Lec 1

NLP — Natural Language Processing

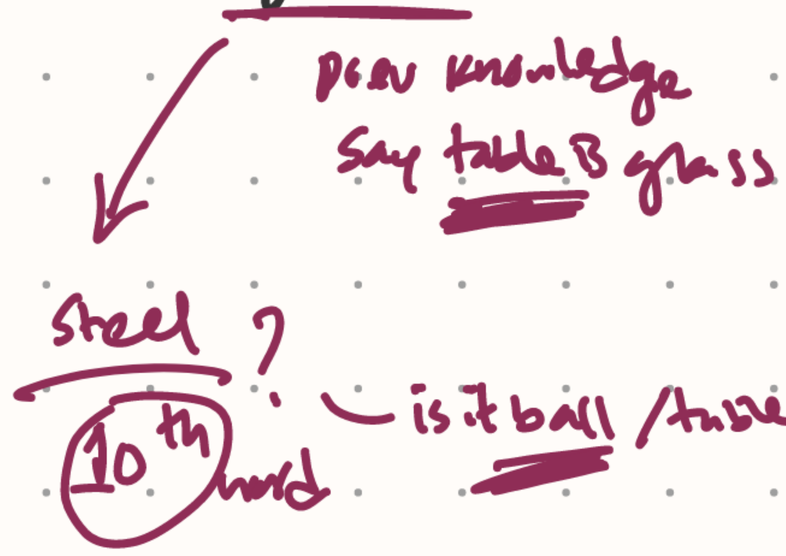
Natural lang — related by humans

Software controls Hardware

Natural lang. defines how humans perceive things.

the ball smashed through the table,

because it was glass



CV uses pattern recognition

But NLP B diff.

Images are same, lang diff (Korean, Chinese, English)

Text Representation & classification

1. NLP Fundamentals
   → Preprocessing techniques

2. N-gram models
   → Smoothing techniques
   (HLI)

3. ML models /techniques
   (to classify texts)
   → Naive Bayes
   → Sentiment Classification
   → Logistic Regression

4. Vector Semantics,
   Embeddings,
   Neural language models
   → Numerically represent lang.

Natural Language understanding

(Parts of speech)
1. POS Tagging,
   Sequence Processing;
   RNNs
   → time series/sequence data

   → Markov model

2. Encoders-Decoders,
   Attention,
   Contextual Embeddings

3. ~~Constituency~~ Parsing
   → Advanced way of representing lang.
   (ToC)

4. Dependency Parsing,
   Logical Representation

Discourse & Dialogue:

1. Information Extraction, word sense, wordnet

2. Question-Answering, Dialogue Systems, Chatbots

Project

- Define the Research Question
  - Introduction
- Collect the Datasets
  - from Prev Research
  - Scraping
- Do experiments & compare w/ baselines
- Prepare the report

Zipf's law : frequency $\propto \frac{1}{\text{rank}}$

↓ word → most used ranking (the in 1)

Properties of language

Regular expression

Text Normalization

Edit Distance

3000 words will cover 95% of everyday writing - Robert Charles

Synsets

Stop words : Removing them, meaning remains same

Why regular expression ?

(

String search in Google doc - Primitive string result ✗

b : start

n : end

: String search ✓ → using regular exp.

re python library : regular exp.

>> import re

re.findall (regex, text) → find bubblegum

re.sub(regex, repl, text) → Replace (search substring)

s = "Jack is a boy"

Index of a

/a/_a

re.search("(^|\s)a", s)

↳ /a/_a

whitespace

re.search('a', s).span() → Return index of 'a' (only 1 instance)

findall + search → Return all indexes of 'a' (all instances)

[{"indices : m.span()" for m in (re.finditer('a', s))]

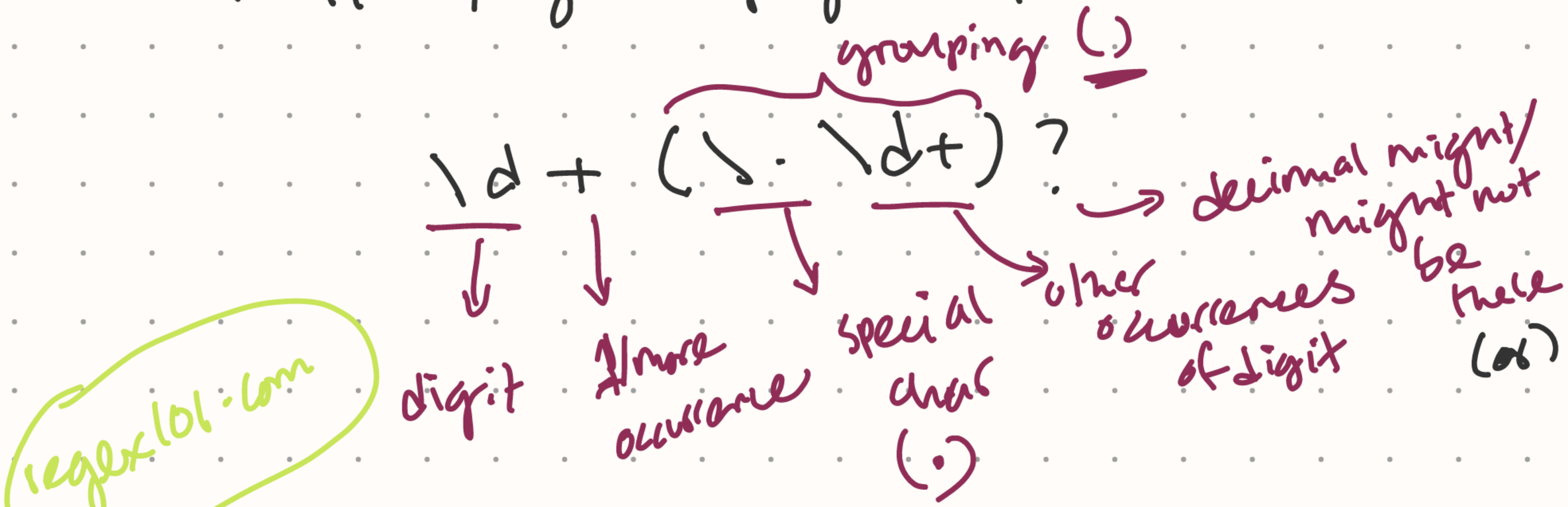
\d → any digit

\d+

metacharacters → For this reason regular exp is used instead of string search

s = "Price? 45 or 55.33?"

To specify any digit using regular exp → \d

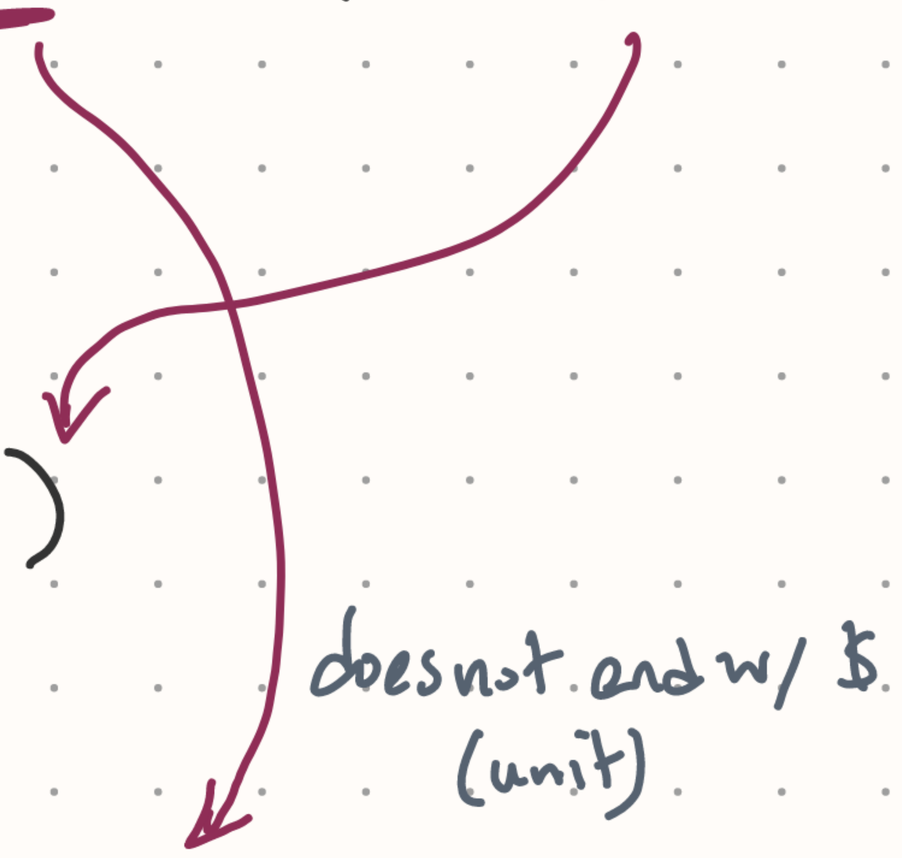


re.findall("\d+(\.\d+)?", s) → Find all numbers from string s → 45 55.33

s = "Price? The price of 1 turkey is 30$."

only product prices → $

(\d+(\.\d+)?(?=\$))



Positive Lookahead (30$)

Negative Lookahead

Positive Lookbehind ($30)

Negative Lookbehind

Negative Lookahead → Do not want sth that ends w/ sth ($)

Regular exp. (cont.)

1 turkey costs 30 $ → Positive lookahead

Is there a digit followed by a $

\d+ (?=\$)

30

Positive lookahead

Is there a digit not followed by $

\d+ (?!\$) → Negative lookahead

1

zero/none

(?<=\$)\d+ → positive lookbehind

(?<!\$)\d+ → Negative lookbehind

Text Normalization

A set of collective tasks
Primary pre-processing tasking before feeding to model

1. Tokenization
2. Normalizing word formats
3. Segmenting sentences

1 " The U.S. is a big nation.

