AUTOMATIC TAGGING

Proposal Skripsi

Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer

Mencerdaskan dan
Memartabatkan Bangsa

Oleh:
Muhammad Zhafran Bahij
1313619012

PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI JAKARTA

2022

LEMBAR PENGESAHAN

Dengan ini saya mahasiswa Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta

Nama : Muhammad Zhafran Bahij

No. Registrasi : 1313619012

Program Studi : Ilmu Komputer

Judul : Automatic Tagging

Menyatakan bahwa proposal skripsi ini telah siap diajukan untuk seminar pra skripsi.

Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Muhammad Eka Suryana, M. Kom	**Med Irzal, M.Kom.**
NIP. 19851223 201212 1 002	NIP. 19770615 200312 1 001

Mengetahui,

Koordinator Program Studi Ilmu Komputer

Ir. Fariani Hermin Indiyah, M.T.

NIP. 19600211 198703 2 001

HALAMAN PERSEMBAHAN

Untuk Ayah dan Ibu.

KATA PENGANTAR

Ungkapan Puji dan Syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan proposal skripsi ini dengan baik. Adapun jenis penelitian yang dengan judul *Automatic Tagging*

Dalam menyelesaikan proposal ini, penulis selalu mendapat dorongan dan bantuan. Oleh karena itu, penulis menyampaikan terima kasih kepada:

1. Para petinggi di lingkungan FMIPA Universitas Negeri Jakarta.
2. Ibu Ir. Fariani Hermin Indiyah, M.T selaku Koordinator Program Studi Ilmu Komputer.
3. Bapak Muhammad Eka Suryana, M.Kom selaku Dosen Pembimbing I yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap proposal skripsi ini.
4. Bapak Med Irzal M.Kom selaku Dosen Pembimbing II yang telah memberikan arahan dan bimbingan dalam penulisan proposal skripsi ini.
5. Ayah dan Ibu penulis yang selama ini telah mendukung dan membantu menyelesaikan proposal skripsi ini.
6. Teman-teman yang telah memberikan dukungan moral kepada penulis sehingga mampu menulis proposal skripsi sampai sejauh ini.

Dalam penulisan proposal skripsi ini, penulis menyadari bahwa dengan keterbatasan ilmu dan pengetahuan penulis, proposal ini masih jauh dari sempurna, baik dari segi penulisan, penyajian materi, maupun bahasa. Oleh karena itu, penulis

sangat membutuhkan kritik dan saran yang dapat dijadikan sebagai pembelajaran serta dapat membangun penulis agar lebih baik lagi kedepannya.

Akhir kata, penulis berharap ini bermanfaat bagi semua pihak khususnya penulis sendiri, serta menjadi semangat dan motivasi bagi rekan-rekan yang akan melaksanakan skripsi berikutnya. Semoga Tuhan Yang Maha Esa senantiasa membalas kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan proposal ini.

Terima kasih,

Jakarta, ... 2023

Penulis

DAFTAR ISI

HALAMAN PERSEMBAHAN iii

KATA PENGANTAR iv

DAFTAR ISI vii

DAFTAR GAMBAR viii

DAFTAR TABEL ix

I PENDAHULUAN 1

1.1 Latar Belakang Masalah 1

1.2 Rumusan Masalah 8

1.3 Batasan Masalah 8

1.4 Tujuan Penelitian 8

1.5 Manfaat Penelitian 8

II KAJIAN PUSTAKA 9

2.1 Representasi Bipartite Graph 9

2.1.1 Normalisasi dan Aproksimasi 10

2.1.2 Bipartite Graph Partitioning 11

2.1.3 Dengan Cluster Node Ranking 12

2.2 Online Tag Recommendation 14

2.3 Algoritma *Automatic Tag* 15

2.3.1 Two-way Poisson Mixture Model 17

2.3.2 Tag Recommendation for New Documents 20

2.3.3 Mixture Model 21

III DESAIN MODEL **24**

3.1 Flowchart Automatic Tag 24

3.2 Desain *Automatic Tag* 24

3.3 Tahapan Perancangan Automatic Tag 25

3.3.1 Penginputan 25

3.3.2 Menentukan Matriks W 26

3.3.3 Normalisasi W menggunakan *Normalized Laplacian* 29

3.3.4 Menghitung *Low Rank Approximation Matrix* menggunakan algoritma Lanczos 30

3.3.5 Melakukan partisi $\hat{W}$ ke dalam klaster K menggunakan *SRE* . 34

3.3.6 Melakukan pelabelan setiap dokumen 36

3.3.7 Menghitung *Node Rank* Rank(T) untuk Setiap Tag 36

3.3.8 Membuat Two Way Poisson Mixture Model 37

3.3.9 Rekomendasi Tag Untuk Dokumen Baru 38

3.3.10 Rekomendasi Tag Berdasarkan Ranks Tag 38

3.4 Algoritma *Automatic Tag* 38

DAFTAR PUSTAKA **41**

DAFTAR GAMBAR

Gambar 1.1 Pengguaan search engine terpopuler (Christ, 2022) 1

Gambar 1.2 High Level Google Architecture (Brin & Page, 1998) 2

Gambar 1.3 Bagian *Automatic Tagging* pada *Indexer* 4

Gambar 1.4 Relasi antara user, tag, dan dokumen (Song et al., 2008) . . . 5

Gambar 1.5 Skema *Mashup*, *Tag*, dan *API* (Shi et al., 2016) 7

Gambar 2.1 Suatu *bipartite graph* X dan Y Song et al. (2008) 9

Gambar 2.2 Smoothed Ranking Function Song et al. (2008) 14

Gambar 2.3 Dua bipartite graph dari dokumen-dokumen, kumpulan kata, dan kumpulan tag. Song et al. (2008) 15

Gambar 2.4 Distribusi Poisson dalam dua klaster. Bagian atas menggambarkan histogram dari *mixture components*. Bagian bawah menggambarkan hasil dari klasifikasi *mixture model*. Bagian (a) *three component mixtures* dan bagian (b) *two component mixtures* Song et al. (2008) 18

Gambar 3.1 Diagram alir untuk algoritma tag recommendation 24

Gambar 3.2 Melakukan online recommendation berdasarkan hasil data training 24

Gambar 3.3 Contoh simpel dua *bipartite graph* 27

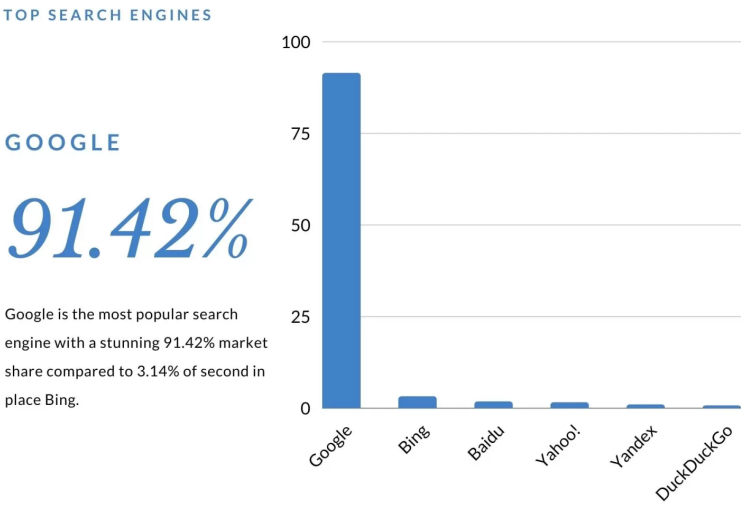
DAFTAR TABEL

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

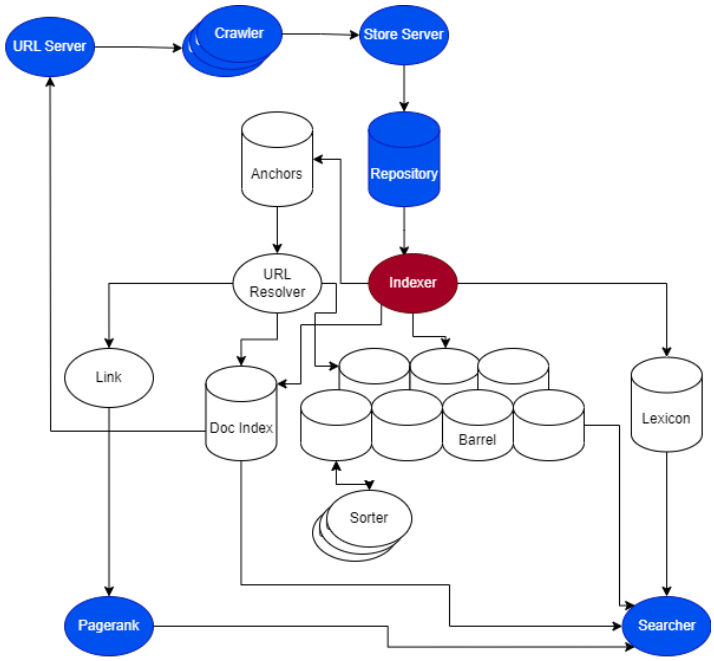
Saat ini, penggunaan *search engine* atau mesin pencari telah digunakan oleh khalayak umum untuk mencari berbagai informasi yang ada. Berdasarkan data dari (Christ, 2022), Google merupakan *search engine* yang menempati urutan pertama terpopuler, selanjutnya diikuti dengan Bing, Baidu, Yahoo!, Yandex, dan DuckDuckGo.



Gambar 1.1: Pengguaan search engine terpopuler (Christ, 2022)

Web Search Engine atau mesin pencari web merupakan suatu perangkat lunak yang digunakan untuk mencari sesuatu di internet berdasarkan kata-kata yang diberikan oleh pengguna sebagai *search terms*. Pembuatan *search engine* pertama kali dilakukan oleh Alan Emtage, Bill Heelan, dan J. Peter Deutsch pada tahun 1990. Mereka menamai *search engine* tersebut yaitu Archie (Seymour et al., 2011).

