Teknik Estimasi dalam Pemrosesan Bahasa Alami

Kali ini kita akan mempelajari suatu topik penting di dalam bidang Pengolahan Bahasa Alami (*Natural Language Processing*), yaitu Pendekatan Prediksi Kata untuk memperkirakan kehadiran suatu kata. Beberapa contoh dan penjelasannya akan digunakan untuk memudahkan pemahaman terhadap topik ini. Tulisan ini tidak mengandung review penelitian dan keterbaruan pendekatan solutif, meski tetap mencoba untuk melihat beberapa contoh aplikasi terkini dari topik yang dibahas. Hal ini pula yang menjadi alasan digunakannya buku, catatan kuliah dan situs web tertentu sebagai referensi dalam menyelesaikan naskah ini.

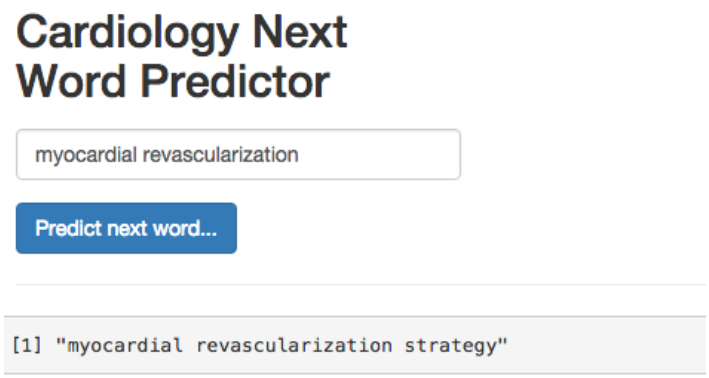
Makalah ini akan menjelaskan beberapa pendekatan dasar dalam mengestimasi kata yang akan hadir, baik setelah suatu deretan kata lain atau di awal kalimat. Topik mengenai prediksi kata selanjutnya sangat berkaitan dengan pemodelan suatu bahasa, karena itu juga akan dijabarkan mengenai model bahasa tertentu, terutama n-grams. Teknik estimasi yang dipilih dapat memberikan hasil yang tidak sesuai dengan kecerdasan manusia, sehingga diperlukan beberapa pendekatan untuk memperbaiki kualitas mesin prediksi “next word” ini.

1. Prediksi Kata dan Model N-Grams

Pekerjaan memprediksi kata terlihat tak-relevan jika kita berpikir NLP hanya untuk pemrosesan pemahaman semantik. NLP juga meliputi pemrosesan data “*noisy*” dan pemeriksaan error terhadap teks. Data *noisy* dapat dihasilkan dalam pengenalan ucapan (speech) dan tulisan tangan (handwriting), sedangkan komputer mungkin tidak dengan tepat mengenali kata dikarenakan tidak jelasnya *speech* dan *handwriting* yang secara signifikan berbeda dari model komputer. Teknik prediksi kata dapat pula melengkapi tugas *spelling checker* (Roberts, 2004) dan penerjemahan mesin, *augmentative communication* (Jurafsky, 2015). Kalimat “bersedekah adalah pekerjaan cuci” oleh *spell checker* dianggap betul karena kata “cuci” ada di dalam kamus Bahasa Indonesia. Berdasarkan statistik corpora tertentu, diketahui bahwa kalimat “bersedekah adalah pekerjaan suci” lebih tepat daripada kalimat sebelumnya dan sistem akhirnya memberikan saran bahwa kata “cuci” sebaiknya digantikan dengan kata “suci”.

Salah satu metode tertua yang digunakan dalam percobaan menghitung peluang kehadiran kata dalam suatu kalimat adalah memanfaatkan model n-grams. Model ini berusaha menebak suatu kata ke-n berdasarkan kehadiran kata sebelumnya (n-1), yaitu dengan melihat probabilitasnya. Pendekatan ini tidak melibatkan konteks. Akibatnya, karena "yang" adalah kata yang lebih umum daripada "hijau" maka "yang" lebih mungkin untuk hadir berikutnya. Pada kalimat "Balon besar itu mempunyai warna...", kata apa yang berpeluang hadir setelah "warna"?. Pada model bigram (2 kata), kita dapat menghitung P(hijau | warna) dan P(yang | warna) untuk mengetahui tebakan mana yang lebih banyak digunakan. Kita mungkin membayangkan bahwa tebakan kata akan lebih akurat jika dilakukan perhitungan P(hijau | Balon besar itu mempunyai warna) menggunakan model 6-grams. Pendekatan demikian memerlukan daya komputasi lebih besar dan waktu yang lebih panjang daripada bigram. Karena estimasi yang baik dapat dibangun berdasarkan model yang lebih kecil, maka lebih praktis hanya menggunakan model bigram atau trigram.

Dalam rangka implementasi teknik prediksi kata, aplikasi NLP harus mengakses probabilitas seberapa sering kehadiran dari kata tertentu dan seberapa sering kata tertentu itu hadir setelah kata tertentu lainnya. Aplikasi komputer yang melibatkan probabilitas tidak dapat dibangun tanpa adanya data yang mendasari. Cara paling sederhana adalah dengan menghitung secara manual jumlah kehadiran kata-kata di dalam kompulan teks contoh, namun dapat memunculkan error, dan manusia tidak mampu untuk menangani jutaan kata dalam waktu singkat. Model n-gram biasanya dilatih pada suatu *corpora*, yaitu file teks besar yang dapat diekstrak untuk menentukan properti statistik dari kata dan kalimat di dalamnya.



