistic Regression is robust to noise and can handle irrelevant features in the dataset.
- **Performance:** Logistic Regression can achieve high performance on a wide range of classification tasks, especially when the classes are well-separated.
- **Used in various applications:** Logistic Regression has been used in various applications, including healthcare, finance, marketing, and social sciences.
- **Towards Neural Networks:** A neuron in a neural network can be understood as a single Logistic Regression unit. Thus, understanding Logistic Regression helps to also understand neural networks.

Linear Models

- Underlying assumption:
  - There exists linear relationship between $x^{(j)}$ and dependent variable $y^{(j)}$

feature

class labels.

estimate

bias

coefficient.

$$\hat{y}^{(j)} = h_\theta\left(x^{(j)}\right) = f\left(b + \theta_1 x_1^{(j)} + \theta_2 x_2^{(j)} + \cdots + \theta_n x_n^{(j)}\right)$$

$$= f\left(\left[\sum_{i=1}^{n} \theta_i x_i^{(j)}\right] + b\right)$$

from data.

Predicted value which is hopefully close to $y^{(j)}$

true value.

$$\theta = \{b, \theta_1, \theta_2, \ldots, \theta_n\},\ b \in \mathbb{R},\ \theta_i \in \mathbb{R}$$

These are the parameters we need to learn
➜ Learning = finding the "right" parameter values

estimate close to the true value.

Linear Models — More User-Friendly Notation

- Vector representation
  - **Bias Trick:** Introduce constant feature $x_0^{(j)}$

$$h_\theta\left(x^{(j)}\right) = f\left(\theta_0 \underbrace{x_0^{(j)}}_{=1} + \theta_1 x_1^{(j)} + \theta_2 x_2^{(j)} + \cdots + \theta_n x_n^{(j)}\right)$$

b = bias, $= \theta_0$

  - Represent $x^{(j)}$ with new constant feature

artificial feature

$$x^{(j)} = \left(1, x_1^{(j)}, x_2^{(j)}, \ldots, x_n^{(j)}\right)$$

  - Rewrite linear relationship using vectors representing $x^{(j)}$ and θ

$$h\left(x^{(j)}\right) = f\left(\theta^T x^{(j)}\right) \qquad \theta = \{\theta_0, \theta_1, \theta_2, \ldots, \theta_n\},\ \theta_i \in \mathbb{R}$$

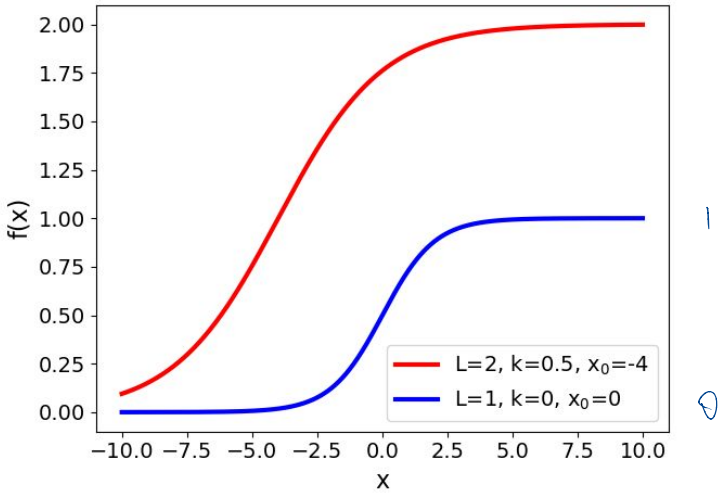
Note: Throughout the rest of the slide, we drop to superscript in $x^{(j)}$ and $y^{(j)}$ if there is no ambiguity.

Logistic Regression

- Logistic Regression ➜ Real-valued predictions interpreted as probability
  - Function *f* is the standard **Logistic Function** (Sigmoid function)

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \xrightarrow{L=1,\ k=1,\ x_0=0} f(x) = \frac{1}{1 + e^{-x}}$$

general form



Logistic Regression — Probabilistic Interpretation

- $\hat{y}$ interpreted as a probability

$$\hat{y} = h_\theta(x) = f(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \qquad \text{with} \quad \hat{y} \in [0, 1]$$

text document

➜ $\hat{y} = h_\theta(x)$ is the estimated probability that $y = 1$ given x and θ

$$\hat{y} = P(y = 1 | x, \theta)$$

true value

➜ Given only discrete, binary outcomes: $P(y = 1 | x, \theta) + P(y = 0 | x, \theta) = 1$

↳ binary classification

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

Logistic Regression is a Linear Model and as such relies on the assumption that there is a linear relationship between the input features *x* and the output *y* (in case of classification: a class label). This is expressed using the following formula:

$$y = h_\theta(x) = f\left(\theta^T x\right) = f\left(\theta_0 + \theta_1 x_1 + \theta_1 x_1 + \ldots + \theta_n x_n\right)$$

This formula naturally requires that each feature x_i is a numerical value. A text document does not satisfy this requirement. We therefore need to convert any text document into a set of numerical features to represent that document. This task is called **Feature Extraction**, and we will cover this topic in the following subsection.

Throughout this subsection about Logistic Regression, we perform this step using hand-crafted features – that is, we define features we deem useful for describing a text document for a given task. In this subsection, we consider the task of sentiment analysis, where we want to classify whether movie review expresses a positive or negative sentiment. Here, meaningful features can be:

- The number of positive and negative words
- The number of emoticons
- The number of exclamation marks
- The length of the movie review
- etc.

All of these extracted numeric values form our input feature vector *x*. The following video details this feature extraction step and introduces the worked example we will use throughout this subsection.

Logistic Regression — Worked Example (Part 1)

- Sentiment Analysis for movie reviews

positive or negative.
⇒ binary classification.

"It's hokey. There are no surprises, the writing is poor. 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."

Feature	Description	Value
x_1	Number of positive words	
x_2	Number of negative words	
x_3	1 if "no" in text; 0 otherwise	
x_4	Number of 1st & 2nd person pronouns	
x_5	1 if "!" in text; 0 otherwise	
x_6	*ln* of word/token count	

Side notes:

- Naive Bayes and Logistic Regression require feature engineering as they do not combine primitive features into composite ones.
- The 6 features on the left are chosen for simplicity; in practice these can be the **tf-idf** weights (we talk about this later in this section)

Logistic Regression — Worked Example (Part 1)

- Step 1: Extract feature values

仿效作时，虚情假意

"It's hokey. There are no surprises, the writing is poor. 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."

Feature	Description	Value
x_1	Number of positive words	3
x_2	Number of negative words	2
x_3	1 if "no" in text; 0 otherwise	1
x_4	Number of 1st & 2nd person pronouns	3
x_5	1 if "!" in text; 0 otherwise	0
x_6	*ln* of word/token count	ln(66) = 4.19

X vector

Logistic Regression — Worked Example (Part 1)

- Step 2: Factor in weights θ
  - Let's assume some oracle gave us those weights
  - It's time to include the bias using the "bias trick"

Feature	Description	Value	Weight θ_i
x_0	Bias b	1	0.1
x_1	Number of positive words	3	2.5
x_2	Number of negative words	2	-5.0
x_3	1 if "no" in text; 0 otherwise	1	-1.2
x_4	Number of 1st & 2nd person pronouns	3	0.5
x_5	1 if "!" in text; 0 otherwise	0	2.0
x_6	*ln* of word/token count	4.19	0.7

Logistic Regression — Worked Example (Part 1)

- Step 4: Compute linear signal (sum of weighted features)

Feature	Description	Value	Weight θ_i	$\theta_i x_i$
x_0	Bias *b*	1	0.1	0.1
x_1	Number of positive words	3	2.5	7.5
x_2	Number of negative words	2	-5.0	-10.0
x_3	1 if "no" in text; 0 otherwise	1	-1.2	-1.2
x_4	Number of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in text; 0 otherwise	0	2.0	0
x_6	*ln* of word/token count	4.19	0.7	2.933

$$\sum = 0.833$$

Vector notation:

$$x = (1, 3, 2, 1, 3, 0, 4.19)^T$$

$$\theta = (0.1, 2.5, -5.0, -1.2, 0.5, 2.0, 0.7)^T$$

$$\rightarrow \theta^T x = 0.833$$

dot product

Logistic Regression — Worked Example (Part 1)

$$\theta^T x = 0.833$$

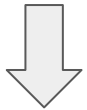
- Step 4: Compute probabilities

movie review.

0.833 = linear signal.

$$P(+|x) = P(y = 1|x, \theta) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} = \frac{1}{1 + e^{-0.833}} = 0.7$$

$$P(-|x) = P(y = 0|x, \theta) = 1 - P(y = 1|x, \theta) = 0.3$$



$$P(+|x) > 0.5 \quad \rightarrow \quad \hat{y} = + \text{ (positive)}$$

Classify movie review as "positive"

Logistic Regression

- So, where did the values for θ come from?
  (in the example, they were simply given to us)
  - Of course, different θ values would have resulted in different probabilities

- Break down into 2 questions

  (1) *How can we quantify how good a set of θ values is?*

  ➜ **Loss function** (also: cost function, error function)

  (2) *How can we systematically find the best θ values?*

  ➜ **Gradient Descent** (numerical method to minimize loss function)

⇒ learning or training.
(find best θ value)

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

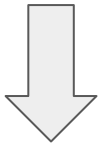
Section 4 — Logistic Regression II (Loss Function)

Logistic Regression — Loss Function

- Intuition: A set of values for θ is good if
  - the correct label y (0 or 1; coming from the dataset)
  - the model's estimated label $\hat{y} = \sigma(\theta^T x)$

  are similar for all $\langle x, y \rangle$ pairs

➜ Find θ that **minimizes the difference** between $\hat{y}$ and y



loss function

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

Logistic Regression — Loss Function

also called Cost function

$$\hat{y} = \frac{1}{1 + e^{-\theta^T x}}$$

- Goal: Maximize probability of the correct label $P(y|x)$

$$h_\theta(x) = \hat{y} = P(y = 1|x, \theta) = 1 - P(y = 0|x, \theta)$$

- Intermediate step: Combine both case into one formula
  - $P(y|x)$ is a Bernoulli distribution (2 discrete outcomes)

$$P(y|x) = \begin{cases} \hat{y} & , y = 1 \\ 1 - \hat{y} & , y = 0 \end{cases}$$

2 Case.

➜ Combine into: $\boxed{P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}}$

$y = 1$: $\hat{y}^1 (1 - \hat{y})^0 = \hat{y}$

$y = 0$: $\hat{y}^0 (1 - \hat{y})^1 = 1 - \hat{y}$

Logistic Regression — Loss Function

$$\hat{y} = \frac{1}{1 + e^{-\theta^T x}}$$

- Goal: Maximize probability of the correct label $P(y|x)$
  - Find θ that **maximizes**

