[] = disjunction operator

PAGE NO. :
DATE : / /

Alphabet = {a-z, A-Z, space, 0-9, punctuations, emoji, reserve unicode block}

tokens ⇒ the constituent chars of a token is a subset of alphabet;

LLMs design context-sensitive languages very well. It learns contextual-embeddings

[P] so P એક pattern છે જે find કરશે

^ = negation so [^a-f 0-4] so not in a to f, 0 to 4

eg ^[^a-f0-4][A-Z]
if aA, bC મા X [P-O]
7A. મા ✓
7ABC ✓

^[^a-f0-4][A-Z]$ → end with
So શરૂઆત અને 2 char on string accept થશે.

* = 0 or more
+ = 1 or more

? = exists eg [A-Z][0-9]? so A9

{n} = repeat n times

a\+ = a[+]

[.]{2} is better.

a{2,5} can be written as a{2,2} | a{3,3} |
a{4,4} | a{5,5}

Q write a pattern to identify the middle character in a string of length 7 where the string should start and end with a digit. The starting digit can not be zero. The middle character should be one of a, b, e, f, g, -, _, +

ans ^[1-9][.][.][.][.][.][0-9]$
[abe-g\-_\+]

\w = alphanumeric character set
[a-zA-Z0-9_]

\d = digits [0-9]

\W = non-alpha numeric

\s = whitespace

\S = non-whitespace

so \d ગાંધાર, કોઈર ઓળખો જણાવો ત્રેગાય.

Q x = I am going home ← This has 5 boundaries
('\S+', x)
ans = I

\b = boundary [.] is any character.

Q.12
→ \d+\.\d

eg: 2/3 or \d+\/\d+

eg: 1,23,732 ⇒ \d+,\d+,\d+
so to remove redundancy we use groups.

(\d+,)+\d+

⇒ Order of precedence in RE
- groups
- wildcard +, *, ?
- [ ]
- OR abd|acd

Nesting of groups is also possible ((\d+)(,))+\d+

Q RE for URL
(^(https?\:\/\/)?www\.)?([\w]+\.)+(in|org|com)

Q (^|[^a-zA-Z0-9])[tT]he([^a-zA-Z0-9]|$)

[^a-zA-Z0-9] → empty string or not (alpha numeric is not)

[tT]he → "the/The"

[^a-zA-Z0-9]|$ → not alpha numeric or end empty string

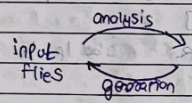
Q ([a-zA-Z])([a-zA-Z]+) \s \1 \2

where ([a-zA-Z]) = \1

([a-zA-Z]+) = \2

Finite state machine can only recognize but finite state transducers can generate + recognize

⇒ There are 2 types of task
Analysis (recognition)
Generation



Do pattern based recogding

Now create a list "ending -y", "just adding -s", etc.
These are called Paradigms.

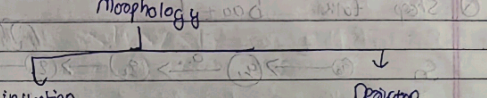
We store all the words in a table

eg: walk, walks, walking, walked
root = walk (this is called Lemma)

Stemming = process of reducing words to their root or base form, often by removing prefixes or suffixes.

y → ies ss → es

Morphology

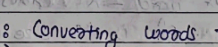


Inflection	Derivation
Adding suffix, category (noun/verb) remains the same	Adding suffix, category changes.
eg: walk, walking	Verb = Realize
	Noun = Realization

porter Stemmer

⇒ Morphology is the study of word structure and how words are formed from smaller meaningful units called morphemes

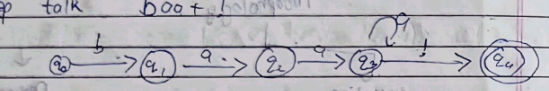
Lemmatization: Converting words to their base or dictionary form (lemma) considering the context
"better" → "good"



Content (open class)	function (closed class)
Nouns	Prepositions (on, to, of, nor)
Verbs	articles (a, an, the)
Adjectives	Conjuctions (but, and, although)
Adverbs	Pronouns (he, she, it)

Compute as verbs : Googling

Q) Sheep talk boo+!