Gambar 1. Aplikasi Cardiology Next Word Predictor

(URL: http://amunategui.github.io/speak-like-a-doctor/)

Teknik prediksi kata berbasis model N-grams dan corpora telah banyak dimplementasikan. Cardiology Next Word Predictor merupakan salah satu aplikasinya, sebagaimana terlihat pada Gambar 1. Proses inti dari aplikasi ini dibangun menggunakan bahasa pemrograman R, sedangkan corpora yang dijadikan basis pelatihan dari model adalah PubMed (khusus domain kesehatan). Aplikasi ini diharapkan mampu menyarankan kata di bidang kesehatan, terutama Cardiology, saat pengguna mencari artikel yang terkait dengan bidang kesehatan.

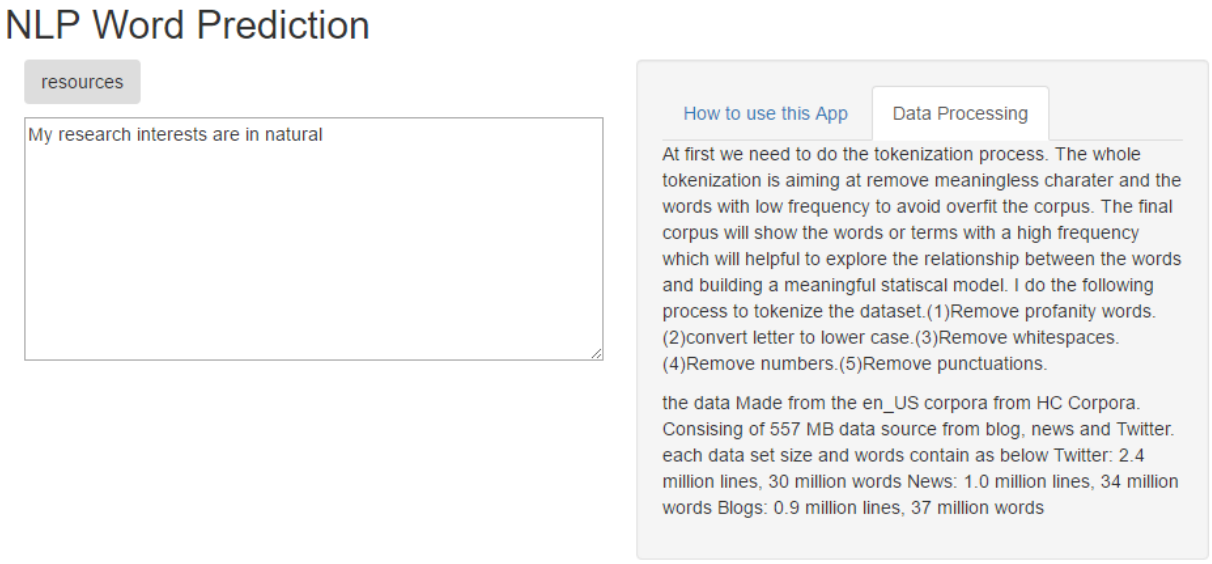
Adanya pelatihan model n-grams menggunakan corpora dapat menghemat waktu dan kerja manusia, tetapi menghadirkan pertanyaan mengenai gaya bahasa yang sangat tergantung pada sumber file dari corpora. Seorang pengguna mengharapkan kata dan kalimat yang sangat berbeda saat membaca koleksi novel karya Dan Brown dibandingkan dengan membaca surat kabar The New York Times. Akibatnya, suatu model yang dilatih secara utuh menggunakan koleksi novel karya Dan Brown kemungkinan akan tidak bagus ketika menebak "next word" dalam kalimat-kalimat dari The New York Times". Salah satu solusi yang disarankan adalah melatih model menggunakan corpora yang melibatkan banyak sumber. Pilihan ini memunculkan masalah baru. The New York Times kemungkinan tidak mempunyai kalimat dan kata-kata serupa dengan yang ada di dalam karya Dan Brown, meskipun suatu model yang dilatih dengan corpora dari kedua teks menganggap sama ketika menebak "next word" dalam suatu artikel surat kabar. Model n-grams akan tetap mempunyai kinerja lebih baik ketika dilatih berbasis corpora yang se-domain dengan tujuan aplikasi (*multi spesific domain corpora*).

Tabel 1. Rangkuman teknik dasar dari estimasi kata dalam NLP

No.	**Pendekatan**	**Potensi Masalah**	**Solusi**
1.	Spell checking	Kesalahan pemilihan kata dari dalam kosa kata	Melihat informasi statistika kata atau kalimat
2.	n-grams, n besar	Harus tersedia daya komputasi besar dan waktu panjang	Cukup bi- dan tri-gram
3.	bi, tri-grams dengan corpora	Gaya bahasa dari sumber berbeda	Memperbanyak sumber corpora
4.	bi, tri-grams dengan banyak sumber corpora	Tidak semua kalimat atau kata hadir di setiap sumber berbeda	Kembali fokus pada domain dari aplikasi, gunakan sumber terkait domain saja.
5.	bi, tri-grams dengan corpora domain khusus	Ada kombinasi kata yang tidak terobservasi selama pelatihan model	Teknik smoothing

Potensi problematik lain dari pelatihan model n-grams di atas adalah tertinggalnya beberapa bahkan banyak kombinasi kata. Kombinasi ini secara otomatis akan mempunyai probabilitas nol, meskipun sebenarnya kombinasi tersebut dapat diterima oleh bahasa yang digunakan. Akhirnya, banyak peneliti mengembangkan teknik *smoothing* untuk memberikan probabilitas kecil kepada kombinasi yang tidak terlihat dalam corpus pelatihan. Teknik ini memungkinkan model menebak suatu kata yang mengikuti kata lain dan tidak pernah terobservasi sebelumnya. Pendekatan dan potensi masalah yang mungkin hadir di dalam aplikasi prediksi kata ini, serta solusi yang dapat ditempuh, sebagaimana telah dijelaskan, diperlihatkan pada tabel 1.