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

$$\log P(y|x) = \log \left[\hat{y}^y (1 - \hat{y})^{1-y}\right]$$

$$= y \log \hat{y} + (1 - y) \log (1 - \hat{y})$$

$\log ab = \log a + \log b$

$\log a^b = b \log a.$

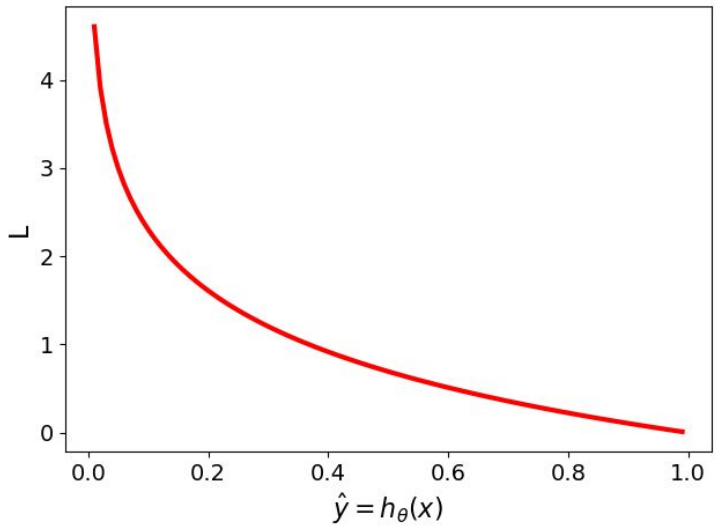
  - Find θ that **minimizes**

$$L_{CE}(\hat{y}, y) = -P(y|x) = \underbrace{-\left[y \log \hat{y} + (1 - y) \log (1 - \hat{y})\right]}_{\textbf{Cross-Entropy Loss}}$$

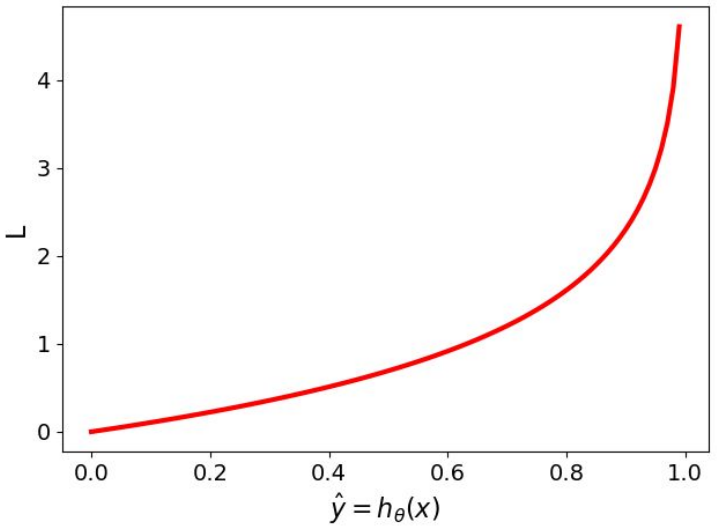
Cross-Entropy Loss — Visualization

$$L_{CE}(\hat{y}, y) = -\left[y \log \hat{y} + (1-y) \log (1-\hat{y})\right]$$

if $y = 1$



if $y = 0$



Cross-Entropy Loss — Total Loss

- Loss for all training samples (given *m* data samples)

$$L_{CE} = \frac{1}{m}\sum_{j=1}^{m} L_{CE}\left(\hat{y}^{(j)}, y^{(j)}\right)$$

average.

$$= -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\log\hat{y}^{(j)} + \left(1-y^{(j)}\right)\log\left(1-\hat{y}^{(j)}\right)\right]$$

$$= -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\log\sigma\left(\theta^T x^{(j)}\right) + \left(1-y^{(j)}\right)\log\left(1-\sigma\left(\theta^T x^{(j)}\right)\right)\right]$$

true class label

$$= -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\log\frac{1}{1+e^{\theta^T x^{(j)}}} + \left(1-y^{(j)}\right)\log\left(1-\frac{1}{1+e^{\theta^T x^{(j)}}}\right)\right]$$

text document.

Cross-Entropy Loss — Worked Example (Part 2)

Recall:

movie review.

$$P(+|x) = \sigma(\theta^T x) = 0.7$$

$$P(-|x) = 1 - \sigma(\theta^T x) = 0.3$$

Feature	Description	Value	Weight θ_i	$\theta_i x_i$
x_0	Bias *b*	1	0.1	0.1
x_1	Number of positive words	3	2.5	7.5
x_2	Number of negative words	2	-5.0	-10.0
x_3	1 if "no" in text; 0 otherwise	1	-1.2	-1.2
x_4	Number of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in text; 0 otherwise	0	2.0	0
x_6	*ln* of word/token count	4.19	0.7	2.933

$$L_{CE}(\hat{y}, y) = -\left[y \log \hat{y} + (1-y) \log (1-\hat{y})\right]$$

Assume the model was right ($y = 1$)

$$L_{CE}(\hat{y}, y) = ???$$

Assume the model was wrong ($y = 0$)

$$L_{CE}(\hat{y}, y) = ???$$

Cross-Entropy Loss — Worked Example (Part 2)

$P(+|x) = \sigma(\theta^T x) = 0.7$

$P(-|x) = 1 - \sigma(\theta^T x) = 0.3$

$$L_{CE}(\hat{y}, y) = -\left[y \log \hat{y} + (1-y) \log (1-\hat{y})\right]$$

Assume the model was right ($y = 1$)

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[\log \hat{y}] \\ &= -[\log 0.7] \\ &= 0.36 \end{aligned}$$

Assume the model was wrong ($y = 0$)

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[\log (1-\hat{y})] \\ &= -[\log 0.3] \\ &= 1.2 \end{aligned}$$

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

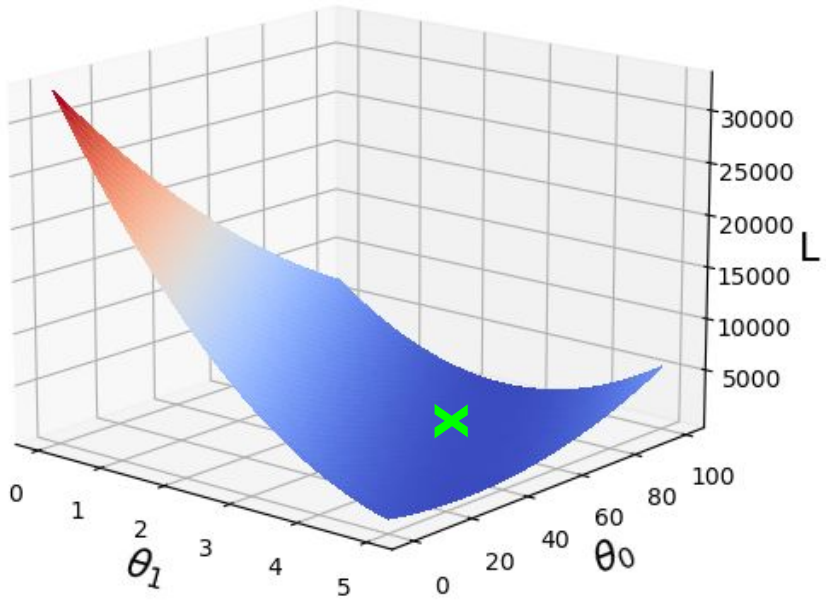
Section 4 — Logistic Regression III (Gradient Descent)

Learning — Minimizing the Loss Function

$$L_{CE} = -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\log\frac{1}{1+e^{\theta^T x^{(j)}}} + \left(1-y^{(j)}\right)\log\left(1-\frac{1}{1+e^{\theta^T x^{(j)}}}\right)\right]$$

Visual illustration of loss function

- Just 1 feature θ_1 and bias θ_0
- Good news: L_{CE} for Logistic Regression is a convex function ➜ 1 global minimum



➜ How to find the minimum of L_{CE}?

Minimizing L using Calculus

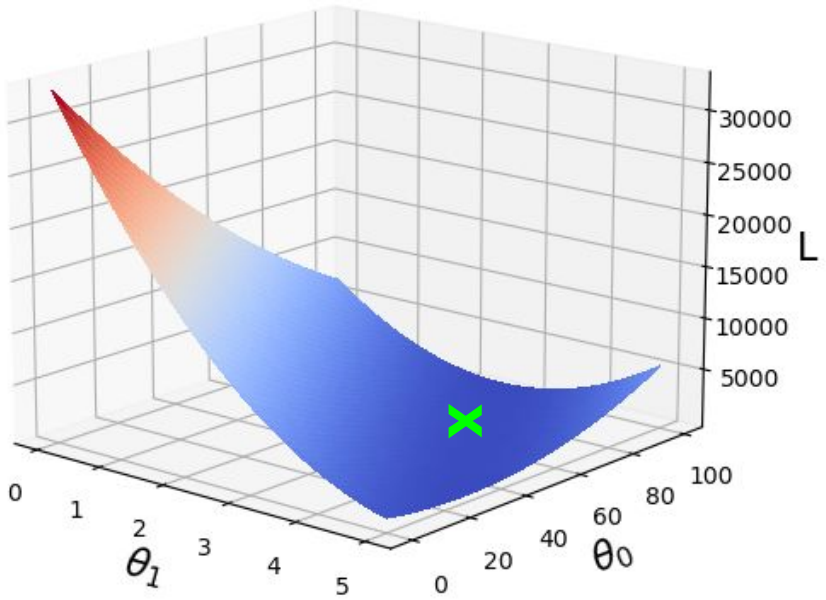
- Minimum of loss function L
  - Partial derivatives w.r.t. to all θ_i are 0

$$\frac{\partial L}{\partial \theta_0} = 0, \quad \frac{\partial L}{\partial \theta_1} = 0, \quad \frac{\partial L}{\partial \theta_2} = 0, \quad \ldots, \quad \frac{\partial L}{\partial \theta_n} = 0$$

n feature
+ 1 bais

  - *n*+1 equations with *n*+1 unknowns
    (➜ 1 unique solution ➜ 1 global minimum)

➜ **What we need:** $\frac{\partial L}{\partial \theta}$



Loss Function — Derivatives

$$L_{CE} = -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\log\sigma\left(\theta^T x^{(j)}\right) + \left(1-y^{(j)}\right)\log\left(1-\sigma\left(\theta^T x^{(j)}\right)\right)\right]$$

…lots of tedious math here…

for single θ_i:

$$\frac{\partial L_{CE}}{\partial \theta_i} = \frac{1}{m}\sum_{j=1}^{m}\left[\sigma\left(\theta^T x^{(j)}\right) - y^{(j)}\right]x_i^{(j)}$$

for all θ:

$$\frac{\partial L_{CE}}{\partial \theta} = \frac{1}{m}X^T\left[\sigma\left(X\theta\right) - y\right]$$