Pekerjaan utama dari *search engine* ada tiga yaitu *web crawling, indexing,* dan *searching*. *Search engine* bekerja dengan cara mengirimkan informasi tentang halaman web, Halaman tersebut di dapat dari *web crawler* suatu *automated web browser* yang mengikuti seluruh pranala yang ada di situs. Pengecualian situs yang dicari dapat dilakukan melalui "*robots.txt*". Kemudian, konten dari setiap halaman akan dianalisis untuk menentukan urutan index. Data tentang halaman web dikirim ke dalam *index database* yang nantinya akan dilakukan *query*. *Query* bisa satu kata atau lebih. Tujuan pengindeksan adalah untuk menemukan informasi secepat mungkin (Seymour et al., 2011).



Gambar 1.2: High Level Google Architecture (Brin & Page, 1998)

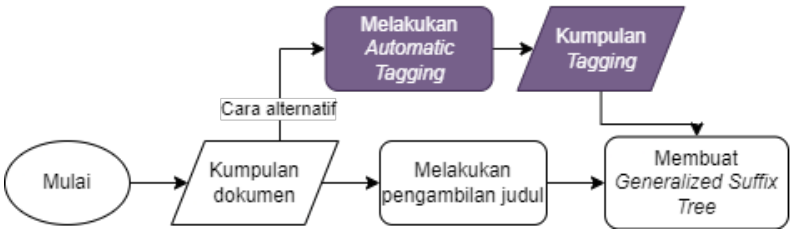
Pada Gambar 1.1, telah diperlihatkan bahwa google merupakan *search engine* terfavorit. Google ditemukan oleh Larry Page dan Sergey Brin pada tahun 1998. Salah satu keunggulan Google adalah pengaplikasian PageRank yaitu mengatasi *underspecified queries*. Sebagai contohnya, jika kita mencari “Real Madrid”, maka situs pertama kali yang terlihat adalah situs resmi “Real Madrid”.

Gambar 1.2 merupakan struktur arsitektur Google. Warna biru menunjukkan hasil penelitian yang dilakukan oleh Lazuardy Khatulistiwa dalam penelitian yang berjudul "Perancangan Arsitektur Search Engine dengan Mengintegrasikan *Web Crawler, Algoritma Page Ranking*, dan *Document Ranking*" dan warna merah menunjukkan proses penelitian dari Zaidan Pratama dalam judul "Perancangan Modul Pengindeks pada *Search Engine* Berupa *Induced Generalized Suffix Tree* untuk Keperluan Perangkingan Dokumen". Salah satu komponen dalam *search engine* milik Google adalah Indexer. Dalam penelitian Zaidan Pratama, di sana menjelaskan tentang melakukan pengindeksan melalui algoritma GST (General Suffix Tree) yang termodifikasi. (Pratama (2022))

Akan tetapi, penelitian ini memiliki kekurangan yaitu hanya bisa mengambil judul dari suatu dokumen. Beberapa dokumen terkadang tidak memiliki relevansi dengan judulnya dan itu bisa saja menyebabkan masalah sehingga diperlukan adanya tag.

Proses pengindeksan dimulai dengan memasukkan kumpulan dokumen yang dibuat menjadi GST. Kemudian, membuat GST dari kumpulan dokumen tersebut. Dari GST tersebut, nantinya akan dilakukan reduksi untuk node yang redundan atau node yang mengalami perulangan yang tidak diperlukan sehingga akan terbentuk pohon yang terinduksi untuk frekuensi f yang bernama Induced Generalized Suffx Tree-f. IGST-f ini menjadi komponen utama dalam pengindeksan. Setelah itu, program menerima masukan berupa pola kata dan batas k untuk dicari pada kumpulan dokumen. Dari sinilah kita akan mencari nilai count dari setiap node.Kemudian, mencatat jumlah dokumen dalam sublist yang tereduksi untuk setiap node.Selanjutnya, mencari nilai counter lowest common ancestor dari setiap node. Terakhir, mengenai Indeks Efisien. Untuk Top-k Document Retrieval Problem dilakukan pengurutan terhadap representasi array IGST-f yang sudah memiliki nilai

count dan mengembalikan hasil top-k yang meiliki pola P. Pratama (2022)



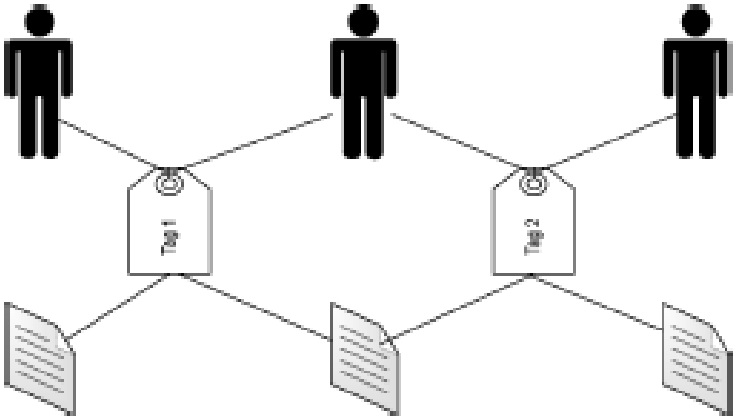
Gambar 1.3: Bagian *Automatic Tagging* pada *Indexer*

Warna ungu adalah salah satu alternatif yang dapat dilakukan yaitu melalui *automatic tagging* yang nantinya peneliti akan lakukan. Hal ini dapat dimanfaatkan agar bisa melakukan pengindeksan lebih akurat.

Tagging merupakan hal yang biasa dilakukan untuk menggambarkan suatu kata kunci yang relevan atau frasa kunci pada suatu dokumen, gambar, atau video. Dalam merambatnya perkembangan Web 2.0 aplikasi seperti Del.icio.us dan Flickr, pelayanan *tagging* mulai populer dan menarik perhatian pihak akademis dan industri. Penelitian tentang cara *Automatic tag* membuahkan hasil. Cara melakukannya dengan algoritma *Poisson Mixture Model*. Dengan cara ini, kecepatan untuk membuat *auto tag* bisa lebih cepat dibandingkan SimFusion dan VS+IG. Contohnya pada saat *Delicious Test Time*, PMM mampu menghasilkan 1,23 detik saat proses *auto tag*, sedangkan SimFusion membutuhkan waktu 6,4 detik dan VS+IG membutuhkan waktu 77,43 detik. Selain kecepatan, PMM juga mampu di atas SimFusion serta VS+IG secara signifikan dalam hal akurasi, presisi, dan *recall*. (Song et al., 2008)

Selain itu, *tagging* juga digunakan untuk membantu pengorganisasian, *browsing*, dan pencarian. Seperti *image tagging* yang digunakan oleh Flickr, *web* page tagging yang digunakan oleh Del.ico.us, dan *social tagging* yang digunakan oleh Facebook, semua sistem tersebut menjadi populer dan dipergunakan di penjuru Web. (Sood, 2007)

Secara umum, sumber yang ter-*tag* biasanya berasosiasi dengan satu atau lebih *tag*. Selain itu, sumber yang ter-*tag* berasosiasi terhadap penggunanya sendiri. Sebagai contoh, *tagging* terhadap dokumen d yang dilakukan oleh pengguna u dengan *tag* t dapat direpresentasikan sebagai tiga kesatuan (u, d, t). Dengan menggunakan pendekatan itu, dapat terbentuk suatu graf yang digambarkan sebagai berikut.



Gambar 1.4: Relasi antara user, tag, dan dokumen (Song et al., 2008)

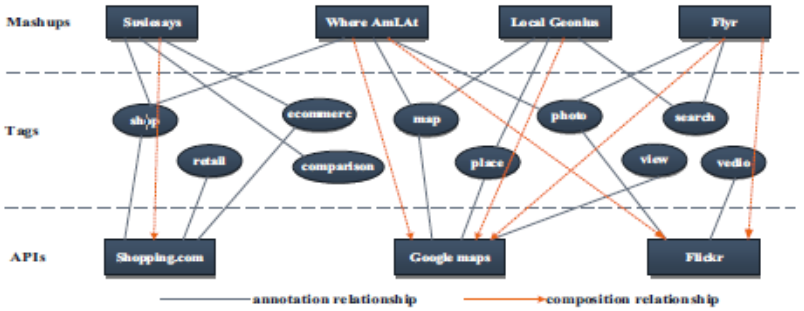
Dengan relasi pada gambar 1.4 , rekomendasi *tag* dapat dilakukan dengan dua jenis menurut Song yaitu jenis pendekatan melalui pengguna dan jenis pendekatan melalui dokumen. Rekomendasi *tag* merupakan suatu sistem yang di mana sistem tersebut menampilkan *tag* yang relevan pada suatu dokumen agar pengguna bisa memperhitungkan apakah *tag* yang ditampilkan itu ingin dipakai atau tidak. Melalui pendekatan *user*, sistem ini akan mengolah rekomendasi *tag* berdasarkan *tag-tag* yang telah dilakukan *user* sebelumnya dan merekomendasikan tag yang mirip dengan *user* ini atau kelompok dari *user* tersebut. Berbeda halnya dengan pendekatan dokumen, cara ini dilakukan dengan cara menklasterisasikan dokumen-dokumen tersebut ke dalam topik-topik yang berbeda. Topik yang sama pada suatu dokumen akan memiliki *tag* yang diasumsikan lebih mirip dibandingkan

dokumen yang berbeda topik. Namun, di antara kedua cara ini, yang dinilai kurang efektif adalah melalui pendekatan *user*. Pertama, berdasarkan penelitian dari Farooq, distribusi dari *user* vs *tag* mengikuti *long tail power law distribution*. Itu artinya, hanya sebagian kecil porsi dari *user* yang melakukan *tag* dengan panjang atau meluas. Sebagai tambahan, penggunaan *tag* yang berulang juga terbilang rendah, tetapi pertumbuhan perbendaharaan *tag* terus berkembang. Dengan sedikit pengguna relatif yang di dapat, pendekatan *user* akan sulit untuk mencari model mana yang cocok buat untuk melakukan rekomendasi *tag* yang efektif. Berbanding terbalik dengan pendekatan dokumen yang lebih kokoh karena kekayaan informasi yang ada di dokumen. Bahkan, *tag* dan kata akan menciptakan relasi yang potensial antara topik dan konten di suatu dokumen yang di mana tag dianggap sebagai kelas label untuk dokumen dalam skenario *supervised learning* atau kesimpulan dari dokumen dalam skenario *unsupervised learning*. (Song et al., 2011)

Namun, percobaan ini hanya terbatas pada CiteULike dan del.icio.us. Selain itu, situs dari CiteULike yang notabene-nya adalah tagging dataset untuk dokumen saintifik kini berubah menjadi situs gambling, sedangkan situs dari del.icio.us kini tidak bisa dibuka secara umum.

