n-gram Modelling

Katja Markert

Institut für Computerlinguistik
Uni Heidelberg
markert@cl.leeds.ac.uk

November 24, 2022

Bis jetzt und heute

- Bis jetzt: Einzelne Wörter erkennen, vergleichen und zählen. Tokenisierung und Wortverteilungen.
- Jetzt: Sprachmodelle: Verteilungen über Wortsequenzen, Sätze, Texte
- *n*-gram Modelle: bestimmte Art von Sprachmodellen

Übersicht

Übersicht

Sprachmodelle

Sprachmodell/language model

- Ein Sprachmodell ist eine Wahrscheinlichkeitsverteilung über Wortsequenzen
- Werden trainiert, indem Sie Textkorpora in einer (oder mehreren) Sprache “lesen” (und eventuell darauf bestimmte Aufgaben erlernen)
- Was passiert mit Sequenzen, die nicht vorher gelesen wurden?

Beispiele:

- ***n*-gram Modellle**
- BERT
- GPT-2/GPT-3

Probabilistische Sprachmodellierung und Wortvorhersage

Wir wollen herausfinden: Ist

$$P(\text{He drank white wine})$$

wahrscheinlicher als?

$$P(\text{He drank white rum})$$

wahrscheinlicher als?

$$P(\text{He ate white wine})$$

Wie wahrscheinlich ist es, dass *wine* dem Anfang *He drank white* folgt?

Anwendungen

- Spracherkennung
- Handschriftenerkennung
- Kontextsensitive Rechtschreibkorrektur
- Textvervollständigung: texting
- Beurteilung von generierten Sprach- oder Textsequenzen in Summarization, Machine Translation ...

Viele Korrekturprogramme sowie Generierungsprogramme nutzen Wahrscheinlichkeiten von Zeichenketten, um ungewöhnliche Sequenzen zu erkennen und wahrscheinlichere Sequenzen vorherzusagen

Formalisierung

Wahrscheinlichkeit $P(t)$ für Wortsequenz
$t = (i_1 = w_1, i_2 = w_2, \ldots, i_m = w_m) = (w_1, w_2, \ldots w_m)$ nach Kettenregel:

$$\begin{aligned} P(t) &= P(i_1 = w_1, i_2 = w_2, \ldots, i_m = w_m) \\ &= P(i_1 = w_1) P(i_2 = w_2 | i_1 = w_1) \ldots P(i_m = w_m | i_1 = w_1, \ldots, i_{m-1} = w_{m-1}) \\ &= \prod_{j=1}^{m} P(i_j = w_j | i_1 = w_1, \ldots, i_{j-1} = w_{j-1}) \end{aligned}$$

$$P(i_j = w_j | i_1 = w_1, \ldots, i_{j-1} = w_{j-1}) = \frac{P(i_1 = w_1, \ldots, i_j = w_j)}{P(i_1 = w_1, \ldots, i_{j-1} = w_{j-1})}$$

Gleichwertigkeit von Sequenzmodellierung und Vorhersage

n-gram Modelle

Markovannahme

w_j hängt nur von den Tokens unmittelbar davor ab.

- 0te Ordnung = Unigramme

$$P(w_j|w_1 \dots w_{j-1}) \approx P(w_j)$$

- Erste Ordnung = Bigramme

$$P(w_j|w_1 \dots w_{j-1}) \approx P(w_j|w_{j-1}) = \frac{P(w_{j-1}w_j)}{P(w_{j-1})}$$

- $n-$te Ordnung parallel
- Meist benutzt (Zweite Ordnung, Trigramme)

$$P(w_j|w_1 \dots w_{j-1}) \approx P(w_j|w_{j-2}w_{j-1}) = \frac{P(w_{j-2}w_{j-1}w_j)}{P(w_{j-2}w_{j-1})}$$

Damit Vereinfachung der Kettenregel

Wahrscheinlichkeit $P(t)$ für Wortsequenz
$t = (i_1 = w_1, i_2 = w_2, \ldots, i_m = w_m) = (w_1, w_2, \ldots w_m)$ nach Kettenregel und dann Markovannahme mit Bigrammen:

$$\begin{aligned} P(t) &= P(w_1, w_2, \ldots, w_m) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_m|w_1, \ldots w_{m-1}) \\ &\approx P(w_1)P(w_2|w_1)P(w_3|w_2)\ldots P(w_m|w_{m-1}) \end{aligned}$$

Trigram?

Vorteile und Nachteile

Wenn Sie sich folgende Beispiele anschauen, bei denen Sie jeweils X vorhersagen sollen, welche Vor- und Nachteile sehen Sie dann für n-gram Modelle?

- He did not like red wine, so he drank white X
- He did not like brown rum, so he drank white X
- He drank white X

Warum funktionieren Sie oft dennoch?

The **man** that I was watching without pausing to look at what was happening down the street, and quite oblivious to the situation that was about to befall him confidently **strode** into the center of the road.

- Das passiert nicht oft (im Englischen)
- Collins (1996): 74% der Dependenzen in Penn Treebank gehören zu einem angrenzenden Wort

Collins, Michael John. *A new statistical parser based on bigram lexical dependencies.* Proceedings of the 34th annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics, 1996.

Außerdem

Viele Konversationen sind konventionell und enthalten Redundanz

- One skinny latte!
- Einen Kaffee bitte
- Can I have an extension?
- Sorry for any . . . caused!

