



**NUS**  
National University  
of Singapore

| **Computing**

# **CS4248: Natural Language Processing**

## **Lecture 4 — Text Classification**

# Recap of Week 03

## Probabilities of Sentences — Example

### (1) Application of Chain Rule

$$P(\text{"remember to submit your assignment"}) = P(\text{"remember"}) \cdot P(\text{"to"} \mid \text{"remember"}) \cdot P(\text{"submit"} \mid \text{"remember to"}) \cdot P(\text{"your"} \mid \text{"remember to submit"}) \cdot P(\text{"assignment"} \mid \text{"remember to submit your"})$$

### (2) Maximum Likelihood Estimation

$$P(\text{"remember"}) = \frac{\text{Count}(\text{"remember"})}{N}$$

$$P(\text{"to"} \mid \text{"remember"}) = \frac{\text{Count}(\text{"remember to"})}{\text{Count}(\text{"remember"})}$$

...

$$P(\text{"assignment"} \mid \text{"remember to submit your"}) = \frac{\text{Count}(\text{"remember to submit your assignment"})}{\text{Count}(\text{"remember to submit your"})}$$

😬 **Foreshadowing:**  
Do you see any problems?

14

## Markov Assumption

- Probabilities depend on only on the last  $k$  words

$$P(w_1, \dots, w_N) = \prod_{n=1}^N P(w_n | w_{1:n-1}) = \prod_{n=1}^N P(w_n | w_{n-k:n-1})$$

- For our example:

$$P(\text{"assignment"} \mid \text{"remember to submit your"}) \approx P(\text{"assignment"} \mid \text{"your"})$$

$$P(\text{"assignment"} \mid \text{"submit your"})$$

$$P(\text{"assignment"} \mid \text{"to submit your"})$$

...

18

## Smoothing

### Basic idea

- Avoid assigning probabilities of 0 to unseen n-grams
- "Move" some probability mass from more frequent n-grams to unseen n-grams
- Also called: **discounting**



### Basic method: Laplace Smoothing (also: Add-1 Smoothing)

- Example for bigrams

	i	like	the	story
i	0	693	20	0
like	326	0	1,997	8
the	15	42	0	5,171
story	23	16	16	0



	i	like	the	story
i	1	694	21	1
like	327	1	1,998	9
the	16	43	1	5,172
story	24	17	17	1

35

## Kneser-Ney Smoothing — Wrapping it Up

$$P_{KN}(w_n | w_{n-1}) = \frac{\max[Count(w_{n-1}w_n) - d, 0]}{Count(w_{n-1})} + \underbrace{\lambda(w_{n-1})}_{\text{last missing puzzle piece}} P_{KN}(w_n)$$

### Normalizing factor $\lambda$

- Required to account for the probability mass we have discounted

$$\lambda(w_{n-1}) = \underbrace{\frac{d}{Count(w_{n-1})}}_{\text{normalized discount}} \cdot \underbrace{|\{w' : Count(w_{n-1}w') > 0\}|}_{\substack{\text{\# words that can follow} \\ = \text{\# words that have been discounted} \\ = \text{\# times the normalized discount has been applied}}}$$

53

# Outline

- **Text Classification**
  - **Common Applications**
  - Formal Setup
- **Naive Bayes Classifier**
  - Basic Intuition & BoW Representation
  - Definition & Practical Considerations
  - Complete Runthrough
  - Discussion & Limitations
- **Evaluation of Classifiers**
- **Vector Space Model**
  - Vector Representation of Documents
  - Document Similarity

# 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 document 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

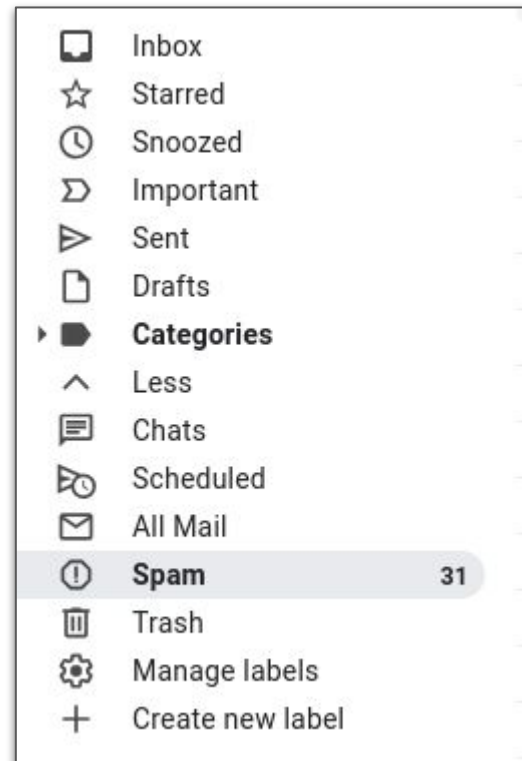
- Identification of the language
  - Relatively straightforward in case of unique alphabets/characters
  - More tricky in case of (closely) related languages