Ide mengenai n-grams dimunculkan oleh Markov pada tahun 1913. Model-model lebih canggih telah dibuat oleh banyak peneliti akhir-akhir ini. Dalam rangka menuntaskan beberapa isu terkait perbedaan *stylistic* (gaya bahasa) antar teks, ilmuwan bereksperimen dengan memberikan bobot lebih besar untuk kombinasi-kombinasi tertentu dan dengan melibatkan banyak kata yang hadir sebelumya (nilai n yang jauh lebih besar) untuk mempengaruhi tebakan. Sebagian juga menggunakan thesauri untuk meningkatkan cakupan kosa kata dari n-grams. Akibatnya adalah diperoleh bahasa yang lebih lengkap dan prediksi kata yang lebih tepat (Roberts, 2004). Kemajuan di bidang perangkat keras juga memungkin proses komputasi dituntaskan dalam waktu yang lebih singkat.



Gambar 2. Aplikasi NLP Word Prection yang memperlihatkan kata “resources” mempunyai probabilitas kehadiran tertinggi setelah kata “natural”

(URL: https://rpubs.com/Trent/88182)

Contoh aplikasi prediksi kata yang lebih luas cakupan domainnya adalah NLP Word Prediction yang dikeluarkan oleh Rpubs (Gambar 2). Aplikasi ini menggunakan model n-gram dimana nilai n dimulai dari 1 (unigram) sampai dengan 4. Saat ini sedang dilakukan pengembangan agar prediksi dapat melibatkan sampai n bernilai 5 (4 kata sebelum tebakan).

NLP Word Prection membuat suatu dataset yang berisi deretan kata 4-grams. Ini dilakukan agar sistem mampu membandingkan suatu frase input terhadap kombinasi kata mencakup 1-gram ("How"), 2-grams ("How are"), 3-grams ("How are you"), dan 4-grams ("How are you doing"). Secara garis besar modul prediksi kata dari sistem ini bekerja sebagai berikut:

- Ketika hanya ada input 1 kata maka aplikasi ini membandingkan semua kata-kata berikutnya (*subsequent words*) dalam dataset. Ini berarti dilakukan perbandingan kata pertama dari deretan 4-grams dengan satu kata inputan kemudian mengembalikan kata kedua dari 4-grams yang paling sering.
- Ketika pengguna memasukkan 2 kata, sistem membandingkan dua kata pertama dari 4-grams dengan dua kata inputan dan kemudian mengembalikan kata ketiga dari 4-grams yang paling sering.
- Pada saat pengguna memasukkan 3 kata, sistem ini mengambil 3 kata dari 4-grams dan membandingkannya dengan 3 kata inputan dan mengembalikan kata keempat dari 4-grams yang paling sering.

NLP Word Prection masih terus dikembangkan dengan menambahkan fitur dan kemampuan tertentu, di antaranya adalah:

- Aplikasi ini menggunakan teknik NLP untuk memprediksi "next word" dalam medan 1-gram, 2-grams, 3-grams dan 4-grams. Perbaikan akan dilakukan sehingga aplikasi mampu memroses input sampai dengan 4 kata (model 5-grams).
- Dua parameter untuk membangun aplikasi yang baik adalah akurasi dan kecepatan. Aplikasi ini rata-rata memerlukan ~0.5 detik untuk menyelesaikan prediksi. Akurasi dari sistem akan ditingkatkan dengan menambahkan sumber daya data (memperbesar corpora), sedangkan penanganan query akan dipercepat dengan menyertakan pustaka bahasa R.

2. Menghitung Probabilitas Estimasi

Probabilitas kehadiran kata *w* setelah diberikan suatu sejarah (daftar kata sebelumnya) *h* dinyatakan sebagai *P(w|h)*. Jika teks sejarahnya adalah "Balon besar itu mempunyai warna" dan kata yang akan diprediksi kemunculan setelahnya adalah "hijau" maka ditulis:

P(hijau|Balon besar itu mempunyai warna)

Cara paling sederhana dalam mengestimasi probabilitas ini adalah dengan menghitung frekuensi relatif h diikuti oleh w berdasarkan suatu corpus. Rumus yang digunakan adalah (Jurafsky, 2015; Manning, 1999):

$$P(w|h) = \frac{P(h\,w)}{P(h)} \quad (1)$$

Sehingga contoh di atas dapat ditulis menjadi:

$$\text{P(hijau|Balon besar itu mempunyai warna)} = \frac{\text{P(Balon besar itu mempuyai warna hijau)}}{\text{P(Balon besar itu mempunyai warna)}}$$

Bagaimana probabilitas dari rangkaian kata keseluruhan dihitung, misalnya $P(w_1, w_2, \ldots, w_n)$? Di dalam statistika dilakukan dengan mendekomposisi probabilitas menggunakan aturan rantai probabilitas:

$$P(X_1 \ldots X_n) = P(X_1)\,P(X_2|X_1)\,P(X_3|X_1^2) \ldots P(X_n|X_1^{n-1})$$

$$= \prod_{k=1}^{n} P(X_k|X_1^{k-1}) \quad (2)$$

Artinya:

*P(Balon besar itu mempunyai warna) = P(Balon)*P(besar|Balon)*P(itu|Balon besar) * P(mempunyai|Balon besar itu) * P(warna|Balon besar itu mempunyai)*

Dengan mengikuti asumsi Markov persamaan di atas dapat disederhanakan, misalnya pada model bigrams menjadi:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \quad (3)$$

Sehingga:

*P(Balon besar itu mempunyai warna) = P(Balon)*P(besar|Balon)*P(itu| besar) * P(mempunyai| itu) * P(warna|mempunyai)*

Bagaimana mengestimasi probabilitas dari bigram atau N-grams?. Suatu cara intuitif memperkirakan probabilitas adalah *maximum likehood estimation* (MLE). MLE melakukan ini dengan menghitung jum

lah dari suatu corpus, menormalisasi jumlah tersebut sehingga berada antara 0 dan 1. Misalnya untuk menghitung probabilitas bigram dari kata b setelah diberikan kata sebelumnya a, maka dilakukan perhitungan frekuensi bigram C(ab) dan menormalkannya dengan jumlah semua brigram yang berbagi-pakai kata pertama a yang sama:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \quad (4)$$