Zwischenfazit

- Sprachmodelle sind Wahrscheinlichkeitsverteilungen über Sätze
- *n*-gram Modelle eine Art von Sprachmodellen
- basierend auf recht kurzer Historie
- In Anwendungen oft kombiniert mit anderen Restriktionen: wie häufigen Rechtschreibfehlern und Minimum Edit Distance
- * Jurafsky und Martin, Kapitel 3, 3rd/online edition
- Übungsblatt 5

Übersicht

Generelles Framework

- Nehme Korpus C
- Nimm den größten Teil des Korpus (oft 80%) für das Trainieren des Modells: **Trainingsset**
- Nimm einen kleineren Teil des Korpus (oft 10%), um eventuelle Metaparameter zu schätzen: **Development Set/Held-Out Set**
- Nimm einen kleineren Teil des Korpus (oft 10%), um das Endmodell zu testen: **Test Set**
- Das Testset darf während der Modellentwicklung nicht benutzt werden. Warum?

Wir kümmern uns jetzt um das Training eines n-gram Modells, sprich wie bekomme ich die Wahrscheinlichkeiten.

Wie leite ich ein *n*-gram Modell in der Praxis her?

- Starte mit Vokabular V (word types):
  - alle types in einem (Trainings)Korpus;
  - oder die k häufigsten;
- Wähle Modelltyp:
  - Nullte Ordnung= unigram
  - Erste Ordnung = bigram
  - Zweite Ordnung = trigram
- Parameterschätzung e.g. $P(X_1 = a, X_2 = b)$ für zwei Worte a,$b \in V$

Wie viele Parameter muss ich schätzen?

Maximum likelihood Schätzung (MLE)

Relative Häufigkeit = **maximum likelihood Schätzung** auf Korpus der Grösse N.

Wir bezeichnen nun die Positionen mit Variablen X, X_1, X_2, X_3 etc.

$$P(X = a) = \frac{f(a)}{N}$$

(unter Annahme von Positionsunabhängigkeit)

$$\begin{aligned} P(X_2 = b | X_1 = a) &= \frac{P(X_1 = a, X_2 = b)}{P(X_1 = a)} \\ &= \frac{f(X_1 = a, X_2 = b)}{f(X_1 = *, X_2 = *)} \cdot \frac{f(X_1 = *, X_2 = *)}{f(X_1 = a, X_2 = *)} \\ &= \frac{f(X_1 = a, X_2 = b)}{f(X_1 = a, X_2 = *)} \end{aligned}$$

Problem, wenn wir keine Bigramme über Satzgrenzen hinaus benutzen wollen:

$f(X_1 = a, X_2 = *)$ ist nicht gleich $f(X = a)$. Warum nicht?

Für Sätze und Korpora

Setze $< s >$ als spezielles Zeichen für Satzanfang und $< /s >$ als spezielles Zeichen für Satzende. Damit gilt für ein Bigramm-Modell:

$$
\begin{aligned}
P(s) &= P(< s > w_1 w_2 \dots w_m < /s >) \\
&= P(w_1 | < s >) P(w_2 | < s > w_1) \dots \\
&\quad P(w_m | < s > w_1, \dots w_{m-1}) \cdot P(< /s > | < s > w_1 \dots w_m) \\
&\approx P(w_1 | < s >) \cdot P(w_2 | w_1) \cdot P(w_3 | w_2) \dots P(< /s > | w_m)
\end{aligned}
$$

Für Trigramme füge zwei Anfangstokens $< s >< s >$ ein.

Vorteile der Satzanfangs- und Endtokens

Damit hat jedes Unigramm immer auch ein nachfolgendes “Wort” und damit gilt

$$
\begin{aligned}
P(X_2 = b | X_1 = a) &= \frac{P(X_1 = a, X_2 = b)}{P(X_1 = a)} \\
&= \frac{f(X_1 = a, X_2 = b)}{f(X_1 = *, X_2 = *)} \cdot \frac{f(X_1 = *, X_2 = *)}{f(X_1 = a, X_2 = *)} \\
&= \frac{f(X_1 = a, X_2 = b)}{f(X_1 = a, X_2 = *)} \\
&= \frac{f(X_1 = a, X_2 = b)}{f(X_1 = a)} \\
&= \frac{f(a, b)}{f(a)}
\end{aligned}
$$

Und für Trigramme

$$P(X_3 = c | X_1 = a, X_2 = b) = \frac{f(a, b, c)}{f(a, b)}$$

n-gram Modell: Beispiel

Bigram Model

$$P(<\mathrm{s}>\mathrm{I\ drank\ white\ wine}<\!/\mathrm{s}>) \approx$$
$$P(\mathrm{I}|<s>)P(\mathrm{drank}|\mathrm{I})P(\mathrm{white}|\mathrm{drank})P(\mathrm{wine}|\mathrm{white})$$
$$\cdot P(<\!/s>|\mathrm{wine})$$

- Wahrscheinlichkeiten, dass bestimmte Worte Sätze beginnen oder beenden, werden ebenfalls modelliert
- $<s>$ nicht Teil des Vokabulars und des Tokencounts, $</s>$ schon.