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

## ACM Computing Classification System (very small snippet)

### 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>

### Information systems

#### World Wide Web

Web applications

Internet communications tools

Email

Blogs

Texting

Chat

Web conferencing

Social networks

Crowdsourcing

Answer ranking

Trust

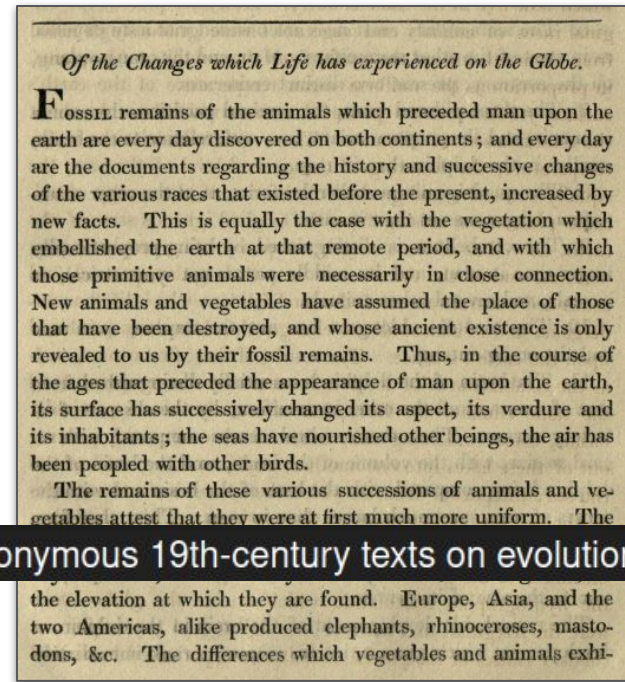
Incentive schemes

Reputation systems

Electronic commerce

# 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.



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.



## Fantastic Stay

"I had a wonderful stay at Tower 1 on the 47th floor as it was for my honeymoon. The view was great as it was facing the city. Great spacious room and the loved the amenities in the room. I would like to give a shoutout to Lifeguard Ryan who made my first trip to the infinity pool memorable for me and my partner. Loved the view from the pool. Also would like to comment on the front office, housekeeping and valet for a job well done. 👍"



However amazing the trickery may be... the characters fall awkwardly into the crack between animal and human, and the plot, which requires them to sing and dance in competition with one another, is scarcely more convincing.

January 3, 2020 | [Full Review...](#)



# Positive or Negative Movie Reviews





# Positive or Negative Movie Reviews





# In-Lecture Activity (5 mins)



# Outline

- **Text Classification**
  - Common Applications
  - **Formal Setup**
- **Naive Bayes Classifier**
  - Basic Intuition & BoW Representation
  - Definition & Practical Considerations
  - Complete Runthrough
  - Discussion & Limitations
- **Evaluation of Classifiers**
- **Vector Space Model**
  - Vector Representation of Documents
  - Document Similarity

# 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  $Y$       $h : X \rightarrow Y$

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

↑  
"True" mapping which  
is unknown in practice

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

**Note 2:** Our SLP3 textbook uses  $d$  for  $x$  and  $c$  for  $y$ . We'll use both interchangeably.

# Text Classification

- Goal of a classification task

- Find the best  $\hat{h}(x)$  to approximate the true mapping  $h(x)$  → But how?

- Two main approaches

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

*IF “good”  $\in x$  OR “great”  $\in x$  OR “nice”  $\in x$  OR ...*

*$h(x) = \text{“positive”}$*

*ELSE IF “bad”  $\in x$  OR “boring”  $\in x$  OR “dumb”  $\in x$  OR ...*

*$h(x) = \text{“negative”}$*

- (2) **Supervised Learning** (machine learning classifiers)

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

# Outline

- Text Classification
  - Common Applications
  - Formal setup
- **Naive Bayes Classifier**
  - **Basic Intuition & BoW Representation**
  - Definition & Practical Considerations
  - Complete Runthrough
  - Discussion & Limitations
- Evaluation of Classifiers
- Vector Space Model
  - Vector Representation of Documents
  - Document Similarity



# 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  $P(y_i|x) \rightarrow y_{NB} = \operatorname{argmax}_{y_i \in Y} P(y_i|x)$
- Calculate  $P(y_i|x)$  using Bayes Rule  $\rightarrow P(y_i|x) = \frac{P(x|y_i)P(y_i)}{P(x)}$

- Example (sentiment analysis)

hopefully :)

↓

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

- Relies on a very simple representation of documents: **Bag-of-Words (BoW)**

# Bag-of-Word (BoW) Representation

- Simplifying assumptions
  - Represent a document as a bag (i.e., multiset) of words  
(i.e., we also keep track of the word counts)
  - Ignore any word order or any other grammar
- BoW representation affected by
  - Tokenization
  - Normalization