Basic approach to find the minimum

(1) Set derivative to 0 ➜ $\frac{1}{m}X^T\left[\sigma\left(X\theta\right) - y\right] = 0$

(2) Solve for θ

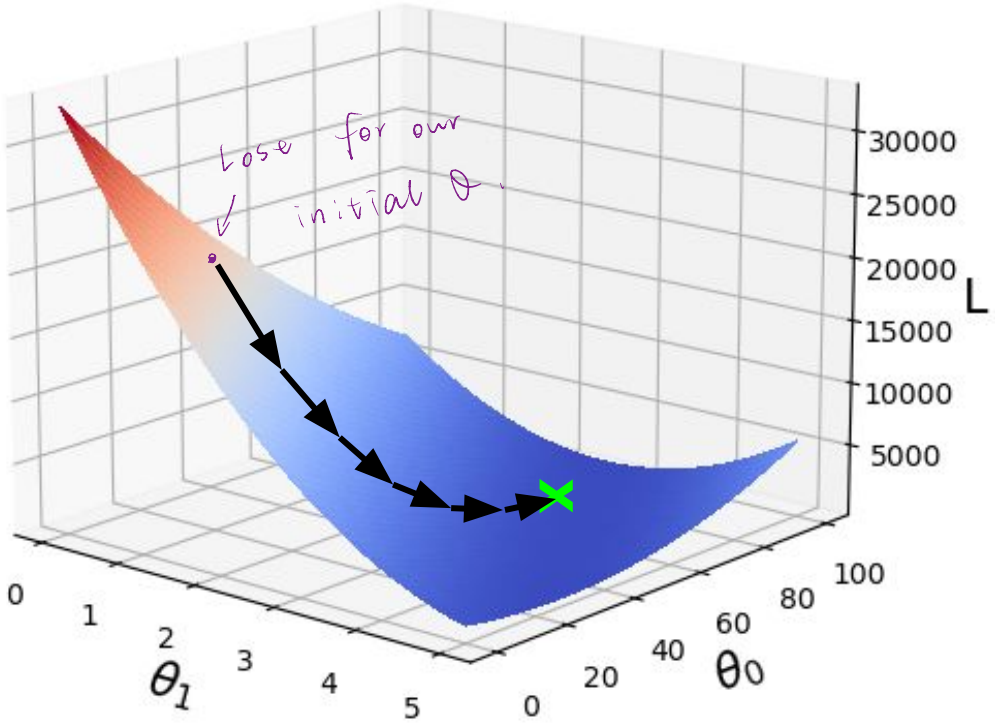
So are we done here?

Gradient Descent

- Problem: $\frac{1}{m} X^T \left[\sigma\left(X\theta\right) - y\right] = 0$ has no closed-form solution for θ

➜ **Gradient Descent**

- Start with a random setting of θ
- Adjust θ iteratively to minimize L



Gradient — Quick Refresher

- Gradient
  - Vector of partial derivatives of a multivariable function (e.g., $\theta_0, \theta_1, \ldots, \theta_n$)
  - Partial derivative: slope w.r.t. to a single variable given a current set of values for all $\theta_0, \theta_1, \ldots, \theta_n$
  - Points in the direction of the steepest ascent

$$\nabla_\theta L = \frac{\partial L}{\partial \theta} = \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \vdots \\ \frac{\partial L}{\partial \theta_n} \end{bmatrix}$$

Gradients — Worked Example (Part 3)

- Calculate Gradients (assuming $y = 1$)

$$\frac{\partial L_{CE}}{\partial \theta} = \frac{1}{m} X^T \left[\sigma\left(X\theta\right) - y\right]$$

Feature	Description	Value	Weight θ_i	$\theta_i x_i$	*Gradients*
x_0	Bias *b*	1	0.1	0.1	**-0.30**
x_1	Number of positive words	3	2.5	7.5	**-0.91**
x_2	Number of negative words	2	-5.0	-10.0	**-0.61**
x_3	1 if "no" in text; 0 otherwise	1	-1.2	-1.2	**-0.30**
x_4	Number of 1st & 2nd person pronouns	3	0.5	1.5	**-0.91**
x_5	1 if "!" in text; 0 otherwise	0	2.0	0	**0.0**
x_6	*ln* of word/token count	4.19	0.7	2.933	**-1.27**

$$\rightarrow \quad \nabla_\theta L_{CE} = \begin{bmatrix} -0.30 \\ -0.91 \\ -0.61 \\ -0.30 \\ -0.91 \\ 0.0 \\ -1.27 \end{bmatrix}$$

Gradients — Worked Example (Part 3)

- Interpretation of gradients
  - Negative values: a small increase in, e.g., θ_0 or θ_1 will decrease the loss
  - A small change in θ_1 affects the loss more than the same change in θ_0
    (since the absolute value of θ_1 is larger than the one of θ_0)
  - Absolute values of gradient not a direct indicator of how to update θ

➜ **So how do we adjust θ to decrease the loss?**

$$\nabla_\theta L_{CE} = \begin{bmatrix} -0.30 \\ -0.91 \\ -0.61 \\ -0.30 \\ -0.91 \\ 0.0 \\ -1.27 \end{bmatrix}$$

Gradient Descent Algorithm

- Important concept: learning rate
  - Scaling factor for gradient (typical range: 0.01 - 0.0001)

text doc

class label.

Input : data (X, y), loss function L, learning rate η

Initialization : Set θ to random values

while true :

step 1: Calculate gradient $\nabla_\theta L$

step 2: $\theta \leftarrow \theta - (\eta \cdot \nabla_\theta L)$

scale gradient use learning rate.

In practice: stop loop when θ converges

Gradient Descent — Worked Example (Part 4)

- Update weights θ
  - Learning rate: $\eta = 0.1$

$$\theta \leftarrow \theta - (\eta \cdot \nabla_\theta L)$$

Feature	Description	Value	Weight θ_i	$\theta_i x_i$	*Gradients*	New Weight θ_i
x_0	Bias *b*	1	0.1	0.1	-0.30	**0.13**
x_1	Number of positive words	3	2.5	7.5	-0.91	**2.59**
x_2	Number of negative words	2	-5.0	-10.0	-0.61	**-4.94**
x_3	1 if "no" in text; 0 otherwise	1	-1.2	-1.2	-0.30	**-1.17**
x_4	Number of 1st & 2nd person pronouns	3	0.5	1.5	-0.91	**0.59**
x_5	1 if "!" in text; 0 otherwise	0	2.0	0	0.0	**2.0**
x_6	*ln* of word/token count	4.19	0.7	2.933	-1.27	**0.83**

➜ **1st iteration of Gradient Descent done!**

$$L_{CE} = 0.12$$

(down from 0.36)

Gradient Descent — Worked Example (Part 4)

- Update weights θ
  - Learning rate: $\eta = 0.1$

after the 1st iteration

$$\theta \leftarrow \theta - (\eta \cdot \nabla_\theta L)$$

Feature	Description	Value	Weight θ_i	$\theta_i x_i$	*Gradients*	New Weight θ_i
x_0	Bias *b*	1	0.13	0.13	-0.11	**0.14**
x_1	Number of positive words	3	2.59	7.77	-0.33	**2.62**
x_2	Number of negative words	2	-4.94	-9.88	-0.22	**-4.92**
x_3	1 if "no" in text; 0 otherwise	1	-1.17	-1.17	-0.11	**-1.16**
x_4	Number of 1st & 2nd person pronouns	3	0.59	1.77	-0.33	**0.62**
x_5	1 if "!" in text; 0 otherwise	0	2.0	0	0.0	**2.0**
x_6	*ln* of word/token count	4.19	0.83	3.46	-0.46	**0.87**

➜ **2nd iteration of Gradient Descent done!**

$$L_{CE} = 0.075$$

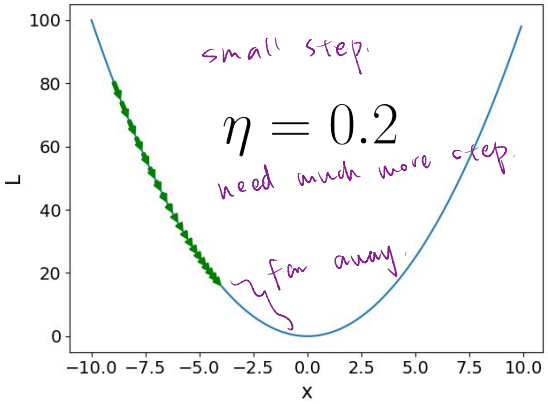
(down from 0.12)

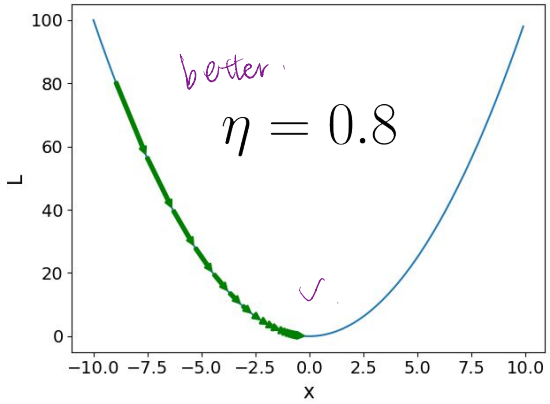
NUS | Computing
National University of Singapore

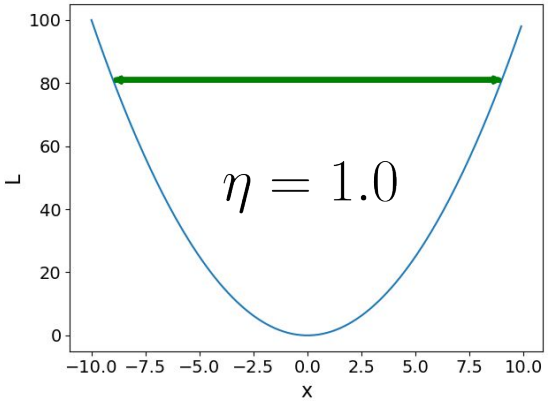
Natural Language Processing: Foundations

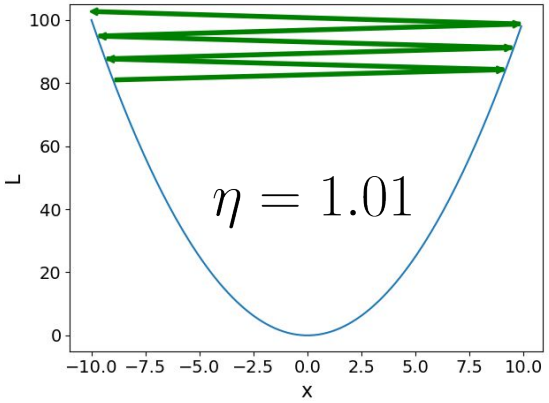
Section 4 — Logistic Regression IV (Practical Considerations)

Effects of Learning Rate for $L = x^2,\ \frac{\partial L}{\partial x} = 2x,\ 20 \text{ steps}$