$\Sigma = \{b, a, !\}$ Σ^+ = closure property.

δ is	b	a	!
q_0	q_1	∅	∅
q_1			
q_2			
q_3			
q_4			

Q) Given a dictionary, can you design an FSA/FSM
ans if length of max word = n chars
then 26^n is the size of transition table.

So we can use dictionary and prune the words not in dictionary.

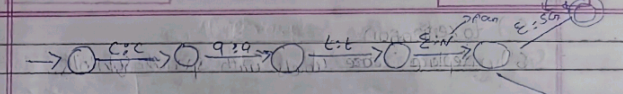
Finite state transducer (mealy, moore)

$Q, \Sigma, O, \delta, F, S$

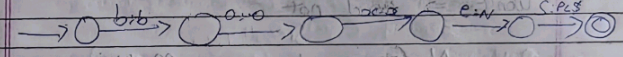
So $\delta: Q \times \Sigma \times O \to Q$

Sub (1 is {2,3}, ' ', text)

cat, cats



box, boxes

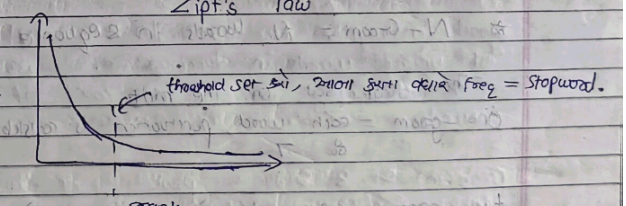


for this also we need transition table

⇒ Stopwords are those words which are very frequent but they do not have any "meaning" value.

We generally remove stop words

Zipf's law



Normalization / Standardization techniques.

1) Stopword removal
2) Stemming
3) Lemmatization
4) Spelling / text normalization

5) tokenization
6) Replace rare words with special symbols.

To handle noisy text we do expanding
I'd go ⇒ I would go
won't ⇒ would not

N-Gram Language Modelling

$P(x,y) = P(x) \cdot P(y|x)$

$P(x,y,z) = P(x) \cdot P(y|x) \cdot P(z|x,y)$

$P(w_1, w_2, \dots w_n) = P(w_1) * \prod_{i=2}^{N} P(w_i | w_{1:i-1})$

* N-Gram = N words in sequence

The cat sat on the mat.
One-gram = each word, punctuation is considered 1
∴ 7

two-gram = 2 words ना 1 consider करो
∴ 5

For handling prob. = 0 use will take log and "add" instead of "multiply"

1-gram model

$P(w_1, w_2, \dots, w_n) = P(w_1) * P(w_2|w_1) * \dots P(w_n|w_{n-1})$.

2-gram model

$P(w_1, w_2, \dots, w_n) = P(w_1) * P(w_2|w_1) * P(w_3|w_2, w_1)$
$\dots P(w_n | w_{n-1}, w_{n-2})$.

For bigram model we add 2 sentence, <s> <begin of sentence> and <end of sentence> </s>.

$P[\langle eos \rangle | w_{n-1}, w_n)] \; P(w_n | w_{n-1}, w_n)$

A bigram language model is a statistical language model used in NLP to predict the likelihood of a word in a sequence based on the preceding word.

Bigram assumes that probability of a word depends only on the word immediately preceding it.

How to actually calculate probability.

$P(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})}$

$C(w_{n-1})$ = how many times w_{n-1} occurs alone

How many times w_{n-1} and w_n are occurring (यानि, bigram)

but $C(w_{n-1}) = \sum_{w_i} C(w_{n-1}, w_n)$

so to → the, be, so

so number of "to" is go to calculate यानि.