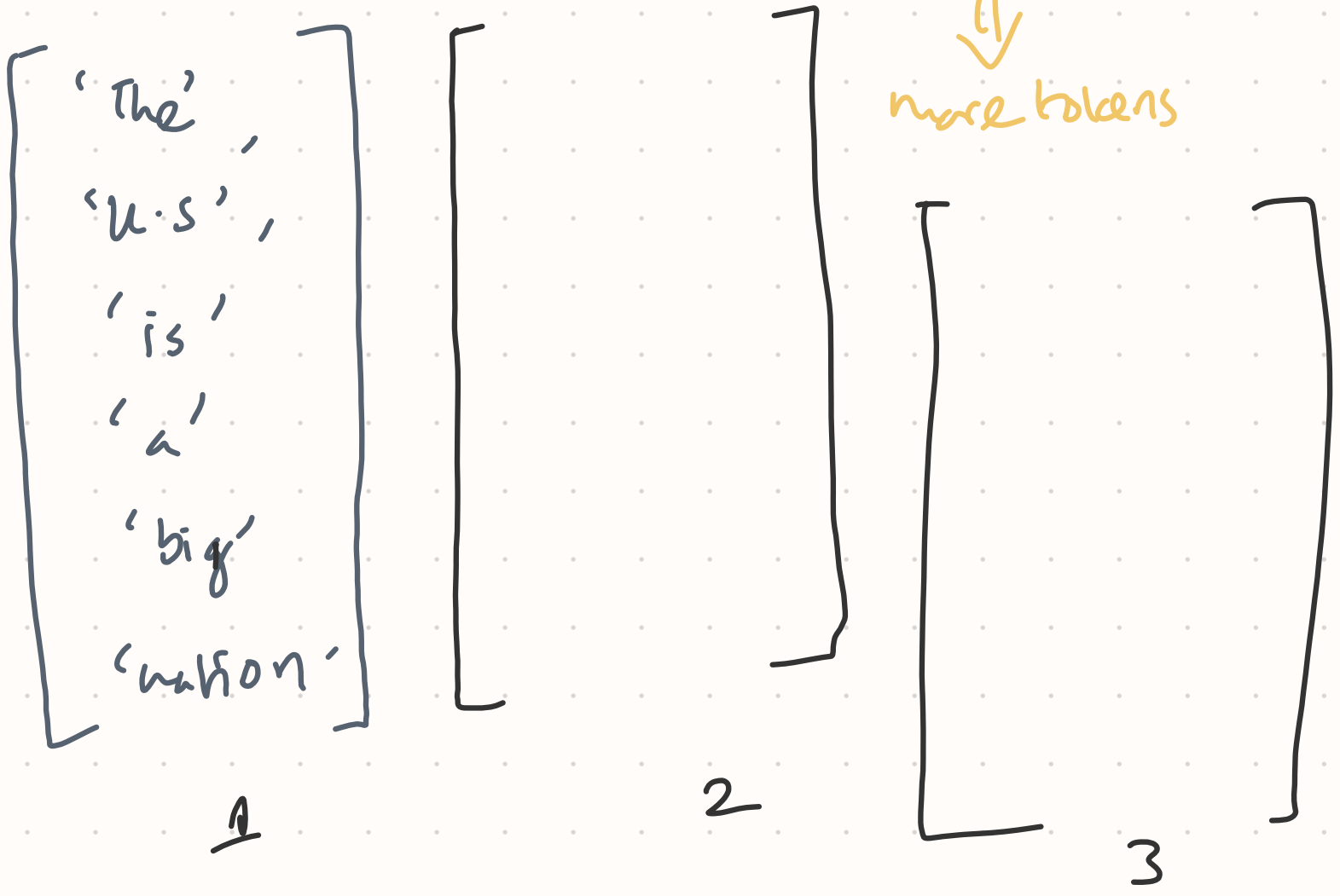
2 The U.S.A poster-print costs 30 $.

3 The driving speed unit here is m.p.h, not km. "

Embedded vectors

more words

more tokens



Tokenization : segmenting a running text into words (tokens)

Represent tokens (words) as numbers

3 diff size vectors

Rule based tokenization — prev.

Each sentence end w/ . → for 1st sentence you get 3 diff sentences

abbreviations periods] .

white space split →

↙ prob

Penn Treebank standard (PTS) → not all . are sentence breakers

↙ soln

post-print — 1 single token

'30', '5'

PTS + regex

Based on tokenization, it depends how effective embedding will be.

Before bert/transformer based models — RNN earlier

bert-based model — embedded vector

Power of transformer / pretraining

Knows

which words are similar

king → queen → cluster1
student → teacher → cluster2
country → patriot

} cluster of tokens

corpus

'nation' : similar vector value like country (Embedding vector)

RNN doesn't have this info only bert has

bert tokenizer / bert embedding

better representation of words ✓ → optimizing parameters

automatic token parsing techniques: (Tokenization)

1. Byte pair Encoding (BPE)
2. unigram language modeling (ULM)
3. wordpiece → bert

} 2 parts:
- token learner
- token segmenter

[sentence piece : BPE + ULM]

Corpus → { low, new, newer, wider, lowest }

PTS fails to understand what to do w/ unknown words.

BPE: Corpus – 18 tokens

Divide all words based on whitespace and word frequency

all unique chars ← vocabulary

count	corpus	vocabulary
5	low_ → whitespace	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_
2	lowest_	
6	newer_	
3	wider_	
2	new_	

morphemes

Task: create a vocab

List most frequent pair of symbols together

er = 6 + 3 = 9

n e w er_

w i d er_

n e w er_

w i d er_

(Learns an unseen data.
new word:
lower rule
low, er_
present in learner vocab
low er_
diff token (merged)

morphemes: smallest unit of

BPE → lower < low, er > } 2 tokens

Subwording: Dividing the tokens into smaller parts

Subword = smartphone

ne w er_

ne w _

BPE: K parameter ⇒ how many cycles you want to run

BPE(corpus, K)

9

Normalizing:

am is are — same word containing same meaning

case folding:

(

lower case everything

Woodhouse
woodhouse
} same value for embedding vector

→ Info retrieval ✓

Text/sentiment classification
+ machine translation

(

not doing lower casing

US → country
us → we

Lemmatization: Determine which words are the same root

addition
additional
} same root

I have some additional information.

I can do some addition on information.

same

words (additional) ⟨ stems (addition)
                    ⟨ Affixes (al)

replace each word w/ roots → Lemmatization ✓ (better)
using pre existing dictionary
sings, sang, sung

Do Lemmatization before feeding corpus to language model.

Stemming : Remove affixes from words
✗

~~additional~~

Changed actual meaning

(doing → doe) → error prone (stemming)

(doing → do) → not error prone (Lemmatization)

happiness → happy

"ness" "ness"

Lemmatization over stemming ✓

Term Paper

Performance Review Research

Language model optimized with least amount of data
(child language acquisition)

Bert, Roberta — in traditional sense, they are not LLMs
(at least a billion parameter)

Chatgpt - General task solvers

Benchmark of chatgpt in summarization datasets

General task solvers are better than specific task solvers.

Performance of LLM in low research language.

only good at previously seen data

Task contamination

SOTA

Class Notes

Edit distance : way to quantify string similarly

spell mistake: graffe

giraffe graf grail | spell checking

Harvard President Claudine

Harvard university President Claudine | coreference

↓

if two strings contain same entity

Define cost — LEVENSHTEIN

The apple was red

1. The apple was ~~big~~ ~~and~~ red. → 4$ (2$ 2$)
2. The ~~fish~~ was ~~blue~~. → 8$ (2$ 2$ 2$ 2$)

task	cost
i insertion	2$
d deletion	2$
s substitution	

1st was more similar to the string.

↳ minimum Edit distance (check least similarity)

INTENTION → EXECUTION

d s s - i s (cost=8)

INTENTION

EXECUTION

min Edit Distance ?

↓ can start from anywhere

i=1$
d=1$
s=d+i

use DP (Dynamic Programming)

- search problem

Find most cost efficient approach

Edit Distance Algo

✓1. Levenshtein
✓2. Hamming Distance
3. Jaro-winkler

Token-based

✓1. Cosine-similarity (similarity in text/sent)
✓2. Jaccard index
3. Tversky Index

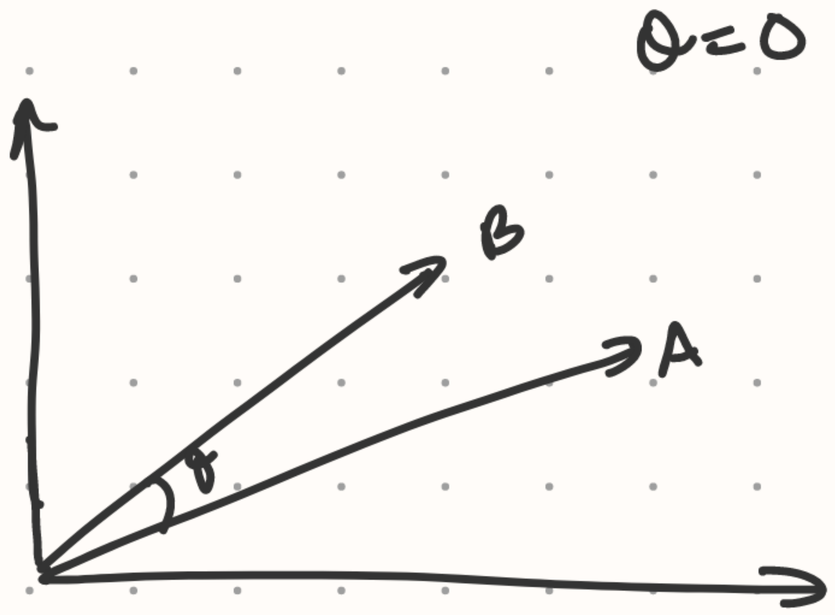
Sequence-based (programming)

1. LCS (Longest common substring)
2. Ratcliff-Obershelp similarity

Cosine-similarity

converts the whole corpus into same dim. vectors and projects it.

similar sentence → similar vectors
more similar → less distance
2 same sentence → project on each other



cosine similarity (doc1, doc2)

Document 1: The cat in the hat.