Rumus ini dapat disederhanakan menjadi (untuk bigram):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \quad (5)$$

MLE untuk bigram di atas dapat digeneralkan untuk N-gram menjadi:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \quad (6)$$

Contoh 1:

Sebuah corpus kecil hanya mengandung 3 kalimat berikut:

<s> I am Sam </s>

<s> Sam I am </s>

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

Dimana <s> menyatakan awal kalimat dan </s> sebagai tanda akhir kalimat. Perhitungan beberapa probabilitas bigram dari corpus ini adalah

P(I|<s>) = 2/3 = 0.67 P(Sam|<s>) = 1/3 = 0.33 P(am|I) = 2/3 = 0.67

P(</s>|Sam) = ½ = 0.5 P(Sam|am) = ½ = 0.5 P(do|I) = 1/3 = 0.33

Contoh 2:

Berikut ini adalah frekuensi bigram dari beberapa kata dalam corpus Berkeley Restaurant Project:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Terlihat jelas ada 827 kali kata “want” hadir setelah kata “i”, 82 kali kata “food” mengikuti kata “chinese”, dan 686 kali kata “eat” hadir setelah kata “to”.

Berikut ini adalah frekuensi kemunculan beberapa kata di dalam corpus tersebut:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Berdasarkan nilai ini, dilakukan normalisasi terhadap frekuensi bigram untuk mendapatkan probabilitas bigramnya. Misalnya “i want” yang frekuensi bigramnya 827 dinormalkan menjadi:

$$P(want|i) = P(want \mid i) / P(i) = 827/2533 = 0.3265 = 0.33$$

Berikut ini adalah daftar probabilitas semua bigram di atas (frekuensi bigram yang telah mengalami normalisasi):

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Beberapa probabilitas bigram lain yag diperlukan adalah

$P(i|<s>) = 0.25$ $P(english|want) = 0.0011$

$P(food|english) = 0.5$ $P(</s>|food) = 0.68$

Berapa probabilitas dari kalimat “i want English food” dan “I want Chinese food”? Kalimat mana yang lebih sering hadir di dalam corpus? Berdasarkan rumus (3) di atas (*product*), probabilitas kalimat “i want English food” adalah:

P(<s> i want english food </s>)

= P(i|<s>)P(want|i)P(english|want)P(food|english)P(</s>|food)

= 0.25 * 0.33 * 0.0011 * 0.5 * 0.68

= 0.000031

Sedangkan probabilits kalimat “i want Chinese food” adalah

P(<s> i want chinesse food </s>)

= P(i|<s>)P(want|i)P(chinese|want)P(food|chinese)P(</s>|food)

= 0.25 * 0.33 * 0.0065 * 0.52 * 0.68

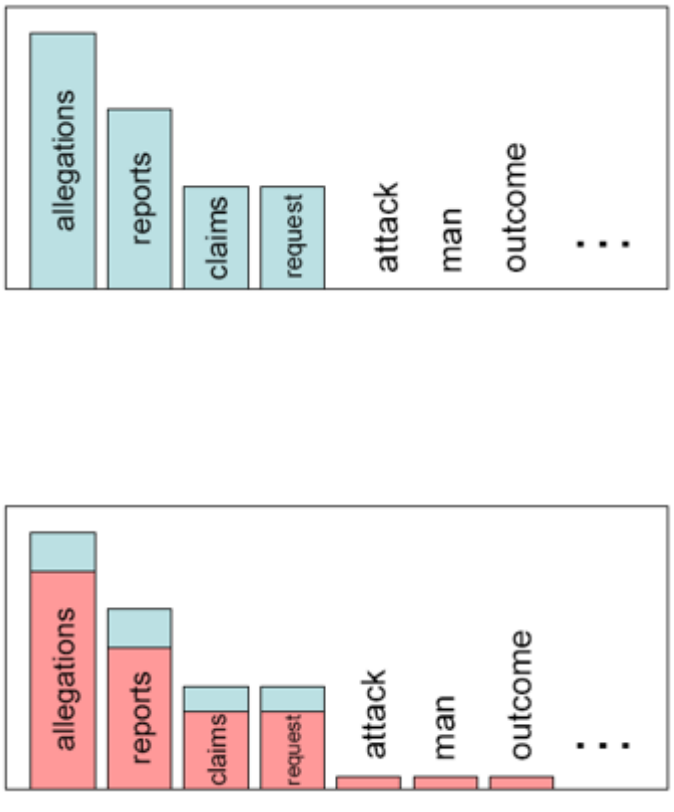
= 0.000189

Dari hasil di atas dapat diketahui bahwa kalimat “i want chinese food” mempunyai peluang lebih sering dihadirkan daripada kalimat “i want english food”.

3. Smoothing

Pada model bahasa N-grams, sering ditemukan adanya kata-kata yang tercatat di dalam kosakata (bukan kata yang tidak dikenal) tetapi hadir di dalam himpunan uji dalam suatu konteks yang tidak terlihat (misalnya kata itu hadir setelah suatu kata yang sebelumnya tidak pernah muncul di dalam training). Ini sejalan dengan hukum distribusi Zipfian: frekuensi suatu kata kira-kira berbanding terbalik dengan peringkatnya di daftar distribusi kata. Di dalam suatu corpus terlihat dengan jelas bahwa (1) sejumlah kecil kata hadir sangat sering dan (2) sejumlah besar kata terlihat muncul hanya sekali. Tanpa pendekatan khusus maka secara otomatis probabilitas dari deretan kata ini menjadi nol. Probabilitas nol dari suatu bigram atau N-grams akan mengakibatkan probabilitas nol dari kalimat dimana bigram tersebut berada. Salah satu teknik yang dapat digunakan agar model bahasa tidak memberikan probabilitas nol untuk peristiwa yang tak terlihat ini adalah melakukan subsidi silang probabilitas. Pendekatan ini dinamakan *smoothing* (memperlancar) atau *discounting* (pengurangan), di antaranya mencakup add-1, add-k, *stupid backoff* dan Kneser-Ney.