Lidstone.

So in trigram $P(w_n \mid w_{n-2}\, w_{n-1}) = \frac{C(w_{n-2}, w_{n-1}, w_n)}{C(w_{n-2}, w_{n-1})}$

if the w_n is a very new word, count 0 so Prob. 0

Smoothing.

- Add one smoothing (Laplace smoothing)
  count of new word માં 1 add કરી દેવો.

$P(w_n) = \frac{C(w_n)}{N}$, $p(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)+1}{C(w_{i-1})+V}$

$= \frac{C(w_n)+1}{N+|V|}$ (→0 હોય)

good assumption for $|V| = 10k$, $|V|$ is number of unique words in language i.e. vocabulary, N= કે dictionary માં હોય એવુ count of unique words.

problem એ છે આમા કે probability of those words which are already frequent that also decreases.

$P(to) = 0.3 = \frac{3}{100}$ હતો

now $P(to) = \frac{30+1}{100+10000} = 0.003$

lidstone
↑
So solution is Add k smoothing

$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}, w_n)+k}{C(w_{n-1})}$

3rd type. (add types *)

$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}, w_n)+k}{C(w_{n-1}) + k * T(w_{n-1})}$

here $T(w_{n-1})$ = no. of unique bigrams starting with w_{n-1}

now logic is કે bigrams etc depend on continuity છે પાછળ ના word ની predict કરી so logic maintain કરવા માટે $T(w_{n-1})$ લીધુ

now to make it a probability
smoothed count ← $C_i^* = (C_i+1) \times \frac{N}{N+1}$

So $P(C_i^*) = \frac{C_i^*}{N} = \frac{C_i+1}{N+1}$

$d = \frac{C_i^*}{C_i}$

Types = no. of unique words

PAGE NO.: DATE: / /

Witten - Bell Smoothing

$$\sum_{w_i \in C(w_i)=0} P_i = \frac{T}{N+T}$$

(T → currently unique words in model)

Z = count of all zero frequency words
$= \sum w_i \mid C(w_i) = 0$

$$Z \times P_{unseen} = \frac{T}{N+T}$$

$$P_{unseen} = \frac{1}{Z} \cdot \frac{T}{N+T} \quad ; Z = V - T \text{ for unigram}$$

$$P_{seen} = \frac{C(w_i)}{N+T}$$

For bigram.

$$P_{unseen} = \frac{1}{Z(w_x)} \times \frac{T(w_x)}{N(w_x)+T(w_x)}$$

(T(w_x) → starting with w_x bigrams which are unique; N(w_x)+T(w_x) → Total bigrams starting with w_x.)

$$Z(w_x) = V - T(w_x)$$

$\frac{N-N_0}{N_0}$

All these are Maximum Likelihood Estimation

PAGE NO.: DATE: / /

Good turing Smoothing

$N_0 = ?$

N_1 = no. of words with 1 freq

N_2 depends on N_1, N_3 depends on N_2

$$N_r^* = (r+1)\frac{N_{r+1}}{N_r}$$

For unseen n-grams

$$P^*(n\,T) = \frac{N_1}{N}$$

(N_1 → n-grams of freq. 1; N → Total no. of n-grams)

let there be a bowl and f(red) = 4, f(black) = 5, f(green) = 3
f(orange) = f(violet) = 2, f(yellow) = f(blue) = 1

so $N_1 = 2$ $N_2 = 2$
$N_3 = N_4 = N_5 = 1$

lets say Cyan, $P(\text{Cyan}) = \frac{0}{18} = 0$

so apply good turing smoothing

$$P_0^* = \frac{N_1}{N} = \frac{2}{18}$$

now yellow's count will be updated.