Document 2: A black cat in a black hat.

DOC → vectors : Find all unique tokens from the docs.

	The	cat	in	hat	a	black
Doc1	2	1	1	1	0	0
Doc2	0	1	1	1	2	2

Frequency

$$A = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 2 \\ 2 \end{bmatrix}$$

cosine similarity (doc1, doc2)

$$= \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{2.0 + 1.1 + 1.1 + 1.1 + 0.2 + 0.2}{\sqrt{7} \cdot \sqrt{11}}$$

$$= 0.342$$

similar → close to 1
diff → close to -1
⊥ vectors → close to 0
(no similarity / dissimilarity)

— completely diff set of words (0 frequency)

There should be a (-ve) value theoretically.

Insight

1) The patient died before he doc came.
2) the doc came after the patient died.

cosine similarity cannot detect semantic meaning of sentence.

(cons)

computationally strong (Pros)

- sensitive to sentence long. long vector (not necessary) | cons

sparsity — having a lot of 0s in vectors/matrix

cosine similarity create sparse vectors that will dominate the entire value

Jaccard Index is more sensitive to context in finding text similarity.

$$J.I = \frac{A \cap B}{A \cup B}$$

Jaccard similarity (doc1, doc2) = $\frac{doc1 \cap doc2}{doc1 \cup doc2}$

if 2 docs talking about same context

docs → tokens

J.S = $\frac{\{\text{'The'}, \text{'cat'}, \text{'in'}, \text{'the'}, \text{'hat'}\} \cap \{\text{'A'}, \text{'black'}, \text{'cat'}, \text{'in'}, \text{'a'}, \text{'black'}, \text{'hat'}\}}{\cup}$

$= \frac{3}{\{\text{'the'}, \text{'cat'}, \text{'in'}, \text{'hat'}, \text{'a'}, \text{'black'}\}}$

$= \frac{3}{6} \approx 50\%$ similarity.

Imp of tokenization/ lemmatization.

throw, throws, throwing → W/O lemmatization, consider them diff.
(verb terms)

N-gram language models

↳ lang models that assign probabilities to N-grams

Please turn in your homework.

word bigrams (2)

Please turn
turn in
in your
your homework

word trigrams (3)

please turn in
turn in your
in your homework

character bigrams

Pl, le, ea ...

character trigrams

Ple, lea, eas, ase ...

skip grams: Nm continuous

Skip bi grams

please in
turn your
in homework

} skip 1 element

Language models : Probabilistic model

Assign probabilities to lang units (sequence of words)

Chatgpt learnt probabilities of words
Assigns probabilities to the possible ans.
Highest probability ans chosen. ✓

Bigram model : Trained w/ simple bigrams
Probability of next word to occur

$P(W_n | W_{n-1})$ → bigram Markov model (Not look too far into past)
Ans closest to you.

$P(\text{turn} | \text{please}) = ?$ → Probability that turn will come after please

$P(W_n | W_{n-1} W_{n-2})$ → trigram markov model

$$P(W_n \mid W_{n-1}) = \frac{C(W_{n-1} W_n)}{C(W_{n-1})}$$

$C(W_{n-1} W_n)$ → prev word occurrence w/ this word

$C(W_{n-1})$ → occurrence of prev word

↳ MLE (maximum Likelihood Estimation)

conditional probability eqn

$P(a|b) = P$

<s> I am Abrar </s> → end of string/whitespace
<s> Abrar I am </s>
<s> I do not like eggs.. </s>

↑ start of string

need to define <s> and </s> to find occurrences.

$$P(I | \langle s \rangle) = \frac{C(\langle s \rangle I)}{C(\langle s \rangle)} = \frac{2}{3}$$

$$P(am \mid I) = \frac{C(I\ am)}{C(I)} = \frac{2}{3}$$

$$P(\langle /s \rangle \mid Abrar) = \frac{C(Abrar\ \langle /s \rangle)}{C(Abrar)} = \frac{1}{2}$$

↳ Probability that Abrar is the end of string.

Probability → that string ends w/ Abrar

$$P(\quad) = \frac{C(Abrar\ \langle /s \rangle)}{C(\langle /s \rangle)}$$

Long models learn from data that you provide.

Longer gram — sparse data → capture more context
(N↑) (less matches)

shorter gram — many matches → less context
(N↓)

[large corpus → trigram / 5-gram as features]

Feature extraction → NLP

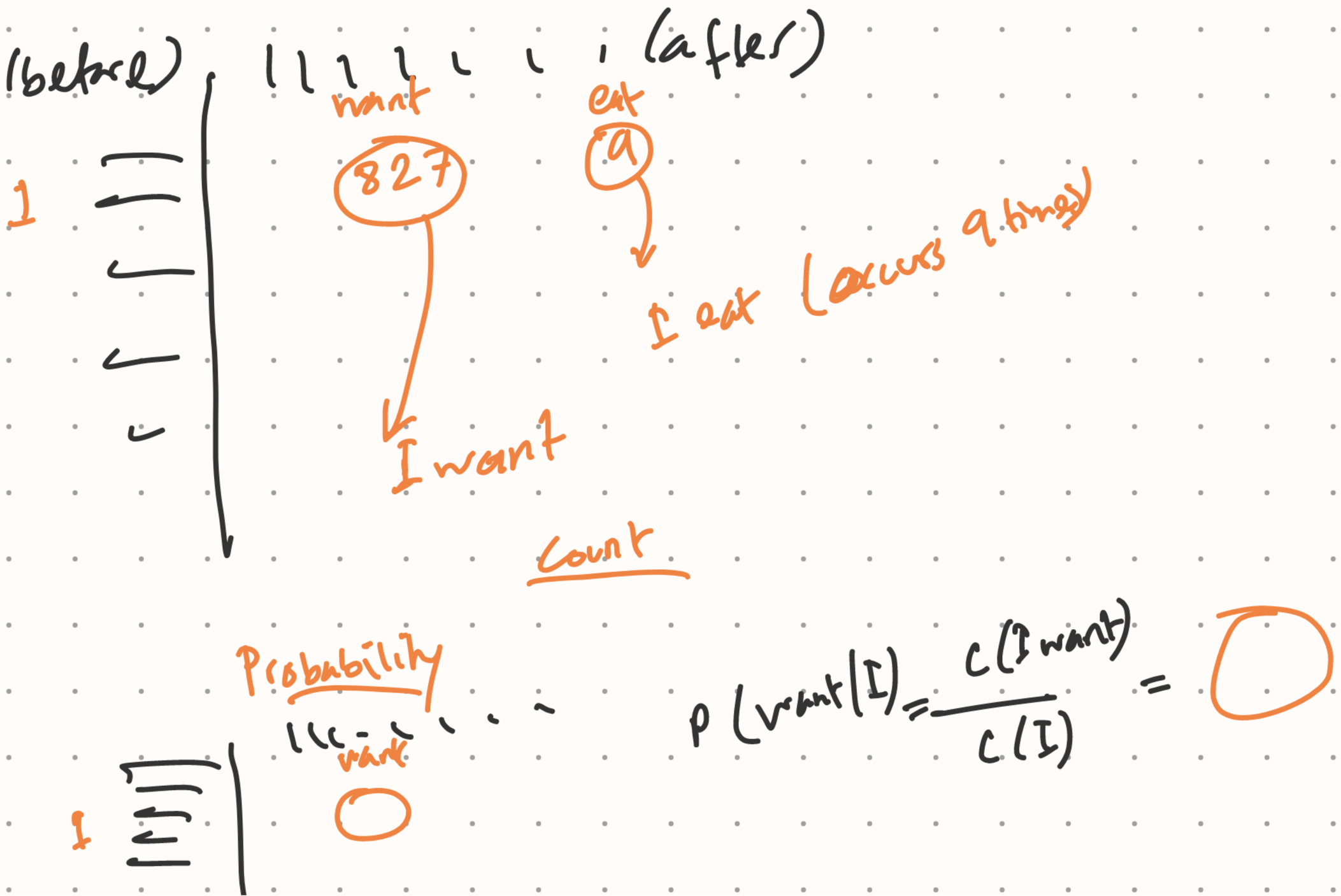
CountVectorizer → bigram, 5-grams
Bag of words

use character/text/word N gram embedding

fastText embedding → N-gram words + chars
(not looking any features)
↓
neural networks

Rule based vs feature based
↓ ↓
fixed cases all cases

Berkley Restaurant Project — Dataset



<s> I want chinese food </s>

$P(i \mid <s>) = 0.25$

$P(\text{food} \mid </s>) = 0.68$

bigram probabilities given

$= P(i \mid <s>)\ P(want \mid i)\ P(chinese \mid want)\ P(food \mid chinese)\ P(</s> \mid food)$

$= 0.25 \times 0.33 \times 0.0065 \times 0.52 \times 0.68$

$= 0.0002$

$\log(0.25) + \log(0.33) + \log(0.0065) + \log(0.52) + \log(0.68) = (\ldots)$ larger number

$\log(P_1 \times P_2 \times P_3) = \log P_1 + \log P_2 + \log P_3$

$\Rightarrow P_1 \times P_2 \times P_3 = \exp(\ldots)$

exponential

working w/ small numbers $\Rightarrow$ log

Evaluating language models (LM)

- Evaluate the system which embed the LM = **Extrinsic Evaluation** (expensive)
  - bug fixing expensive
- **Intrinsic Evaluation**: simulate the environment where users use LM (use test set to evaluate)
  train model using training corpus

★ one LM fits the test set better than the other one.
How to define this performance?

(Higher probability to the words

Assign probabilities better → better LM

(Probability of occurrence of a sentence in a corpus).