Beberapa peneliti menganalogikan proses *smoothing* sebagai perilaku legenda Robin Hood yang mengambil harta orang kaya dan memberikannya kepada orang-orang miskin. Gambar 3 mengilustrasikan jumlah kemunculan (*count*) dari 7 kata setelah "denied the" di dalam suatu corpus. Awalnya ada 4 kata yang hadir setidaknya satu kali ("allegations" 3 kali, "reports" 2 kali, "claims" 1 kali dan request 1 kali), sedangkan 3 kata ("attack", "man", dan "outcome") tidak hadir sama sekali. Subsidi silang dilakukan dengan mendiskon jumlah kemunculan dari 4 kata pertama masing-masing sebesar 0.5 sehingga diperoleh total diskon 2.0. Nilai ini kemudian dibagikan kepada secara merata kepada 3 kata yang sebelumnya mempunyai jumlah kehadiran nol. Sekarang kata "attach", "man" dan "outcome" mempunyai "probabilitas" kehadiran setelah "denied the" masing-masing sebesar 0.67.



Jumlah kemunculan kata setelah "denied the":

allegations 3

reports 2

claims 1

request 1

Setelah smoothing, jumlah kehadiran kata menjadi:

allegations 2.5

reports 1.5

claims 0.5

request 0.5

lainnya 2

Gambar 3. Contoh smoothing terhadap probabilitas kehadiran kata

3.1 Laplace Smoothing

Lapache *smoothing* merupakan teknik *smoothing* paling sederhana, yaitu dengan menambahkan nilai satu ke setiap frekuensi kehadiran bigram sebelum dilakukan normalisasi untuk memperoleh probabilitas. Akibatnya semua frekuensi awal nol berubah menjadi 1. Laplace *smoothing* tidak cukup baik saat digunakan dalam model N-grams modern tetapi menjadi konsep dasar (juga sebagai *baseline*) bagi banyak algoritma *smoothing* yang hadir kemudian.

Probabilitas unigram dari suatu kata w_i pada pendekatan estimasi MLE adalah frekuensi kehadiran c_i yang dinormalkan oleh jumlah total token kata N:

$$P(w_i) = \frac{c_i}{N} \quad (7)$$

Laplace *smoothing* menaikkan satu (*increment*) frekuensi c_i (pembilang dari rumus di atas). Karena ada sebanyak *V* kata di dalam kosakata dan semuanya telah ditambahkan satu, maka penyebut juga harus ditambahkan dengan V, sehingga diperoleh rumus:

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V} \quad (8)$$

Pembilang dari persamaan di atas perlu disesuaikan agar mudah dibandingkan secara langsung dengan frekuensi MLE dan diubah ke dalam probabilitas seperti frekuensi MLE yang dinormalisasi oleh N. Ini dapat dilakukan dengan mengalikan pembilang dengan N/(N+V) sehingga diperoleh:

$$c_i^* = (c_i + 1)\frac{N}{N + V} \quad (9)$$

Sekarang c_i^* dapat digunakan untuk memperoleh P_i^* dengan normalisasi oleh N.

Pada model bigram rumus probabilitas Lapache di atas berubah menjadi:

$$P^*_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \quad (10)$$

Sebagaimana terlihat pada contoh 2, sebagian besar bigrams Berkeley Restaurant Project mempunyai frekuensi nol. Laplace smoothing menambahkan nilai satu untuk setiap frekuensi bigram sehingga diiperoleh matriks berikut (total bigram yang tercatat adalah V = 1446):

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Matriks berikut memperlihatkan probabilitas bigram yang telah di-*smooth* (misal P(want|i) = 828/(2533 + 1446) = 0.208, dibuat dua angka dibelakang koma menjadi 0.21):

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Bagaimana jika matriks bigram di atas dibangun ulang setelah terkena pengaruh *smoothing*? Rumus berikut dapat digunakan untuk menghitung ulang frekuensi bigram terbaru dari Berkeley Restaurant Project:

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V} \quad (11)$$

Misalnya:

$$c^*(i\ want) = \frac{[C(i\ want) + 1] \times C(i)}{C(i) + V} = \frac{828 \times 2533}{2533 + 1446} = 527$$

Berikut ini adalah matriks baru untuk semua bigram yang hadir dalam contoh 2.

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Matriks ini tentu mempengaruhi probabilitas dari suatu bigram dan kalimat. C(want to) berubah dari 608 menjadi 238. P(to|want) mengalami penurunan: sebelumnya 0.66 berubah menjadi 0.26 setelah dilakukan *smoothing*. Setiap bigram mengalami diskon sebesar d (rasio antara frekuensi baru dan lama). Diskon untuk bigram "want to" adalah 0.39, sedangkan diskon untuk "chinese food" sebesar 0.10.

3.2 Add-k Smoothing

Pada pendekatan *smoothing* ini dilakukan pengalihan sedikit (sebesar k) probabilitas dari N-grams yang terlihat ke kejadian yang tak-tampak. Konstanta k harus bernilai pecahan (misalnya k bernilai 0.5, 0.05 atau 0.01) dan tidak boleh bernilai 1 (jika k = 1 berarti sama dengan *smoothing* add-1). Add-k mengharuskan kita mempunyai metode untk menentukan nilai k, misalnya dengan mengoptimalkan suatu **devset**. Meskipun *add-k* bermanfaat bagi beberapa tugas seperti klasifikasi teks, kinerjanya masih belum begitu bagus untuk pemodelan bahasa, membangkitkan frekuensi dengan variansi buruk dan sering memberikan diskon yang tidak tepat.