$$r^* = (r+1)\frac{N_2}{N_{r+1}}$$

$r_1^* = \frac{(1+1)\,N_2}{N_1} = \frac{2\,(2)}{2} = 2$

$r_2^* = \frac{(2+1)\,N_3}{N_2} = \frac{3 \times 1}{2} = \frac{3}{2}$

till r_4, last again or same as next

problem: In bigram and Trigram we won't have all so use next available higher frequency

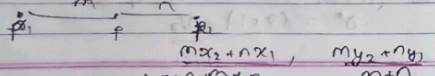
Backoff Models (interpolation)

⇒ Stupid backoff: if $C(w_{n-2}, w_{n-1}, w_n) = 0$ then fall back to bigram model

if $C(w_{n-2}, w_{n-1}) = 0$, fall back to unigram model.

* in fallbacks don't use smoothened models.

if nothing is there put $1/N$.



⇒ $P(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i)$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, $\lambda_1 \neq 0$ $\lambda_2 \neq 0$, $\lambda_3 \neq 0$

now Grid search करेंगे to find $\lambda_1, \lambda_2, \lambda_3$

λ_1	λ_2	λ_3
0.1	0.3	0.6
0.2	0.2	0.6

now we use this values and apply this model on dev (val) set and find errors.

Errors = LHS − RHS

= $P(w_i \mid w_{i-2}, w_{i-1})$ we have from trigram and RHS λ and values so we can calculate

For this we can use smooth or unsmooth models

Deleted interpolation method और बाकी बाद में

Katz backoff

$$P_{KATZ}(w|h) = \begin{cases} d_{w,h} \times \hat{P}_{MLE}(w|h) & \text{where } C(w|h) > k \\ \alpha(\text{discounted KATZ history}) \times P_{KATZ}(w|\text{smaller history}) & \text{where } C(w|h) \leq k \end{cases}$$

$d_{w,h} = \frac{r^*}{r}$ where r^* can be found using good turing smoothing & $d_{w,h} = (r+1)\frac{N_{r+1}}{N_r} \times \frac{1}{r}$ $\left[\frac{C^*(\text{freq})}{C}\right]$

$$\alpha_{KATZ}(w_i|w_{i-1}) = \frac{1 - \sum_{w_i : C(w_{i-2}, w_{i-1}, w_i) > 0} P_{KATZ}(w_i|w_{i-2}, w_{i-1})}{1 - \sum_{w_i : C(w_i, w_{i+1}) > 0} P_{KATZ}(w_i|w_{i-1})}$$

So here third discount says and α are distributed in $\leq k$ ones.

$r^* = \frac{(r+1)N_{r+1}}{N_r}$ [r = observed freq. of n gram]

N_r = no. of n grams that occur exactly r times

Kneser – Ney Smoothing

$$P(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}, w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) \times P(w_i)$$

$d = 0.75$, $\in [0,1]$

bigrams starting with w_{i-1} ← $\lambda(w_{i-1}) \times P(w_i)$ → Continuation

bigrams ending with w_i

$$\lambda(w_{i-1}) = \frac{d}{C(w_{i-1})} \times \{w_i : \text{no. of bigrams starting with } w_{i-1}\}$$

Normalization

no. of unique bigrams starting with w_{i-1}.

$$P(w_i) = \frac{\sum_{w_x} C(w_x, w_i)}{\text{total bigrams}}$$

Continuation

so ending with w_i

no. of unique bigrams

Q $C(\text{the, cat}) = 3$, $C(a, \text{cat}) = 2$, $C(\text{the, dog}) = 4$, $C(a, \text{dog}) = 1$, $C(\text{cat, sleeps}) = 1$, $C(\text{dog, sleeps}) = 3$

$C(\text{cat}) = 5$

now let's do for trigram.

$$P_{KN}(w_i \mid w_{i-1}, w_{i-2}) = \frac{\max(C(w_{i-2,i-1,i}) - D, 0)}{C(w_{i-2}, w_{i-1})} + \lambda(w_{i-2}, w_{i-1}) \cdot P_{KN}(w_i \mid w_{i-1})$$

here $\lambda(w_{i-2}, w_{i-1}) = \frac{d}{C(w_{i-2}, w_{i-1})} \times$ unique trigrams following w_{i-2}, w_{i-1}

P માટે પાછલી formula માં મૂકી દેવો બસ.