Dev set / validation set — optimize hyperparameter

unseen test set — test performance

Assign abnormally higher probabilities to the seen data.
^
overfitting : Higher bias towards training set (memorize rather than generalize)
highly unlikely to get 99% accuracy

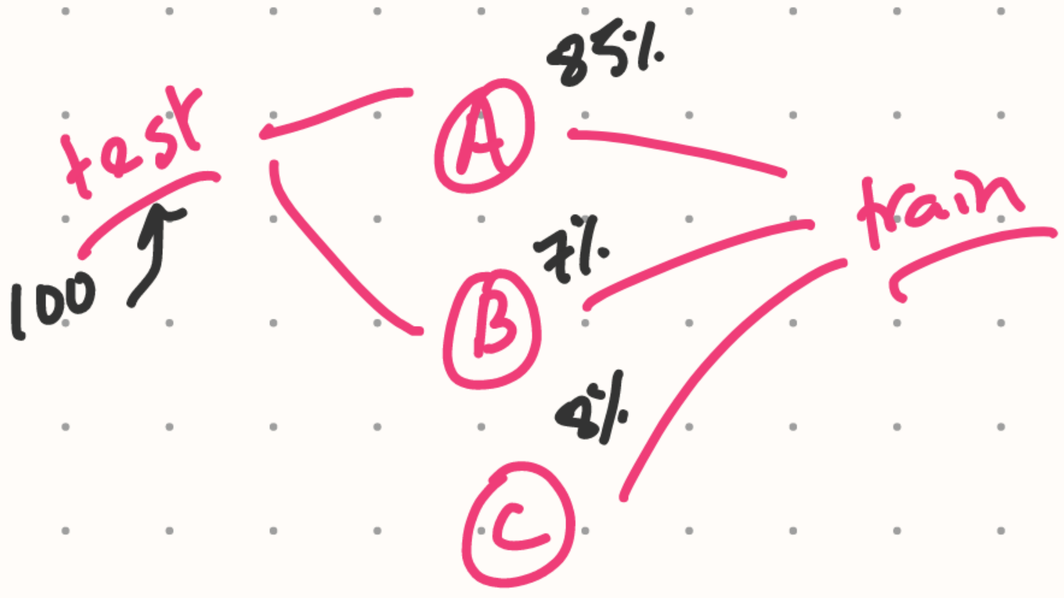
but poor on unseen data.

underfitting : performs poor on the training set as well as unseen data

model cannot find any generalized pattern in training data.
- cannot assign probabilities to seen data

Large Dataset
training : dev : test
80 : 10 : 10
70 : 15 : 15
60 : 20 : 20 → why not this?

low amount data — unrepresentative of actual scenario



biased test set — cannot evaluate LM

maintain ratio :
80 — training
7 — dev
8 — test

train_test_split

Evaluation matrix:

For general task:

Next word / Sentence prediction — doctor ankay
doctors far away

Summarization
Paraphrasing

Diff eval metrics for diff tasks

perplexity: To assess performance of LM in next word prediction

$$PPL(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

$$= 2^{-\frac{1}{N}\sum_{i=1}^{N} \log_2 P(x_i)}$$

N = Length of any test sample

PPL = logarithmic exponential of loss function (cross entropy)

"The cat is on the mat" → Test sample

P("the") = 0.1

P("cat") = 0.2

P("is") = 0.15

P("on") = 0.1

P("the") = 0.2

P("mat") = 0.25

Learnt from training corpus

N = 6 words/tokens in the test set

$$PPL = 2^{-\frac{1}{6}\left[\log_2(0.1) + \log_2(0.2) + \log_2(0.15) + \log_2(0.1) + \log_2(0.2) + \log_2(0.25)\right]}$$

$$= 2^{-\frac{1}{6} \times (-15.02)}$$

$$= 2^{2.5}$$

$$\simeq 5.66$$

→ 6 branches in training corpus where these probabilities will be found

more branches — more non deterministic — more difficult

PPL → weighted avg branching factor of a LM

PPL ↓ ✓ (less branches preferred)

★ Assignment

task + documentation

edit distance - brief explanation / defn

Notebook

Documentation - hard copy

- Islam public bias
- Detection
- Ukraine vs Palestine

★ Don't beat accuracy !

- transliteration
- express differing
- alphabet
- bangla
- English

interesting prob.

Prompt Engineering

- Parsing (extracting info from samples)

Sentiment analysis - classification

context/culture

- the sentiment in britain not same in BD
- negate sentence (how good → isn't good (not same in Bangla)

How do -ve patterns work in Bangla compared to English?

Linguistic pattern, not model

Design experiment

- mental health datasets - sample labels

LLM → depressing

- elderly people LLM lonely chat DO chatbot provide friends environment ...

- kids

Perplexity ↗ Probability

↓

better than

diff representations of probability

Project idea: Optimum

Large N-gram to model lang., what are the problems?

- less matches → parsing
  (many rows 0 in probability distribution matrix)

Berkley Restaurant Project corpus → Probability / count of a word given another word

Probabilities = bigram count / unigram count

$$P(i|i) = \frac{\text{count}(i|i)}{\text{count}(i)} = \frac{5}{2533}$$

Smoothing/Discounting: Take from other probabilities and give to the 0 probability

collective probability = 1

Laplace smoothing: Add 1s to all the 0 counts

$$P = \frac{\text{count}(w, v) + 1}{\text{all counts of bigrams} + 1}$$

After laplace smoothing: $P(i|i) = \frac{6}{253}$

I want food lunch.

$P = P(I \mid <s>) \times P(want \mid I) \times P(food \mid want) \times$

$P(lunch \mid food) \times P(<\backslash s> \mid lunch)$

$P(lunch \mid food)$ → 0 → whole probability becomes 0.

For a particular bigram

Soln Smoothing (Add 1)

Laplace/Add-K (another variant of laplace)

$$P = \frac{count(w|v) + k}{\text{all counts of bigrams} + K}$$

try diff k.

For which K, we get max probability

Good for text classification but not lang. model

Backoff

$P(w_n | w_{n-1} w_{n-2}) \rightarrow$ trigram probability

$P(w_n | w_{n-1}) \rightarrow$ bigram probability

If you don't find probability of a trigram, back off to bigram.

- - . - - - bigram, . - - - unigram

$P(\text{lunch} | \text{food}) = 0$ Bigram

↓ backoff

$P(\text{lunch}) =$ unigram

Interpolation : Advanced technique of backoff

$$P(w_n | w_{n-1} w_{n-2}) = \lambda_1 \cdot P(w_n) + \lambda_2 \cdot (\text{Bigram Prob.}) + \lambda_3 (\text{Trigram Prob.})$$

(unigram prob.)

$$\sum_{i}^{n} \lambda_i = 1$$

Kneser-Ney smoothing: sophisticated way

I can't see without my reading ____

✓ glasses

✗ Francisco (more based on Prob. count)

↓

If you only depend on Prob. count

Though Francisco is very frequent, only appropriate after

San.

San Francisco ✓

$P_{continuation}$ = the no. of diff contexts the word has appeared in.

P_{w} continuation $\propto \{ v : \text{unique } vs \text{ between } w \}$

$$P_{w} \text{ continuation} \propto \{ v : c(v, w) > 0 \}$$

(reading, glass)

glasses - 2

Francisco - 4

San Francisco - 1

$$\frac{P_c(\text{glasses})}{P_c(\text{Francisco})} = \frac{2}{1}$$

occurs in 2 context only

glasses ✓

(higher prob of occuring in diff contexts)

Preprocessing ✓
N gram LM

LM calculates probability distribution.

Stopwords — high probability?

Shouldn't be only depend on word count

Text/sentiment classification

Linear ML models — Naive Bayes, Logistic Regression

Evaluation metrics

Sentiment Classification

↳ sentiment extraction

what particular document is about

positive/negative

spam detection : email is spam/not

previously — hand rules (if else ...)

Soln: supervised ML

↳ input data w/ output labels

ML learn from these samples

Classifications:

→ Binary (+/−, spam/non-spam)

→ multinomial (series of emotion — anger, joy, or-de)

↳ discrete

→ ordinal (child, youth, teen, elder)

↳ series

→ multiclass (happy, youth)

↳ n. no. of classes

classify k out of n classes

from multiple classes

→ multilevel (happy, youth)

↳ multiple level for a particular element

Naive Bayes

$$p(x|y) = \frac{p(x) \times p(y|x)}{p(y)}$$

Naive assumption about how features interact

Bag of words
↳ frequency based

Does not consider order of the elements

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; p(c|d)$$

Calculate prob of all classes → max posterior prob. choose

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \frac{p(c) \times p(d|c)}{p(d)}$$

(p(c): class; p(d|c): document given class)

p(d): document

p(d) = 1 (have a doc.)

win a free vacation
document

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; P(c) \times P(d|c)$$

Prior probability

likelihood estimation

Prob. of doc given that it is inside that class

100 training samples — 50 spam, 50 non-spam

$$P(\text{spam}) = \frac{1}{2}$$

$$P(\text{not spam}) = \frac{1}{2}$$

Even in real scenario, model assume we get 1/2 always.

$$P(d|c) = P(f_1\ f_2\ f_3 \ldots f_n \mid c)$$

↳ All possible combinations of feature inside the class.