Beberapa tahun kemudian, suatu penelitian membahas mengenai *automatic mashup tag*. Secara sederhana, *mashup* adalah suatu *web service* yang di mana merupakan kumpulan dari kombinasi beberapa Web *API* dan konten dari berbagai sumber. Berbeda dengan rekomendasi *tag* yang menggunakan pendekatan dengan konten tekstual, di dalam *Web services* terdapat banyak sekali relasi seperti komposisi relasi antara *mashup* dengan *API* dan anotasi berelasi antara *API* dan *tag*. (Shi et al., 2016)

Selain teks, *tag* juga digunakan dalam hal yang bersifat non teks seperti video, musik, dan gambar. Dalam suatu video, *tag* sangat diperlukan untuk



Gambar 1.5: Skema *Mashup*, *Tag*, dan *API* (Shi et al., 2016)

menentukan relevansi antara pencarian yang diinginkan dengan isi video. Meskipun beberapa platform video seperti Youtube menyediakan judul dan deskripsinya, bisa saja judul tersebut tidak ada keterkaitannya dengan video dan deskripsinya yang sangat panjang sehingga orang malas untuk membaca. Manfaat dalam *tag* video ada dua yaitu bisa menemukan daftar video yang representatif dan *tag* dapat mendukung untuk melakukan penemuan tentang video yang kontennya berhubungan dengan video yang telah ditonton. (Parra et al., 2018)

Untuk kasus *automatic tag* pada musik, *Automatic music tagging* adalah *multi label binary classification* yang bertujuan untuk memprediksi *tag* yang relevan pada suatu lagu. *Tag* tersebut membawa informasi musik semantik yang nantinya dapat digunakan untuk membuat aplikasi seperti rekomendasi musik. (Won et al., 2020)

Setelah pemaparan di atas, peneliti ingin membuat penelitian terkait *automatic tagging* melalui pendekatan dokumen yang nantinya pengindeksan dengan algoritma GST dari penelitian Zaidan Pratama pada search engine dari penelitian Lazuardy Khatulistiwa. Awalnya pengindeksan dilakukan hanya dengan judulnya, tetapi setelah penelitian *automatic tagging*, diharapkan akan diproses melalui *tag*.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah diuraikan, maka perumusan masalah pada penelitian ini adalah "Bagaimana cara melakukan Automatic Tagging melalui bahasa pemrograman Python".

1.3 Batasan Masalah

Batasan masalah dalam penelitian ini yaitu:

1. Hanya menggunakan bahasa pemrograman Python

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk merancang dan membangun sebuah sistem yang dapat melakukan automatic tagging terhadap dokumen.

1.5 Manfaat Penelitian

Beberapa manfaat yang ingin diperoleh dari penelitian ini adalah sebagai berikut:

1. Bagi Peneliti

Menambah pengetahuan penulis tentang automatic tag terhadap dokumen

2. Bagi Peneliti Selanjutnya

Diharapkan metode yang diusulkan pada penelitian ini dapat membantu penelitian selanjutnya dalam mengembangkan sistem yang lebih kompleks dan bermanfaat.

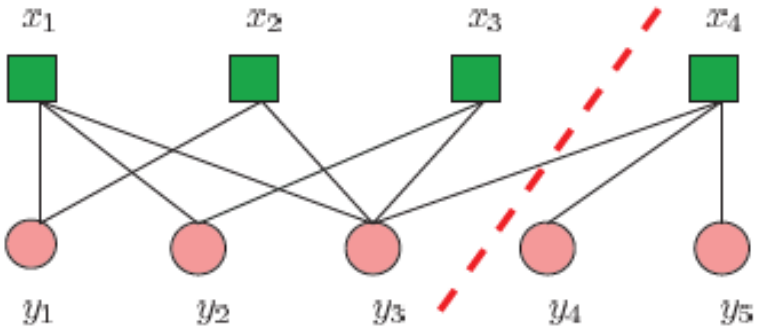
BAB II

KAJIAN PUSTAKA

2.1 Representasi Bipartite Graph

Mendefinisikan suatu graf $G = (V, E, W)$ sebagai suatu set dari titik-titik V dan hubungan mereka dengan garis E, dengan W sebagai bobot dari garis-garis tersebut. Sebagai contoh, w_{ij} sebagai bobot dari garis di antara titik i dan j.

Graf G disebut *bipartite* jika mengandung dua kelas titik X dan Y sebagai berikut $V = X \cup Y$ dan $X \cap Y = \emptyset$ setiap garis $e_{ij} \in E$ memiliki *endpoint* i di dalam X dan *endpoint* lain j di dalam Y. Biasanya, X dan Y merujuk kepada perbedaan tipe objek dan E merepresentasikan relasi antara keduanya. Di dalam konteks representasi dokumen, X merupakan suatu set dokumen, sedangkan Y adalah $terms$ dan w_{ij} merupakan banyaknya *term* j yang muncul di dalam dokumen i. Perlu dicatat bahwa *weighted adjacency matrix* W untuk *bipartite graph* selalu simetris. Sebagai contoh, gambar berikut adalah *undirected bipartite graph* dengan 4 dokumen dan 5 *term*.



Gambar 2.1: Suatu *bipartite graph* X dan Y Song et al. (2008)

2.1.1 Normalisasi dan Aproksimasi

Normalisasi biasanya digunakan pertama kali untuk bobot matriks W untuk mengeliminasi *bias*. Jalur yang paling lurus untuk menormalisasikan W adalah normalisasi baris yang tidak mengambil akun dari simetri pada W. Namun, untuk memahami simetri pada W, Song dkk menggunakan *normalized graph Laplacian* untuk aproksimasi W. *Normalized Laplacian* $L(W)$ sebagai berikut.

$$L(W)_{ij} = \begin{cases} 1 - \frac{w_{ij}}{d_i} & \text{jika } i = j, \\ -\frac{w_{ij}}{\sqrt{d_i d_j}} & \text{jika } i \text{ dan } j \text{ berdekatan} \\ 0 & \text{selain itu,} \end{cases}$$

di mana d_i adalah *degree* luar dari titik i, pada persamaan $d_i = \sum w_{ij}, \forall j \in V$. Kemudian, kita bisa mendefinisikan matriks diagonal D di mana $D_{ii} = d_i$. Oleh karena itu, *normalize Laplacian* dapat direpresentasikan sebagai berikut.

$$L(W) = D^{(-1/2)} W D^{(-1/2)} \tag{2.1}$$

Untuk *dataset* berskala besar seperti situs *corpora* dan koleksi gambar, spasi fitur mereka biasanya mengandung jutaan vektor dengan dimensi yang sangat tinggi (contohnya, x = 10^6, y = 10^7). Oleh karena itu, biasanya ini sering diinginkan untuk mencari *low rank matrix* W *komplemen* untuk aproksimasi $L(W)$ dengan tujuan untuk mengurangi biaya komputasi, mengekstrak korelasi, dan menghapus *noise*. Metode dekomposisi matriks tradisional misalnya adalah *Single Value Decomposition* dan *eigenvalue decomposition*, membutuhkan waktu *superlienar* untuk perkalian matriks vektor jadi biasanya mereka tidak menggunakannya sampai pengaplikasian di dunia nyata.

Untuk *symmetric low rank apporiximation*, di sini menggunakan algoritma Lanczos yang secara iteratif mencari nilai eigen dan vektor eigen dari matriks persegi. Diberikan nxn *sparce symmetric matrix* A dengan nilai eigen:

$$\lambda \geq ... \geq \lambda_n \geq 0 \tag{2.2}$$

Algoritma Lanczos mengkomput k x k *symmetric tridiagonal matrix* T, yang nilai eigennya mengaproksimasi nilai eigen dari A, dan vektor eigen dari T bisa digunakan untuk mengaproksimasi vektor eigen A, dengan k lebih kecil dibandingkan n. Dengan kata lain, T memuaskan:

$$||A - T||_F \leq e||A||_F \tag{2.3}$$

di mana $||.||_F$ denotasi *Frobenius norm*, dengan e sebagai variabel terkendali. Sebagai contoh, untuk menangkap 95% varians dari A, e diatur sebagai 0.05.

2.1.2 Bipartite Graph Partitioning

Untuk melakukan multi-klastering pada *bipartite graph*, Song et al. (2008) menggunakan algoritma *Spectral Recursive Embedding (SRE)*. Secara essensial, *SRE* digunakan untuk menkontruksi *partition* dengan meminimalisir normalisasi dari total pada bobot garis di antara pasangan yang tidak cocok pada suatu garis, misalnya $min_{\Pi(A,B)}Ncut(A,B)$, di mana A dan B adalah pasangan yang cocok dalam partisi dengan A^c dan B^c menjadi yang lain. Normalisasi varian dari *edge cut* $Ncut(A,B)$ didefinisikan sebagai berikut.

$$Ncut(A,B) = \frac{cut(A,B)}{W(A,Y) + W(X,B)} + \frac{cut(A^c,B^c)}{W(A^c,Y) + W(X,B^c)}, \tag{2.4}$$

di mana

$$\begin{aligned} cut(A,B) &= W(A,B^c) + W(A^c,B) \\ &= \sum_{i\in A, j\in B^c} w_{ij} + \sum_{i\in A^c, j\in B} w_{ij}, \end{aligned} \tag{2.5}$$

Rasional dari $Ncut$ tidak hanya mencari partisi dengan perpotongan garis kecil, tetapi juga mepartisinya sepadat mungkin. Ini berguna untuk aplikasi dari *tagging document* di mana dokumen dengan setiap partisi secara ideal berfokus kepada satu topik spesifik. Sebagai hasil, semakin padat suatu partisi, semakin baik yang dokumen relevan dan *tag* terkelompokkan bersamaan.