Entropy = highest of uniform distribution
= measure of randomness

$$H(P) = -\sum_{x_i \in X} P(x_i) \log_2 P(x_i)$$

perplexity $= 2^H$

$$PP(W) = P(w_1 w_2 \ldots w_n)^{-\frac{1}{n}}$$
Perplexity

$W = w_1 w_2 \ldots w_n$ sequence

normally perplexity complete corpus on લાગે.

Eg. Model can predict = a, the, red, fox, dog, .
so P ('a red fox .')

$= P(a) \times P(red \mid a) \times P(fox \mid a\ red) \times P(. \mid a\ red\ fox)$

$= 0.4 \times 0.27 \times 0.55 \times 0.79$
$= 0.0469.$

but P_{norm}('a red fox .') $= P^{1/n}$
$= (0.0469)^{1/4}$
$= 0.465$

$$\text{Perplexity} = \frac{1}{P_{norm}} = 2.15$$

perplexity measures how well the language model will predict the next word.

$$\text{perplexity} \propto \frac{1}{\text{Probability of sequence}}$$

Uniform distribution હોય તો perplexity વધી જાય which is not good.

likelihood = for a particular point
probability = Area in the graph over a range.

$PP(w) = \frac{1}{P(w)^{1/n}}$

$= e^{\frac{-1}{n} \log(P(w))}$

→ $= e^{\frac{-1}{n} \sum_{i=1}^{n} \log(P(w_i | w_i^{i-1}))}$

Generally used formula

↓ Chain multiplication of history (ijjat thi koi pan model ની prob. કાઢો and લીધ ૬ like prev. example)

eg: <s> I am going </s>
so no need to do P(<s> | </s>)

bigram consider: $= P(I | \langle s \rangle) \times P(am | I) \times P(going | am)$

Text Representation

So a sentence $w_1, w_2, \ldots, w_n$ can be represented as vector of prob.

$(P(w_1 | \langle s \rangle), P(w_2 | w_1), \ldots, P(w_n | w_{n-1}))$

1) Label encoding (Bag of words representation)
So let's say 10k unique words છે in vocab then we give 0 to ~~1000~~ 9999 index and word ને index થી represent કરી દઈશું.

problem: no meaning.

2) OHE

3) eg1: I am in Surat.
eg2: I am in Delhi.

rep. 1: Sentence can be represented in vector of 10000 dimension
and I ના index પર 1, am ના index પર 1, ... બાકી 0,

rep. 2: Corpus level 10k dimension vector
freq (word) in corpus થી ભરો.

word
team frequency

PAGE NO.:
DATE: / /

TF-IDF

tf_i = no. of times (frequency) of a team i in a document d.
= raw count

but prob. 0 આવે છે તો do smoothing
$f_{t,d}$ = freq. of team in document

$LF = \log(1 + f_{t,d})$

Normalization we do because freq. can be very high.

M-1) $tf = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$

M-2) $tf = \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}}$

M-3) Double normalization.

$tf = 0.5 + 0.5 \times \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}}$

General formula.

$tf = k + (1-k) * \frac{f_{t,d}}{\max f_{t,d}}$

PAGE NO.:
DATE: / /

Inverse Document Frequency (IDF)

idf weight (t) = $\frac{N}{|\{d \in D : t \in d\}|}$
N → no. of documents
↳ no. of document જેમા આ term આવે છે

but we take log and +1
= $\log(N/4) + 1$

M-2: $idf(t) = \log\left(\frac{N}{\max_{t' \in d} |\{d \in D : t' \in d\}|}\right)$
↳ max no. times this word comes in which doc
↳ ~~the~~ doc maximum docs માં જે word આવે તે

M-3: $idf = \log\left(\frac{N+1}{idf(t)+1}\right) + 1$

Finally: TF-IDF = TF * IDF

How to generate new sentences using LMs?
= let's take a ~~bi~~gram and જેમા last 2 વખત most probable જે word હોય તે લઈ લેવો. End @ </s>

problem is we can generate only 1 sentence.

