

# Implementera en ordprediktor

Anton Johansson & Nathalie Haghshenas

# Uppgiftsbeskrivning för ordprediktorn

- Ordprediktor likt de som finns i telefoner, uppdateras vid varje knapptryck
- Eventuellt föreslå rättningsar och/eller lära sig nya uttryck användaren nyttjar



“En bra ordprediktor använder ord-n-gram-sannolikheter för att kunna föreslå den troligaste kompletteringen”

# Bakgrund 1 - Statistiska språkmodeller & Markov-antagande

- Antar att ord är slumpmässiga och genereras av stokastisk process
- Genererar sannolikheter av ords förekomst givet kontext (föregående ord)
- Kan användas för prediktion, stavningskontroll, översättning m.m.
- Begränsar kontext genom Markov-antaganden
- “Assume that a word only depends on the previous couple of words”

$P(I \text{ really like ants}) =$

$P(\text{ants}|I \text{ really like})P(\text{like}|I \text{ really})P(\text{really}|I)P(I)$

N-gram-modeller tar de föregående n-1 orden i beaktande vid förutsägande av nästa ord:

**Unigram:**  $P(I)P(\text{really})P(\text{like})P(\text{ants})$

**Bigram:**  $P(I)P(\text{really}|I)P(\text{like}|\text{really})P(\text{ants}|\text{like})$

$$P(w_i|w_1, \dots, w_{i-1}) = P(w_i|w_{i-n+1}, \dots, w_{i-1})$$

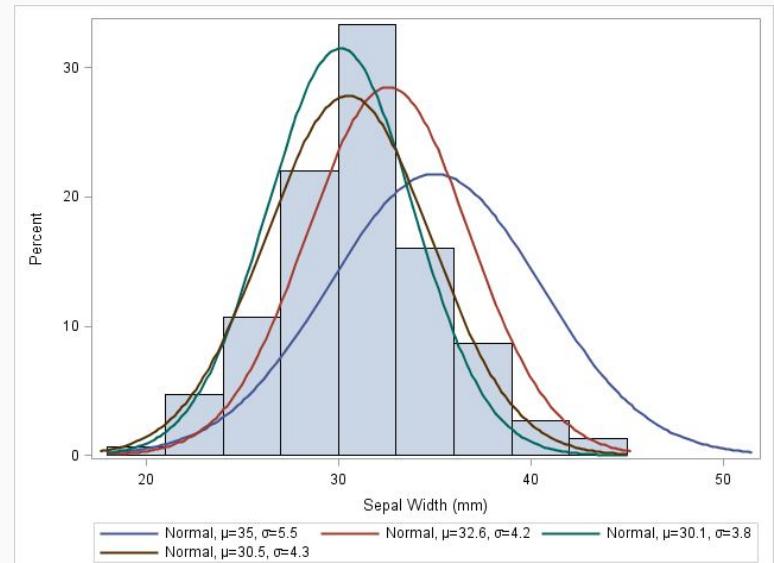
\*Kallas även stokastisk/probabilistisk språkmodell

# Bakgrund 2 - MLE

Beräkna sannolikheten av olika ords förekomst i kontext

$$P(\text{to}|\text{like}) = C(\text{like to})/C(\text{like})$$

$$P(w_i | w_{i-n+1}, \dots, w_{i-1}) = \frac{c(w_{i-n+1}, \dots, w_i)}{c(w_{i-n+1}, \dots, w_{i-1})}$$



# Bakgrund 3 - Felstavning & Levenshtein

- Ger en “kostnad” för insättning, borttagning eller ersättning.
- Manipulerar ett ord mot mål-ordet
- Summan av kostnaderna är avståndet

	m	e	i	l	e	n	s	t	e	i	n	
l	0	1	2	3	4	5	6	7	8	9	10	11
e	1	1	2	3	3	4	5	6	7	8	9	10
v	2	2	1	2	3	3	4	5	6	7	8	9
e	3	3	2	2	3	4	4	5	6	7	8	9
e	4	4	3	3	3	3	4	5	6	6	7	8
n	5	5	4	4	4	4	3	4	5	6	7	7
s	6	6	5	5	5	5	4	3	4	5	6	7
h	7	7	6	6	6	6	5	4	4	5	6	7
t	8	8	7	7	7	7	6	5	4	5	6	7
e	9	9	8	8	8	7	7	6	5	4	5	6
i	10	10	9	8	9	8	8	7	6	5	4	5
n	11	11	10	9	9	9	8	8	7	6	5	4

- Ord som inte förekommer i vårt corpus,  $C(w) = 0$  kan vi anta är felstavade\*
- Tillsammans med dess sannolikhet att förekomma i kontexten kan vi ge användare förslag på “rätt” ord

\*Alternativt låtit användaren utöka corpus med ordet

# Snabbt om Hypotes

- Trigram > bigram
- Textcorpus begränsande
- Troligtvis innehåller slang
- Större träningscorpus → mer träffsäker prediktion

# Implementation 1: Återanvänt

- BigramTrainer → TrigramTrainer
- Utökades för att beräkna trigramsannolikheter och skapa vår språkmodell utifrån ett corpus
- Beräknar uni-, bi- och trigrams-sannolikheter genom MLE
- BigramTester → TrigramTester
- Utökades för att kunna utvärdera språkmodellen.
- TrigramTester beräknar kors-entropin mellan modell och en testmängd
- Summera modellens entropi för varje enskilt ord.