} Choice depending on the application/task

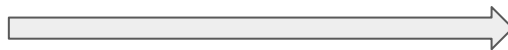
# Bag-of-Words Representation — Example

## Movie review for "Airplane!" (1980)



Nov 03, 2019

"Airplane" is a landmark of American cinema, one of the parents of the subgenre "Besteiorl" and one of the most referenced American comedies of cinema, and opens the door to the 1980s with the golden key the story that tells a love story while a plane crashes and satirizes various Hollywood classics, from "Shark" to "The Wind Gone", with a sour, black and very caricature humor, but that works to this day, with low budget, using a lot of mockup and ready-made scenarios, the Classic entertains generations even today with its simple, bold and silly humor. The script is simple and cliché, which is part of the joke, full of hype and nonsense with a good direction that goes beyond just driving the film, it serves as a pillar for several jokes that is based on other Hollywood movies, this trail voices to performances, not all jokes are explicit, but everything in "Airplane" is a joke. Important movie for those who like cinema to fish a little of the references and cinematic background of the time, besides being a film that contributes a lot to the comedy genre in Hollywood, the film has aged well. Note 7



## Normalization steps:

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





# Quick Quiz



# Outline

- Text Classification
  - Common Applications
  - Formal setup
- Naive Bayes Classifier
  - Basic Intuition & BoW Representation
  - **Definition & Practical Considerations**
  - Complete Runthrough
  - Discussion & Limitations
- Evaluation of Classifiers
- Vector Space Model
  - Vector Representation of Documents
  - Document Similarity

# Naive Bayes Classifier — Annotated

- Basic setup

- Document  $x \in X$  with  $x = w_1, w_2, \dots, w_n$  (BoW representation)
- Class label  $y \in Y$

The diagram illustrates the Naive Bayes formula with four annotated components:

- Likelihood:** Probability of  $x$  given that it belongs to class  $y$ . This points to the numerator's first term,  $P(w_1, w_2, \dots, w_n | y)$ .
- Prior:** Probability that  $x$  belongs to class  $y$  without seeing any data. This points to the numerator's second term,  $P(y)$ .
- Posterior:** Probability of class  $y$  given document  $x$ . This points to the entire left side of the equation,  $P(y | w_1, w_2, \dots, w_n)$ .
- Marginal:** Probability of  $x$  under any class. This points to the denominator,  $P(w_1, w_2, \dots, w_n)$ .

$$P(y | w_1, w_2, \dots, w_n) = \frac{P(w_1, w_2, \dots, w_n | y) P(y)}{P(w_1, w_2, \dots, w_n)}$$

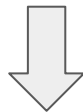
# Naive Bayes Classifier

- 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  $P(y_j|x)$

$$\frac{P(w_1, w_2, \dots, w_n|y_i)P(y_i)}{P(w_1, w_2, \dots, w_n)} \stackrel{?}{\geq} \frac{P(w_1, w_2, \dots, w_n|y_j)P(y_j)}{P(w_1, w_2, \dots, w_n)} \stackrel{?}{<}$$

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



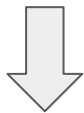
$$P(y|w_1, w_2, \dots, w_n) \propto P(w_1, w_2, \dots, w_n|y)P(y)$$

# Naive Bayes Classifier — The “Naive” Part

- Simplifying assumption

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

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



“Naive” assumption

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

How to calculate  
these probabilities?



# Naive Bayes Classifier — Maximum Likelihood Estimates

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

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

# documents of class  $y$

# documents (total)

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

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

# occurrences of  $w_i$  in documents of class  $y$

# words (total) in documents of class  $y$

Does this look familiar?

# Naive Bayes Classifier — Practical Considerations

- Risk of arithmetic underflow → Calculate log probabilities

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

- Out-of-vocabulary (OOV) words + unrepresented classes

- Unseen words  $w_i$  during test/prediction time →  $\text{Count}(w_i, y) = 0$  →  $P(w_i|y) = 0$
- No document of class  $y$  →  $P(y) = 0$

e.g.: Add-k Smoothing:

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

# Deriving the NB Classifier in One Slide

$$y_{MAP} = \arg \max_{y \in \mathcal{Y}} P(y|x)$$

Most likely class (*Maximum a priori*)

$$= \arg \max_{y \in \mathcal{Y}} \frac{P(x|y) \cdot P(y)}{P(x)}$$

Bayes Rule

$$= \arg \max_{y \in \mathcal{Y}} P(x|y) \cdot P(y)$$

Dropping the prior  $P(\mathbf{w})$  in the denominator

$$= \arg \max_{y \in \mathcal{Y}} P(w_1, \dots, w_n|y) \cdot P(y)$$

Doc represented as words  $\mathbf{w}_1, \dots, \mathbf{w}_m$   
(such as word counts) using BoW  
assumption

$$= \arg \max_{y \in \mathcal{Y}} P(w_1|y) \cdot \dots \cdot P(w_n|y) \cdot P(y)$$

Independence Assumption

$$= \arg \max_{y \in \mathcal{Y}} \prod_{i=1}^N P(w_i|y) \cdot P(y)$$

Equation for Naive Bayes

# NB Algorithm Summary

**TrainNaiveBayes**( $D, \mathcal{Y}$ ) returns  $\log P(y)$  and  $\log P(w|y)$  :

$N \leftarrow |D|$

**for each class**  $y \in \mathcal{Y}$  : // calc prior terms

$N[y] \leftarrow D_y$

$\logprior[y] \leftarrow \log(|N[y]|/N)$

$V \leftarrow$  vocabulary of  $D$

$bigdoc[y] \leftarrow \text{append}(d)$  **forall**  $d \in N[y]$

**for each word**  $w \in V$  : // calc likelihood terms

$c(w, y) \leftarrow \#$  of occurrences of  $w$  in  $bigdoc[y]$

$\loglikelihood[w, y] \leftarrow \log \frac{c(w, y) + 1}{\sum_{w' \in V} (c(w', y) + 1)}$