Rumus probabilitas dari add-k adalah

$$P^*_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV} \quad (12)$$

3.3 Backoff dan Interpolasi

Discounting mampu menyelesaikan masalah frekuensi nol dari N-grams. Jika kita mencoba menghitung $P(w_n|w_{n-2}w_{n-1})$ tetapi tidak mempunyai contoh trigram $w_{n-2}w_{n-1}w_n$ maka kita dapat mengestimasi probabilitasnya memanfaatkan probabilitas bigram $P(w_n|w_{n-1})$. Dengan cara serupa, jika kita tidak mempunyai frekuensi untuk menyelesaikan $P(w_n|w_{n-1})$ maka kita melihat ke unigram $P(w_n)$. Ada dua pendekatan genaralisasi demikian: backoff dan interpolasi.

Pada *backoff*, kita menggunakan trigram jika fakta tersedia, jika tidak kita menggunakan bigram, jika tidak ada fakta juga maka memanfaatkan unigram. Kita hanya melakukan *backoff* ke N-gram orde lebih rendah jika ditemukan frekuensi nol pada N-gram orde lebih tinggi. Agar model *backoff* memberikan distribusi probabilitas yang tepat maka harus diterapkan diskon terhadap N-grams orde lebih tinggi. Kumpulan diskon tersebut dialihkan ke N-grams orde lebih

rendah. Pendekatan *backoff* dengan diskon ini dinamakan *Katz backoff*. Probabilitas dengan *backoff* dirumuskan sebagai:

$$P_{BO}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & if\ C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{BO}(w_n|w_{n-N+2}^{n-1}), & jika\ tidak \end{cases} \quad (13)$$

Katz backoff sering dikombinasikan dengan metode *smoothing* bernama Good-Turing. Algoritma Good-Turing Backoff meliputi komputasi sangat detail untuk mengestimasi Good-Turing smoothing, nilai P^* dan α.

Pendekatan interpolasi sangat berbeda. Kita harus menggabungkan probabilitas dari semua N-gram, memberikan bobot dan mengkombinasikan frekuensi trigram, bigram dan unigram. Berikut ini adalah rumus untuk mengestimasi probabilitas menggunakan interpolasi linier:

$$P(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n) \quad (14)$$

Dimana jumlah total λ sama dengan 1 dan agar lebih dapat dipercaya sebaiknya probabilitas N-grams orde lebih tinggi dibobot dengan λ yang lebih besar.

3.4 Kneser-Ney Smoothing

Algoritma Kneser-ney terinterpolasi merupakan metode *smoothing* berkinerja terbaik dan sangat sering diaplikasikan. Akar dari metode ini adalah *discounting* absolut. Misalnya kita mempunyai suatu N-grams dengan frekuensi 4 dan perlu didiskon, tetapi berapa besar diskon yang tepat? (Church dan Gale, 1991) menghitung tata bahasa bigram dari 22 juta kata dari newswire AP dan kemudian memeriksa frekuensi setiap bigram ini dalam 22 juta kata lainnya. Rata-rata, suatu bigram yang hadir 4 kali dalam 22 juta kata pertama terjadi sebanyak 3.23 kali dalam 22 juta kata berikutnya. Berikut ini adalah frekuensi kehadiran bigram dengan *c* dari 0 sampai dengan 9.

Bigram count in training set	Bigram count in heldout set
0	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

Rumus probabilitas dari *Absolute Discounting* ini adalah

$$P_{AbsoluteDiscounting}\,(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{C(w_{i-1})} + \lambda(w_{i-1})P(w_i) \qquad (15)$$

Dimana λ biasanya dipilih 0.75 atau secara khusus 0.5 untuk bigram berfrekuensi 1.

Kneser-Ney *discounting* (Kneser dan Ney, 1995) memanfaatkan *discounting* absolut dengan cara yang lebih canggih untuk menangani distribusi unigram orde lebih rendah. Jika kita mempunyai kalimat:

I can't see without my reading _____

Apakah kata yang tepat mengikuti potongan kata di atas seandainya interpolasi dilakukan terhadap model bigram dan unigram?

Kata "glasses" kemungkinan besar lebih tepat daripada kata "Francisco". Tetapi faktanya (sesuai corpus) kata "Francisco" lebih umum karena "San Francisco" adalah kata yang sangat sering hadir. Model unigram standart akan memberikan "Francisco" probabilitas lebih tinggi daripda a"glasses". Secara intuisi kita dapat mengatakan bahwa "Francisco" memang sering tetapi hanya dalam frase "San Francisco", setelah kata "San". Kata "glasses" mempunyai distribusi yang jauh lebih luas. Ada konteks yang dilibatkan. Intuisi ini menjadi basis estimasi $P_{Kontinuasi}$ pada jumlah konteks berbeda dimana kata w hadir di dalamnya, yaitu jumlah tipe bigram yang dilengkapinya. Setiap tipe bigram adalah suatu sambungan baru (*novel continuation*) saat pertama kali terlihat. Hipotesanya adalah kata-kata yang telah hadir di dalam lebih banyak konteks di masa lalu juga lebih mungkin untuk hadir di dalam beberapa konteks baru. Berapa kali suatu kata w_i muncul sebagai sambungan baru dapat diekspresikan sebagai:

$$P_{Kontinuasi}(w_i) \propto |\{w_{i-1} : c(w_{i-1}\,w_i) > 0\}| \qquad (16)$$

Dan setelah dinormalkan dengan jumlah total dari tipe bigram, diperoleh

$$P_{Kontinuasi}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}\, w_i) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}\, w_j) > 0\}|} \quad (17)$$