# Implementation 2: Nyskapat

- Prediktorn = Predictor.py.
- Enstaka likheter i struktur med Generator.py från labb 2
- GUI i klass “GuiWindow”. Knappttryck binds till att kalla på särskilda metoder i prediktorn. Deras innehåll samt användarens inmatningspanel uppdateras efter tangent-tryck eller knappttryck
- Prediktor i klass “Predictor”. Innehåller metoder för att (förutom att läsa in modell) generera nya ord beroende på kontexten. Olika metoder beroende på antalet tidigare ord (Backoff) och huruvida användaren skriver på ett ord eller inte

Backoff - en teknik där man, om en sannolikhet från en högre ordnings n-gram-modell inte är tillgänglig, väljer att använda sannolikheter från en n-gram-modell av lägre ordning

# Data - “Mobile text dataset”

Hämtat från studien “Mining, analyzing, and modeling text written on mobile devices” av Kristensen & Vertanen, 2021

En studie från Cambridge & Michigan Technical University där forskarna försöker tillgängliggöra data för studier om hur vi skriver på telefoner och andra små enheter, genom web scraping och utsållning genom enheters signaturer

Resultatet är flertalet txt-filer innehållandes inlägg som användare gjort på diverse forum/medier från olika typer av enheter, bl.a. Tränings- utvecklings- och evaluerings-set för språkmodellering samt en intressant diskussion gällande användarvanor och språkmodellers (bristande) träffsäkerhet

[Vertanen K, Kristensson PO. Mining, analyzing, and modeling text written on mobile devices. Natural Language Engineering. 2021;27\(1\):1-33. doi:10.1017/S1351324919000548](#)

# DEMO

The screenshot shows a Mac desktop environment with a Python code editor open in the foreground. The code editor is displaying a file named `Predictor.py` which contains code for a word predictor using bigram models. A floating window titled "Word Predictor" is overlaid on the code editor, showing the text "Input text here" and three suggestions: "seen", "if", and "as". The code editor's sidebar shows other files in the project, including `TrigramTester.py`, `Trigramtrainerv3.py`, `NgramTrainer.py`, `smallmodel3.txt`, `halvering.py`, and `Planering.txt`. The bottom status bar indicates the file is "Projektarbete". The desktop dock at the bottom features various application icons.

```
if wordstoreturn:
    # Sort the suggestions and remove values from the bottom so that only "wordstoreturn" many words are in the list
    sorted_suggestions = sorted(suggestions, key=lambda x: x[1], reverse=True)
    suggested_words = [word for word, _ in sorted_suggestions[:wordstoreturn]]
    print("SUGGESTED WORDS FROM BIGRAM_COMPLETE IF WORDS TO RETURN: " + str(suggested_words))

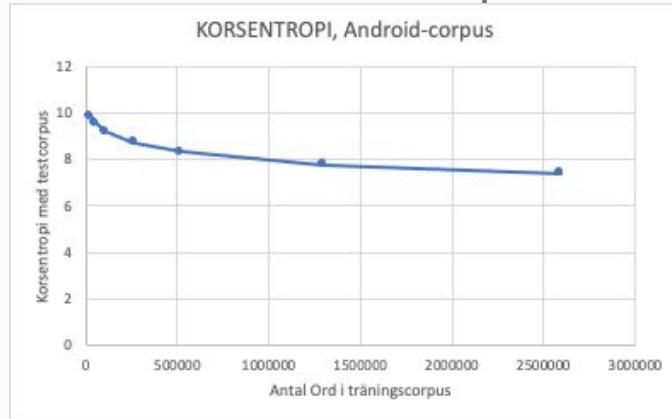
# Sort the suggestions based on the score in descending order
sorted_suggestions = sorted(suggestions, key=lambda x: x[1], reverse=True)
# Extract the top three words with the highest scores
top_suggestions = sorted_suggestions[:3]
suggested_words = [word for word, _ in top_suggestions]
print("SUGGESTED WORDS FROM BIGRAM_COMPLETE: " + str(suggested_words))

return suggested_words

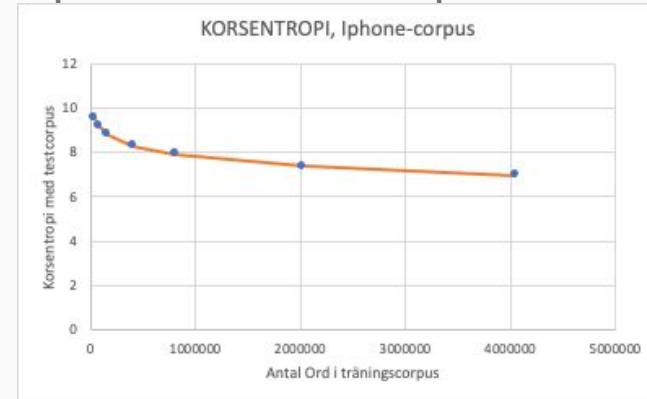
except KeyError:
    # Handle KeyError by returning unigram completion
    print("FEGAR UR COMPLETE BIGRAM")
```

# Resultat - Korsentropi med test-set i TrigramTester

Android-andvändar-corpus:



Iphone-användar-corpus



- Modellen blir mer träffsäker med större corpus
- Fortfarande relativt hög entropi, beror med stor sannolikhet (delvis) på testprogrammet

# Slutsats

Modellen följer de förväntningar vi har (om än med hög entropi) på statistiska språkmodeller

Antagandet om ord som slumpmässiga, stokastiskt uppkommande fenomen ignoreras  
grammatik och begränsar starkt prediktorns förmåga att ge användbara rekommendationer

Bättre lösning med Levenshteinavstånd eller annan metod för  
stavningskontroll/ordkomplettering bör finnas