Expensive — need to simplify

Naive simplification

All features are independent

$$P(d|c) = P(f_1\ f_2\ f_3 \ldots f_n \mid c) = P(f_1|c) \cdot P(f_2|c) \cdot \ldots P(f_n|c)$$

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\ P(c) \prod_{f \in F} P(f|c) = \underset{c \in C}{\operatorname{argmax}}\ P(c) \prod_{i \in \text{position}} P(w_i|c)$$

↳ go to every position in doc, check the probability of that doc in that class

features — words w/ (tve) sentiments

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

Naive Bayes eqn for text

↳ generative classifier ↳ add features (linear classifiers)

Logistic Regression — Discriminative classifier
(classification (not generation))

Task: If incoming email is spam/not

Sentiment classification:

Prob. of the class that we're looking for

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

N_{doc} = #training samples

N_c = #samples in each class

$P_{spam} = \frac{2}{5}$ | classes = spam, non-spam

(total #doc)

$P_{Non\text{-}spam} = \frac{3}{5}$

Likelihood $P(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$

(particular word)

(all words in that class)

vocabulary

word you're looking for is not inside spam class

but in non-spam class

Soln Laplace smoothing

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

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

↙ all words in vocab

☆ Spam Detection using Naive Bayes classifier

Feature Extraction using bag of words

test sample, S = "Win a free vacation"

$P(Spam) = \frac{3}{6}$

$P(ham) = \frac{3}{6}$

a → unknown word (didn't occur in training sample) → ignore (OR, randomly assign a probability (cannot apply Laplace smoothing))

6 doc
- 3 spam
- 3 ham

Laplace smoothing → when word is here but particular bigram/trigram not here
(Add 1)

Prob. of occurrence of each word in each of the classes:

$P(win|spam) = \frac{1+1}{10+16}$

$P(free|spam) = \frac{2+1}{10+16}$

$P(vacation|spam) = \frac{0+1}{10+16}$

$P(win|Ham) = \frac{0+1}{8+16}$

$P(free|Ham) = \frac{0+1}{8+16}$

$P(vacation|Ham) = \frac{1+1}{8+16}$

16 unique words — vocab (unique ignore duplicate) $|V| = 16$

2 bag of words (1 for spam
1 for ham)

Count (not unique) — count duplicate words

$$\hat{c} = P(c) \cdot \prod P(w_i | c)$$

P(s in spam)

$$P(spam) \times P(S | spam)$$

$$= \frac{3}{6} \times \frac{2 \times 3 \times 1}{26^3}$$

$$\approx 0.000171 \longrightarrow \checkmark \text{ (bigger)}$$

P(s in ham)

$$\hookrightarrow P(Ham) \times P(S | Ham) = \frac{3}{6} \times \frac{1 \times 1 \times 2}{24^3}$$

$$\approx 0.000072$$

model classifies S as Spam (Naïve Bayes)

I really like this book

I don't like this book.

positive class

negative meaning

not

bag of words

In case of negation, using bag of words as feature extraction cannot handle these cases.

Prepending (preprocessing)

I don't NOT_like NOT_this NOT_book.

inside negative class

until punctuation

Prepending (preprocessing) should be done on test samples too.

→ Insufficient word in training sample

Pretrained, pre annotated corpus → true, +ve words

LIWC (Linguistic Inquiry word count)

MPQA lexicon — root word

| tve : admirable (0.9), beautiful (0.4) ... catastrophe (0.1)
| -ve : catastrophe (0.9), envious (0.4)

Catastrophe might be in tve sample with low probability.
lets, assume 0.1

spam >>> compared to ham

correctly classify ham (less in number in real life)

100 emails — unbalanced dataset — only knows spam (seen more spam)

99% (spam) 1% (ham) ← mistake

Predict every thing as spam

Evaluating lang. model (LM):

Accuracy = 99% → 1% classifying ham as spam

useless LM

↓

risky Evaluation metric

Soln Confusion matrix

(visualization of how an algo is performing in diff classes)

Use category for evaluation

spam

ham

Gold labels / Human Gold labels — in the test doc,

gold labels are used for evaluation metrics

$$F\text{-measure} = \frac{(\beta^2+1)\cdot Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$$

($\beta > 1$) → more weight to recall (R)

($\beta < 1$) → more weight to Precision (P)

($\beta = 1$) → equal weight P and R

Precision & Recall Calculation:

Confusion matrix for 3 classes:

Urgent
spam
ham

P = Precision
R = Recall

	u	s	h
u	✓ 8	10	1
s	5	60	50
h	3	30	200

$P_u = \frac{8}{8+10+1}$

$r_u = \frac{8}{8+5+3}$

Performance of whole model?

* macro-averaging
* micro-averaging

class-1 : urgent

8	11
8	340

$P_{r_1} = \frac{8}{8+11}$

class-2 : spam

60	55
40	212

P_{r_2}

class-3 : ham

200	33
51	83

P_{r_3}

Macro-avg precision $= \frac{P_{r_1} + P_{r_2} + P_{r_3}}{3} = 0.60$

↓

all classes equally imp.

268	99
99	635

} From 3 classes (3 matrixes)

Micro-avg precision = $\frac{268}{268+99} = 0.73$

↓ dominating class

Determining span → micro avg

- Naive Bayes — generative classifier
- Logistic Regression — Discriminative "

Classify cats & dogs:

(Naive Bayes) — generate as much features as possible

$$\hat{c} = P(c) \cdot P(d|c)$$

likelihood

— given cat, what are the features he eyes might have

— given dog, - - - - - - - -

tail might have

create dog/cat features by itself

(Logistic Regression)

↳ calculates P(d|c) directly

Put more weight one feature

model and prediction

1. feature Representation → find discriminative feature

2. classification function

binary / multinomial (based on need)

↙ hard void

↘ soft void

3. objective/ function

loss (objective/ loss function: how your prediction is diff from goal

prediction and gold label

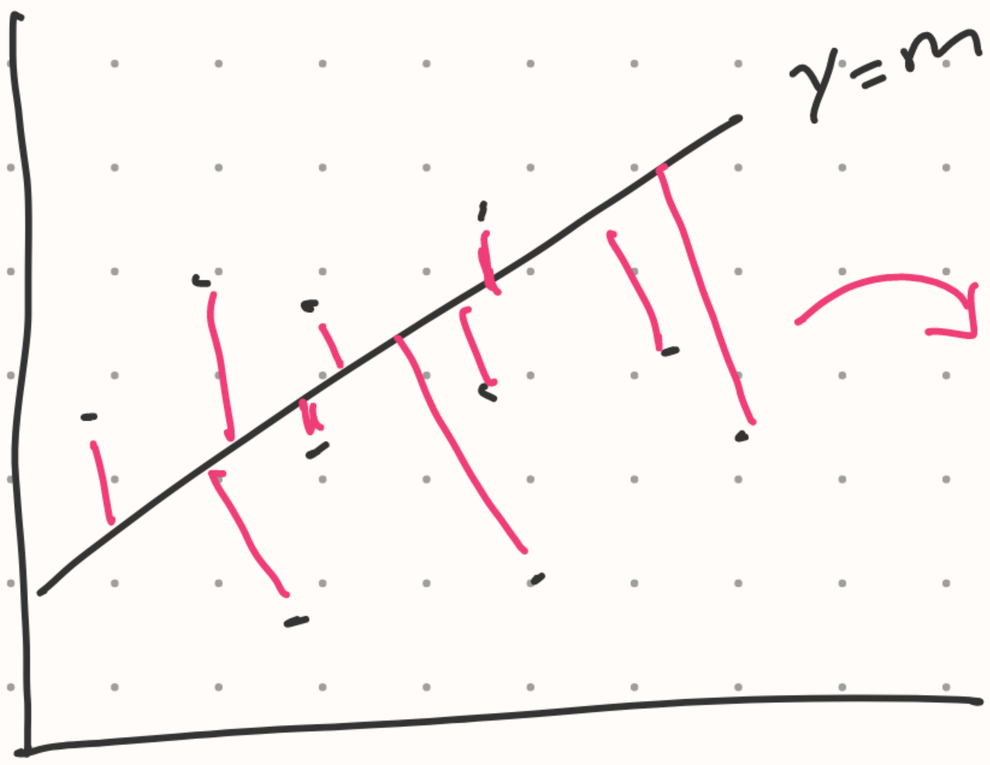
min. distance

4. optimization function : to optimize objective func.

| Gradient Descent

Linear Regr..

str. line (bad algo)



errors

MSE

(mean square Error) → L2 loss function

$$= \frac{\sum_{i=1}^{n} (y_i - y_i^{pred})^2}{n}$$

MAE = (mean Absolute error) → L1 loss

$$= \frac{\sum |y_i - y_i^{pred}|}{n}$$

How to optimize loss?

↳ m and b value

Gradient Descent : matrix multiplication formula
partial Derivative
} 2 ways

Random Hill climbing Algo : random m and b
(Global)
Calculate mSE, MAE
(like 20 times randomly)

minimizing error → go in that way
(decreasing loss func.)
stop when loss func. increasing again

Outliers → skewed data → inside outliers

Avoid outliers — L2 loss plays big price
(high penalty)

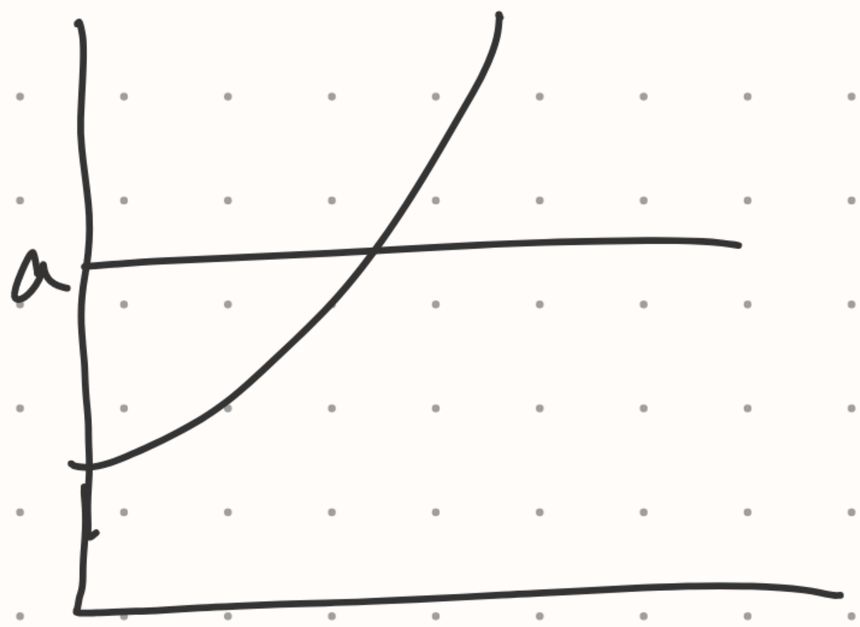
use L1 loss ✓

flattens out graph data points — Logistic Regr. soln
(use such a loss function that
flattens out w/ datapoints)