**return**  $\logprior, \loglikelihood, V$

**TestNaiveBayes**( $x, \logprior, \loglikelihood, \mathcal{Y}, V$ )

**returns**  $y$  :

**for each class**  $y \in \mathcal{Y}$  :

$sum[y] \leftarrow \logprior[y]$

**for each position**  $i$  in  $x$  :

$w \leftarrow x_i$

**if**  $w \in V$  :

$sum[y] += \loglikelihood[w, c]$

**return**  $\operatorname{argmax}_y(sum[y])$

# Outline

- Text Classification
  - Common Applications
  - Formal setup
- Naive Bayes Classifier
  - Basic Intuition & BoW Representation
  - Definition & Practical Considerations
  - **Complete Runthrough**
  - Discussion & Limitations
- Evaluation of Classifiers
- Vector Space Model
  - Vector Representation of Documents
  - Document Similarity

# Naive Bayes Classifier — Runthrough

- 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
<i>very good and funny movie!</i>	pos
<i>what a funny cast!</i>	pos
<i>a very boring movie and boring cast</i>	neg
<i>very boring cast!</i>	neg
<i>such a funny movie!</i>	pos
<i>really good cast, really good movie</i>	pos
<i>boring...such a boring movie!!!</i>	neg

# Naive Bayes Classifier — Runthrough

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

- Number of reviews  $N = 7$
- Number of positive reviews  $N_{pos} = 4$
- Number of negative reviews  $N_{neg} = 3$

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

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

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

# Naive Bayes Classifier — Runthrough

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

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

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

...

$w_i$	$P(w_i \text{pos})$	$P(w_i \text{neg})$
<i>funny</i>	4/16	1/14
<i>boring</i>	1/16	6/14
<i>movie</i>	4/16	3/14
<i>cast</i>	3/16	3/14
<i>good</i>	4/16	1/14

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





# Naive Bayes Classifier — Inference

- Predict the class for a new review

Review	Class
<i>a funny movie and cast</i>	???

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

$w_i$	$P(w_i pos)$	$P(w_i neg)$
<i>funny</i>	4/16	1/14
<i>boring</i>	1/16	6/14
<i>movie</i>	4/16	3/14
<i>cast</i>	3/16	3/14
<i>good</i>	4/16	1/14

→ Label review with ?



# Naive Bayes Classifier — Inference

- Predict the class for a new review

Review	Class
<i>a funny movie and cast</i>	???

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

$w_i$	$P(w_i pos)$	$P(w_i neg)$
<i>funny</i>	4/16	1/14
<i>boring</i>	1/16	6/14
<i>movie</i>	4/16	3/14
<i>cast</i>	3/16	3/14
<i>good</i>	4/16	1/14





# How does NB compare with LM? (5 mins)



# Outline

- Text Classification
  - Common Applications
  - Formal setup
- Naive Bayes Classifier
  - Basic Intuition & BoW Representation
  - Definition & Practical Considerations
  - Complete Runthrough
  - **Discussion & Limitations**
- Evaluation of Classifiers
- Vector Space Model
  - Vector Representation of Documents
  - Document Similarity

# Naive Bayes Classifier + BoW — 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 is an LM! It 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 + BoW — Limitations

- Example: Sentiment Analysis

- BoW incapable to handle some relevant linguistic phenomena
- Most prominently: **negation** (typically flips the sentiment)

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

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

- 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 it often works well)

*"the movie is not very funny."* → *"the movie is not NOT\_very NOT\_funny."*



**To think about:** Where would this simple heuristic fail? Examples?

# Naive Bayes Classifier + BoW — 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



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, but more prominent due to NB's BoW approach

# 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 (at least somewhat) holds
  - Maybe some tweaks possible address this issue (e.g., negation handling)



# Break

# Outline

- Text Classification
  - Common Applications
  - Formal setup
- Naive Bayes Classifier
  - Basic Intuition & BoW Representation
  - Definition & Practical Considerations
  - Complete Runthrough
  - Discussion & Limitations
- **Evaluation of Classifiers**
- Vector Space Model
  - Motivation & Goals
  - Vector Representation of Documents
  - Document Similarity

# Pre-Lecture Activity from Last Week

- Assigned Task

- Post a 1–2 sentence answer to the following question into the Pre-Lecture Discussion  
(you will find the thread on Canvas > Discussions)

*"When we want to evaluate classifiers,  
why is **accuracy** alone often not a good metric?"*

**Side notes:**

- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers, but this won't help you learn better



# Evaluating Classifiers — Error Types

- Recall from Week 02: 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 label →