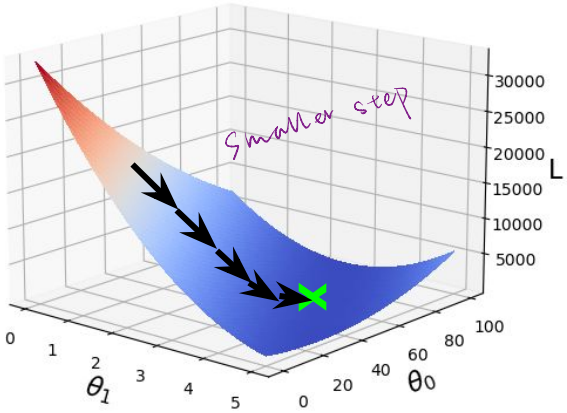


Gradient Descent — Variations

- (Basic) Gradient Descent
  - Calculate gradient und update θ for whole dataset
- Stochastic Gradient Descent (SGD)
  - Calculate gradient und update θ for each data sample
- Mini-batch Gradient Descent
  - Calculate gradient und update θ for batches of sample
  - e.g., batch = 64 data samples
  - In practice often referred to as SGD

Gradient Descent — Variations

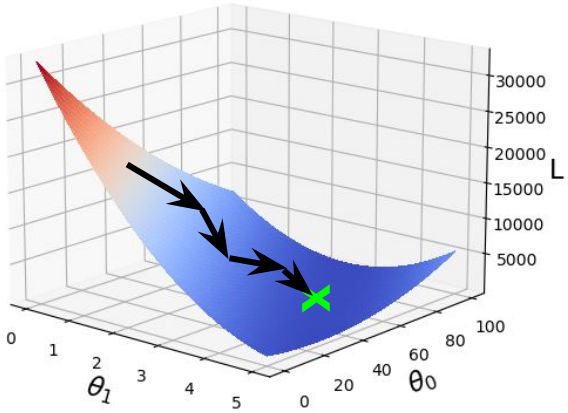
Gradient Descent



Gradient averaged over <u>all</u> data items

- Smooth descent
- Small(er) gradients
- Small(er) update steps

Mini-Batch Gradient Descent

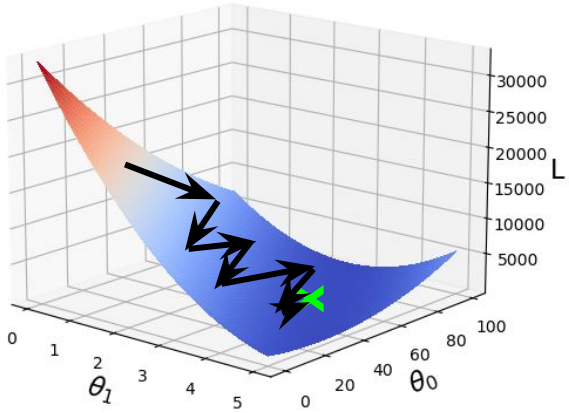


Gradient averaged over <u>some</u> data items

- Well, "somewhere in-between" :)

↑ Common in practice.

Stochastic Gradient Descent



Gradient for each data item considered

- Choppy descent
- Large(r) gradients
- Large(r) steps

Gradient Descent — When to Stop?

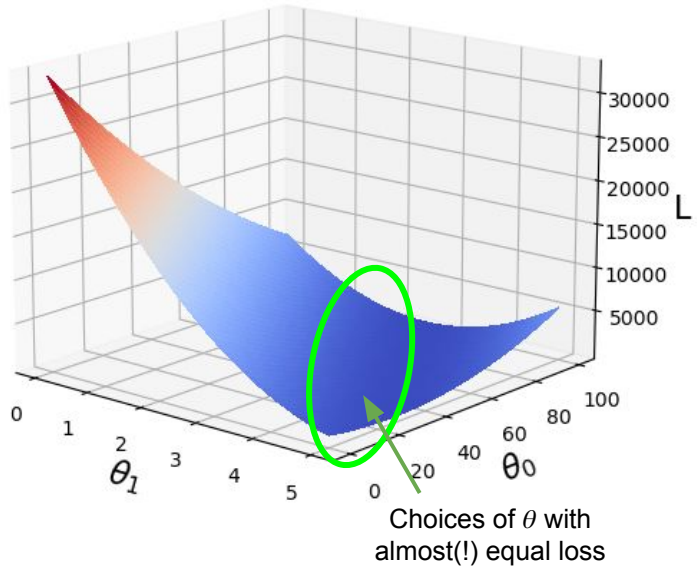
- Intuition: $\nabla_\theta L_{CE} < threshold$

Problem: regions of "near-plateaus":

➜ Gradient $\nabla_\theta L$ very small

➜ Step $\eta \nabla_\theta L$ extremely small

➜ Very slow convergence



Choices of θ with almost(!) equal loss

flat → small step.

- Alternative stop conditions:
  - Loss is small (enough)
  - Change in loss is small enough
  - Max. #iterations reached

Note: This problem is much more pronounced for non-convex loss function with multiple local minima

We showed how we can solve Logistic Regression using Gradient Descent. In practice, however, you won't implement Logistic Regression from scratch but you use existing packages like Python's scikit-learn to train a Logistic Regression Model. Let's have a look at a small screenshot from the documentation showing the constructor with all the parameters.

sklearn.linear_model.LogisticRegression

class sklearn.linear_model.**LogisticRegression**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Do you notice anything odd?

We said that an important parameter when performing Gradient Descent is the learning rate. We used the learning rate to scale the gradient before updating our weights. However, the Logistic Regression implementation of scikit-learn does not seem to feature any parameter reflecting the learning rate. What's going on?

The simple answer is that the implementation of Logistic Regression in the scikit-learn package does not use the basic Gradient Descent algorithm but more sophisticated methods. In fact, the implementation allows you to choose from different so-called solvers; see the screenshot above. For example, the SAG (Stochastic Average Gradient) solver aims to determine how much to change the weights as part of the algorithm. In other words, this method aims to find the best learning rate for each iteration.

However, all solvers are still iteration-based solutions that start with an initial choice of the weights and update those weights in each iteration to minimize the loss function. We introduced Gradient Descent as it is the most basic but also generally applicable method. It is also the core method of choice when we extending the idea of Logistic Regression to Neural Networks.

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Logistic Regression V (Overfitting & Regularization)

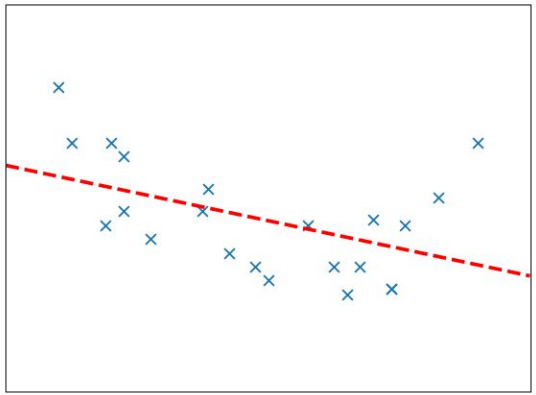
Overfitting — Basic Intuition

有用的 data 和 noisy data.

Overfitting ≒ learning "too much"

- Overfitting — Visualized using curve fitting
  - Task: Find a polynomial for degree *p* that best fit the data points

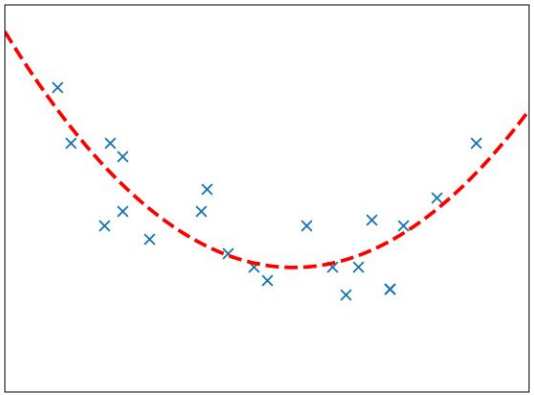
$p = 1$



Underfitting

- Polynomial of degree 1 just a line
- Not capable to fit non-linear data

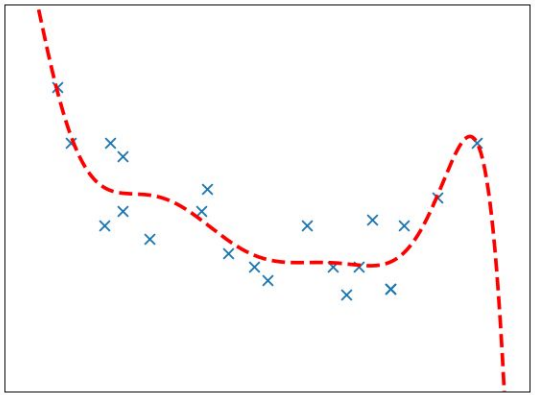
$p = 2$



Good fit

- Model captures the overall trend
- Probably good fit for unseen data

$p = 8$



Overfitting

- Model has too much capacity to exactly fit individual data points
- Probably bad fit for unseen data

Overfitting — Intuition

- Scenario — movie reviews
  - (Very) low number of reviews
  - Assume the following artifact (noise)

    All positive reviews contain many pronouns

    Almost no negative reviews contain pronouns

Feature	Description	Value	Weight θ_i	$\theta_i x_i$
x_0	Bias *b*	1	0.1	0.1
x_1	Number of positive words	3	2.5	7.5
x_2	Number of negative words	2	-5.0	-10.0
x_3	1 if "no" in text; 0 otherwise	1	-1.2	-1.2
x_4	Number of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in text; 0 otherwise	0	2.0	0
x_6	*ln* of word/token count	4.19	0.7	2.933

➜ Effect of Logistic Regression classifier

- Classifiers over-emphasizes the importance of pronouns
  ➜ large value for θ_4 (compared to other θ_i)
- Unseen negative review with many pronouns will most likely be misclassified

Regularization

- Observation
  - Model "too powerful" ⇔ (very) large θ values

➜ **Regularization**: Penalize large θ values

- Extend loss function by penalty term
- For example, for Cross-Entropy loss

λ: Regularization Parameter to control the "strength of the regularization"

$$L = -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\log\sigma\left(\theta^T x^{(j)}\right) + \left(1-y^{(j)}\right)\log\left(1-\sigma\left(\theta^T x^{(j)}\right)\right)\right] + \lambda\sum_{i=1}^{n}\theta_i^2$$

L2 Regularization
("Ridge Regression")

$$L = -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\log\sigma\left(\theta^T x^{(j)}\right) + \left(1-y^{(j)}\right)\log\left(1-\sigma\left(\theta^T x^{(j)}\right)\right)\right] + \lambda\sum_{i=1}^{n}|\theta_i|$$

L1 Regularization
("Lasso Regression")

New Loss ➜ New Gradient

- Since we change L, the gradient $\nabla_\theta L = \frac{\partial L}{\partial \theta}$ also changes
  - No big deal, regularization is just an added term
  - For example, for L2 Regularization (Ridge Regression)

$$\frac{\partial L_{CE}}{\partial \theta} = \frac{1}{m} X^T \left[\sigma\left(X\theta\right) - y\right] + \lambda \frac{2}{n} \theta$$

  - No changes to Gradient Descent Algorithms

Computing

Basic Logistic Regression together the the cross-entropy loss we have introduced so far is defined over a binary classification tasks, i.e., where we have only 2 possible class labels. Practical use cases include spam detection (*"spam"* vs. *"no spam"*) or binary sentiment analysis (*"positive"* vs. *"negative"*).