2.1.3 Dengan Cluster Node Ranking

Song et al. (2008) mendefinisikan dua pengukuran baru *N-Precision* dan *N-Recall* untuk *node ranking*. *N-Precision* dari titik i merupakan jumlah total dari bobot pada garis yang terkoneksi kepada titik dalam klaster yang sama, dibagi dengan jumlah total dari bobot garis yang ada di dalam kluster. Label klaster i sebagai $C(i)$,

$$np_i = \frac{\sum_{j=1} nw_{ij}\Pi[C(j) = C(i)]}{\sum_{j,k=1} nw_{jk}\Pi[C(j) = C(k) = C(i)]}, j, k \neq i, \tag{2.6}$$

Untuk graf tidak berbobot, persamaan di atas setara dengan banyaknya garis yang berasosiasi dengan titik i di dalam klaster $C(i)$, dibagi dengan total garis yang ada di dalam klaster $C(i)$. Secara umum, *N-precision* mengukur seberapa pentingnya titik pada suatu klaster, di dalam perbandingan dengan titik lain. Dalam konteks pada suatu dokumen, klasternya adalah suatu set topik dari dokumen dan bobot dari titik-titik kata menunjukkan frekuensi dari kata-kata yang muncul pada topik tersebut.

Dengan *cluster determined*, *denominator equation* bersifat konstan, jadinya semakin banyak bobot yang dimiliki pada suatu titik, semakin penting pula.

Kontrasnya, *N-recall* digunakan untuk menghitungkan probabilitas posterior dari titik i pada klaster yang diberikan dan *inverse* pembagian dari garis i yang berasosiasi dengan klasternya.

$$nr_i = \frac{|E_i|}{\sum_{j=1} nw_{ij} \Pi[C(j) = C(i)]}. \tag{2.7}$$

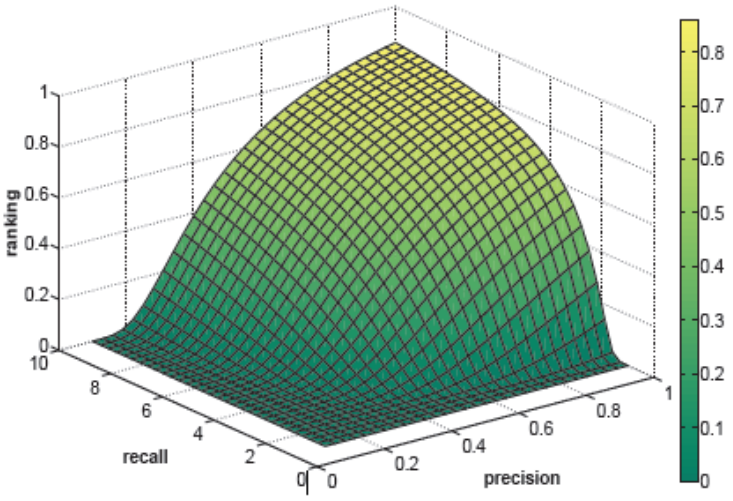
Hal tersebut merupakan bukti bahwa *N-Recall* selalu tidak kurang dari satu. Semakin lebar *N-Recall*, maka semakin mungkin bahwa kata tersebut berasosiasi dengan suatu topik yang spesifik.

Diberikan np_i dan nr_i, kita akan melakukan estimasi dari peringkat i:

$$Rank_i = \begin{cases} \exp(-\frac{1}{r(i)^2}) r(i) \neq 0, \\ 0 r(i) = 0, \end{cases} \quad \textbf{di mana}\, r(i) = (np_i) * log(nr_i) \tag{2.8}$$

Berdasarkan gambar berikut, fungsi perangkingan dari Song et al. (2008) merupakan pengganti yang dihaluskan yang proporsional untuk presisi titik (*node precision*) dan *recall*, hasilnya terjamin berada di kisaran *range (0, 1)*.

Potensi pengalikasian dari metodologi *bipartite graph node ranking* termasuk interpetasi relasi dokumen dan *author* yaitu menentukan relasi sosial (misal "*hub*" dan "*authority*") dari *author* di dalam satu topik penelitian yang sama, dan mencari representatif dokumen dari suatu topik. Di sini, mereka mengaplikasikan *framework* ini untuk melakukan rekomendasi tag dengan *ranking node* yang merepresentasikan tag pada setiap klaster.



Gambar 2.2: Smoothed Ranking Function Song et al. (2008)

2.2 Online Tag Recommendation

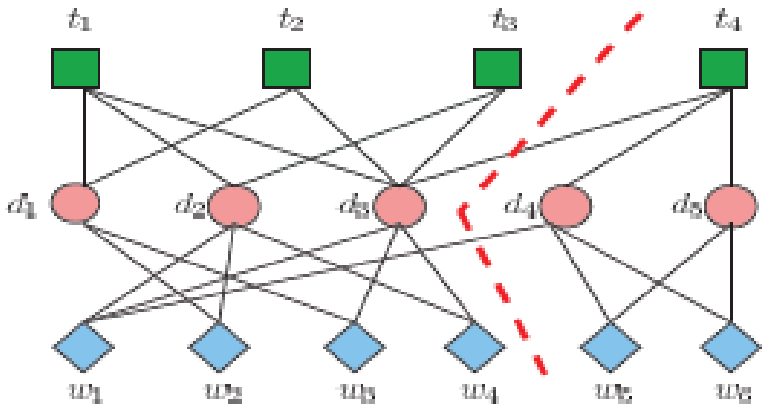
Suatu dokumen biasanya mengandung beberapa kata dan beberapa *tag* yang dianotasi oleh *user*. Hubungan antara dokumen, kata-kata, dan *tag* bisa direpresentasikan dengan gambar dua *bipartite graph* yang akan ditunjukkan oleh gambar setelah ini.

Graf yang berbobot dapat ditulis dsebagai berikut

$$W = \begin{pmatrix} 0 & A & 0 \\ A^T & 0 & B \\ 0 & B^T & 0 \end{pmatrix} \tag{2.9}$$

di mana A dan B sebagai matriks-matriks inter-relasi antara *tag* dan dokumen-dokumen, dokumen-dokumen, dan kata-kata berturut-turut.

Diberikan suatu representasi matriks, pendekatan lurus untuk *tag* rekomendasi adalah dengan melihat kemiripannya antara dokumen *query* dan dokumen *training* dengan fitur-fitur kata, kemudian lakukan *top ranked tags* dari



Gambar 2.3: Dua bipartite graph dari dokumen-dokumen, kumpulan kata, dan kumpulan tag. Song et al. (2008)

dokumen yang paling mirip. Pendekatan ini biasanya direferensikan sebagai filter kolaborasi. Namun, pendekatan ini tidaklah efisien untuk skenario dunia nyata. Untuk mengambil kelebihan dari algoritma *node ranking*, Song dkk menggunakan *Possion Mixture Model (PMM)* yang secara efisien menentukan sampel *membership* sebaik klastering kata-kata dengan makna yang mirip. Sebelum melakukan *mixture model*, di sini terdapat rangkuman algoritma yang digunakan untuk rekomendasi *tag* di dalam Algoritma 1.

2.3 Algoritma *Automatic Tag*

Dua tahap *framework* ini bisa diinterpretasikan sebagai prosedur *unsupervised-supervised learning*. Saat tahap *offline learning stage*, titik-titik akan dipartisi ke dalam klaster-klaster menggunakan *unsupervised learning*, label klaster dipasangkan kepada *document node* sebagai *class label* mereka, dan tag akan diberikan *rank* di dalam setiap klaster. *Mixture model* kemudian dibangun berdasarkan distribusi dari dokumen-dokumen dan kata-kata. Di dalam *online recommendation stage*, suatu dokumen diklasifikasikan ke dalam *predefined cluster*

Algorithm 1 Online Tag Recommendation

1: **Input** $(\mathcal{D}, S, T), K, M, L$
Kumpulan dokumen: $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$
Word vocabulary: $S = \{S_1, \ldots, S_k\}$
Tag vocabulary: $T = \{T_1, \ldots, T_n\}$
banyaknya klaster: $K \in R$
banyaknya komponen-komponen: $M \in R$
banyaknya klaster-klaster kata: $L \in R$

Offline Computation

2: Menunjukkan bobot terdekat matriks W seperti persamaan (2.9)
3: Normalisasi W menggunakan *Normalized Laplacian*
$L(W) = D^{(-1/2)} W D^{(-1/2)}$ persamaan (2.1)
4: Komputasi *low rank approximation matrix* menggunakan Lanczos:
$\tilde{W} \simeq L(W) = Q_k T_k Q_k^T$
5: Partisi $\tilde{W}$ ke dalam klaster K menggunakan SRE,
$\tilde{W} = \left\{\tilde{W}_1, \ldots, \tilde{W}_K\right\}$
6: Tandai label ke dalam setiap dokumen $\mathcal{D}_j, j \in \{1, \ldots m\}$
$C\left(\mathcal{D}_j\right) \in \{1, \ldots, K\}$
7: Hitung *node rank* $Rank(T)$ untuk setiap tag $T_{i,k}$ in cluster
$k, i \in \{1, \ldots, n\}, k\{1, \ldots, K\}$ persamaan (2.8)
8: Buat *Poisson Mixture Model* untuk $(\tilde{B}, C(\mathcal{D}))$ dengan M komponen-komponen dan L klaster kata-kata, di mana $\tilde{B}$ denotasi matriks inter-relationship pada suatu dokumen-dokumen dan kata-kata di dalam $\tilde{W}$ persamaan (2.9)

Online Recommendation

9: Untuk setiap dokumen tes Y, kalkulasikan posterior probabilitas
$P(C = k \mid D = Y)$ di dalam setiap klaster k, dan denotasi membership pada Y sebagai $C(Y) = \{c(Y, 1), \ldots, c(Y, K)\}$ persamaan (2.16)
10: Tag rekomendasi berdasarkan perangkingan pada tag, yaitu *joint probability* pada tag-tag T dan dokumen Y, $R(T, Y)$ persamaan (2.17)

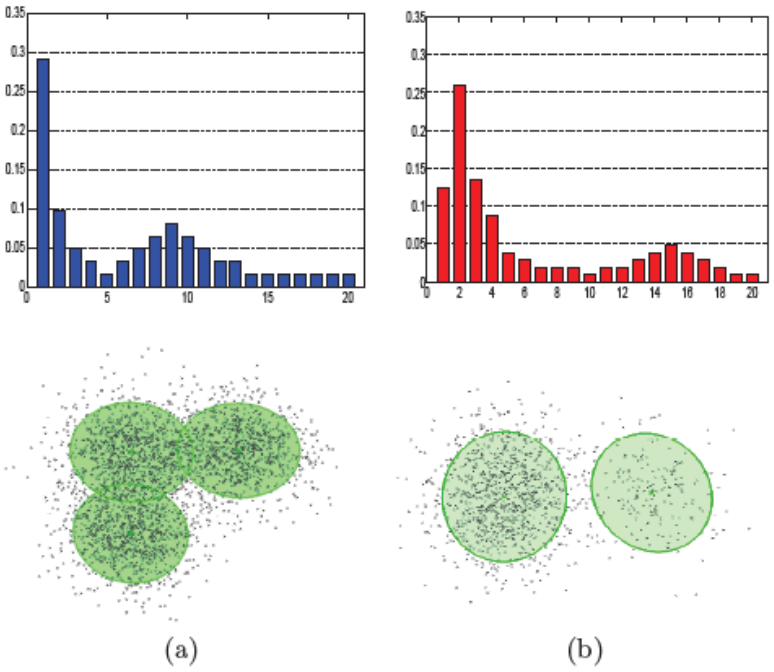
yang telah didapatkan di dalam tahap pertama oleh Naive Bayes jadi *tag* tersebut bisa direkomendasikan di pengurutan secara terbalik dari rank mereka. Untuk menghindari kebingungan, Song et al. (2008) akan merujuk kepada klaster yang diinginkan dengan mempartisi algoritima di dalam tahap pertama sebagai *classess* di dalam sesi selanjutnya.