So what we can do is generate 1st word a) randomly
b) based on next best probability ←

use beam search
k best prob.

eg: TF-IDF example.

1) This is a document.
2) The second document is the document.
3) The third document is over the first document.
4) The first document is there.
5) is there a document?

ans. Pre-processing: 1) word tokenize
2) Case same કરો.

now total no. of unique words are 12

tf is on single document, here document is each sentence.

$tf = \{ t_i : t \in d \}$
$f t_i d_i$

For doc 1

so tf(this) = 1
tf (is) = 1
tf (a) = 1
tf (document) = 1
tf (.) = 1

so

.	?	a	doc.	first	is	over	second	the	there	third	this
1	0	1	1	0	1	0	0	0	0	0	1
1	0	0	2	0	1	0	1	2	0	0	0
1	0	0	2	1	1	1	0	2	0	1	0
1	0	0	1	1	1	0	0	1	1	0	0
0	1	1	1	0	0	0	0	0	1	0	0

now for IDF

$idf_t = \log_2 \frac{N}{d_t} + 1$ | idf for term is across all documents

d_t → કેટલા doc માં t આવે

So 1vector જ આવશે

N = 5

df table.

.	?	a	doc.	first
4	1	2	5	2

in idf common words = low score.

so $idf_{.} = \log_2\left(\frac{5}{4}\right) + 1$

= some number

$idf_{?} = \log_2\left(\frac{5}{1}\right) + 1 = 3.32$

so tf matrix × idf matrix = tf_idf

now in testing phase, there is a new sentence so is ~~there~~ this a document?

testing માં tf બદલાય but idf remains same and then tf × idf કરવાનું

to find vocab. we can use train + test data

now if some new word comes in testing so આવશે idf = ∞ so we use.

$$idf = \log\left(\frac{N+1}{d_t+1}\right)+1$$

instead of words if we want to use n-grams તો n-gram ની freq. ના table બનાવશે in tf

eg: The boy is sitting

bigram ની example.

<s> The | The boy | boy is | is sitting

[tf ના matrix]

Note: if we are dealing with n-grams then we need n-1 start and end of sentence markers.

Also in n-grams vocab. size increases as
$V_1 + V_2 + V_3 + V_4$ → $(w_1 w_2 w_3 w_4)$ ની frequencies.
↳ one word freq.

Also like words tf-idf can be done on character level that too in 2 types:

1) without spaces
2) with spaces

$$|V_{word}| >>> |V_{char}|$$

↳ 26 + punctuations only

Pointwise mutual information ← $PMI(x, y) = \log\left(\frac{P(x,y)}{P(x) \cdot P(y)}\right)$

When we combine word and character level tf-idf we will get best accuracy.

★ TF-IDF doesn't take position into accounts (n gram થી thoda kinda)

COLLOCATION

So we can maintain 2 lists, 1 to find out how many words preceeds 'I' (કોઈ પણ word) and other to find how many succeeds.

we can do like V × V matrix બનાવી શકીએ and જે ના પછી words પહેલા આવે તો 1 else 0 (case of preceed list)

or V × V matrix and count of words coming before a word.

$$P(x, y) = \frac{C(x,y)}{C(x)}$$ } conditional distribution

$$P(x,y) = \frac{C(x,y)}{\text{total no. of bigrams}}$$

$$P(x) = \frac{C(x)}{N}$$

} No smoothing

so for words like 'San Fransico' માં $P(x,y) \neq P(x).P(y)$ as mostly સાથે જ આવે. (related to each other)

main problem of n-gram = out of vocabulary (OOV)

Byte pair Encoding (BPE)

See the non-overlapping most frequently occuring pair.

and pair को एक नए character which is not in vocab से replace करो.

also if freq. में tie then prefer original chars.