Of course even more use cases for classification tasks are conceivable that assume more than 2 class labels. This can be as obvious as considering a 3rd class label (e.g., *"neutral"*) for sentiment analysis. Another common example is to assign news articles to a predefined set of categories (*"politics"*, *"finance"*, *"sports"*, *"entertainment"*, *"lifestyle"*, etc.).

Natural Language Processing: Foundations

Section 4 — Logistic Regression VI (Multiclass Logistic Regression)

Binary LR ➜ Multiclass LR

- Multiclass LR: Classification beyond 2 classes
  - Let's assume we have *C* classes: *c* = 1..*C*
  - Separate weights θ_c for each class *c* ➜ *C* output probabilities

C > 2

linear signal

Binary Logistic Regression

$$P(y=1|x) = \sigma(\theta_1^T x)$$

Multiclass Logistic Regression

$$\begin{bmatrix} P(y=1|x) \\ P(y=2|x) \\ \dots \\ P(y=C|x) \end{bmatrix} = f_{mystery} \left(\begin{bmatrix} \theta_1^T x \\ \theta_2^T x \\ \dots \\ \theta_C^T x \end{bmatrix} \right)$$

Probabilities need to sum up to 1

➜ **How can we ensure that?**

$f_{mystery}$ → Softmax

- Softmax function
  - Converts any vector of scores into a vector of probabilities

$$P(y = c|x) = \frac{\exp(\theta_c^T x)}{\sum_{i=1}^{C} \exp(\theta_i^T x)}$$

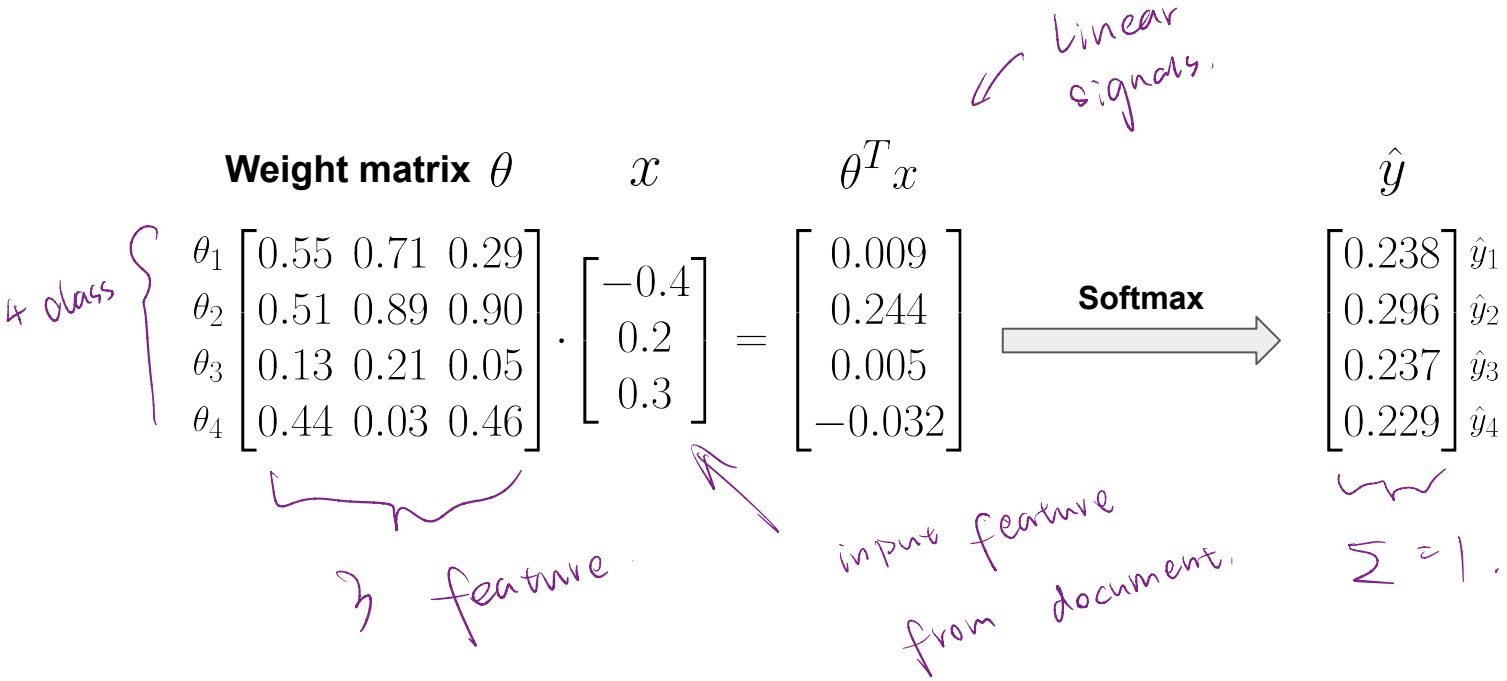
a) values are in [0, 1]

b) all value sum up to 1.

$$\begin{bmatrix} P(y=1|x) \\ P(y=2|x) \\ \dots \\ P(y=C|x) \end{bmatrix} = \frac{1}{\sum_{i=1}^{C} \exp(\theta_i^T x)} \begin{bmatrix} \exp(\theta_1^T x) \\ \exp(\theta_2^T x) \\ \dots \\ \exp(\theta_C^T x) \end{bmatrix}$$

Example

- Example with 4 classes and 3 input features



Cross-Entropy Loss

Cross-Entropy Loss for Binary Logistic Regression

$$L_{CE}(\hat{y}, y) = -\left[y \log \hat{y} + (1-y) \log (1-\hat{y})\right]$$

$y_1 = y$

$\hat{y}_1 = \hat{y}$

$y_2 = 1 - y_1 = 1 - y$

$\hat{y}_2 = 1 - \hat{y}_1 = 1 - \hat{y}$

Generalized Cross-Entropy Loss for Multiclass Logistic Regression

probability output after Softmax

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

$y_i = 1$ for correct class, 0 otherwise

$C = 2$

$-\left[y_1 \log(\hat{y}_1) + y_2 \log(\hat{y}_2)\right]$

New gradient $\nabla_\theta L_{CE}$ but beyond the scope here.

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Evaluation of Classifiers

Evaluating Classifiers — Error Types

- Recall from Section 2: Two basic types of errors
  - Assume there are only 2 classes: **Positive (1)** & **Negative (0)** ➜ binary classification
  - There are 2 ways for a classifier to get it wrong

The classifier incorrectly predicts the label	➜	**False Positives** (Type I Errors)
The classifier incorrectly fails to predict the label	➜	**False Negatives** (Type II Errors)

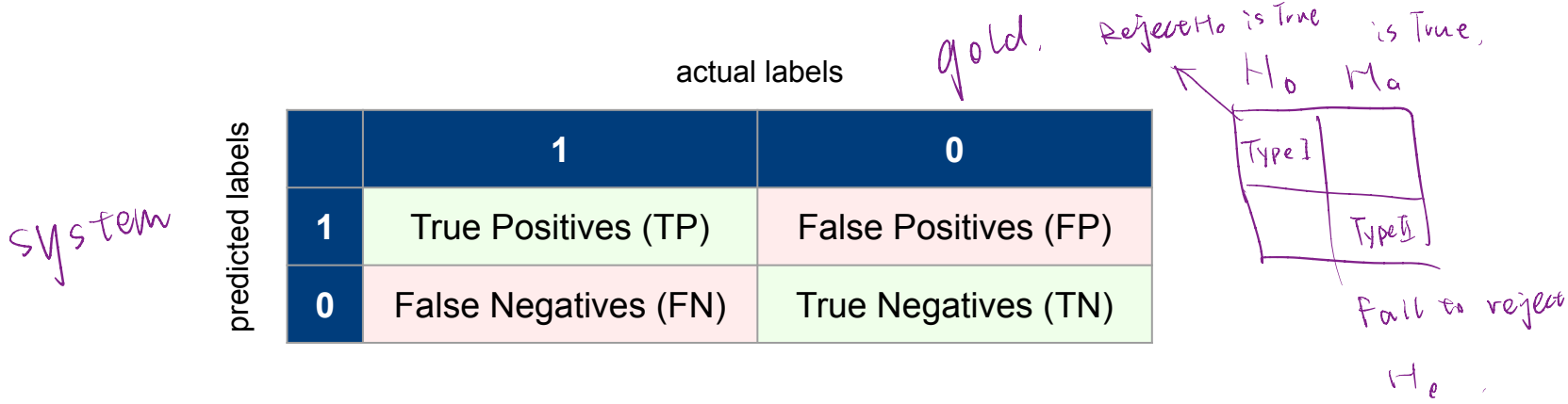
  - Analogously, there are 2 ways to get it right

The classifier correctly predicts the right label	➜	**True Positives**
The classifier correctly fails to predict the label	➜	**True Negatives**

Classification: Evaluation — Confusion Matrix

actual labels

predicted labels	1	0
1	True Positives (TP)	False Positives (FP)
0	False Negatives (FN)	True Negatives (TN)



True Positives (TP): Number of positive classes that have been correctly predicted as positive

True Negatives (TN): Number of negative classes that have been correctly predicted as negative

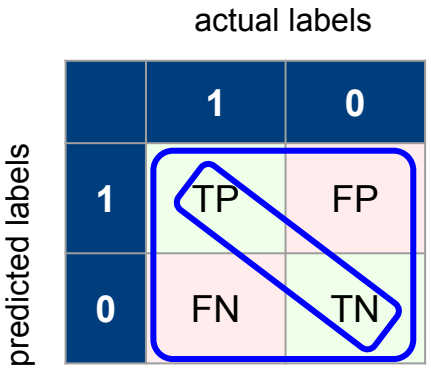
False Positives (FP): Number of negative classes that have been incorrectly predicted as positive

False Negatives (FN): Number of positive classes that have been incorrectly predicted as negative