Minibeispiel an Tafel

Trainingskorpus:

$< s >$ I am Sam $< /s >$
$< s >$ Sam I am $< /s >$
$< s >$ I do not like green eggs and ham $< /s >$

Berechne Wahrscheinlichkeit des Testsatzes: $< s >$ Sam I am Sam $< /s >$

Beispiele

Welche der folgenden Sequenzen sind wahrscheinlicher in einem Unigrammodell auf Brown/BNC? In einem Bigrammodell?

1. $< s >$ I drank white rum . $< /s >$
2. $< s >$ I drank white water . $< /s >$
3. $< s >$ I drank white wine . $< /s >$
4. $< s >$ I drank white milk . $< /s >$

Was passiert wohl in einem Trigram-Modell?

Teil eines Bigram-models (cased) des Brown Korpus

Erstes Wort in Zeile.

	I	drank	white	wine	milk	rum	water	.	$< /s >$	
$< s >$	1527	0	0	0	0	0	2	87	0	57, 484
I	0	5	0	0	0	0	0	3	0	5840
drank	0	0	0	0	0	0	0	1	0	19
white	0	0	0	5	1	0	0	8	0	254
wine	0	0	0	0	0	0	0	6	0	68
milk	0	0	0	0	0	0	0	4	0	44
rum	0	0	0	0	0	0	0	0	0	2
water	0	0	0	0	0	0	0	0	61	436
.	0	0	0	0	0	0	0	0	49, 491	49,754

Brown corpus hat 1, 014, 312 tokens mit 57, 484 Sätzen.

Teil eines Bigram-Modells (cased) auf dem BNC

Erstes Wort in Zeile

	I	drank	white	wine	milk	rum	water	.	$< /s >$	
$< s >$	231, 411	0	42	7	9	0	37	1930	0	6, 026, 284
I	3855	92	0	0	0	0	4	361	1879	861, 382
drank	1	0	1	6	3	4	16	73	3	1310
white	14	0	22	262	1	6	103	789	90	18, 317
wine	16	0	0	1	0	0	0	859	92	5537
milk	14	0	2	0	0	0	0	491	109	4316
rum	2	0	0	0	0	0	0	48	6	332
water	36	0	0	0	1	0	2	4241	276	31, 091
.	166, 168	0	8	0	0	0	11	1701	4 450 610	4 713 133

BNC hat 110, 691, 482 Tokens und 6, 026, 284 Sätze.

Eigenschaften

- Im Idealfall führt größeres n zu besserer Diskrimination
- Aber: mehr Null-Schätzungen $\longrightarrow$ führt zu Totalversagen
- Beispiel: Shakespeare als Korpus
  - $N = 884,647$ tokens, $|V| = 29,066$
  - 300,000 verschiedene Bigramme von $|V|^2 = 844m$ möglichen
  - 99.96% in Bigramtabelle sind Nullen
- Größeres Korpus kann besser funktionieren als kleineres
- Aber auch in einem großen Korpus werden wir nicht alles vorfinden (Chomsky, Zipfsches Gesetz)
- Mit Maximum Likelihood können wir nur “nacherzählen” $\longrightarrow$ Smoothing nötig

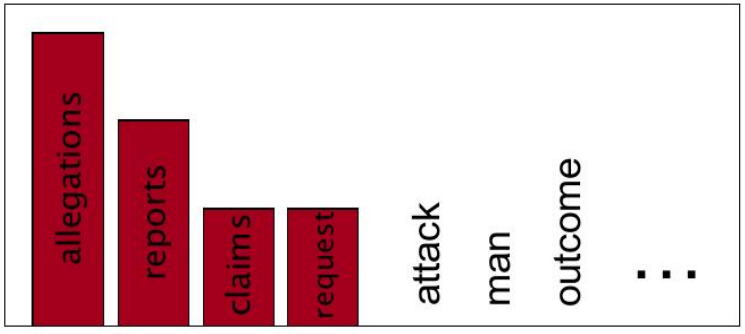
Intuition zu Smoothing (von Dan Klein)

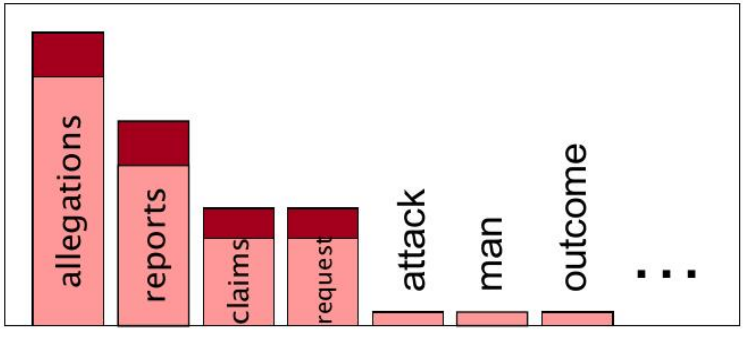
Smoothing

Reevaluation von Nullwahrscheinlichkeiten sowie *n*-grams mit niedrigen Wahrscheinlichkeiten, wobei ihnen höhere Werte zugewiesen werden.