**True Positives**

The classifier incorrectly fails to predict the label →

**True Negatives**

# Classification: Evaluation — Confusion Matrix

		actual labels	
predicted labels		1	0
	1		
	0		

- 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 — Confusion Matrix

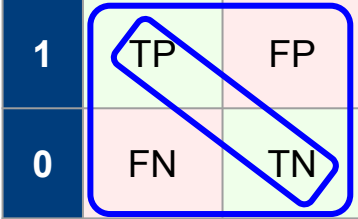


# Classification: Evaluation — Popular Metrics

- Accuracy

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

		actual labels	
		1	0
predicted labels	1	TP	FP
	0	FN	TN





# Classification: Evaluation — Popular Metrics

- Precision, Recall, F1 Score

Harmonic Mean of  
Precision and Recall

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

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

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

		actual labels	
		1	0
predicted labels	1	TP	FP
	0	FN	TN

		actual labels	
		1	0
predicted labels	1	TP	FP
	0	FN	TN

		actual labels	
		1	0
predicted labels	1	TP	FP
	0	FN	TN

# Which is more important?

Precision or Recall?



# Classification: Evaluation — Why so Many Measures?

- Observation: FP and FN not always equally problematic

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

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

- Example: COVID 19 pre-vaccine

(e.g., from social media content posted by users)

- BAD: misclassifying a high-risk person
- OK-ish: misclassifying a healthy person



**Recall > Precision**

- Example: Web Search Results

(e.g., for search engines such as Google)

- BAD: showing an irrelevant result
- OK: missing a relevant article in result



**Recall < Precision**

# Classification: Evaluation — Why so Many Measures?

- Problem: (Highly) imbalanced datasets
- Example use case: COVID-19 test (binary “classifier”)
  - Most people in a population are not infected
  - Assume a test that always(!) returns “negative”

		actual labels	
		1	0
predicted labels	1	0	0
	0	200	10,000

$$Accuracy = \frac{0 + 10000}{0 + 0 + 10000 + 200} = 98\%$$

→ Very high accuracy despite “useless” test



# Why Harmonic? (5 mins)





# Why Harmonic?



# Classification: Evaluation — Beyond 2 Classes

- Example: 3 classes, 50 samples

actual labels

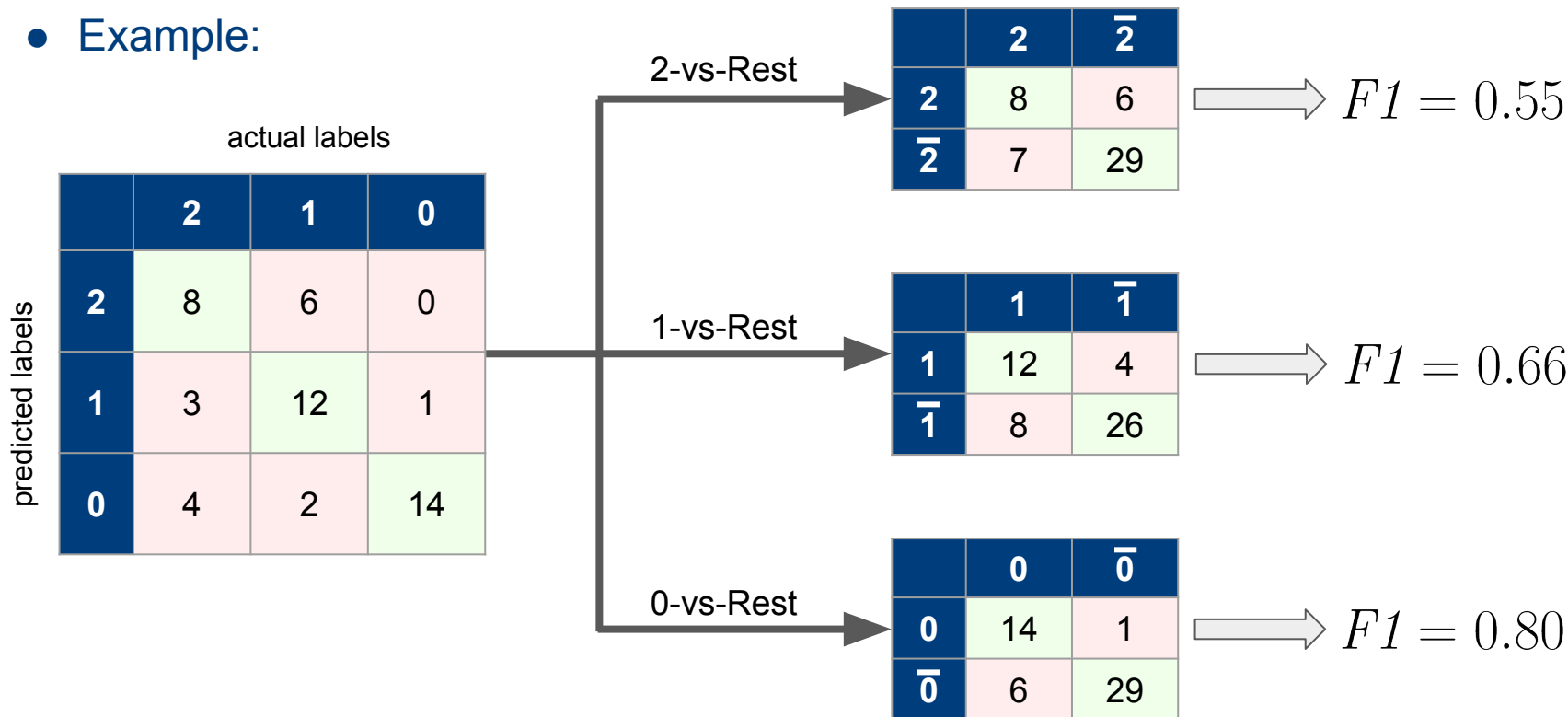
	2	1	0
2	8	6	0
1	3	12	1
0	4	2	14

predicted labels

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

# Multiclass Evaluation — One-vs-Rest Confusion Matrices

- Example:





# One-vs-Rest — Micro Averaging

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

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

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

Average over all  
TP, FP, FN, TN



	c	$\bar{c}$
c	11.33	3.66
$\bar{c}$	7	28



$$F1 = 0.68$$

# One-vs-Rest — Macro Averaging

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

$\Rightarrow F1 = 0.55$

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

$\Rightarrow F1 = 0.66$

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

$\Rightarrow F1 = 0.80$

Average over  
all metrics

$\Rightarrow F1 = 0.67$

# 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)



# Micro / Macro 1





## Micro / Macro 2

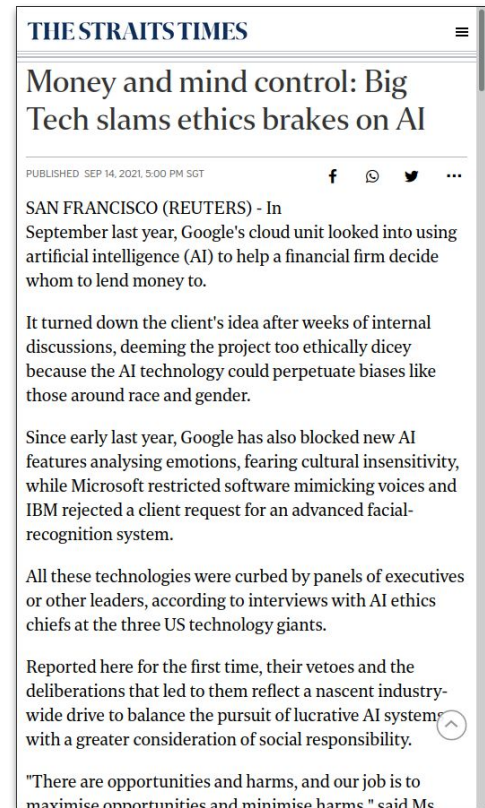


# Outline

- Text Classification
  - Common Applications
  - Formal setup
- Naive Bayes Classifier
  - Basic Intuition & BoW Representation
  - Definition & Practical Considerations
  - Complete Runthrough
  - Discussion & Limitations
- Evaluation of Classifiers
- **Vector Space Model**
  - **Vector Representation of Documents**
  - Document Similarity

# Vector Space Model — Motivation

- Most algorithms do not work on raw text  
— common requirements
  - Numerical input
  - Standardized/canonical input
- Feature extraction → vectorization of text data
  - Represent each text document as a vector of equal size
  - Vector elements = numerical values derived from text



???



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

# “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



# Vector Space Model

- Idea: Vectorize documents based on vocabulary (→ BoW representation)

- Length each document vector is the size of corpus vocabulary  $V$
- Vectors for all documents in dataset  $D$  form the document-term matrix

- Document-term matrix

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

→ 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

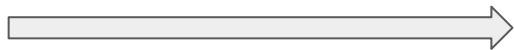
$d_1$ : *Dogs chase cats and other dogs.*

$d_2$ : *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

$d_1$ : *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  $V = \{car, cat, chase, dog, sit, tv, watch\}$

# Document–Term Matrix with Binary 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 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”)

	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
<b>car</b>	0	0	1	0	1
<b>cat</b>	1	1	0	0	1
<b>chase</b>	1	1	1	0	0
<b>dog</b>	1	0	0	1	1
<b>sit</b>	0	0	0	0	1
<b>tv</b>	0	0	1	1	0
<b>watch</b>	0	0	0	1	0

# Document–Term 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

- $w_{t,d}$  : # occurrences 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”?

	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
<i>car</i>	0	0	1	0	1
<i>cat</i>	1	2	0	0	1
<i>chase</i>	1	1	1	0	0
<i>dog</i>	2	0	0	2	1
<i>sit</i>	0	0	0	0	1
<i>tv</i>	0	0	1	1	0
<i>watch</i>	0	0	0	1	0

# $tf_{t,d}$ 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

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

But is  $d_1$  really 10x more important than  $d_2$ ?