Classification: Evaluation — Popular Metrics

- Accuracy

$$Accuracy = \frac{TP + TN}{TP + FP + TN + TF}$$



Classification: Evaluation — Popular Metrics

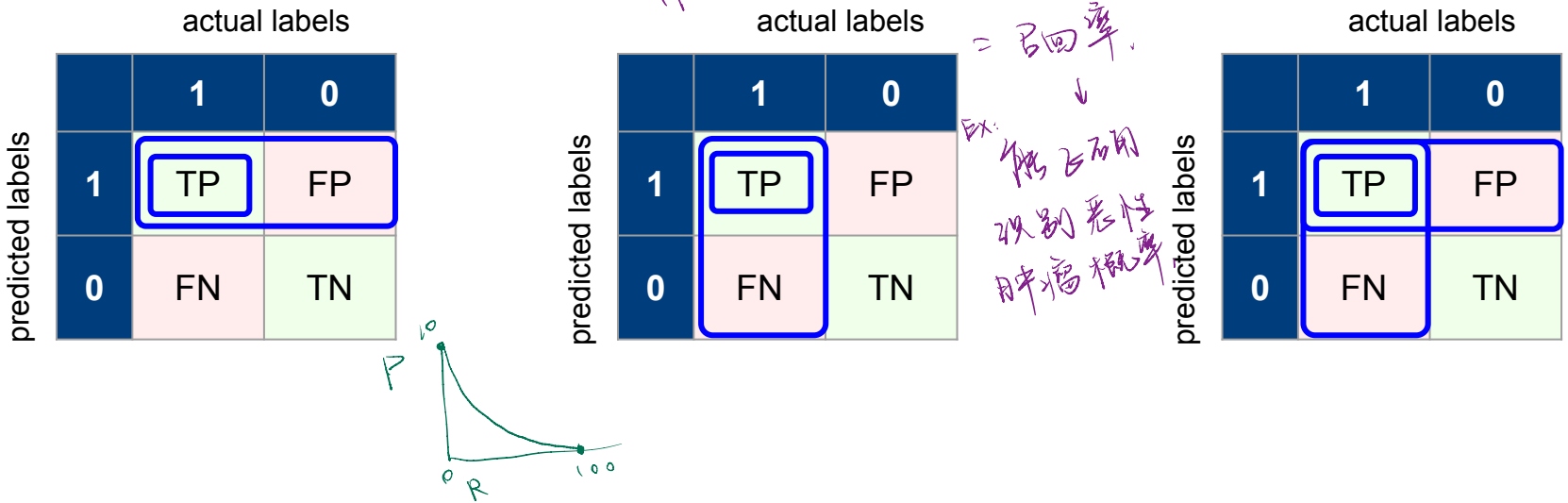
- Precision, Recall, F1 Score

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

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

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Harmonic Mean of Precision and Recall



Type I errors are:

- (●) also known as false positive errors; when the system calls something as positive but it actually negative.
- ( ) also known as false negative errors; when the system calls something as negative but it actually positive.

✔

Summing up all of the (numeric) entries in the confusion matrix yields:

- ( ) the total size of the training set.
- (●) the total size of the number of examples the classifier was applied to.
- ( ) the total number of instances that the classifier got correct.

✔

Select the asymmetric evaluation measures

- [ ] Accuracy.
- [x] Precision.
- [x] F1.
- [ ] None of the above.

✔

Classification: Evaluation — Beyond 2 Classes

- Example: 3 classes, 50 samples

Ex. language check

Malay Chinese English

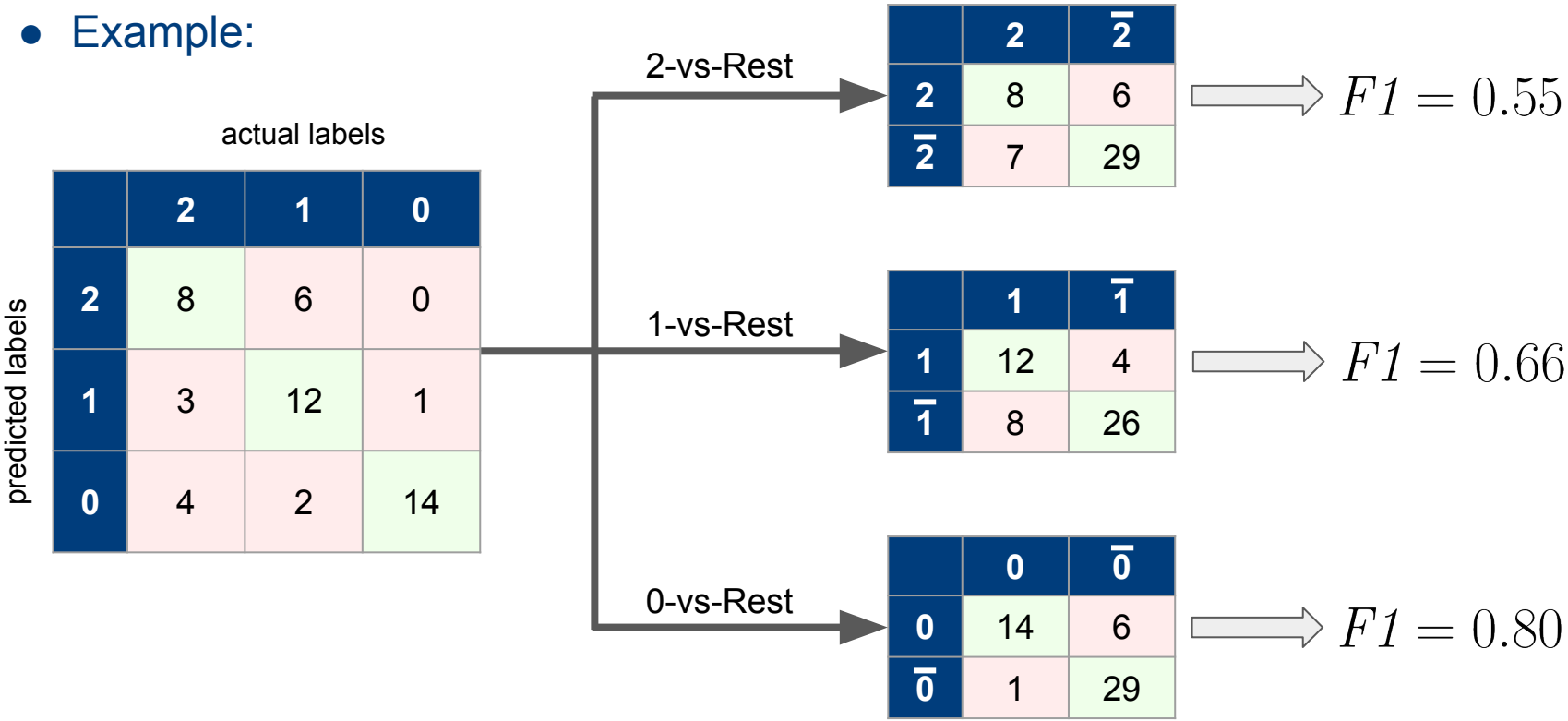
actual labels

predicted labels	2	1	0
2	8	6	0
1	3	12	1
0	4	2	14

$$Accuracy = \frac{8 + 12 + 14}{8 + 12 + 14 + 6 + 3 + 1 + 4 + 2} = 0.68$$

Multiclass Evaluation — One-vs-Rest Confusion Matrices

- Example:



	2	1	0
2	8	6	0
1	3	12	1
0	4	2	14

	2	$\overline{2}$
2	8	6
$\overline{2}$	7	29

	1	$\overline{1}$
1	12	4
$\overline{1}$	8	26

	0	$\overline{0}$
0	14	6
$\overline{0}$	1	29

One-vs-Rest — Micro Averaging

	2	$\overline{2}$
2	8	6
$\overline{2}$	7	29

	1	$\overline{1}$
1	12	4
$\overline{1}$	8	26

	0	$\overline{0}$
0	14	6
$\overline{0}$	1	29

Average over all TP, FP, FN, TN

$\frac{8+12+14}{3} = 11.33$

	C	$\overline{C}$
C	11.33	3.66
$\overline{C}$	7	28

$$F1 = 0.68$$

One-vs-Rest — Macro Averaging

	2	$\overline{2}$
2	8	6
$\overline{2}$	7	29

$F1 = 0.55$

	1	$\overline{1}$
1	12	4
$\overline{1}$	8	26

$F1 = 0.66$

	0	$\overline{0}$
0	14	6
$\overline{0}$	1	29

$F1 = 0.80$

Average over all metrics

$F1 = 0.67$

$\frac{0.55 + 0.66 + 0.8}{3}$

One-vs-Rest — Macro vs. Micro Averaging

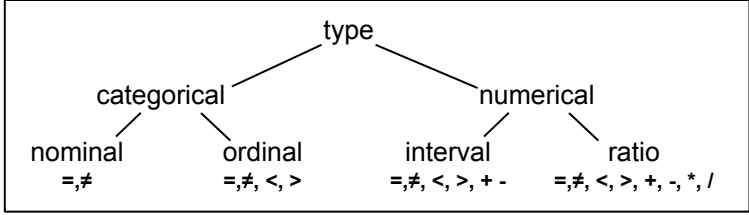
- Both methods use One-vs-Rest confusion matrices
  - All introduced metrics applicable
- Micro-averaging
  - Averaging over TP, FP, FN, TN values of all One-vs-Rest confusion matrices
  - Favors bigger classes (since average over counts)
- Macro-averaging
  - Averaging over metrics derived from each One-vs-Rest confusion matrix
  - Treats all class equally (since metrics are normalized)

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Prerequisite: Vector Space Model I (Basic Approaches)

Motivation — Challenge



- Text is not really "convenient" data
  - Recall: text = sequence of words (well, more generally: tokens)

Variable length

Nominal Data

words are just labels.

vs.

- Many common techniques/algorithms require
  - Standardized/canonical input (particularly fixed length)
  - Numerical input (e.g., to compare, add, scale text documents)

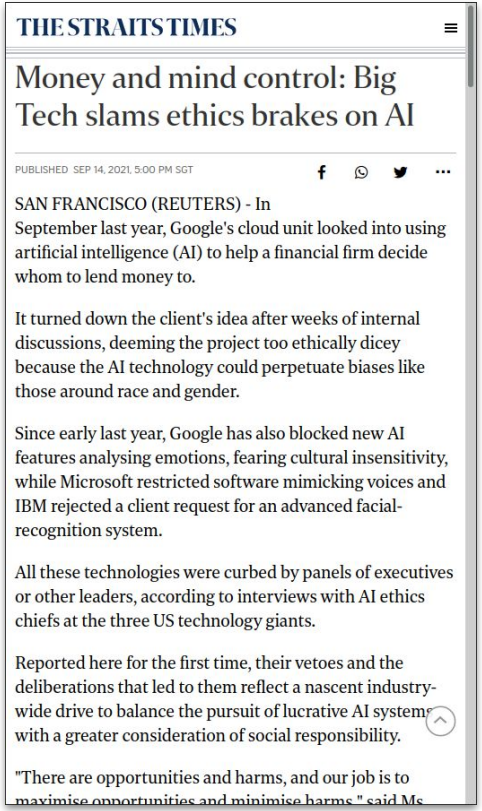
➜ **Many techniques do not work in raw text!**

include logistic regression.