Gegeben

$P(w \mid denied, the)$
3 allegations
2 reports
1 claims
1 request
7 total





Smoothingmethode I: Laplace Smoothing (1851)

$$P_{+1}(w_i) = \frac{f(w_i) + 1}{N + |V|}$$

$$P_{+1}(w_i|w_{i-1}) = \frac{f(w_{i-1}, w_i) + 1}{f(w_{i-1}) + |V|}$$

- $|V|$ ist Größe Vokabular ($V = \{w : f(w) > 0\}$ + UNK)
- N ist Korpusgröße
- Bayesian Schätzung mit uniformer a priori Wahrscheinlichkeit

Allgemeine Version für eine Zufallsvariable

Sei X eine Zufallsvariable, die M Werte annehmen kann. Wir nennen den Werteraum Ω_X. Ich schätze (durch N Experimente) die Wahrscheinlichkeiten der Werte. Sei $f(x)$ die Zählung für einen möglichen Wert x der Zufallsvariable. Laplace-Smoothing heisst dann:

$$P_{+1}(X = x) = \frac{f(x) + 1}{N + M}$$

Damit ist wieder eine Wahrscheinlichkeitsverteilung gegeben denn:

$$\sum_{x \in \Omega_X} P_{+1}(x) = \sum_{x \in \Omega_X} \left(\frac{f(x) + 1}{N + M}\right) = \frac{1}{N + M} \cdot \left(\sum_{x \in \Omega_X} f(x) + \sum_{x \in \Omega_X} 1\right) = 1$$

Probleme im Laplace-Smoothing

Wir nehmen an, wir haben ein Vokabular von 10,000 Worten und einen Trainingskorpus von 1m Wörtern. Das Unigram *data* kommt zehnmal vor. Achtmal ist das nächste Wort *sparseness*.

$$P_{MLE}(sparseness|data) = ??$$

$$P_{+1}(sparseness|data) = ???$$

Quick fix: Lidstone's Gesetz

$$P_{+\lambda}(w_i) = \frac{f(w_i) + \lambda}{N + \lambda|V|}$$

$$P_{+\lambda}(w_i|w_{i-1}) = \frac{f(w_{i-1}, w_i) + \lambda}{f(w_{i-1}) + \lambda|V|}$$

$$P_{+\lambda}(w_1, w_2 \ldots w_n) = \frac{f(w_1, w_2 \ldots w_n) + \lambda}{N + \lambda|V|^n}$$

- $0 < \lambda < 1$
- Wie soll man λ schätzen?
- Immer noch vokabularabhängig
- Für *n*-gram Modelle ist Laplace/Lidstone-Smoothing ungeeignet
- Wenn man weniger Nullen hat, dann kann es benutzt werden
  $\longrightarrow$ nächste Woche für Textklassifikation

Smoothing-Methode II: Interpolations-Smoothing

Grundidee

Wenn wir die langen *n*-gramme nicht oder nicht oft genug sehen, dann vielleicht die kurzen

Interpolation: Kombiniere Unigram, Bigram und Trigram-Modelle.
Einfachste Version:

$$\begin{aligned} P_{intpol}(w_i|w_{i-2}w_{i-1}) &= \lambda_1 P_{MLE}(w_i|w_{i-2}w_{i-1}) \\ &+ \lambda_2 P_{MLE}(w_i|w_{i-1}) \\ &+ \lambda_3 P_{MLE}(w_i) \end{aligned}$$

mit $\sum_{j=1}^{3} \lambda_j = 1$ und $0 < \lambda_j < 1$ für $j = 1, 2, 3$

Woher bekomme ich die Lambdas?

Teile Korpus in

1. Trainingskorpus. Für Originalschätzungen. Größter Teil des Korpus
2. Held-out Korpus/Development Set.

Fixiere *n*-gram Wahrscheinlichkeiten auf Trainingskorpus. Suche Lambdas, die die Wahrscheinlichkeiten auf Held-Out Korpus maximieren.

Maximiere (*m* Länge des held-out Korpus)

$$P_{interpol}(w_1, \ldots, w_m) = \prod_{i=1}^{m} P_{interpol}(w_i | w_{i-2}, w_{i-1})$$

Smoothing-Methode III: Absolute Discounting

Church und Gale (1991):

- Teilten 44m Korpus (AP newswire) gleichmässig in Training und held-out
- Für jedes Bigram im Trainingskorpus, schaue die wirkliche Häufigkeit im Held-out Korpus an
- Berechne Mittelwerte für Häufigkeitsklassen

Training	Held-Out
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Absolute Discounting

$$P_{AD}(w_i|w_{i-1}) = \frac{max(f(w_{i-1}, w_i) - d, 0)}{f(w_{i-1})} + \lambda_{w_{i-1}} P(w_i)$$

- Eine Kombination von einfachem Discounting und Interpolation.
- d (normalerweise zwischen 0 und 1), wird auf dem held-out Korpus geschätzt; könnte im vorherigen Beispiel 0.75 sein
- Damit sich alles auf 1 aufaddiert, muss gelten:

$$\lambda_{w_{i-1}} = \frac{d}{f(w_{i-1})} |w : f(w_{i-1}, w) > 0|$$

  wobei der zweite Multiplikator: Anzahl der Worte, die auf w_{i-1} folgten (im Training) = Anzahl der Worttypen, für die wir einen Discount vorgenommen haben.
- Ist das Unigram am Ende der beste Backoff? Nein: Kneser-Ney (Optional)

Smoothing-Methode IV: (Modified) Kneser-Ney Smoothing (Optional)

Sollte $p(w_i)$ einfach wie immer geschätzt werden?

- Vervollständige *I don't want to visit . . . . . .*
- Vielleicht ist *Francisco* häufiger als *Leavenworth*. Ist das trotzdem beste Ergänzung?