Store all the transformations in lookup table.

eg: aaabdaaabac

→ ZabdZabac (Z = aa)

→ ZYdZYac (Y = ab)

→ XdXac (X = ZY)

Stop here as no > 1 freq.

S-1: Count all word level frequencies.

h ug : 10

p ug : 5

p un : 5

b un : 4

h ugs : 5

S-2: Split character wise, for each word.

h ug , p ug , ---.

Byte is represented as unicode generally.

S-3: Vocabulary is of all appearing unique chars.

V = { h, u, g, p, b, n, s }

S-4: Find char pair frequency for all character pair

hu : 10

ug : 20

pu : 17

un : 16

bu : 4

gs : 5

S-5: Merge the ones (char pairs) with highest frequency.

h ug : 10

p ug : 5

p u n : 12

b u n : 4

h ug s : 5

merge = { (u, g) : ug }

include this char pair in vocabulary

V = { h, u, g, p, b, n, s, ug }

now like Step-4, 5

Stop: nothing left to combine

This is Sentence Piece → Byte Pair

iteration-2: un = 16 = max.
& V માં add કરો છે.

h ug : 10
p ug : 5
p un : 12
b un : 4
h ug s : 5

then hug : 15 pug : 5 pun : 12 bun : 4
ug s : 5

So now hug લેવો
(h, ug) : {hug}

now આગળ iteration થઈ કરીયે.

why useful: if a new word out of vocabulory comes

e.g.: thug આવે તો t, hug split થશે (longest possible merged in vocab find કરીને split કરે).

t, hug = unk hug
↖ unknown

Word Piece

⇒ Change-1: we also show કે split કર્યા પછી પહેલું કે નહિ

Walking → walk + ##ing
↖ Comes in middle

h u g → h ##u ##g

⇒ Change-2: to merge we don't just see max freq. but we see score.

h ##u ##g
↳ hu ##g માં merge થશે so # વચ્ચેની દેખાશે

This score penalises good frequent subwords

$$Score = \frac{P(A, B)}{P(A) \times P(B)}$$

Chinese માં whitespace ના હોય so it is called text segmentation.

And is text splitting = english માં જ્યારે split કરીએ.

$$P(A,B) = \frac{C(A,B)}{C(A) \times C(B)}$$

if A, B generally comes together then score = higher
if combination is rare then score = lower.

eg: I am going to class.
_I _am _go ##ing _to _Class.

N-gram જેવું જ આવે skip-gram so we need to skip

This captures long context better
eg: I am going home

So I < going, home
am – home
going – I
home < I, am

Total skip bigrams possible = $(n-2)(n-3) + 2(n-2)$

now for skip-trigrams we can do:
1) 3 unigrams all on diff spaces.
2) 1 unigram પછી skip પછી 1 bigram.
3) 1 bigram પછી skip પછી 1 unigram.

P (skip trigram) = Count (x, y, z ભેગા જોઈએ in context) / Count (x, y in context).

$$P(x,y,z) = \frac{C(x,y,z)}{C(x,y)}$$

m-2: $P(x,y,z)$ = Just like word-piece.

For filling the collocation table we use bigram + skip-gram
So code માં just put 1 for all the words in the particular sentence other than this word.

So I am going home

So I, going, home બધા માં 1-1 મૂકી દેવો.

now method-2 છે કે instead of putting 1 we use TF-IDF of all and એના થી ભરો.

Negative Sampling

So similar words ના એ છે જે context માં આવે છે and for this we know these words which are in V−W → similar words, words in context ગણાવી |W| ના proportion માં sampling કરો ને એને ઓછી probability આપો like 0.000001 so neural network માં this will help

So કોના સાથે નહીં આવશે તે આના થી ખબર પડે.