Vector Space Model — Motivation

- Goal: Suitable representation of a text document
  - Numerical input
  - Standardized/canonical input → fixed length.
- Feature extraction ➜ vectorization of text data
  - Represent each text document as a vector of equal size
  - Vector elements = numerical values derived from text

↳ need to preserve interesting characteristic of our text.



THE STRAITS TIMES

Money and mind control: Big Tech slams ethics brakes on AI

PUBLISHED SEP 14, 2021, 5:00 PM SGT

SAN FRANCISCO (REUTERS) - In September last year, Google's cloud unit looked into using artificial intelligence (AI) to help a financial firm decide whom to lend money to.

It turned down the client's idea after weeks of internal discussions, deeming the project too ethically dicey because the AI technology could perpetuate biases like those around race and gender.

Since early last year, Google has also blocked new AI features analysing emotions, fearing cultural insensitivity, while Microsoft restricted software mimicking voices and IBM rejected a client request for an advanced facial-recognition system.

All these technologies were curbed by panels of executives or other leaders, according to interviews with AI ethics chiefs at the three US technology giants.

Reported here for the first time, their vetoes and the deliberations that led to them reflect a nascent industry-wide drive to balance the pursuit of lucrative AI systems with a greater consideration of social responsibility.

"There are opportunities and harms, and our job is to maximise opportunities and minimise harms," said Ms

???

(0.42, 0.02, 0.53, 0.91, 0.21, 0.74, 0.04, …, 0.16, 0.76)

Hand-crafted features are a set of manually designed features that can be used to represent the text data in a numerical format for sentiment analysis. The choice of hand-crafted features can significantly impact the performance of a sentiment analysis model. Here are some commonly used hand-crafted features for sentiment analysis:

- **Word Counts:** The frequency of occurrence of certain words in a text can provide important information about the sentiment. For example, words like "happy," "excited," and "love" are often associated with positive sentiment, while words like *"sad," "angry,"* and *"hate"* are often associated with negative sentiment.
- **Sentiment Lexicons:** Sentiment lexicons are lists of words and their associated sentiment scores, such as positive or negative. Sentiment scores can be assigned to words using various methods, such as crowd-sourcing or using pre-trained models. The sentiment scores of words in a text can be aggregated in various ways, such as by taking the sum or average, to provide a numerical representation of the text's sentiment.
- **N-grams:** N-grams are contiguous sequences of n words in a text. N-grams can capture information about the relationships between words in a text, such as the presence of certain phrases or collocations. For example, the bigram *"not good"* is often associated with negative sentiment.
- **Part-of-Speech Tagging:** Part-of-speech (POS) tagging involves annotating words in a text with their corresponding POS tags, such as noun, verb, adjective, etc. The frequency of certain POS tags in a text can provide information about the sentiment, as certain POS tags are more indicative of positive or negative sentiment.
- **Capitalization Features:** The capitalization of words in a text can provide information about the sentiment, as capitalized words are often associated with strong emotions or emphasis. For example, words in all capital letters are often associated with negative sentiment.

These are some of the commonly used hand-crafted features for sentiment analysis. The choice of hand-crafted features can depend on the specific problem and the type of text data being analyzed. It is important to experiment with different hand-crafted features to determine which ones perform best for a given sentiment analysis task.

"Manual" Approach — Handcrafted Features

- Example: Sentiment Analysis
  - Length of text document (number of tokens or characters)
  - Number of positive and negative emoticons
  - Number of words associated with positive or negative mood

Finding good features can be tricky in practice

- 2 movie reviews
  - R_1: *"The movie was so boring - I hated it after just 20 minutes! :((("*
  - R_2: *"Dune is a such a brilliant and beautiful movie!"*

	#char	#tokens	#emoticons+	#emoticons-	#words+	words-
R_1	64	15	0	1	0	2
R_2	47	10	0	0	2	0

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

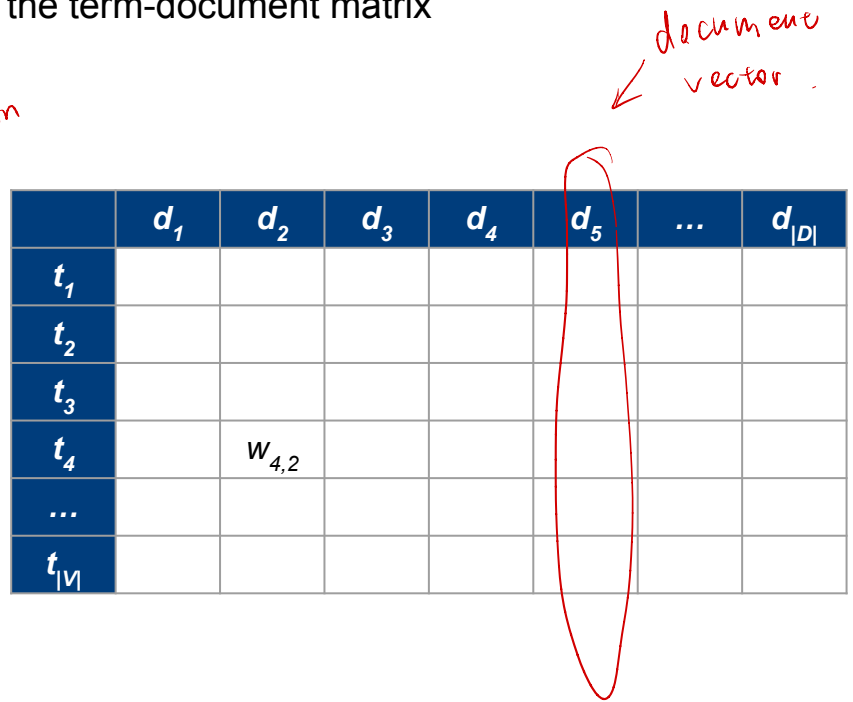
Section 4 — Prerequisite: Vector Space Model II (Set & Bag of Words)

Vector Space Model

- Idea: Vectorize documents based on vocabulary
  - Length each document vector is the size of corpus vocabulary V
  - Vectors for all documents in dataset D form the term-document matrix

- Term-document matrix
  - Set of documents $d_1, d_2, \ldots, d_{|D|}$
  - Set of unique terms $t_1, t_2, \ldots, t_{|V|}$

➜ weight $w_{t,d}$: matrix value depending on representation



	d_1	d_2	d_3	d_4	d_5	…	$d_{\|D\|}$
t_1							
t_2							
t_3							
t_4		$w_{4,2}$					
…							
$t_{\|V\|}$							

Vector Space Model — Example Corpus

d_1 : *Dogs chase cats and other dogs.*

d_2 : *Cats chase other cats.*

d_3 : *There is a car chase on the TV.*

d_4 : *My dog watches other dogs on TV.*

d_5 : *My dog and cat sit in the car.*

Normalization steps:

- Removal of non-words
- Removal of stopwords
- Case-folding (lowercase)
- Lemmatization

Dogs → dog.

d_1 : *dog chase cat dog*

d_2 : *cat chase cat*

d_3 : *car chase tv*

d_4 : *dog watch dog tv*

d_5 : *dog cat sit car*

➜ Vocabulary V = {*car*, *cat*, *chase*, *dog*, *sit*, *tv*, *watch*}

$|V| = 7$

Term-Document Matrix with Binary Weights

- Matrix elements are either 0 or 1
  - $w_{t,d} = 1$: document d contains term t
  - $w_{t,d} = 0$: otherwise

- Interpretation
  - Weights reflect presence or absence of a term in a document
  - No differentiation between words of a document
  - Suitable for basic filtering of documents (e.g., find all documents containing "dog")

d_1 : dog chase cat dog
d_2 : cat chase cat
d_3 : car chase tv
d_4 : dog watch dog tv
d_5 : dog cat sit car

Set of Words (SoW) Representation

	d_1	d_2	d_3	d_4	d_5
car	0	0	1	0	1
cat	1	1	0	0	1
chase	1	1	1	0	0
dog	1	0	0	1	1
sit	0	0	0	0	1
tv	0	0	1	1	0
watch	0	0	0	1	0

Term-Document Matrix with Term Frequencies

d_1 : *dog chase cat dog*
d_2 : *cat chase cat*
d_3 : *car chase tv*
d_4 : *dog watch dog tv*
d_5 : *dog cat sit car*

- Matrix elements are integers
  - $w_{t,d}$: #occurences of term t in document d

    ➜ **term frequency** $tf_{t,d}$

- Interpretation
  - Assumption: more frequent terms in a document are more important

BUT: Does "more frequent" always mean "more important"?

↳ generally No.

这个表示次数.

Bag of Words (BoW) Representation

	d_1	d_2	d_3	d_4	d_5
car	0	0	1	0	1
cat	1	**2**	0	0	1
chase	1	1	1	0	0
dog	**2**	0	0	**2**	1
sit	0	0	0	0	1
tv	0	0	1	1	0
watch	0	0	0	1	0

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 4 — Prerequisite: Vector Space Model II (TF-IDF)

$tf_{t,d}$ as a Indicator for a Term's Importance

- Consideration 1: Relative importance
  - Assume 2 documents d_1 and d_2 containing the term *"NLP"*
  - d_1 contains *"NLP"* 100 times, d_2 contains *"NLP"* 10 times

$tf_{NLP,d_1} > tf_{NLP,d_2}$ ➜ d_1 more important than d_2 w.r.t. *"NLP"* ✓

But is d_1 really 10x more important than d_2?

No

Optional by common.

to damped the effect

➜ Extension: Use a sublinear function to model importance based on $tf_{t,d}$

- Common: **logarithm**
- Different functions possible and not always required

$$w_{t,d} = min \begin{cases} 1 + \log_{10} tf_{t,d} & , \text{if } tf_{t,d} > 0 \\ 0 & , \text{otherwise} \end{cases}$$

$tf_{t,d}$ as a Indicator for a Term's Importance

- Consideration 2: Cross-document importance
  - Assume a document d_1 containing the term *"NLP"* many times
  - Let *"NLP"* also be frequent in many to most other documents

Is *"NLP"* really important (i.e., characteristic, informative) for d_1?

- Intuition — example: "dog watch dog tv"
  - "dog" appears 2x in the document, but also in 3/5 of the other documents
  - "watch" appears 1x in the document, but also only in this document

$tf_{t,d}$ as a Indicator for a Term's Importance

➜ Extension: **Inverse Document Frequency** idf_t as additional factor

- Document frequency df_t: #document containing t
- Inverse measure of a terms importance, relevance, informativeness

➜ Inverse Document Frequency: $idf_t = \log \frac{|D|}{df_t}$

Again, log to dampen the effect of the inverse document frequency

Term-Document Matrix with tf-idf Weights

- Putting it all together

$$w_{t,d} = (1 + \log_{10} tf_{t,d}) \cdot \log_{10} \frac{|D|}{df_t}$$

damped tf

damped inverse df.