2.3.1 Two-way Poisson Mixture Model

Song et al. (2008) mengusulkan untuk menggunakan Poisson Mixture Model untuk mengestimasi distribusi data pada vektor dokumen sebab algoritma tersebut cocok digunakan dibandingkan standard Poissons dengan memproduksi estimasi lebih baik pada data varians dan cukup mudah untuk estimasi parameter. Akan tetapi, itu membutuhkan waktu untuk mencocokkan data latihan. Algoritma ini efisien untuk memprediksi label kelas dari dokumen baru setelah model tersebut selesai dibuat. Karena stabilitas numerikal pada pendekatan statistika ini, biasanya hasilnya dapat diandalkan. Sejak hanya estimasi probabilitas yang dilibatkan, ini dapat diandalkan untuk real-time process.

Namun, pendekatan tradisional unsupervised learning dari mixture model tidak selalu diandalkan untuk menghadapi klasifikasi dokumen. Mempertimbangkan kelemahan dan tingginya dimensi pada matriks document-word di mana kebanyakan masukkan berupa 0 dan 1, model bisa saja gagal untuk memprediksi distribusi yang benar (yaitu *probability mass faunction*) pada komponen yang berbeda. Sebagai hasilnya, klastering kata adalah langkah yang diperlukan sebelum mengestimasi komponen-komponen di dalam model. Di sini akan dilakukan *two-way Poisson Mixture Model* untuk secara bersamaan melakukan *cluster word feature* dan klasifikasi dokumen.

Diberikan suatu dokumen $D = \{D_1, ..., D_p\}$, di mana p adalah dimensi,



Gambar 2.4: Distribusi Poisson dalam dua klaster. Bagian atas menggambarkan histogram dari *mixture components*. Bagian bawah menggambarkan hasil dari klasifikasi *mixture model*. Bagian (a) *three component mixtures* dan bagian (b) *two component mixtures* Song et al. (2008)

distribusi pada vektor dokumen di setiap kelas dapat diestimasikan dengan menggunakan *parametric mixture model*.

kelas label $C = \{1, 2, ..., K\}$, kemudian

$$P(D = d|C = k) = \sum_{m=1}^{M} \pi_m \Pi(F(m) = k) \prod_{j=1}^{p} \phi(d_j|\lambda_{j,m}), \tag{2.10}$$

di mana π_m adalah *prior probability* dari komponen m, dengan $\sum_{m=1}^{M} \pi_m = 1.\Pi(F(m) = k)$ adalah fungsi indikator yaitu apakah komponen m milik kelas k, dan ϕ merujuk kepada *probability mass function* dari distribusi Poisson, $\phi(d_j|\lambda_{j,m}) = e^{-\lambda_{j,m}} \lambda_{j,m_j^d} \, d_j!$.

Pada jalur ini, setiap kelas adalah *mixture model* dengan distribusi yang multivariasi dengan memiliki Variabel yang mengikuti distribusi Poisson. Gambar

2.4 menunjukkan histogram pada dua *mixture* yang bisa dianggap sebagai pmf dari dua *Poisson mixture*.

Asumsi Song et al. (2008) terkait setiap kelas, kata-kata pada dokumen yang berbeda memiliki parameter Poisson yang setara, saat dokumen-dokumen di dalam kelas yang berbeda, kata-kata bisa saja mengikuti perbedaan distribusi Poissson. Untuk mempersimpel, Song et al. (2008) mengasumsi bahwa semua kelas memiliki nomor yang dari klaster-klaster kata. *Denote* $l = \{1,,,L\}$ untuk menjadi klaster-klaster kata, kata-kata yang sama klaster kata m akan memiliki parameter yang sama yaitu $\lambda_{i,m} = \lambda_{j,m} = \lambda_{l,m}$ untuk $c(i,k) = c(j,k)$ di mana $c(i,k)$ *denote* label klaster pada kata i di dalam kelas k. Berarti, persamaan sebelumnya dapat dipermudah menjadi (dengan $L \ll p$):

$$P(D = d|C = k) \propto \sum_{m=1}^{M} \pi_m \Pi(F(m) = k) \prod_{l=1}^{L} \phi(d_{k,l}|\lambda_{l,m}), \tag{2.11}$$

Estimasi Parameter Dengan kelas yang dideterminasi, berikutnya masukkan algoritma EM untuk mengestimasi parameter Poisson $\lambda_{l,m}, l \in \{1,...,L\}, m \in \{1,...,M\}$, *prior of mixture component* π_m, dan indeks klaster kata $c(k,j) \in \{1,...,L\}, k \in \{1,...,K\}, j \in \{1,...,p\}$.

Estimasi *E-step posterior probability* $p_{i,m}$ sebagai berikut.

$$p_{i,m} \propto \pi_m^{(t)} II(C(i)) \prod_{j=1}^{p} \theta\left(d(i,j) \mid \tilde{\lambda}_{m,i,j}^{(t)}\right) \tag{2.12}$$

M-step menggunakan $p_{i,m}$ untuk memaksimalkan *objective function*

$$\begin{aligned} &L\left(\pi_m^{(t+1)}, \tilde{\lambda}_{m,l}^{(t+1)}, c^{(t+1)}(k,j) \mid \pi_m^{(t)}, \tilde{\lambda}_{m,l}^{(t)}, c^{(t)}(k,j)\right) \\ &= \max \sum_{i=1}^{n} \sum_{m=1}^{M} p_{i,m} \log\left(\pi_m^{(t+1)} I(C(i)) \prod_{j=1}^{p} \theta\left(d(i,j) \mid \tilde{\lambda}_{m,i,j}^{(t+1)}\right)\right), \end{aligned} \tag{2.13}$$

dan *update* parameter

$$\pi_m^{(t+1)} = \frac{\sum_{i=1}^{n} p_{i,m}}{\sum_{m'=1}^{M} \sum_{i=1}^{n} p_{i,m'}}, \tag{2.14}$$

$$\tilde{\lambda}_m^{(t+1)} = \frac{\sum_{i=1}^{n} p_{i,m} \sum_j d(i,j) I(C(i))}{|d(i,j)| \sum_{i=1}^{n} p_{i,m}}, \tag{2.15}$$

di mana $|d(i,j)|$ denotasi bilangan dari j di dalam komponen l.

Setelah $\tilde{\lambda}_m^{(t+1)}$ telah diperbaiki, indeks klaster kata $c^{(t+1)}(k,j)$ bisa ditemukan dengan melakukan *linear search* pada semua komponen-komponen:

$$c^{(t+1)}(k,j) = \arg\max_l \sum_{i=1}^{n} \sum_{m=1}^{M} \log\left(d(i,j) \mid \tilde{\lambda}_{m,l}^{(t+1)}\right)$$

2.3.2 Tag Recommendation for New Documents

Normalnya, label kelas $C(d_t)$ dari suatu dokumen baru d_t ditentukan oleh $\hat{C}(x) = \arg\max_k P(C = k \mid D = d_t)$. Namun, di kasus ini, Song et al. (2008) determinasi *mixed membership* pada suatu dokumen dengan menkalkulasi *posterior probabilities* pada suatu kelas dengan $\sum_{k=1}^{K} P(C = k \mid D = d_t) = 1$. Mengaplikasikan persamaan (2.12) dan *Bayes Rule*,

$$\begin{aligned} P(C = k|D = d_t) &= \frac{P(D = d_t|C = k)P(C = k)}{P(D = d_t)} \\ &= \frac{\sum_{m=1}^{M} \pi_m II(F(m) = k) \prod_{l=1}^{L} \phi(d_{k,l}\tilde{\lambda}_{l,m}) P(C = k)}{P(D = d_t)} \end{aligned} \tag{2.16}$$

di mana $P(C = k)$ adalah *prior probabilites* dari kelas k dan set seragam. Akhirnya, probabilitas untuk setiap *tag* $T_i, i \in \{1, \ldots, n\}$ diasosiakan dengan sampel adalah

$$\begin{aligned} R(T_i, d_t) &= P(T = T_i | D = d_t) \\ &= Rank_{T_i} * P(C = x | D = d_t) \end{aligned} \tag{2.17}$$

Dengan perangkingan *tag* dari terbesar ke terkecil pada probabilitas mereka, *top ranked tags* dipilih untuk rekomendasi.

2.3.3 Mixture Model

Salah satu *Mixture Model* yang biasa ... yaitu *Gaussian Mixture Model* (*GMM*), *GMM* adalah suatu *parametric probability density function* yang merepresentasikan *weighted sum* dari kepadatan komponen *Gaussian*. Biasanya, *GMM* digunakan sebagai *parametric model* dari distribusi probabilitas pada pengukuran yang kontinu. Parameter *GMM* diestimasi dari data latih menggunakan algoritma *Expectation-Maximization* (*EM*).