Logistic Regression

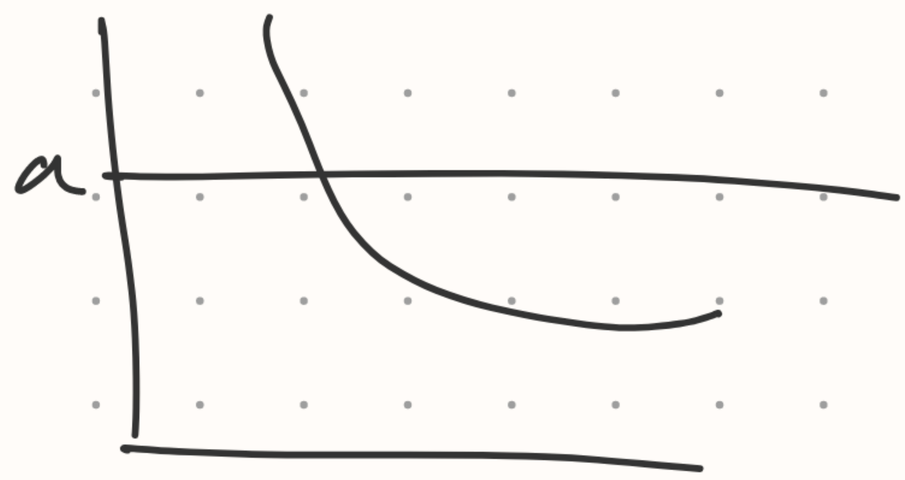
$$y = mx + b$$

$$y = a$$



$$y = e^{mx+b}$$

$$Y = e^{-(mx+b)}$$

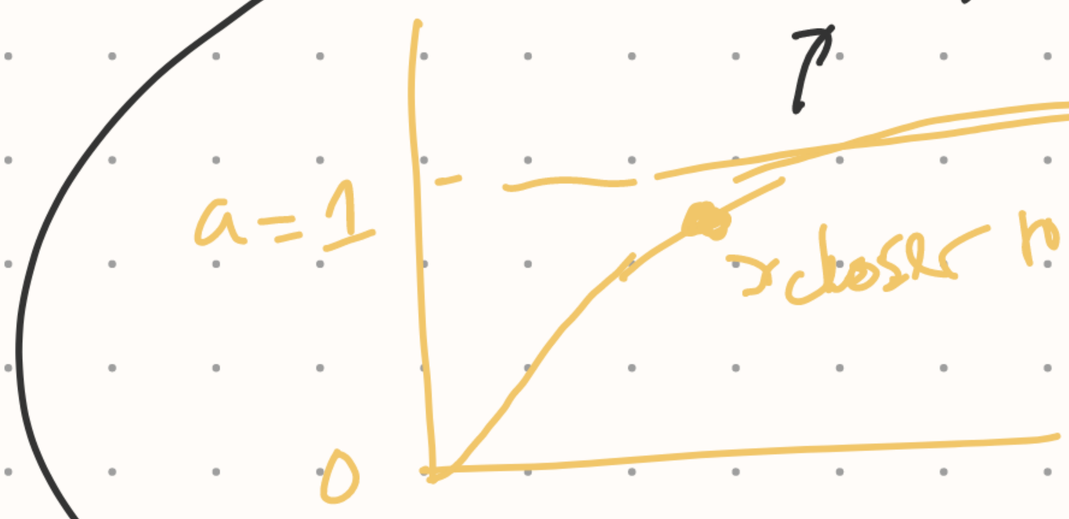


→ Don't rise above 'a'

$$y = \frac{a}{1 + e^{-(mx+b)}}$$

$$y = \frac{1}{1 + e^{-(mx+b)}}$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Sigmoid function

m = weight

b = bias

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

sigmoid func.
= fitting
= utilis ✓
- differentiable (optimizable)

* 6 features

weight more for evil words — 1
↑ (+ve) ~ -ve ↑ — 0

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

$w = [2.5, -5, -1.2, 0.5, 2, 0.7]$ $[b = 0.1]$
(weight)

high-value

Bias — prevent overfitting data

$P(+ \mid doc) = \sigma(w \cdot x + b)$

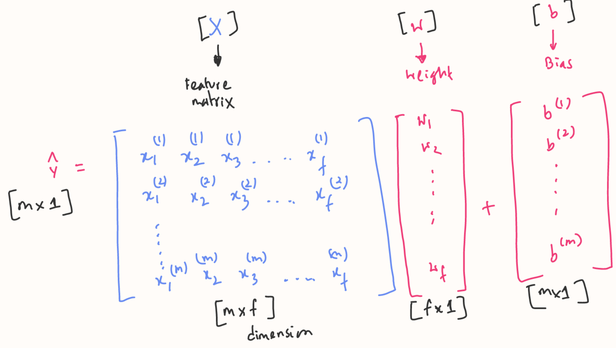
$= 3(2.5) + 2(-5) + \cdots + 0.1$

$= \sigma(0.833) = \frac{1}{1 + e^{-0.833}}$

≈ 0.7

$P(- \mid doc) = 1 - \sigma(z) = 1 - 0.7 = 0.3$

Feature template → Representation learning → Autoencoder (BART) (advanced RL)
(prev) → Dimensionality reduction (PCA) (appropriate features that represent data) (now) → SVD (Singular Value Decomposition)



m = # docs

$z = wx + b$

z = weight x feature + bias

- Tradeoffs between naive Bayes & Logistic Regression

matrix multiplication — costly
- small datasets ✓
- discriminate between classes ✓ (discriminative features)
- enough datasets

- Binary classification → use sigmoid function

$\hat{y}$ → positive

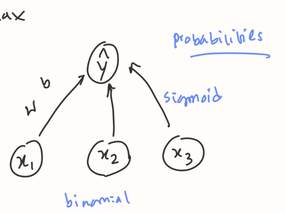
$1-\hat{y}$ → negative

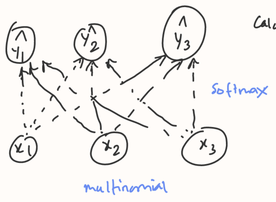
- multinomial logistic regression

sigmoid function designed to classify between 2 classes: 0 and 1.

change objective function ist to classify 72 classes.

use softmax





Calculate probabilities for n number of classes

$z = wx + b$

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

$\hat{y}_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z}}$

$z = [0.7, 0.15, 0.02 \ldots]$ (n number of classes)

- How to learn weights? (w and b)

Objective function / Loss: how likely the predictions are to my gold labels (true)

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

(ŷ: pred, y: gold labels)

Cross Entropy

outcome: 0 or 1

Bernoulli distribution → only 2 possible outcomes

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

(prob., feature of doc)

(y=1) if true label true ie 1 class → $P(y|x) = \hat{y}$

(y=0) if true label -ve ie 0 class → $P(y|x) = 1-\hat{y}$

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

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

(Cross Entropy Loss)

$= -\left[y \log \sigma(wx+b) + (1-y) \log (1 - \sigma(wx+b)) \right]$

Target: minimize loss function

If the classifier is perfect, you have zero loss.

. . . . . perfectly imperfect? - gold label = 0, classifying it as 1.

$\log(0) = \infty$

$\log(1) = 0$

range: $[-\infty, 0]$ (of loss function)

bad classifier close to $-\infty$

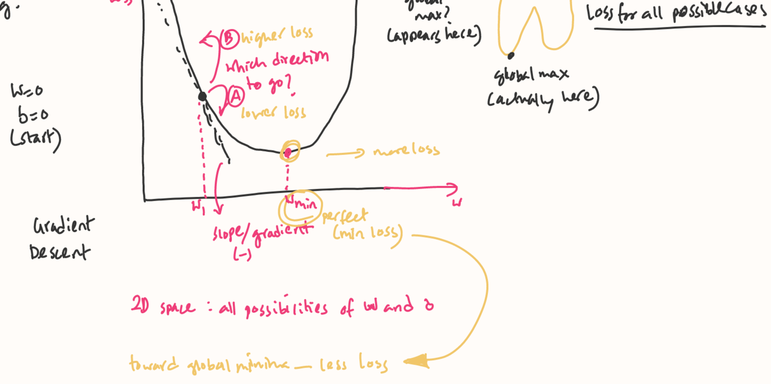
good 0

Loss function → θ

$\hat{\theta} = \underset{\theta}{\text{argmin}} = \frac{1}{m} \sum_{i=1}^{m} L_{CE}\left(f(x^{(i)}, \theta), y^{(i)}\right)$ (m: no. of samples)

$(\hat{w}, \hat{b})$ ← (w, b)

For each sample, minimize loss func.



future weight prediction:

$$w^{t+1} = w^t - \eta \frac{d}{dw} \left(L(f(x,w),y) \right)$$

(η) learning rate : 0.01

Gradient descent algo decide which way to go — go down

How much to go? — decided by η

η very high — cross global minima

η very low — take long time to find global minima

Hyperparameter tuning : take large η and decrease slowly

- How to calculate gradient descent?

2 feature: $x = [x_1, x_2]$

x_1 = (+ve) lexicons = 3

x_2 = (−ve) lexicons = 2

$w_1 = w_2 = b = 0$ (starts w/)

$\eta = 0.1$ (learning rate)

$$\theta^{t+1} = \theta^t - \eta \nabla L(f(x_i;\theta), y^i)$$

(gradient descent of loss func. (inverse delta))

$$\nabla_{w,b} L = \begin{bmatrix} \frac{d}{dw_1} L_{CE}(\hat{y},y) \\ \frac{d}{dw_2} L_{CE}(\hat{y},y) \\ \frac{d}{dw_b} L_{CE}(\hat{y},y) \end{bmatrix}$$

partial derivatives wrt w_j

loss func. for logistic regression:

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

$$\frac{d}{dw_j} L_{CE} = (\hat{y} - y)\, x_j$$

$\hat{y} = \sigma(w \cdot x + b)$ sigmoid func.