→ 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  $df_t$ : # documents 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

# Document-Term Matrix with $tf$ - $idf$ Weights

- Putting it all together

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

- Side notes

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



# Document-Term 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

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

- Example

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

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

# Document-Term Matrix with $tf-idf$ Weights

- Matrix elements =  $tf-idf$  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

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

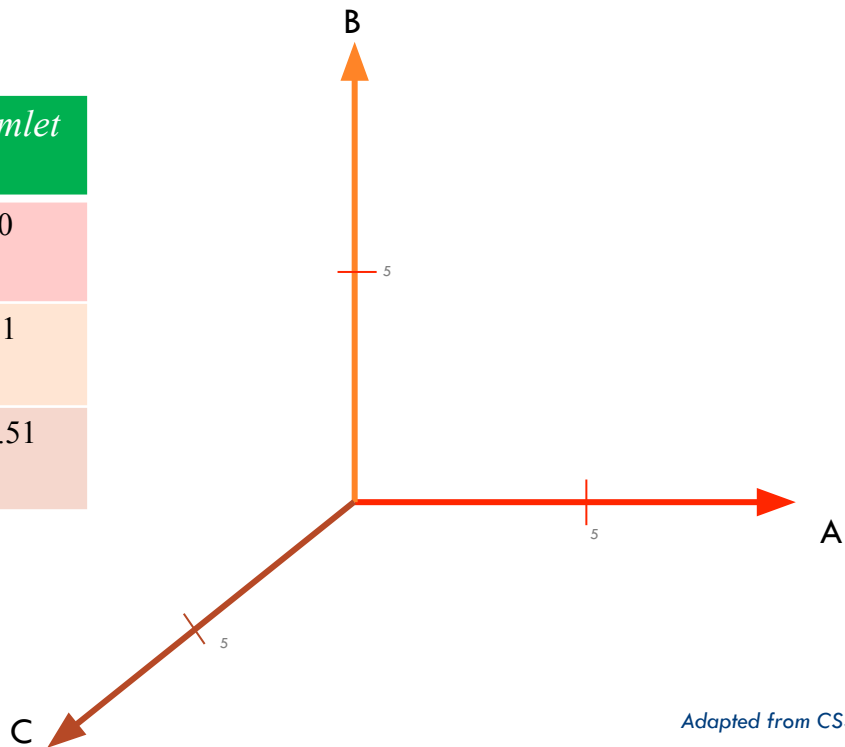
	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
<b>car</b>	0	0	0.4	0	0.4
<b>cat</b>	0.22	0.29	0	0	0.22
<b>chase</b>	0.22	0.22	0.22	0	0
<b>dog</b>	0.29	0	0	0.29	0.22
<b>sit</b>	0	0	0	0	0.7
<b>tv</b>	0	0	0.4	0.4	0
<b>watch</b>	0	0	0	0.7	0

# Outline

- Text Classification
  - Common Applications
  - Formal setup
- Naive Bayes Classifier
  - Basic Intuition & BoW Representation
  - Definition & Practical Considerations
  - Complete Runthrough
  - Discussion & Limitations
- Evaluation of Classifiers
- **Vector Space Model**
  - Vector Representation of Documents
  - **Document Similarity**

# Vector Space

	<i>Antony &amp; Cleopatra</i>	<i>Julius Caesar</i>	<i>Hamlet</i>
Antony	5.25	3.18	0
Brutus	1.21	6.1	1
Caesar	8.59	2.54	1.51



Adapted from CS3245

# Vector Space



# Vector Space



# 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

$$\text{dot}(v, w) = v \cdot w = v_1w_1 + v_2w_2 + \dots v_nw_n = \sum_{i=1}^n v_iw_i$$

- Interpretation

- $\text{dot}(v, w)$  is high if  $v$  and  $w$  have large values in the same dimensions

→  $\text{dot}(v, w)$  represents a similarity metric between vectors, but...



# Document Similarity

- Limitations of Dot Product

- $\text{dot}(v, w)$  is higher if a vector has higher values in many dimensions

→  $\text{dot}(v, w)$  favors long vectors

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

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

- Effects in document vectors

- $\text{dot}(v, w)$  favors frequent words  
(since they occur many times with other documents)
- $\text{dot}(v, w)$  favors long documents  
(since the raw term frequencies are higher)



→  $\text{dot}(v, w)$  overly favors frequent words

# Document Similarity — Cosine Similarity

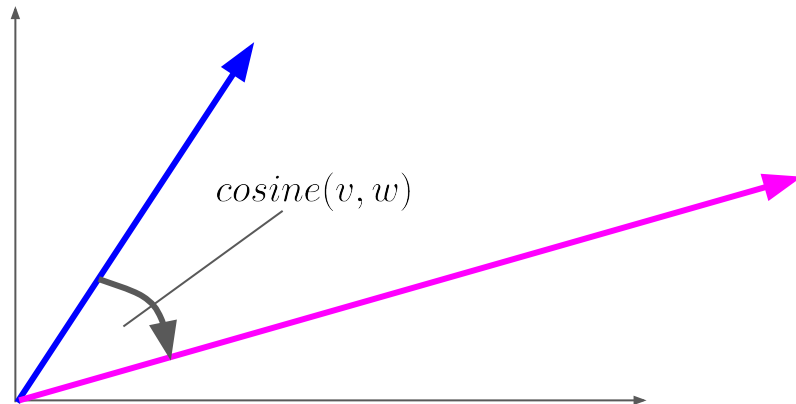
- Approach 2: Cosine Similarity (dot product normalized by length of vectors)