Gaussian Mixture Model adalah *weighted sum* dari *M* kepadatan komponen *Gaussian* dengan persamaan sebagai berikut.

$$p(x|\lambda) = \sum_{i=1}^{M} w_i g(x|\mu_i, \Sigma_i) \tag{2.18}$$

Di mana x adalah D *dimensional continuous* vektor data, w_i, i = 1, ...,M adalah bobot mikstur, dan $g(x|miu_i, \Sigma_i)$, i = 1,...,M adalah kepadatan komponen *Gaussian*. Kepadatan komponen *Gaussian* adalah sebagai berikut.

$$g(x|\mu_i, \Sigma_i) = \frac{1}{(2\Phi)^{D/2}|\Sigma_i|^{1/2}} exp\{-\frac{(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i)}{2}\} \tag{2.19}$$

dengan μ_i sebagai *mean vector* dan σ_i sebagai matriks kovarian. *Mixture*

Weights memenuhi persamaan $\sum_{i=1}^{M} w_i = 1$.

Beberapa parameter tersebut dikumpulkan menjadi persamaan berikut.

$$\lambda = \{w_i, \mu_i, \sigma_i\} i = 1, ..., M \tag{2.20}$$

Pada tahap selanjutnya adalah menghitung *Maximum Likelihood Parameter*. Diberikan data latih berupa vektor-vektor dan konfigurasi *GMM*. Dari sini akan dilakukan estimasi dari parameter *GMM* λ. Metode paling populer dalam menentukan ini adalah estimasi *Maximum Likelihood*.

Tujuan *Maximum Likelihood* adalah mencari parameter model yang memaksimalkan *likelihood* pada GMM dari data latih yang diberikan. Untuk setiap T *training vector* X = $\{x_1, ..., x_T\}$, *GMM likelihood*, asumsikan antar vektor adalah independen, maka sebagai berikut.

$$p(X|\lambda) = \prod_{t=1}^{T} p(x_t|\lambda) \tag{2.21}$$

Sayangnya, persamaan ini adalah fungsi tidak linear dari parameter λ dan tidak mungkin untuk melakukan pemaksimalan secara langsung. Namun, parameter *Maximum Likelihood* dapat diestimasikan dengan cara iteratif dengan menggunakan algoritma *Expectation-maximization* (*EM*).

Ide dasar dari algoritma *EM* adalah memulai model awal λ untuk mengistmasi model baru $\tilde{\lambda}$ lalu menjadi $p(X|\tilde{\lambda}) \geq p(X|\lambda)$. Model baru akan menjadi model awal untuk iterasi selanjutnya dan proses tersebut akan berlanjut sampai batas tertentu yang ditetapkan. Model awal biasanya *derived* dengan menggunakan *VQ estimation*.

Dalam setiap iterasi *EM*, formula reestimasi selanjutnya akan digunakan untuk meningkatkan nilai *likelihood*.

Mixture Weights

$$\tilde{w}_i = \frac{1}{T}\sum_{t=1}^{T} Pr(i|x_t, \lambda) \tag{2.22}$$

Means

$$\tilde{\mu}_i = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda)x_t}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} \tag{2.23}$$

Varians

$$\tilde{\sigma}_i^2 = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda)x_t^2}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} - \tilde{\mu_i^2} \tag{2.24}$$

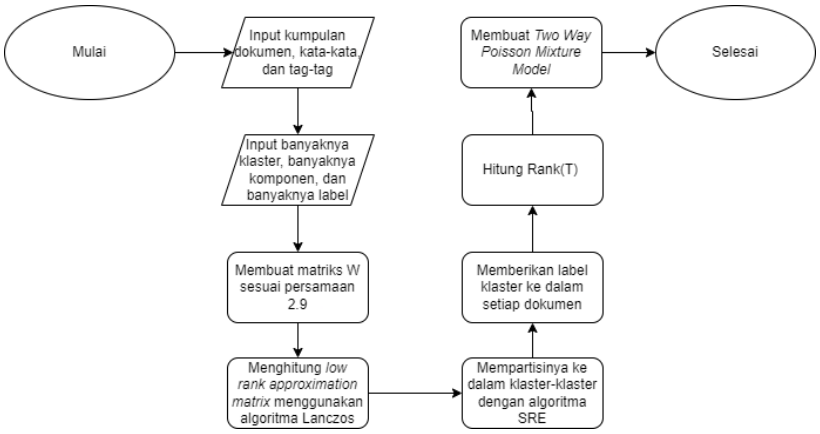
Kemudian *posterior probability* untuk komponen i sebagai berikut.

$$Pr(i|x_t, \lambda) = \frac{w_i g(x_t|\mu_i, \Sigma_i)}{\sum_{k=1}^{M} w_k g(x_t|\mu_i, \Sigma_i)} \tag{2.25}$$

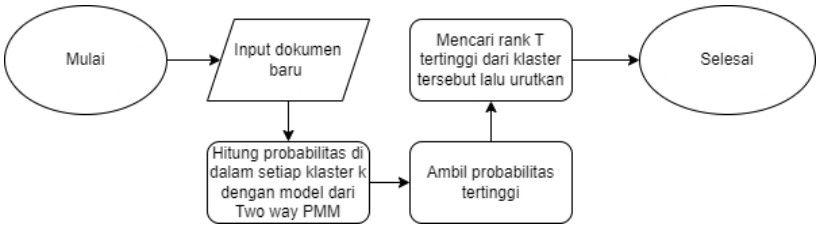
BAB III

DESAIN MODEL

3.1 Flowchart Automatic Tag



Gambar 3.1: Diagram alir untuk algoritma tag recommendation



Gambar 3.2: Melakukan online recommendation berdasarkan hasil data training

3.2 Desain *Automatic Tag*

Pada proses pembuatan algoritma untuk pembuatan *automatic tag*, penulis menggunakan algoritma *Online Tag Recommendation*. Pada algoritma ini, teradapat dua tahap besar yaitu *Offline Computation* dan *Online Recommendation*. Kedua tahapan ini dijalankan dimulai dari *offline computation* dan diakhiri dengan *online recomenndation*.

Offline Computation merupakan tahapan untuk mempelajari data-data dari kumpulan dokumen-dokumen (D), tag-tag (T), serta kata-kata (S) yang telah diberikan. Nantinya, data-data tersebut akan dibentuk ke dalam klaster-klaster (K) lalu setiap klaster akan diberikan label (L). Setiap dokumen-dokumen (D) akan diberikan suatu label yang mewakili bahwa ia termasuk ke dalam golongan klaster tersebut. Lalu, setiap klaster akan diberikan suatu *tag-tag*. Namun, *tag-tag* yang diberikan terdapat suatu *rank* atau peringkat. Terakhir adalah pembuatan *Poisson Mixture Model* yang nantinya akan digunakan untuk membuat suatu *model* untuk digunakan pada tahap *Online Recommendation*.

Tahapan *Online Recommendation* merupakan tahapan di mana suatu dokumen yang baru saja diinput dan bukan yang diinput pada tahap *Offline Computation* akan diberikan *tag-tag* sesuai dengan klaster yang nantinya dokumen baru ini akan tempati.

3.3 Tahapan Perancangan Automatic Tag

Setidaknya terdapat 10 tahapan yang perlu dijalankan agar *automatic tag* ini akan berjalan dengan baik yaitu

3.3.1 Penginputan

Pada tahap ini, akan ditentukan enam komponen atau inputan yang diperlukan adalah dokumen-dokumen (D), *word vocabulary* (S), *Tag vocabulary* (T), banyaknya klaster (K), banyaknya komponen-komponen (M), dan banyaknya klaster-klaster kata (L).

Untuk **dokumen-dokumen (**D**)** yang diinput, bisa berasal dari *PDF* dan dari *URL*. Untuk format *URL*, secara teknis prinsipnya mirip seperti artikel *online* yang di mana pada suatu *link* tersebut tentunya ada isi artikelnya.

Untuk penggunaan bahasa pada ***word vocabulary*** (S), sesuaikan dengan

penggunaan bahasa yang ada di dokumen-dokumen.

Pada ***tag vocabulary*** *(*T*)*, *tag-tag* yang di dapat berasal dari penanaman *tag* pada dokumen-dokumen yang diperoleh di suatu artikel. Beberapa dokumen biasanya diberikan suatu *tag* agar menjadi suatu penandan yang nantinya dokumen tersebut mudah untuk dicari jika lagi butuh topik yang relevan.

Pada **banyaknya klaster (**K**)**, digunakan untuk menentukan berapa klaster yang ingin dibuat saat menjalankan algoritma *SRE*. Banyaknya klaster ini ditentukan sesuai dengan keinginan sang pengguna.

Banyaknya **komponen-komponen (**M**)**, biasanya M ini akan digunakan pada algoritma *PMM* atau *Poisson Mixture Model*.

Terakhir adalah **banyaknya klaster kata** untuk melakukan pelabelan (L). Jumlahnya disesuaikan dengan banyaknya klaster. Nantinya, label ini seabgai penanda untuk suatu dokumen bahwa dokumen tersebut masuk kategori yang mana.

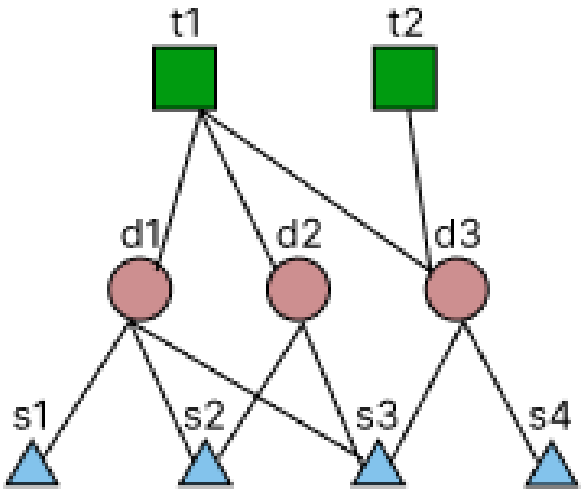
3.3.2 Menentukan Matriks W

Untuk matriks W akan terdiri dari beragam matriks yaitu matriks A, matriks A^T, matriks B, dan matriks B^T.

Untuk matriks A, terbuat dari relasi antara *tag-tag* dengan dokumen-dokumen, sedangkan untuk matriks B diperoleh dari relasi antara dokumen-dokumen dengan *word vocabulary*.

Untuk membuat matriks W akan menggunakan persamaan (2.9). Untuk contohnya, akan mengambil gambar berikut.