Persamaan alternatifnya normalisasi dengan jumlah kata-kata yang mendahului semua kata:

$$P_{Kontinuasi}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}\, w_i) > 0\}|}{\sum_{w_i'} |\{(w_{i-1}') : c(w_{i-1}' w_i') > 0\}|} \quad (18)$$

Persamaan akhir bagi smoothing Kneser-Ney terinterpolasi adalah

$$P_{KN}(w_i|w_{i-1}) = \frac{max(c(w_{i-1}w_i)-d,0)}{c(w_{i-1})} + \lambda\,(w_{i-1})\, P_{Kontinuasi}(w_i) \quad (19)$$

Dimana λ adalah konstanta normalisasi yang digunakan untuk mendistribusikan koleksi probabilitas yang telah didiskon sebelumnya:

$$\lambda\,(w_{i-1}) = \frac{d}{c(w_{i-1})}\, |\{(w: c(w_{i-1}, w)) > 0\}| \quad (20)$$

$\frac{d}{c(w_{i-1})}$ adalah diskon yang telah dinormalkan, sedangkan $|\{(w: c(w_{i-1}, w)) > 0\}|$ adalah jumlah tipe kata yang dapat mengikuti w_{i-1} atau jumlah tipe kata yang didiskon, berapa kali diskon ternormalisasi diterapkan.

Persamaan rekursif umumnya adalah sebagai berikut:

$$P_{KN}\left(w_i|w_{i-n+1}^{i-1}\right) = \frac{max(C_{KN}\left(w_{i-n+1}^{i}\right) - d, 0)}{C_{KN}(w_{i-n+1}^{i-1})} + \lambda\,(w_{i-n+1}^{i-1})\, P_{KN}\left(w_i|w_{i-n+2}^{i-1}\right) \quad (21)$$

Dimana definisi dari hitungan (frekuensi kehadiran) C_{KN} tergantung pada apakah kita menghitung N-gram orde tertinggi yang diinterpolasi (misalnya trigram jika kita menginterpolasi trigram, bigram dan unigram) atau salah satu N-gram orde lebih rendah (bigram atau unigram jika menginterpolasi trigram, bigram dan unigram):

$$C_{KN}(\bullet) = \begin{cases} count(\bullet) & untuk\ orde\ paling\ tinggi \\ frekuensisambungan(\bullet) & untuk\ orde\ lebih\ rendah \end{cases} \quad (22)$$

Frekuensisambungan adalah jumlah konteks kata tunggal yang unik untuk •.

Saat penghentian rekursi, unigram diinterpolasi dengan distribusi seragam:

$$P_{KN}(w) = \frac{\max(C_{KN}(w) - d, 0)}{\sum_{w'} C_{KN}(w')} + \lambda(\varepsilon)\frac{1}{|V|} \quad (23)$$

Jika kita ingin memasukkan kata yang tidak dikenal atau <UNK> maka dapat dilakukan hanya dengan menyertakan kata tersebut sebagai kosakata reguler berfrekuensi nol yang nantinya akan mendapatkan probabilitas otomatis sebesar $\frac{\lambda(\varepsilon)}{|V|}$.

Versi *Kneser-Ney smoothing* berkinerja terbaik dinamakan *modified Kneser-Net smoothing* yang diajukan oleh (Chen dan Goodman, 1998). Pada pendekatan ini tidak digunakan diskon tunggal d, tetapi menggunakan 3 tingkatan diskon: d_1, d_2 dan d_{3+}. Diskon d_1 berlaku untuk N-grams dengan frekuensi 1, d_2 untuk N-grams berfrekuensi 2 dan d_{3+} untuk N-grams yang frekuensinya sama atau lebih dari 3. (Körner, 2013) mengaplikasi teknik *smoothing* modifikasi ini untuk membangun suatu mesin prediksi kata.

3.5 Web dan Stupid Backoff

Teks dari web dapat digunakan untuk membangun model bahasa super besar. Google pada tahun 2006 telah merilis himpunan N-grams sangat besar (1 s.d 5-grams) dari semua deretan lima-kata yang hadir setiaknya 40 kali dari 1 triliun lebih kata dari *running text* di web; termasuk sekitar 1,2 milyar rangkaian lima-kata menggunakan lebih 13 juta tipe kata.

Data raksasa tersebut sangat mungkin untuk membangun model bahasa skala web menggunakan Kneser-Ney *smoothing* secara penuh. (Brants dkk., 2007) memperlihatkan bahwa algoritma yang jauh lebih sederhana sudah cukup untuk menangani model bahasa yang sangat besar. Algoritma bernama ***stupid backoff*** ini memunculkan ide percobaan untuk membuat model bahasa suatu distribusi probabilitas murni. Tidak ada diskon terhadap probabilitas ber-orde lebih tinggi. Jika N-gram orde lebih tinggi mempunyai frekuensi nol maka dilakukan *backoff* ke N-grams ber-orde lebih rendah, dibobot dengan suatu bobot tetap (tidak tergantung pada konteks). Rumus yang digunakan adalah

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{count(w_{i-k+1}^{i})}{count(w_{i-k+1}^{i-1})} & jika\ count\left(w_{i-k+1}^{i}\right) > 0 \\ \lambda S(w_i|w_{i-k+2}^{i-1}) & jika\ tidak \end{cases} \quad (24)$$

Backoff akan berhenti pada unigram yang mempunyai probabilitias $S(w) = \frac{count(w)}{N}$. Konstanta λ biasanya akan memberikan hasil terbaik saat diberikan nilai 0.4.