Benutze statt $p(w)$ besser $p_{Continuation}(w)$: Wie wahrscheinlich ist w als neue Fortsetzung? Nach wie vielen verschiedenen Wörtern (Types) kommt w vor?

Kneser-Ney Smoothing (Optional)

Die verschiedenen Wort types, denen w folgte:

$$p_{continuation}(w) \propto |w_{i-1} : f(w_{i-1}, w) > 0|$$

Muss normalisiert werden durch alle Bigramtypen:

$$p_{continuation}(w) = \frac{|w_{i-1} : f(w_{i-1}, w) > 0|}{|\{(w_{j-1}, w_j) : f(w_{j-1}, w_j) > 0\}|}$$

oder Normalisierung durch die Zahl aller Worte, die einem anderen vorangehen

$$p_{continuation}(w) = \frac{|w_{i-1} : f(w_{i-1}, w) > 0|}{\sum_{w'} |f(w'_{i-1}, w') > 0|}$$

Kneser-Ney Smoothing (Optional)

$$P_{KN}(w_i|w_{i-1}) = \frac{max(f(w_{i-1}, w_i) - d, 0)}{f(w_{i-1})} + \lambda_{w_{i-1}} P_{continuation}(w_i)$$

λ ist eine Normalisierungskonstante, damit sich alles wieder auf 1 aufaddiert:

$$\lambda_{w_{i-1}} = \frac{d}{f(w_{i-1})} |w : f(w_{i-1}, w) > 0|$$

wobei der zweite Multiplikator: Anzahl der Worte, die auf w_{i-1} folgen können = Anzahl der Worttypen, für die wir einen Discount vorgenommen haben.

Kneser-Ney Smoothing (Optional)

Extras:

- Wenn man Kneser-Ney auf höhere n-gramme anwendet, dann verkompliziert sich die Formel noch durch Rekursion (siehe J &M, Kapitel 3)
- Bei Modified Kneser-Ney werden zusätzlich drei verschiedene Discounting-Parameter benutzt. d_1 für alle nur einmal gesehene n-gramme, d_2 für alle zweimal gesehenen n-gramme und d_3 für alle n-gramme, die dreimal oder mehr gesehen wurden. Natürlich muss dann auch die Normalisierung λ entsprechend angepasst werden.

Smoothing-Methode V: Webbasierte Language Modelle und Stupid Backoff

Alle bisherigen Methoden versuchen aus gegebenen Daten mehr herauszuholen. **Wie stehts mit mehr Daten?**

Idee: Häufigkeiten von $w_1 \ldots w_n$ aus dem Web (Zhu:Rosenfeld, 2001)

$$\hat{P}_{web}(w_3|w_1, w_2) = \frac{f_{web}(w_1, w_2, w_3)}{f_{web}(w_1, w_2)}$$

- Hier nur für Nulltrigrams oder wenig gesehenen Trigrams
- Keller and Lapata 2004: Web-Häufigkeiten für **alle** n-gramme
- Wie bekommen wir Webhäufigkeiten?

Webhäufigkeiten

Suchanfragen an Suchmaschine

- *n*-gram als Phrase an Suchmaschine
- wird genaue Phrasensuche ausführen
- Wird Anzahl der Webseiten ausgeben $\longrightarrow$ Schätzung

Was sind die Nachteile des Webs und/oder dieser Methode?

Aber es ist groß

	Types	Nicht abgedeckt von			
		Alta Vista	Lycos	Google	Korpus
1g	327	0	0	0	8
2g	462	4	5	4	68
3g	453	46	46	23	189

Table 1: Novel *n*-gram types in 24 news sentences (Zhu and Rosenfeld, 2001)

- Korpus: 103 M newswire Korpus
- Testsätze noch nicht indiziert

Vermeidung mancher Nachteile mit Web Snapshots

- Verwende Web 1T Korpus: 1-gram bis 5-gramme für alle 5-Wort-Sequenzen, die in mindestens 40 mal in 1 Billion Tokens Webtext vorkommen
- Vermeidet manche Probleme wie das der Reproduzierbarkeit. Welche noch? Welche nicht?
- Beispiel für Trigramme:

ceramics collection $</s>$	120
ceramics collection and	43
ceramics collection at	52
ceramics collection is	68

- Von 2006: `https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html`

Web mit Stupid Backoff (Brants et al (2007))

Keine Wahrscheinlichkeitsverteilung mehr

$$S(w_i|w_{i-n+1,\ldots,i-1}) = \begin{cases} \frac{f(w_{i-n+1,\ldots,i})}{f(w_{i-n+1,\ldots,i-1})}, \text{wenn nicht Null} \\ \lambda S(w_i|w_{i-n+2,\ldots,i-1}) \text{sonst} \end{cases}$$

- Backoff endet beim Unigram mit Wahrscheinlichkeit $S(w) = \frac{f(w)}{N}$
- Oft $\lambda = 0.4$.