$x_1 = 3$, $x_2 = 2$

$$\nabla_{w,b} L = \begin{bmatrix} (\sigma(w \cdot x + b) - y)\, x_1 \\ (\sigma(w \cdot x + b) - y)\, x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} -0.5\, x_1 \\ -0.5\, x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

$$\sigma(0) = \frac{1}{1+e^{-0}}$$

$$\theta^t = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}$$

$$\theta^{t+1} = \theta^t - \eta \nabla L(f(x_i;\theta), y^i)$$

Stochastic grad desc. (one step at a time) — online algo

$$= \begin{bmatrix} w_1 = 0 \\ w_2 = 0 \\ b = 0 \end{bmatrix} - 0.1 \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

$$= \begin{bmatrix} 0.15 \\ 0.1 \\ 0.05 \end{bmatrix} \rightarrow \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}$$ after 1 step

$$\hat{\theta} = \text{argmin} \frac{1}{m} \sum_{i=1}^{m} L_{CE}\left(f(x^{(i)}, \theta), y^{(i)}\right)$$

convex optimization — gradient descent always points towards global min.
optimize weights and biases

Perfectly fit weight on features — model likes

Cannot generalize unseen data — overfitting
performs well on training data

- How to tackle overfitting in optimization problem?

— Regularization
1. L1 norm : Absolute distance from origin
2. L2 norm : Euclidean distance from origin (degree=2)

For L2 norm, eqn: $\hat{\theta} = \text{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^{m} L_{CE}\left(f(x^{(i)}, \theta), y^{(i)}\right) - \alpha \sum_{j=1}^{n} \theta_j^2$ (one example at a time; α: scalar)

↓ distributes weights so that no weight becomes very large (break down weight into smaller weights)

For L1 norm, eqn: $\hat{\theta} = \text{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^{m} L_{CE}\left(f(x^{(i)}, \theta), y^{(i)}\right) - \alpha \sum_{j=1}^{n} |\theta_j|$

↓
sparsity
big weight
Soln randomly make some weights 0

L2 better than L1 ✓

all eg at a time : time and resources ↑

Soln batch_size

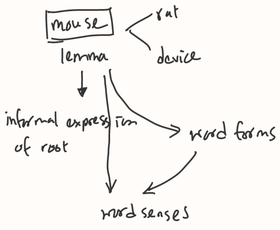
Stochastic gradient descent : one by one eg
(online algo)

Embeddings

Feature Engineering : hand crafted features
not good for large doc

frequency — prev. evaluation metric to represent features

Represent words in vector space using their meanings → vector semantics
(Representation learning)



Synonymy : couch sofa
water H_2O

" No two words are perfectly synonyms " — synonyms are not always substitutable
↳ Principle of

Word Similarity :
no perfect synonyms but have similarity

cat dog — both animals

Random insertion/substitution : cat → dog

Word Relatedness : more broadened scope

coffee cup — related
(used in same context / semantic field)

restaurant : waiter, menu, recipe — related

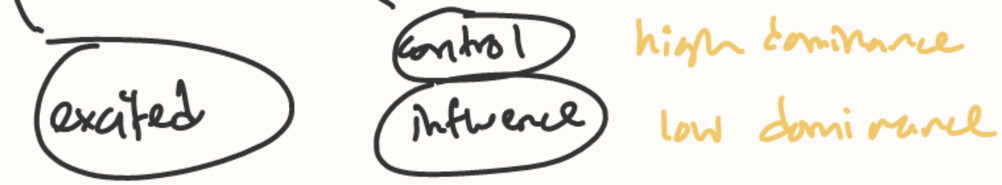
LDA : Latent Dirichlet Allocation
↓
model's topics
uses a context window
makes atomic based on sentences occurring in a context window

document: Doctor nurse hospital ot

Connotations : related to sentiments

innocent, naive
↓ +ve context ↓ -ve context → connotation

Every word has 3 particular characteristics :
1) Balance (pleasantness)
2) Arousal (how intensely tries to simulate)
3) Dominance (degree of control)



Excited (__ , __ , __)

↓ balance ↓ arousal ↓ dominance

Projecting a word in 3D space
(multi dim)

↳ Embeddings

word can be represented as a point in multidim space.

Document similarity – how similar doc is to query → search engine

Info retrieval (IR) – query → docs provided
(most appropriate)

cosine similarity → smaller angle – more relevant to query

col vector : docs as cols
row vector : words as cols

hypothesis → decision

similar words —occurs in→ similar docs

similar docs —contains→ similar words

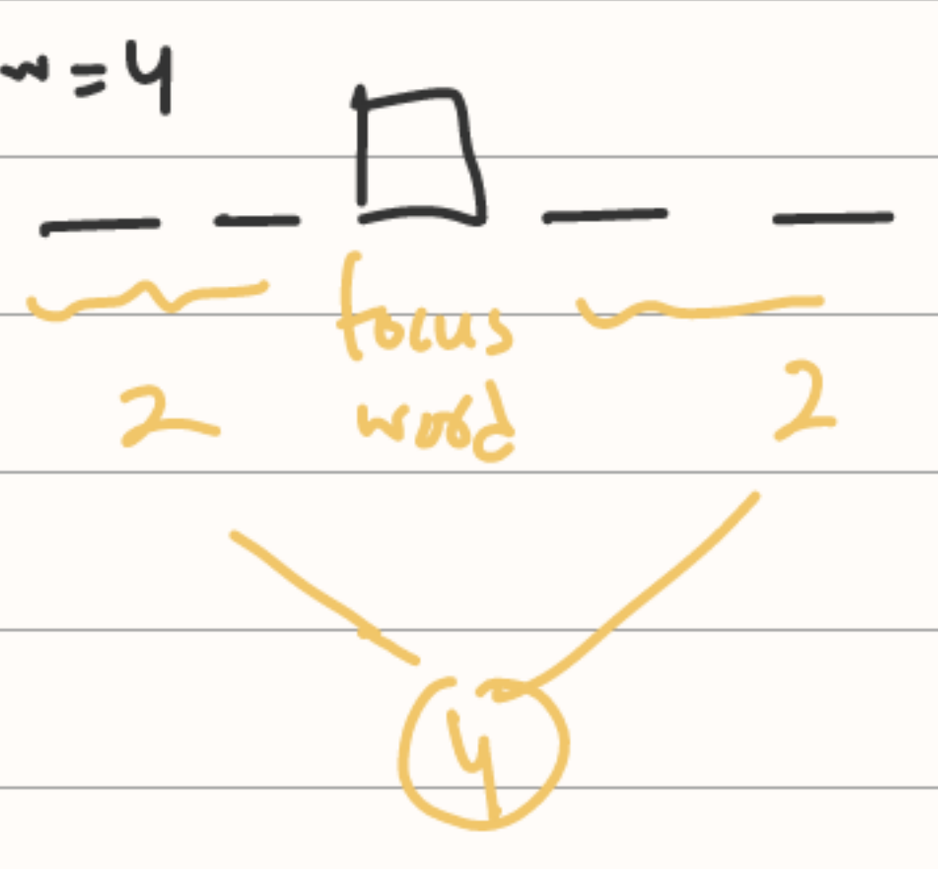
Distributional semantics : Projecting words in vector space based on meanings

- Till now, uses frequency

Term document frequency

\# rows → # vocabs
\# cols → # Docs
represent words in terms of doc — row wise

Represent words in terms of words
- context : 4 words before, 4 words after
  For each word : context window = 4

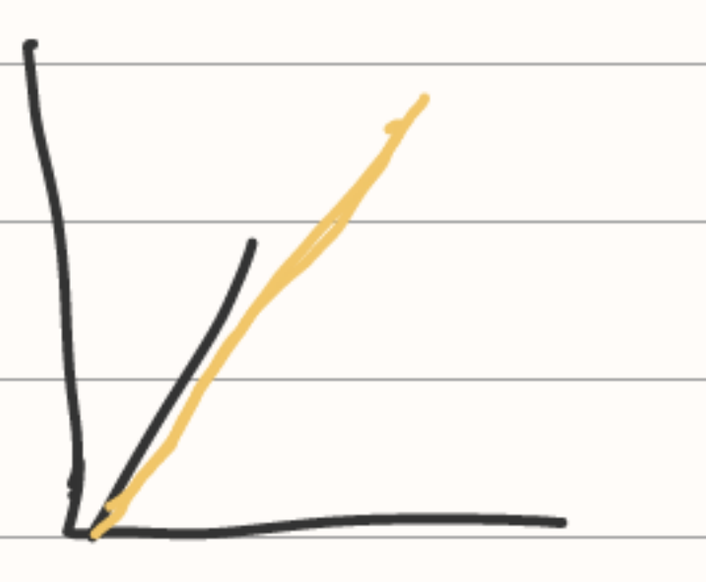


which words occur frequently in same context window
- digital computer
- strawberry pie
- cherry pie

→ strawberry and cherry similar / correlated
→ digital and info correlated

Embedding : Projecting those words in terms of meanings

If two words appear in same context window, they are similar in meaning. (Almost the same vector)



Cosine Similarity

measure similarity between 2 vectors (in 2D)

But in multiDim ?