3.6 Good-Turing Smoothing

Pemikiran dasar dari Good-Turing Smoothing adalah memanfaatkan *count* atau frekuensi sesuatu yang telah dilihat sekali untuk membantu mengestimasi frekuensi dari sesuatu yang belum pernah terlihat (Cao, 2009). Kata atau N-gram (atau suatu kejadian) yang hadir sekali disebut sebagai *singleton*. Dalam rangka menghitung frekuensi *singleton*, kita perlu menghitung Nc (jumlah kejadian yang hadir sebanyak c kali). Pada Good-Turing ada anggapan bahwa semua kata terdistribusi secara binomial. Misalnya N_c jumlah item yang frekuensinya c kali (frekuensi dari frekuensi c) maka N_c dapat digunakan untuk menyediakan estimasi yang lebih baik c. Perhitungan frekuensi yang disesuaikan c* adalah:

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \qquad (25)$$

Pada model bahasa bigram, Good-Turing menganggap bahwa kita mengetahui N_0, jumlah bigram yang belum terlihat. Kita mengetahui ini karena diberikan suatu kosakata berukuran *V*, jumlah total dari bigram adalah V^2, karena itu N_0 adalah V^2 dikurangi semua bigram yang sudah terlihat. Secara umum dan praktis estimasi c* terdiskon tidak digunakan untuk semua hitungan c. Pertama, hitungan besar (dimana c > k untuk beberapa *threshold* k) diasumsikan dapat dipercaya. Katz menyarankan agar k diberikan nilai 5. Jadi diperoleh definisi:

$$c^* = c \; untuk \; c > k$$

Dan estimasi frekuensi kehadiran suatu kata adalah

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - C\frac{(k+1)N_{k\_1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}} \qquad (26)$$

Tabel 2 memperlihatkan Frekuensi bigram dan hasil re-estimasi menggunakan Good-Turing terhadap bigram dari corpus AP yang mengandug 22 juta bigram yang disiapkan oleh (Church dan Gale, 1991) dan dari corpus Berkeley Restaurant Project yang menganung 9332 kalimat. Reestimasi Good-Turing dilakukan menggunakan rumus (26).

Tabel 2. Frekuensi bigram dan hasil re-estimasi dengan Good-Turing Smoothing

AP Newswire			Berkeley Restaurant		
c (MLE)	N_c	c (GT)	c (MLE)	N_c	c (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

Bagaimana menghitung probabilitas dari suatu kalimat? Kita dapat menggunakan rumus berikut:

$$P_{GT}(w_1 \dots w_n) = \frac{1 - \sum_{r=1}^{\infty} N_c \frac{c^*}{N}}{N_0} \approx \frac{N_1}{N_0 N} \qquad (27)$$

Contoh: Estimasi Good-Turing

Diketahui suatu corpus mempunyai kosa kata sebanyak *V = 14585* dan jumlah bigram yang terlihat adalah 199252. Frekuensi dari frekuensi bigram *c* sebagai berikut:

c	N_c
1	138741
2	25413
3	10531
4	5997
5	3565
6	...

Berapa nilai N_0, $C_{tak\text{-}terlihat}$ dan $P_{tak\text{-}terlihat}$?

$N_0 = (14585)^2 - 199252$

$C_{tak\text{-}terlihat} = N_1/N_0 = 0.00065$

$P_{tak\text{-}terlihat} = N_1/(N_0N) = 1.06 \times 10^{-9}$

Jika *c(person she) = 2* dan *c(person) = 233*, maka

$C_{GT}(person\ she) = (2 + 1)(10531/25413) = 1.243$

$P(she|person) = C_{GT}(person\ she)/223 = 0.0056$

4 Rangkuman

Naskah ini telah menjelaskan teknik dasar prediksi kata dan probabilitas kalimat di dalam model bahasa N-grams berdasarkan suatu corpus. Beberapa kekurangan yang muncul diperbaiki dengan teknik smoothing atau discounting. Dua teknik smoothing yang memberikan kinerja paling adalah Kneser-Ney dan Good-Turing. Sampai saat ini diyakini bahwa teknik smoothing Kneser-Ney termodifikasi adalah yang terbaik. Penelitian dan penerapan metode prediksi kata dengan smoothing semakin banyak. Beberapa contohnya telah disebutkan di atas.

Referensi

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large language models in machine translation. In EMNLP/CoNLL 2007.

Cao, D.D. dan Basili, R. (2009) Probability Estimation, handout kuliah Web Mining e Retrieval 2008-9, URL: http://www.uniroma2.it/didattica/WmIR/deposito/estimation_handout.pdf, diakses: 10 Mei 2016

Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. Tech. rep. TR-10-98, Computer Science Group, Harvard University.

Chen, S.F. dan Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. Dalam *Computer Speech and Language* (1999) 13, pp. 359–394

Church, K. W. and Gale, W. A. (1991). A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. Computer Speech and Language, 5, 19–54.

Graber, J. B. (2011). Data-Intensive Information Processing Applications ! Session #6: Language Models (University of Maryland), URL: http://www.umiacs.umd.edu/~jbg/teaching /INFM_718_2011/lecture_6.pdf, diakses: 10 Mei 2016

Jurafsky, Daniel dan Martin, James H. (2014). Speech and Language Processing (Draft of January 9, 2015)

Kneser, R. and Ney, H. (1995). Improved backing-off for M-gram language modeling. In ICASSP-95, Vol. 1, pp. 181–184.

Körner, M.C. (2013). Implementation of Modified Kneser-Ney Smoothing on Top of Generalized Language Models for NextWord Prediction, Bachelor Thesis, Universitat Koblenz Landau, URL: https://west.uni-koblenz.de/sites/default/files/BachelorArbeit_ MartinKoerner.pdf, diakses: 10 Mei 2016

Manning, C. D. dan Schütze, H. (1999). Foundations of Statistical Natural Language Processing, The MIT Press

Roberts. (2004). Word Prediction Techniques, URL: https://cs.stanford.edu/people/ eroberts/courses/soco/projects/2004-05/nlp/techniques_word.html, diakses: 10 Mei 2016

Wikipedia (2016). Zip's Law, URL: https://en.wikipedia.org/wiki/Zipf%27s_law, diakses: 10 Mei 2016