Relasi antara t (*tag*) dengan d (*document*) akan membentuk matriks A dan A^T sebagai berikut.



Gambar 3.3: Contoh simpel dua *bipartite graph*

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

$$A^T = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{3.2}$$

Selanjutnya, mencari relasi antara d (*document*) dengan s (*words*) agar membentuk **matriks** B sebagai berikut.

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \tag{3.3}$$

$$B^T = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

Selanjutnya, adanya penggabungan yang nantinya membentuk **matriks** W sebagai berikut

$$W = \begin{pmatrix} 0 & A & 0 \\ A^T & 0 & B \\ 0 & B^T & 0 \end{pmatrix}$$

Nantinya, matriks A dan matriks B ditempatkan ke dalam matriks W. Angka 0 di atas merupakan matriks 0 yang isinya menyesuaikan baris dan kolom matriks sekitarnya.

$$W = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.5}$$

3.3.3 Normalisasi W menggunakan *Normalized Laplacian*

Pada persamaan normalisasi Laplacian, diperlukan dua jenis matriks yaitu matriks W dan matriks D. Untuk mengisi matriks diagonal D diperlukan mengisi D_{ii} terlebih dahulu. Untuk mendapatkan D_{ii} memerlukan d_i yang berasal dari $d_i = \sum w_{ij}$. w_{ij} didapat dari matriks W.

$$D = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

Setelah mendapatkan D dan W, lalu masukkan ke persamaan normalisasi

Laplacian hingga membentuk $L(W)$. $L(W)$ yang dicontohkan di sini hasil dari pembulatan agar lebih mudah terbaca, tetapi dalam kasus nyatanya, tidak ada pembulatan.

$$L(W) = \begin{pmatrix} 0 & 0 & 0.29 & 0.33 & 0.29 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0.29 & 0 & 0 & 0 & 0 & 0.5 & 0.35 & 0.29 & 0 \\ 0.33 & 0 & 0 & 0 & 0 & 0 & 0.41 & 0.33 & 0 \\ 0.29 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0.29 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.35 & 0.41 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.29 & 0.33 & 0.29 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.7}$$

3.3.4 Menghitung *Low Rank Approximation Matrix* menggunakan algoritma Lanczos

Pada algoritma Lanczos, ada beberapa inputan agar prosesnya bisa berjalan yaitu β_0, q_0, b, dan q_1. Berikut adalah beberapa inputannya.

$$\beta_0 = 0$$

$$q_0 = 0$$

$$b = arbitrary$$

Karena b adalah *arbitrary* yang artinya adalah inputannya bebas. Alasannya $b = arbitrary$ adalah apapun bilangannya, hasilnya akan tetap sama saat perhitungan

q_1. Oleh karena itu, b dibuat menjadi

$$b = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Selanjutnya nilai q_1 adalah

$$q_1 = b/||b|| = \begin{pmatrix} 0.33 & 0.33 & 0.33 & 0.33 & 0.33 & 0.33 & 0.33 & 0.33 & 0.33 \end{pmatrix}$$

$$\hat{W} \simeq L(W) = Q_k T_k Q_k^T \tag{3.8}$$

Karena persamaan ini, hasil dari $L(W)$ juga bisa dicari dengan menggunakan $\hat{W}$. Selain itu, algoritma Lanczos dinilai lebih efisien untuk data yang banyak dibandingkan menggunakan normalisasi Laplacian. Untuk mencari Q_k dan T_k, perlu menggunakan iterasi Lanczos.

Untuk matriks T_k memiliki format sebagai berikut

$$T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{k-1} \\ & & & \beta_{k-1} & \alpha_k \end{bmatrix}.$$

Dalam matriks di atas, diperlukan pencarian nilai α dan β agar bisa melengkapi hasil T_k.

Untuk matriks Q_k didapat dari

$$Q_k = \begin{pmatrix} q_1 & | & q_2 & | & \dots & | & q_k \end{pmatrix}$$

q_1 adalah matriks kolom yang ke-1. q_2 adalah matriks kolom yang ke-2. q_k adalah matriks kolom yang ke-k. q_1 sampai q_k akan didapat melalui iterasi Lanczos sebagai berikut.

Pada algoritma di bawah, asumsikan bahwa $n = k$ dan $A = W$.

Algorithm 2 *Lanczos Iteration*

Require: $\beta_0 = 0, q_0 = 0, b = \textit{arbitrary}, q1 = b/||b||$
  while $n = 1, 2, 3, ..$ **do**
    $v = Aq_n$
    $\alpha = q_n^T v$
    $v = v - \beta_{n-1} q_{n-1} - \alpha_n q_n$
    $\beta_n = ||v||$
    $q_{n+1} = v/\beta_n$
  end while

Hasil yang di dapat sebagai berikut.

$$\alpha = \begin{pmatrix} 0 & 2.44 & -0.59 & -1.56 & -0.02 & -0.02 & -0.27 & 0.01 & 2.69 & -2.69 \end{pmatrix} \tag{3.9}$$

$$\beta = \begin{pmatrix} 0 & 1.17 & 1.34 & 1.07 & 0.99 & 0.89 & 0.72 & 0 & 1.07 & 0.09 \end{pmatrix} \tag{3.10}$$

Lalu, hasil T_k yang didapat adalah

$$T_k = \begin{pmatrix} 2.44 & 1.17 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.17 & -0.59 & 1.34 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.34 & -1.56 & 1.07 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.07 & -0.02 & 0.99 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.99 & -0.02 & 0.89 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.89 & -0.27 & 0.72 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.72 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.69 & 1.07 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.07 & -2.69 \end{pmatrix} \tag{3.11}$$

Kemudian, matriks Q_k sebagai berikut.

$$Q_k = \begin{pmatrix} 0.33 & 0.16 & 0.56 & -0.14 & -0.14 & -0.02 & 0.09 & 0.5 & -0.35 \\ 0.33 & -0.41 & -0.14 & -0.13 & -0.31 & 0.09 & -0.27 & 0.15 & -0.1 \\ 0.33 & 0.44 & -0.26 & 0.08 & 0.07 & 0.78 & 0.01 & 0.36 & 0.51 \\ 0.33 & 0.16 & -0.08 & 0.84 & -0.18 & -0.33 & -0.09 & 0.31 & 0.47 \\ 0.33 & 0.44 & -0.47 & -0.46 & -0.04 & -0.5 & 0.06 & 0.31 & 0.45 \\ 0.33 & -0.41 & -0.14 & 0.07 & 0.24 & 0.01 & 0.8 & 0.17 & -0.11 \\ 0.33 & -0.13 & 0.1 & 0 & 0.82 & -0.1 & -0.43 & 0.33 & -0.21 \\ 0.33 & 0.16 & 0.56 & -0.14 & -0.14 & -0.02 & 0.09 & 0.5 & -0.35 \\ 0.33 & -0.41 & -0.14 & -0.13 & -0.31 & 0.09 & -0.27 & 0.15 & -0.1 \end{pmatrix} \tag{3.12}$$

Lalu dengan menggunakan persamaan (3.8), $\hat{W}$ bisa didapat dengan hasil sebagai berikut.

$$\hat{W} = \begin{pmatrix} -0.02 & 0.01 & 2.1 & 1.98 & 1.95 & 0.01 & 0.01 & 0.02 & 0.01 \\ 0.01 & 0.01 & 0.33 & 0.3 & 1.29 & 0.01 & 0.01 & 0.01 & 0.01 \\ 2.1 & 0.33 & 0.02 & 0.01 & 0.02 & 1.37 & 1.71 & 2.1 & 0.33 \\ 1.98 & 0.3 & 0.01 & 0.03 & 0.01 & 0.33 & 1.64 & 1.98 & 0.3 \\ 1.95 & 1.29 & 0.02 & 0.01 & 0.01 & 0.32 & 0.62 & 1.95 & 1.29 \\ 0.01 & 0.01 & 1.37 & 0.33 & 0.32 & 0.01 & 0.02 & 0.01 & 0.01 \\ 0.01 & 0.01 & 1.71 & 1.64 & 0.62 & 0.02 & 0.02 & 0.01 & 0.01 \\ -0.02 & 0.01 & 2.1 & 1.98 & 1.95 & 0.01 & 0.01 & 0.02 & 0.01 \\ 0.01 & 0.01 & 0.33 & 0.3 & 1.29 & 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix} \quad (3.13)$$

3.3.5 Melakukan partisi $\hat{W}$ ke dalam klaster K menggunakan *SRE*

Untuk membuat algoritma SRE berdasarkan Zha et al. (2001), ada lima langkah yang diperlukan.

Pertama adalah menghitung D_X dan D_Y dan membentuk matriks $\hat{W} = D_X^{-1/2} W D_Y^{-1/2}$. Pada langkah ini, matriks $\hat{W}$ telah dibuat dengan menggunakan algoritma Lanczos atau *normalized Laplacian*. Alasannya adalah pola untuk mendapatkan $\hat{W}$ yang mirip.

Langkah kedua ialah mencari vektor terluas kedua singular kiri dan singular kanan. Untuk melakukan hal ini, perlu algoritma khusus yang bernama *Lanczos Bidiagonalization*.

Dalam algoritma tersebut terdapat, terdapat matriks penting seperti $U^T AV = B$ dengan matriks A bisa didapat dari $\hat{W}$ dan U^T serta V bisa didapat sebagai berikut.

$$U = [u_1, ..., u_m]$$

$$V = [v_1, ..., v_n]$$

$$U^T U = I_m$$

$$V^T V = I_n$$

B adalah matriks *bidiagonalization* sebagai berikut.

$$B = \begin{bmatrix} \alpha_1 & \beta_1 & & \dots & 0 \\ & \alpha_2 & \beta_2 & & \vdots \\ & & \alpha_3 & \ddots & \\ \vdots & & & \ddots & \beta_{k-1} \\ 0 & \dots & & & \alpha_k \end{bmatrix}.$$

Algorithm 3 *Lanczos Bidiagonlaization Algorithm*

$v_1 =$ given unit 2-norm n-vector
$p_0 = v_1$
$\beta_0 = 1$
$k = 0$
$u_0 = 0$
while $\beta \neq 0$ **do**
  $v_{k+1} = p_k / \beta_k$
  $k = k + 1$
  $r_k = A v_k - \beta_{k-1} u_{k-1}$
  $\alpha_k = \|r_k\|_2$
  $u_k = r_k / \alpha_k$
  $p_k = A^T u_k - \alpha_k v_k$
  $\beta_k = \|p_k\|_2$
end while

Dalam algoritma SRE, **langkah ketiga** adalah mencari titik potong yaitu c_x dan c_y untuk $x = D_X^{-1/2}\hat{x}$ dan $y = D_Y^{-1/2}\hat{y}$. Untuk mencari keduanya, strategi

tersimpelnya adalah dengan memasang $c_x = 0$ dan $c_y = 0$.