Dim = V × D
V → # vocab
D → # Docs

If vectors are similar, dot product will be higher

cos(cherry, info) = 0.007
cos(digital, info) = 0.998 ✓ → digital and info more similar

NOT
In 1 doc, high value → negative doc (rules type doc)

Term frequency says about context

sparse vector (0 at most entries)
Freq. words are not good discriminators of context.
Stopwords - highest occurring terms → Doesn't say anything about context

TF-IDF : Term frequency inverse document frequency

Term-Doc
diff dim

Suppress the imp of frequency →
1 → 1
10 → 2
100 → 3

Doc freq. → In how many of docs, the term is appearing

$$idf = \frac{N}{df} = \log_{10}\left(\frac{N}{df}\right) = \log_{10}\left(\frac{4}{4}\right) = \log_{10}(1) = 0$$

$$tf\text{-}idf = tf \times idf$$

A particular term is appearing in all docs → idf = 0, tf-idf = 0

N = # docs
df = # docs the term appears in

Find tf-idf for all words in docs
similar words in terms of context → row vector — cosine similarity

avoid 0 → $idf = \log_{10}\left(\frac{N}{df^{\alpha}}\right)$, $\alpha \to \frac{a}{b}$

Positive Point wise mutual info (PPMI)

Term-Term
same dim

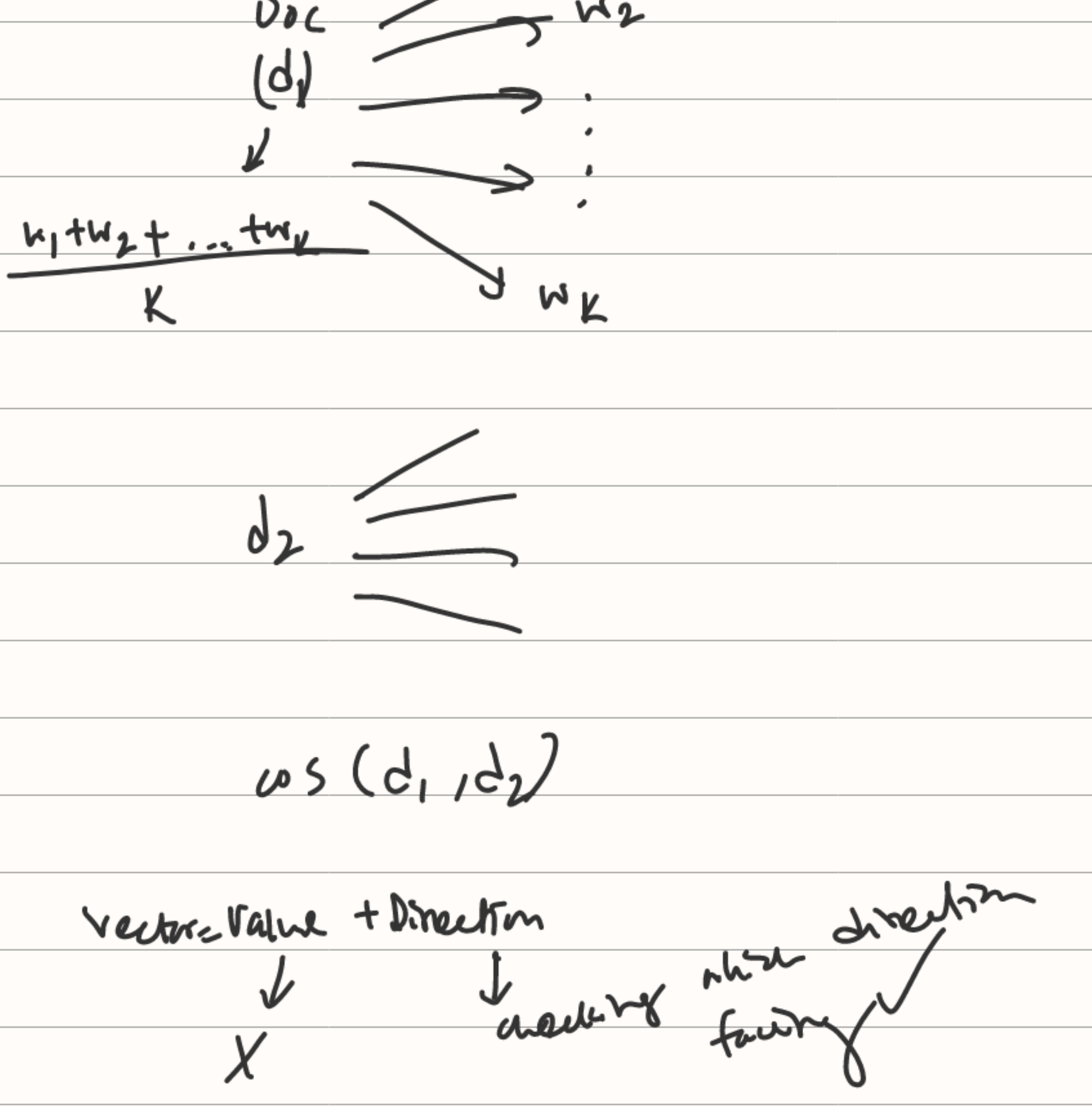
How they appear in same context
. . . . individual context

What is the probability that the terms appears in the same context?
→ ratio betwn 2 prob
work w/ positive PMI

PPMI is biased for infrequent words
- low value of probability
- high value of PPMI

soln scaling

Document similarity



cos (d1, d2)

vector = Value + Direction
Value → X
Direction → checking which direction facing ✓

In which scenario, which embedding works better?

word2vec

Embedding vector : dense vector
(not sparse)
↳ frequently occurring 0s

(term-doc vector)
|v| x f → embedding for tf-idf
|v| x |v| → term-term vector

doc/terms
the ... (f)
cherry
data
⋮
all unique words (|v|)

Prob: tf-idf and pmi
→ frequency dependent
→ sparse vector

Soln
Dense vector : learn weights
(not frequency dependent)

|v| x d
↳ Dimension — random number (300)
↳ optimal result

target word vs context word prediction

Data is processed into information.
(context words, focus word)

Context window = 4

When you know context words, and want to find focus words ⇒ CBOW (continuous bag of words)

When you know focus word, ... context words ⇒ Skip-Ngram

Data is ___ into information. → CBOW, → Skip-Ngram
___ ___ processed ___ ___
↳ likely to produce more rare words
approach to create word 2vec

similar words into similar classes — target of embeddings

Skip Ngrams is more suitable approach for finding embeddings of rare words.
(diversity more)

1m words in vocab
probabilities : $1m C_2$ — not feasible

Training
Negative sampling (|v| x d)
words
2 matrix
context

[PAD] Data is processed into info. [PAD]
c f c — iteration 1
c f c — iteration 2
} context window = 1 (3 words)

Every word might occur as f as c

rows = # vocab
cols = # defined by you (traditionally 300)

Data 0.1 -0.2
is
processed
into
info
cat
dog
⋮
↓

Dim space should be large enough to capture true essence of the words.
but shouldn't be so large that it makes it sparse vector.

take a vector — randomly optimize using grad descent

Prediction Problem

$$\log p(c|w;\theta) =$$

softmax obj function — denominator used to normalize between 0 and 1.

$$\sum_{c'}$$

1m words prob → X
(find context prob of all words)

Soln
Negative sampling
sigmoid obj. function

Neg samples — unigram distribution.

$$P = \frac{c^{\alpha}}{\sum c^{\alpha}}$$

$\alpha = 0.75 \rightarrow \frac{3}{4}$

Brings down frequent words
Brings up infrequent words

use grad descent to optimize

Embeddings ⇒ weights

softmax, loss function

word (W) (C1)
if actually occurring in context window — true (actual)
w.C1 — predicted

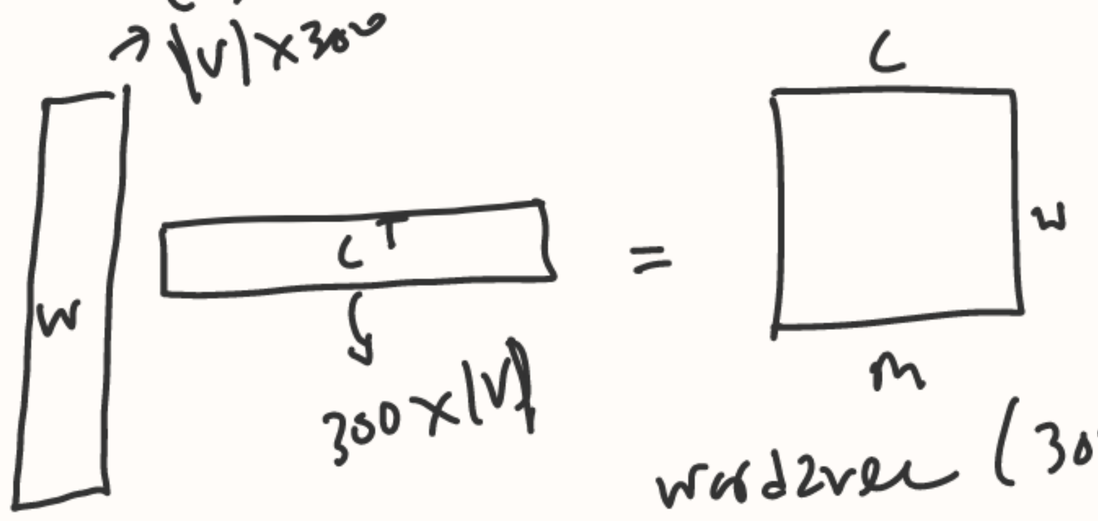
loss function : 1 − W·C1
1 − W·C2

global minima — min. loss

similar contextual words in similar window

word2vec Jordan video

(−) → to minimize



word2vec (300×300) ?

$A = S \Lambda S^T$

SVD
(Singular value Decomposition)
- Dimensionality Reduction
  - 50000 to 300 dim

(SVD cannot run in my PC
can run word2vec

Evolution of words through embeddings
(1850s to 2000s)

Diff word2vec embeddings from same doc

Inherent variability — embeddings are variable
  - bc randomizing weights at first iteration

multiple weights — avg

bootstrapping
(randomize)

Word2vec — static embedding
└ Every word has same embedding regardless of context

Now — Dynamic embedding
└ one word has diff embedding based on context

'King' − 'man' + 'woman' = 'Queen'

man : woman :: King : Queen

Gender stereotyping : male doc — nurse
└ (not female doc)

1) Allocational bias
misogynist
└ homemaker
↓
unfairly for diff people

2) Representational bias
(racial bias)

American names : David — handsome
African names : ____
racial remarks

Internet data contains bias - positively biased towards white

Debiasing techniques
└ reduce bias, not remove
Remove bias from embeddings
(not training sample)

MIP
→ Probability related Ques. ✓
| Derivation ✗
| Application ✓