Zusammenfassung bisher

- Markovmodelle schätzen Satzwahrscheinlichkeiten ab
- MLE Schätzung nicht brauchbar
- Einfaches Laplace-Smoothing vergibt zu viel Wahrscheinlichkeitsmasse an ungesehene Fälle
- Besser: Interpolation, Absolute Discounting, Kneser-Ney
- Im Moment beste Smoothingmethode: Kneser-Ney (Weiterentwicklung von Absolute Discounting mit Interpolation)
- Dennoch: sehr viele Parameterschätzungen, schlechte Generalisierung von Training zu Test $\rightarrow$ Neurale Sprachmodelle
- * Jurafsky und Martin. Kapitel 3
- Übungsblatt 5

Übersicht

Andere praktische Probleme: Unknown words

Oft wissen wir das gesamte Vokabular nicht im Voraus: Out of Vocabulary Words (OOV words)

- kreiiere ein Token $< UNK >$
- Training:
  1. Kreiiere fixes Vokabular V
  2. Jedes Wort im Korpus, das nicht in V ist, wird durch $< UNK >$ ersetzt
  3. Behandele $< UNK >$ wie ein normales Wort im Training
- Beim testen: Benutze $< UNK >$ Wahrscheinlichkeiten für jedes Wort, das wir nicht kennen

Andere praktische Probleme: Underflow

Im Cased Unigrammmodell auf Brown Korpus mit vereinfachtem $N = 1m$:

$$p(\textit{I drank white wine}) = \frac{5840}{1m} \cdot \frac{19}{1m} \cdot \frac{254}{1m} \cdot \frac{68}{1m}$$

Das werden aber kleine Zahlen $\longrightarrow$ Rundungsprobleme

Mache alles im *log*-Raum

Berechne

$$\log p(w_1 \ldots w_m) = \log (p_1 \cdot p_2 \ldots p_m) = \log p_1 + \log p_2 + \ldots + \log p_m$$

Übersicht

Intrinsische Evaluation

Intuition

Ein Textmodell ist besser, wenn es eine höhere Wahrscheinlichkeit den Wörtern und Wortsequenzen zuordnet, die im Testset wirklich vorkommen. D.H. Modell *A* ist besser als *B*, wenn es das Testkorpus als wahrscheinlicher ansieht. Man ist weniger überrascht.

Methode:

1. Spalte Korpus in Trainings- und Testkorpus. (Warum?)
2. Schätze Parameter des Modells auf Trainingskorpus. Dieser ist oft noch mal aufgespalten in Training und Held-out.
3. Teste Performanz auf Testkorpus. Messe zum Beispiel (Kreuz)Entropie und Perplexität des Modells auf Testkorpus.

Cross-entropy und Perplexität

Cross-entropy zwischen Sprache L und Modell p der Sprache gemessen auf Korpus $W := w_1, \ldots w_N$

$$H(W, p) \approx -\frac{1}{N} \log p(w_1, \ldots w_N)$$

Perplexität:

$$PP(W) := 2^{H(W,p)} = 2^{-\frac{1}{N} \log p(w_1, \ldots w_N)} = \sqrt[N]{\frac{1}{p(w_1 \ldots w_N)}}$$

Kettenregel:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i | w_1 \ldots w_{i-1})}}$$

Bigrams:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i | w_{i-1})}}$$

Perplexität: Beispiel

Korpus:

- $< s >$ there is a big house $< /s >$
- $< s >$ i buy a house $< /s >$
- $< s >$ they buy the new house $< /s >$

Testkorpus C: $< s >$ *they buy a big house* $< /s >< s >$ *i buy the new house* $< /s >$

Unter Bigrammodell (kein smoothing):

$$\begin{aligned} p(C) &= p(\textit{they}|< s >) \cdot \ldots \cdot p(< /s > |\textit{house}) \\ &= (\frac{1}{3} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot 1 \cdot 1) \cdot (\frac{1}{3} \cdot 1 \cdot \frac{1}{2} \cdot 1 \cdot 1 \cdot 1) = 0.0138 \end{aligned}$$

Kreuzentropie:

$$H(C,p) = -\frac{1}{12} \log p(C)$$

Perplexität: $PP(C) = 2^{-\frac{1}{12} \log p(C)}$

Perplexität von Sprache

Training 38 m Wörter, Testset 1.5 Millionen Wörter, WSJ

Perplexität:

Unigram	Bigram	Trigram
962	170	109

Tools und Daten:

- SRILM: `http://www.speech.sri.com/projects/srilm/`
- KenLM: `https://kheafield.com/code/kenlm/`
- Natürlich auch Implementationen in NLTK

Übersicht

Mögliche Anwendungen

Wortwahl, wenn es eine kleine Anzahl an Möglichkeiten gibt

Beispiele:

- Handschriftenerkennung: *a big brown b....r*
- MT:
  1. Ich glaube diese Geschichte nicht.
  2. I don't believe this $\{$*history,story,tale,saga,strip.*$\}$
- Rechtschreibkorrektur
- Ordnung in Generierung

Anwendungen sind eine gute Möglichkeit, die Güte eines Modells abzuschätzen

Rechtschreibkorrektur

Resultat	Nichtwörter	Wörter
Grund	typographisch	kognitiv
Fehleranzahl	einer	mehrere

Minimum Edit Distance war gut für das Vorschlagen von Korrekturen bei kleiner Fehleranzahl. Nicht gut für das Auswählen der richtigen Korrektur.

Rechtschreibkorrektur: Unteraufgaben

- **non-word error detection**: Erkenne Nichtwörter
- **isolated-word error correction:** korrigiere Nichtwörter ohne Kontext (Min Edit, isolierte Häufigkeit der Korrektur)
- **context-dependent error detection and correction:** entdecke und korrigiere real word errors.