Langkah keempatnya membentuk suatu partisi dengan $A = \{i|x_i \geq c_x\}$ dan $A^C = \{i|x_i < c_x\}$ untuk *vertex* X dan $B = \{i|y_j \geq c_y\}$ dan $B^C = \{j|y_j < c_x\}$ untuk *vertex* Y.

Langkah kelima adalah mengulangi partisi pada *sub graph* $G(A, B)$ dan $G(A^c, B^c)$ jika diperlukan.

Berikut adalah algoritma dari SRE

Algorithm 4 *Spectral Recursive Embedding (SRE)*

Diberikan *weighted bipartite graph* $G = (X, Y, E)$ dengan bobot garis matriks W
1. Komputasi D_x dan D_y dan bentuk *scaled weight matrix* $\hat{(W)} = D_x^{-1/2} W D_y^{-1/2}$
2. Hitung singular vektor terluas kiri dan kanan kedua dari vektor $\hat{W}$, $\hat{x}$, dan $\hat{y}$
3. Temukan titik potong c_x dan c_y untuk $x = D_X^{-1/2}\hat{x}$ dan $y = D_Y^{-1/2}\hat{y}$, secara berulang.
4. Bentuk partisi $A = \{i|x_i \geq c_x\}$ dan $A^c = \{i|x_i < c_x\}$ untuk verteks set X, dan $B = \{j|y_j \geq c_y\}$ dan $B^c = \{j|y_j < c_y\}$ untuk verteks set Y.
5. Lakukan partisi secara rekursif untuk *sub-graphs* $G(A, B)$ dan $G(A^c, B^c)$

3.3.6 Melakukan pelabelan setiap dokumen

Selanjutnya adalah memberikan label kepada setiap dokumen yang ada sesuai pembagian klasternya. Misalnya, terdapat 4 dokumen yaitu D_1, D_2, D_3, dan D_4 dan ada 2 klaster sesuai jumlah K. Nantinya, dokumen tersebut dimasukkan ke dalam klaster sesuai dengan hasil partisi $\hat{W}$ sesuai dengan banyaknya klaster yang dibentuk.

3.3.7 Menghitung *Node Rank* Rank(T) untuk Setiap Tag

Cara menghitung Rank(T) untuk setiap tag T_i di dalam klaster k dengan menggunakan persamaan (2.6), persamaan (2.7), dan persamaan (2.8).

Untuk persamaan (2.6) adalah menghitung *N-Precision* np_i. Untuk mencari np_i, diharuskan menghitung total seluruh bobot *edges* dari *node* i di dalam klaster

yang sama, lalu hasilnya dibagi dengan total dari seluruh bobot pada *edges* di klaster tersebut. Semakin besar nilai np_i-nya, semakin penting keberadaan *tag* di dalam klaster tersebut.

Untuk persamaan (2.7) adalah menghitung *N-recall*. $|E_i|$ didapat dari banyaknya garis yang terhubung ke Tag i T_i baik itu di dalam klaster maupun di luar klaster. Lalu, hasilnya dibagi dengan banyaknya garis yang terhubung dengan T_i yang di luar klaster.

Setelah berhasil menghitung np_i dan nr_i, $Rank_i$ dapat dihitung dengan $exp(\frac{-1}{r(i)^2})$ untuk $r(i) = np_i * log(nr_i)$ dengan syarat $r(i)$ tidak boleh sama dengan 0. Jika $r(i) = 0$, $Rank_i$ hasilnya adalah 0 berdasarkan persamaan (2.8).

3.3.8 Membuat Two Way Poisson Mixture Model

Untuk membuat salah satu komponen dari *Two Way Poisson Mixture Model*, diperlukan persamaan (2.11). Salah satu komponen dari persaman tersebut adalah ϕ_m yang didapat dari persamaan (2.15).

Untuk $II(F(m) = k)$ adalah suatu *indicator function*. Jika komponen m milik klaster k, berarti bernilai 1. Dalam kondisi sebaliknya, bernilai 0.

Untuk mengisi t pada persamaan (2.14), inisialisasi $t = 0$ (Li & Zha (2004)). Kemudian, saat mencari persamaan (2.14), terdapat komponen $p_{i,m}$ yang perlu dicari terlebih dahulu dengan menggunakan persamaan (2.11).

Pada $p_{i,m}$ terdapat komponen $\phi_m^{(t)}$ yang bisa dianggap sebagai 1 karena seperti inisialisasi di awal bahwa $t = 0$. Kemudian, $d(i, j)$ didapat dari berapa banyaknya kata j di dalam dokumen i. Hal ini ada hubungannya dengan matriks B dari persamaan (2.9). Mencari θ dapat dicari dengan cara $\theta(d_j|\lambda_{j,m}) = e^{-\lambda_{j,m}}\lambda_{j,m}^{d_j}/d_j!$. Untuk p, ia adalah *dimension*. Selain itu, $IIC(i)$ untuk mengecek apakah kata i ada di klaster tersebut.

Selain itu, di persamaan (2.11) terdapat komponen $\tilde{\lambda}_{m,l}$ yang bisa ditemukan di persamaan (2.15). Pada variabel $|d(i,j)|$ dapat dicari dengan menentukan apakah j ada di label l.

3.3.9 Rekomendasi Tag Untuk Dokumen Baru

Lakukan melalui persamaan (2.16) lalu untuk mencari $\frac{P(D=d_t|C=k)P(C=k)}{P(D=d_t)}$ bisa ditemukan dengan persamaan (2.11). Lalu, masukan dokumen yang baru ke dalam suatu klaster yang memiliki probabilitas terbesar.

3.3.10 Rekomendasi Tag Berdasarkan Ranks Tag

Setelah melakukan perhitungan $P(C = k|D = Y)$, langkah berikutnya adalah merekomendasikan *tag-tag* berdasarkan klaster dari dokumen tersebut. Cara melakukannya adalah dengan menggunakan persamaan (2.17) pada setiap *tag* yang ada di klaster tersebut. Kemudian, lakukan pengurutan $R(T_i, d_t)$ dari yang terbesar ke yang terkecil.

3.4 Algoritma *Automatic Tag*

Algorithm 5 Online Tag Recommendation

1: **Input** $(\mathcal{D}, S, T), K, M, L$
Kumpulan dokumen: $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$
Word vocabulary: $S = \{S_1, \ldots, S_k\}$
Tag vocabulary: $T = \{T_1, \ldots, T_n\}$
banyaknya klaster: $K \in R$
banyaknya komponen-komponen: $M \in R$
banyaknya klaster-klaster kata: $L \in R$

Offline Computation

2: Menunjukkan bobot terdekat matriks W seperti persamaan (2.9)
3: Normalisasi W menggunakan *Normalized Laplacian*
$L(W) = D^{(-1/2)} W D^{(-1/2)}$ persamaan (2.1)
4: Komputasi *low rank approximation matrix* menggunakan Lanczos:
$\tilde{W} \simeq L(W) = Q_k T_k Q_k^T$
5: Partisi $\tilde{W}$ ke dalam klaster K menggunakan SRE,
$\tilde{W} = \left\{\tilde{W}_1, \ldots, \tilde{W}_K\right\}$
6: Tandai label ke dalam setiap dokumen $\mathcal{D}_j, j \in \{1, \ldots m\}$
$C\left(\mathcal{D}_j\right) \in \{1, \ldots, K\}$
7: Hitung *node rank* $Rank(T)$ untuk setiap tag $T_{i,k}$ in cluster
$k, i \in \{1, \ldots, n\}, k\{1, \ldots, K\}$ persamaan (2.8)
8: Buat *Poisson Mixture Model* untuk $(\tilde{B}, C(\mathcal{D}))$ dengan M komponen-komponen dan L klaster kata-kata, di mana $\tilde{B}$ denotasi matriks inter-relationship pada suatu dokumen-dokumen dan kata-kata di dalam $\tilde{W}$ persamaan (2.9)

Online Recommendation

9: Untuk setiap dokumen tes Y, kalkulasikan posterior probabilitas
$P(C = k \mid D = Y)$ di dalam setiap klaster k, dan denotasi membership pada Y sebagai $C(Y) = \{c(Y, 1), \ldots, c(Y, K)\}$ persamaan (2.16)
10: Tag rekomendasi berdasarkan perangkingan pada tag, yaitu *joint probability* pada tag-tag T dan dokumen Y, $R(T, Y)$ persamaan (2.17)

DAFTAR PUSTAKA

Brin, S. & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

Christ, A. (2022). Top 10 search engines in the world (2022 update).

Li, J. & Zha, H. (2004). Two-way poisson mixture models for simultaneous document classification and word clustering.

Parra, E., Escobar-Avila, J., & Haiduc, S. (2018). Automatic tag recommendation for software development video tutorials.

Pratama, Z. (2022). Perancangan modul pengindeks pada search engine berupa induced generalized suffix tree untuk keperluan perangkingan dokumen.

Seymour, T., Frantsvog, D., & Kumar, S. (2011). History of search engines. *International Journal of Management & Information Systems (IJMIS)*, 15(4):47–58.

Shi, M., Liu, J., Tang, M., Xie, F., & Zhang, T. (2016). A probabilistic topic model for mashup tag recommendation.

Song, Y., Zhuang, L., & Giles, L. (2011). Real-time automatic tag recommendation.

Song, Y., Zhuang, Z., Li, H., Zhao, Q., Li, J., Lee, W.-C., & Giles, L. (2008). Real-time automatic tag recommendation.

Sood, S. (2007). Tagassist: Automatic tag suggestion for blog posts.

Won, M., Ferraro, A., Bogdanov, D., & Serra, X. (2020). Evaluation of cnn-based automatic music tagging models.

Zha, H., He, X., Ding, C., Simon, H., & Gu, M. (2001). Bipartite graph partitioning and data clustering.