- Side notes
  - No real theoretic underpinning, but tf-idf works best in practice
  - Not all definitions of tf-idf apply a sublinear scaling of $tf_{t,d}$ → optional
  - Alternative names: $tf \cdot idf$, $tf \times idf$
  - There are different weighting functions for calculating tf-idf different log base. $\log_2$, $\ln$

Term-Document Matrix with tf-idf Weights

d_1 : dog chase cat dog
d_2 : cat chase cat
d_3 : car chase tv
d_4 : dog watch dog tv
d_5 : dog cat sit car

- Example

$$w_{t,d} = (1 + \log_{10} tf_{t,d}) \cdot \log_{10} \frac{|D|}{df_t}$$

出现两次

5个文件中出现3次

$$w_{dog,d_4} = (1 + \log_{10} 2) \cdot \log_{10} \frac{5}{3} = (1 + 0.3) \cdot 0.22 = 0.29$$

$$w_{watch,d_4} = (1 + \log_{10} 1) \cdot \log_{10} \frac{5}{1} = (1 + 0) \cdot 0.7 = 0.7$$

Term-Document Matrix with tf-idf Weights

d_1 : dog chase cat dog
d_2 : cat chase cat
d_3 : car chase tv
d_4 : dog watch dog tv
d_5 : dog cat sit car

- Matrix elements = tf-idf weights

numerical

document vector

fixed size

$$w_{t,d} = (1 + \log_{10} tf_{t,d}) \cdot \log_{10} \frac{|D|}{df_t}$$

➜

	d_1	d_2	d_3	d_4	d_5
car	0	0	0.4	0	0.4
cat	0.22	0.29	0	0	0.22
chase	0.22	0.22	0.22	0	0
dog	0.29	0	0	0.29	0.22
sit	0	0	0	0	0.7
tv	0	0	0.4	0.4	0
watch	0	0	0	0.7	0

Consider following query (q) and documents (d1 - d3):

q: Apple ships new Macbook.

d1: TSMC is busy producing new Macbook.

d2: Apple Stores are busy hosting Macbook fans.

d3: New laptop are announced by Tim Cook.

$(1+\log 1)\log_{10} 2$

$(1+\log 0)\log_{10} 2$

d1 : TF

apple : 0

ship : 0

new : 1

Macbook : 1

DF :

apple: 0

ship : 0

new : 1

Macbook : 1 .

$(1+\log 1)$

d2 apple : 1

ship : 0

new : 0

Macbook : 1

apple : 1

ship : 0

new : 0

1

0

0

0

0

0

1

0

Vector Space Model — Document Similarity

- Vector Space Model
  - $|V|$-dimensional vector space
  - Words are axes (i.e., dimensions) of the space
    (each word in vocabulary represent a axis/dimensions)
  - Documents are points or vectors in this space
  - In practice: very high-dimensional space
    (typically tens of thousands of dimensions)

➜ Document vectors are typically very sparse
(i.e., most entries in the vectors are zero)

➜ How can we calculate the **similarity** between text documents

- Many NLP tasks rely on "some meaningful" metric quantifying document similarity
- Using Vector Space Model: document similarity ➜ vector similarity

Document Similarity

- Approach 1: Dot Product
  - The dot product between two vectors v and w is defined as

$$dot(v, w) = v \cdot w = v_1 w_1 + v_2 w_2 + \ldots v_n w_n = \sum_{i=1}^{n} v_i w_i$$

- Interpretation
  - $dot(v, w)$ is high if v and w have large values in the same dimensions

➜ $dot(v, w)$ represents a similarity metric between vectors, but…

Document Similarity

- Limitations of Dot Product
  - $dot(v, w)$ is higher if a vector has higher values in many dimensions

$$dot(v, w) = \sum_{i=1}^{n} v_i w_i$$

➜ $dot(v, w)$ favors long vectors

$$|v| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

- Effects in document vectors
  - $dot(v, w)$ favors frequent words
    (since they occur many times with other documents)
  - $dot(v, w)$ favors long documents
    (since the raw term frequencies are higher)

➜ $dot(v, w)$ overly favors frequent words

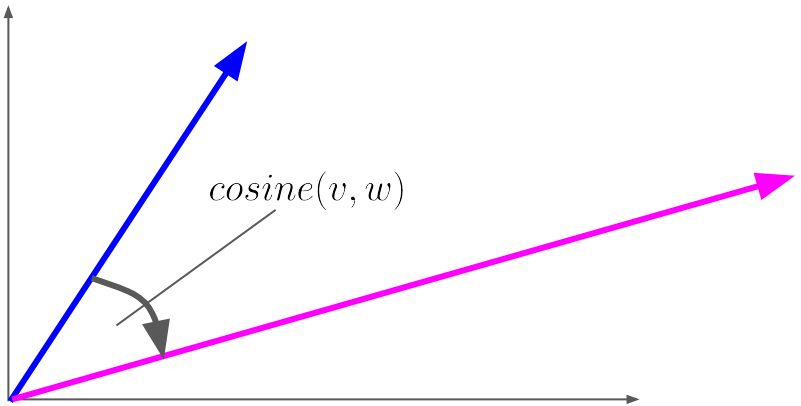
Document Similarity — Cosine Similarity

- Approach 2: Cosine Similarity (dot product normalized by length of vectors)

$$cosine(v,w) = \frac{v \cdot w}{|v| \cdot |w|} = \frac{v \cdot w}{\sqrt{\sum_{i=1}^{n} v_i^2} \cdot \sqrt{\sum_{i=1}^{n} w_i^2}}$$

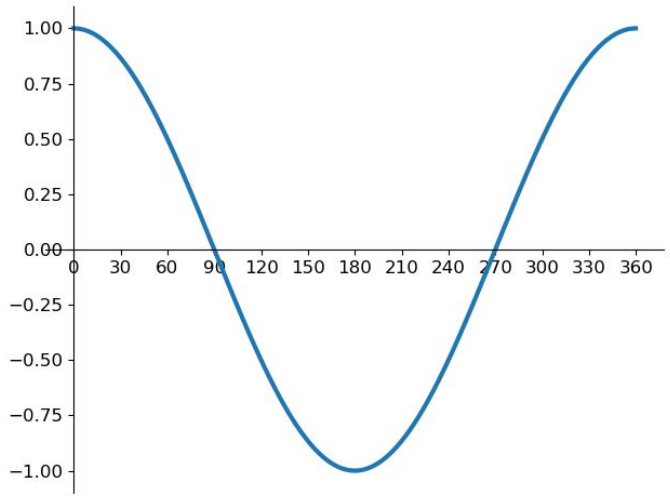
- Geometric interpretation
  - $cosine(v,w)$ measures the angle between vectors

$dot(v,w)$ cares about angle and length



Document Similarity — Cosine Similarity

- Cosine as a similarity metric
  - $cosine(v, w) = -1$
    vectors point in opposite directions
  - $cosine(v, w) = 1$
    vectors point in the same direction
  - $cosine(v, w) = 0$
    vectors are orthogonal



- Cosine similarity for document vectors
  - Vector entries are all positive

➜ $0 \leq cosine(u, v) \leq 1$

Document Similarity — Cosine Similarity

d_1 : dog chase cat dog
d_2 : cat chase cat
d_3 : car chase tv
d_4 : dog watch dog tv
d_5 : dog cat sit car

	d_1	d_2	d_3	d_4	d_5
car	0	0	0.4	0	0.4
cat	0.22	0.29	0	0	0.22
chase	0.22	0.22	0.22	0	0
dog	0.29	0	0	0.29	0.22
sit	0	0	0	0	0.7
tv	0	0	0.4	0.4	0
watch	0	0	0	0.7	0

$$cosine(v, w) = \frac{v \cdot w}{|v| \cdot |w|} = \frac{v \cdot w}{\sqrt{\sum_{i=1}^{n} v_i^2} \cdot \sqrt{\sum_{i=1}^{n} w_i^2}}$$

$$cosine(d_1, d_2) = \frac{(0.22 \cdot 0.29) + (0.22 \cdot 0.22)}{\sqrt{0.22^2 + 0.22^2 + 0.29^2} \cdot \sqrt{0.29^2 + 0.22^2}} = 0.72$$

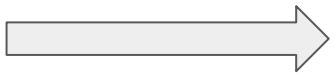
(only non-zero components included)

Document Similarity — Cosine Similarity

d_1 : *dog chase cat dog*
d_2 : *cat chase cat*
d_3 : *car chase tv*
d_4 : *dog watch dog tv*
d_5 : *dog cat sit car*

	d_1	d_2	d_3	d_4	d_5
car	0	0	0.4	0	0.4
cat	0.22	0.29	0	0	0.22
chase	0.22	0.22	0.22	0	0
dog	0.29	0	0	0.29	0.22
sit	0	0	0	0	0.7
tv	0	0	0.4	0.4	0
watch	0	0	0	0.7	0

All pairwise cosine similarities



	d_1	d_2	d_3	d_4	d_5
d_1	1	0.72	0.19	0.23	0.31
d_2		1	0.22	0	0.20
d_3			1	0.31	0.31
d_4				1	0.09
d_5					1

Vector Space Model

- Representing documents as vectors
  - Meaningful way to compute similarities between documents
    (e.g., for ranking documents in information retrieval, clustering)
  - Valid input for other text classifiers beyond Naive Bayes
    (document vectors have no numerical values)

- Limitation: BoW representation of documents
  - Does not consider sequential order of words in a sentence

Step 1: Convert the above sentences in lower case as the case of the word does not hold any information.

Step 2: Remove special characters and stopwords from the text. Stopwords are the words that do not contain much information about text like 'is', 'a','the and many more'.

After applying the above steps, the sentences are changed to

Sentence 1: "welcome great learning now start learning"

Sentence 2: "learning good practice"

Although the above sentences do not make much sense the maximum information is contained in these words only.

Step 3: Go through all the words in the above text and make a list of all of the words in our model vocabulary.

- welcome
- great
- learning
- now
- start
- good
- practice

The scoring method we use here is the same as used in the previous example. For sentence 1, the count of words is as follow:

Word	Frequency
welcome	1
great	1
learning	2
now	1
start	1
good	0
practice	0

Writing the above frequencies in the vector

Sentence 1 → **[1,1,2,1,1,0,0]**

Now for sentence 2, the scoring would be like

Word	Frequency
welcome	0
great	0
learning	1
now	0
start	0
good	1
practice	1

Similarly, writing the above frequencies in the vector form

Sentence 2 → **[0,0,1,0,0,1,1]**

Sentence	welcome	great	learning	now	start	good	practice
Sentence1	1	1	2	1	1	0	0
Sentence2	0	0	1	0	0	1	1