Ein Teilproblem: Konfusionssets

Confusion sets

Wortpaare oder Wortmengen, die leicht füreinander falsch geschrieben werden

principle	principal	
then	than	
affect	effect	
quite	quiet	
peace	piece	
cite	site	sight
you’re	your	
their	there	they’re
weather	whether	

Aufgabe: Confusion Set Disambiguierung

I don't know ***weather*** *I want to go to the party.* $\longrightarrow$ richtig oder falsch?

$w_{-n}\ w_{-n+1} \ldots w_{-1}\ t\ w_1\ w_2 \ldots w_m \longrightarrow t$ richtig oder falsch?

äquivalent zu

I don't know ? I want to go to the party. $\longrightarrow$ wähle *weather* oder *whether*

$w_{-n}\ w_{-n+1}\ \ldots w_{-1}\ ?\ w_1\ w_2 \ldots w_m \longrightarrow$ welches Mitglied t des confusion set soll die Lücke ausfüllen

N-gram algorithmen

Geg. $w_{-n}\ w_{-n+1}\ \ldots\ w_{-1}\ ?\ w_1\ w_2 \ldots w_m$ und $t_1 \ldots t_j$ als confusion set

- Unigram model: $max_{t_i \in t_1 \ldots t_j} f(t_i)$
- Linkes-bigram: $max_{t_i \in t_1 \ldots t_j} f(w_{-1}, t_i)$
- Rechtes bigram : $max_{t_i \in t_1 \ldots t_j} f(t_i, w_1)$
- Mittleres trigram : $max_{t_i \in t_1 \ldots t_j} f(w_{-1}, t_i, w_1)$
- Linkes trigram: $max_{t_i \in t_1 \ldots t_j} f(w_{-2}, w_{-1}, t_i)$
- Rechtes trigram: $max_{t_i \in t_1 \ldots t_j} f(t_i, w_1, w_2)$

Modelle geschätzt auf Korpus C

Modelbeispiele

Testsatz: *I don't know ? I want to go to the party*; confusion set *whether, wheather*

- Unigram: $f(\textit{whether})$ *vs.* $f(\textit{weather})$
- Linkes Bigram: $f(\textit{know whether})$ *vs.* $f(\textit{know weather})$
- Rechtes bigram: $f(\textit{whether I})$ *vs* $f(\textit{weather I})$
- Mittleres Trigram ?
- Rechtes trigram ?
- Linkes trigram?

Experiment zum Testen

Geg. confusion set S mit Elementen $t_1 \ldots t_j$

1. Nimm Korpus $C1$ and extrahiere alle Kookkurenzen $t_1 \ldots t_j$ mit Kontext
2. Ersetze durch Lücke und versuche Original vorherzusagen
3. Messe Akkuratheit auf diesem automatisch generierten Testset

$C1$ muss von C getrennt sein

Beispiel

Aus Korpus $C1$ und confusion set $\{whether,wheather\}$ leite Testset ab:

Kontext	Original	Vorhersage?
I don't know ? I want to	whether	{whether,weather}
The ? is pretty bad today	weather	{whether,weather}
? he likes me or not I can't say	whether	{whether, weather}
I like sunny ?	weather	{whether,weather}

Prädiktion via n-gram model geschätzt von Korpus C

Resultate: BNC

18 confusion sets, 2056 Beispiele, Schätzungen BNC; Keller und Lapata (2005)

Model	BNC
Unigram	71.78
Linkes bigram	**83.29**
Rechtes bigram	81.70
Mittleres trigram	**78.14**
Linkes trigram	74.72
Rechtes trigram	73.46

Web und BNC Resultate

18 confusion sets, 2056 Beispiele

Model	Web	BNC
Unigram	74.79	71.78
Linkes bigram	86.13	83.29
Rechtes bigram	82.36	81.70
Mittleres trigram	**89.35**	**78.14**
Linkes trigram	**89.24**	**74.72**
Rechtes trigram	83.71	73.46

Fazit für Confusion sets

1. Unüberwachte Methode mit sehr kurzem Kontext: 89.35%
2. Web viel besser als BNC, trotz vieler theoretischer Nachteile
3. Gute überwachte Modelle erreichen 94% und mehr %
4. Methode kann auch auf confusion sets, die nichts mit Rechtschreibung zu tun haben, generalisiert werden (*until/by*)

Confusion Set Fazit

N-grams Anwendungen

- Oft gut für Wortwahl zwischen kleiner Menge an Möglichkeiten
- Braucht smoothing und/oder sehr große Korpora
- Kann gute, billige (unüberwachte) Alternative zu überwachtem maschinellem Lernen darstellen