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

- Geometric interpretation

- $\text{cosine}(v, w)$  measures the angle between vectors

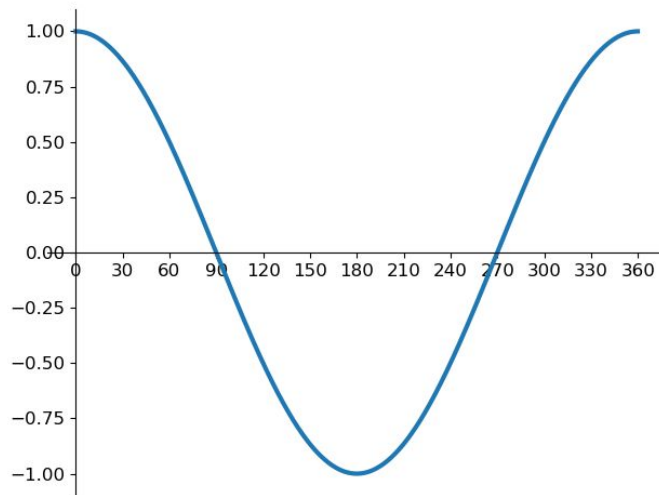
$\text{dot}(v, w)$  cares about angle and length



# Document Similarity — Cosine Similarity

- Cosine as a similarity metric

- $\text{cosine}(v, w) = -1$   
vectors point in opposite directions
- $\text{cosine}(v, w) = 1$   
vectors point in the same direction
- $\text{cosine}(v, w) = 0$   
vectors are orthogonal



- Cosine similarity for document vectors

- Vector entries are all positive
- $0 \leq \text{cosine}(u, v) \leq 1$

# 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

	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
<b>car</b>	0	0	0.4	0	0.4
<b>cat</b>	0.22	0.29	0	0	0.22
<b>chase</b>	0.22	0.22	0.22	0	0
<b>dog</b>	0.29	0	0	0.29	0.22
<b>sit</b>	0	0	0	0	0.7
<b>tv</b>	0	0	0.4	0.4	0
<b>watch</b>	0	0	0	0.7	0

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

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

(only non-zero components included)

# Document Similarity — Cosine Similarity

$d_1$ : dog chase cat dog

$d_2$ : cat chase cat

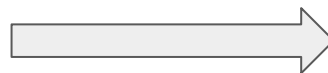
$d_3$ : car chase tv

$d_4$ : dog watch dog tv

$d_5$ : dog cat sit car

	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
<i>car</i>	0	0	0.4	0	0.4
<i>cat</i>	0.22	0.29	0	0	0.22
<i>chase</i>	0.22	0.22	0.22	0	0
<i>dog</i>	0.29	0	0	0.29	0.22
<i>sit</i>	0	0	0	0	0.7
<i>tv</i>	0	0	0.4	0.4	0
<i>watch</i>	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

# Summary

- Text Classification

- Very fundamental NLP task  
(very fundamental machine learning task, in general)
- Supervised machine learning task → we need training data

- Baseline classifier: Naive Bayes

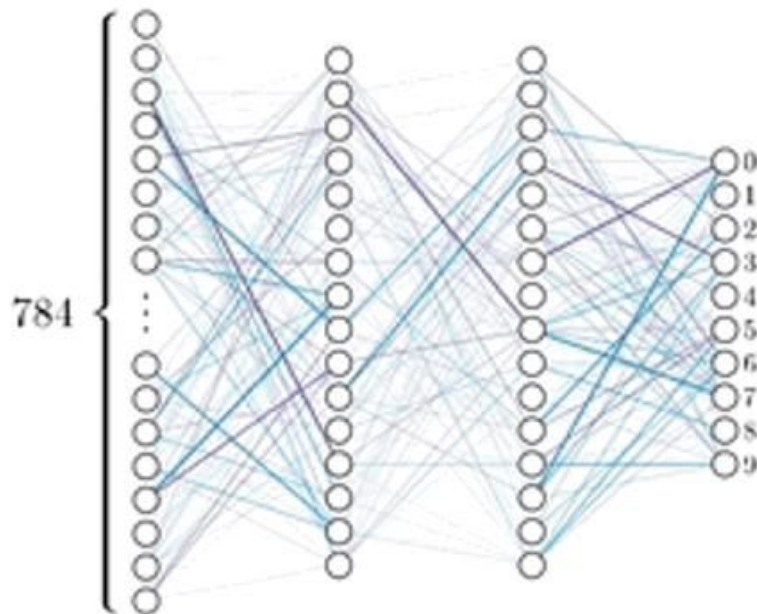
- Very simple classifier related to language models → works directly over words
- Relies on Bag-of-Word Representation of documents (incl. its limitations)

- Vector Space Model

- Derive meaningful vector representation of documents from their vocabulary
- Definition of meaningful similarity between documents → import for many NLP tasks

# Outlook for Next Week: Connectionist ML

Training in  
progress. . .





# Pre-Lecture Activity for Next Week

- **Assigned Task** (due before Feb 16)
  - Post a 1–2 sentence answer to the following question into the Pre-Lecture Discussion (you will find the thread on Canvas > Discussions)

*“What is a common myth about neural networks?”*

Read some blog posts or online articles, and cite them with the links in your answer

**Side notes:**

- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers but this won't help you learn better