PAGE NO. :
DATE : / /

so tf-idf વી ભરો colocation table and ~~[scribble]~~ પછી negative-sampling કરીને એના small value મૂકી દેવો.

* also ~~to~~ each word માટે આપણને 1 vector મળશે

I [_ _ _ _ _ _] → vector

so to find similarity we can use 2 methods:

1) Cosine similarity (value ↑ તો similar)
2) Euclidean distance. (value ↓ તો similar)

Steps:-

1) Tokenization
2) Preprocessing techniques
   - Stopword removal
   - Lemmatization
   - Stemming
   - Normalization.
3) PSA and FST → morphology of words
4) N-gram modelling → freq. distribution.
5) Smoothing techniques
   - add 1
   - add k → add type
   - backoff
     - Stupid
     - Katz
     - deleted (linear) interpolation.
   - Good turing
   - Kneser Ney

prediction of next word
- Greedy
- Beam

PAGE NO. :
DATE : / /

6) Text representation Schemes
   - TF-IDF
     - word
     - Character.
   - PMI

   - Subword
     - BPE
     - Word piece
     - Sentence piece

7) Collocation
   - TF-IDF
   - Negative Sampling.

Naive Bayes (features are words here)

Given a new document, you have to predict the author

$D_1 \to A_1$, $D_2 \to A_2$, $D_3 \to A_3$, ..., $D_n \to A_n$

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

we don't want to model feature dependence so we remove denominators.

$$P(x_1 x_2 x_3 | Y) = P(x_1|Y) \cdot P(x_2|Y) \cdot P(x_3|Y)$$

Prior = $P(y_k)$

Posterior = $P(y_k|x)$

$P(D_i|A_i) = P(w_1|A_i) \times P(w_2|A_i) \times \dots \times P(w_n|A_i)$

D_i has words w_1 to w_n

So create a dictionary

$$\hat{A} = \underset{A_i}{\text{argmax}} \; P(D|A_i) \times P(A_i)$$

$$P(A_i) = \frac{\text{total no. of docs authored by } A_i}{\text{total no. of docs}}$$

$P(D|A_i) = P(w_1|A_i) \times P(w_2|A_i) \times \dots \times P(w_n|A_i)$

Now if A_i has d docs. and w_i word is in m documents (freq. count એટલા વાર, કે કેટલા doc માં આવ્યો)

$$P(w_i|A_i) = \frac{m}{d}$$

We can also find $P(w_i|A_i)$ in a diff way

$$P(w_i|A_i) = \frac{C(w_i, A_i)}{\text{all words written by } A_i}$$

$P(w_n\, w_{n-1} | w_{n-2}) = P(w_{n-1}|w_{n-2}) \times P(w_n | w_{n-1}\, w_{n-2})$

\# = Can we have n-gram features?

Yes

Here replace $w_i \rightarrow w_i, w_{i-1}$

$P(f_1, \dots, f_m|y_k) = P(w_1|y_k) \times P(w_2|y_k) \times \dots \times P(w_n|y_k)$

$P(w_1 w_2|y_k) \times P(w_2 w_3|y_k) \times \dots$

અહીં બધાં $P(y_k)$ multiply થશે.

and $P(w_i w_j | y_i)$ તો થાળી લેવો કે bigram કેટલા docs માં આવે / total docs. written by that author.

problems: tf-idf માં add k smoothing (બોલાવો) or idf માં $\log\left(\frac{N+1}{d+1}\right)+1$ કરો.

Sentence Classification

So we are using tf-idf as features but these are generic features

To add extra feature we can add 3 columns like.

no. of +ve words | no. of -ve words | no. of neutral words

We will have list of +ve, -ve and neutral words and/ each word in a sentence ની list માં find કરો and maintain the count