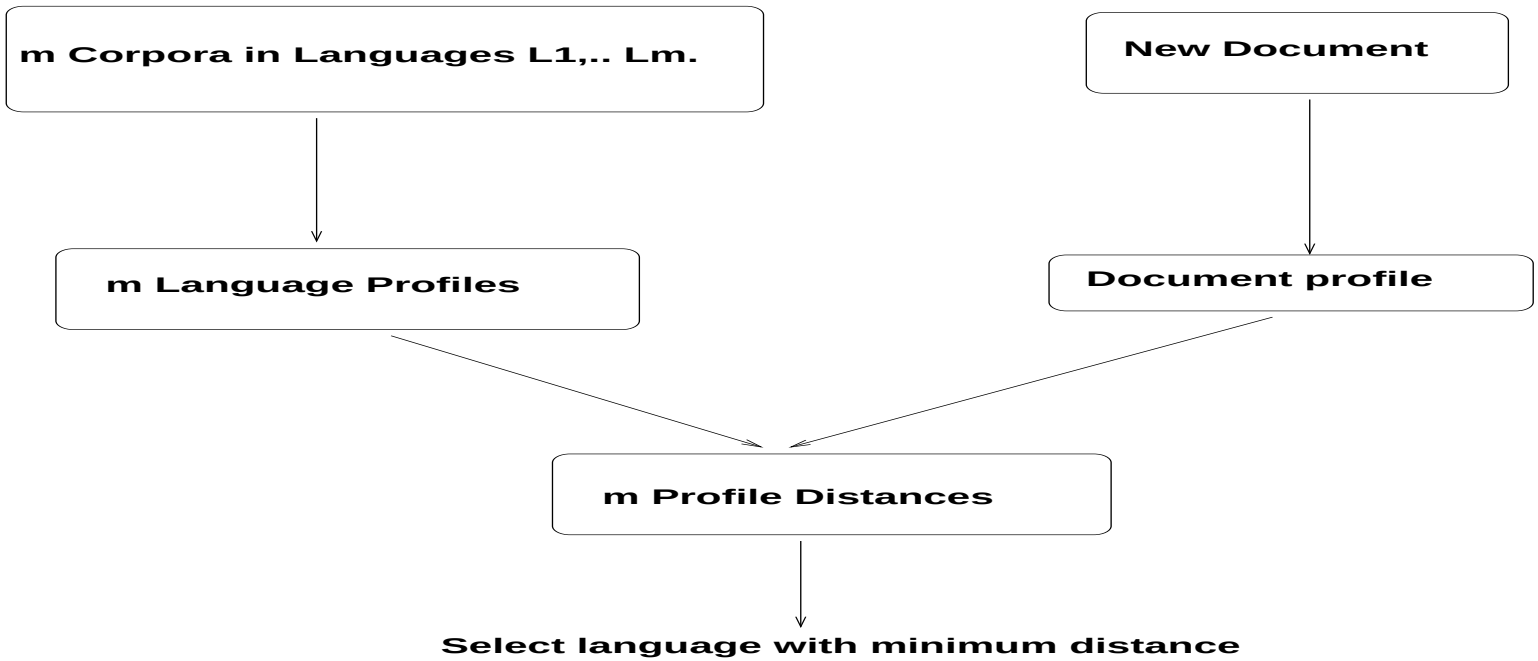
Lapata and Keller (2005): Web-based models for Natural Language Processing. IN ACM Transactions on Speech and Langauge Processing

Noch eine Anwendung: Sprachidentifizierung

Aufgabe: gegeben eine Zeichenkette, welche (menschliche) Sprache ist es?

Vorschläge

Buchstaben *n*-gramme für Sprachidentifikation



Profilgenerierung

- **Schritt 1:** Trainigsdaten für mehrere Sprachen
- **Schritt 2:** Häufigkeiten von H äufigkeiten für Buchstaben *n*grams aus Trainingsset
- Carver und Trenkle (1994): 1-5 gramme

Rang	n-gram
1	e
2	t
...	...
...	th
...	er
...	on
...	le
...	ing
...	and

Sprachidentifizierung

Für Testdokument:

- **Schritt 3**: Profil für das Testdokument
- **Schritt 4**: Vergleiche Sprachprofile mit Testdocumentprofil
- **Schritt 5**: Wähle die Sprache mit dem ähnlichstem Profil

Profildistanz

Rang	Sprachprofil	Testprofil
1	e	e
2	t	t
3	th	th
4	er	ing
5	on	on
6	le	er
7	ing	and
8	and	ed
...	...	...

Rang	Sprachprofil	Testprofil	Unterschied
1	e	e	**0**
2	t	t	**0**
3	th	th	**0**
4	er	ing	**3**
5	on	on	**0**
6	le	er	**2**
7	ing	and	**1**

Resultate

3200 newsgroup Artikel in 8 Sprachen; leave-one-out classification

L	<300	<300	<300	>300	>300	>300
Profil	100	200	300	100	200	300
Acc	92.9%	97.6%	98.6%	97.2%	99.5%	99.8%

L ist Artikellänge in Byte

Fazit

- Funktioniert schon gut auf kurzen Artikeln
- fast perfekt für lange Artikel
- auch gut für verrauschte Daten
- Demo: `https://tools.wmflabs.org/textcatdemo/`

Papiere

- * Jurafsky und Martin, Kapitel 3
- Chen und Goodman (1999): An empirical study of smoothing techniques for language modelling. In *Computer, Speech and Language*.
- Frank Keller und Mirella Lapata (2003): Using the Web to Obtain Frequencies for Unseen Bigrams. In: *Computational Linguistics*.
- Zhu und Rosenfeld (2001): Improving Trigram Language Modelling with the World Wide Web. In: *Proceedings of International Conference on Acoustics, Speech and Signal Processing*
- Cavnar und Trenkle (1994): N-gram-based Text Categorisation. Symposium on Document Analysis and Information Retrieval.
- Lapata und Keller (2005): Web-based models for Natural Language Processing. IN ACM Transactions on Speech and Language Processing
- Brants et al (2007): Large language models in machine translation. In EMNLP 2007.