PERBAIKAN METODE REKOMENDASI DISKUSI PEMROGRAMAN DENGAN NORMALISASI *IDENTIFIER* MENGGUNAKAN LINGUA::IDSPLITTER

Nanang Fakhrur Rozi[1], Daniel Oranova Siahaan[2], Fajar Baskoro[2]

[1]Teknik Informatika ITATS, Jl. Arief Rachman Hakim 100, Surabaya
[2]Teknik Informatika ITS, Jl. Teknik Kimia, Kampus ITS Sukolilo, Surabaya
[1]E-mail: nanang@itats.ac.id

ABSTRACT

Stack Overflow as a question and answer (Q&A) site has often been used as reference by programmers. Information or solutions in the software development process can be searched with the help of search engine on the site. However, differences in writing style, especially on the program identifier writing, often causing the recommendation (search) become incompatible with the programmers need. Some programmers write identifier in abbreviated form while another are not so that it lowers the recommendation performance. This research adopted the Lingua::IdSplitter to normalize the identifier on Stack Overflow discussion. The normalization process is done by separating the identifier comprising the composition of the term as well as expanding the existing abbreviation on the identifier to the full form. The results showed that the identifier normalization using Lingua::IdSplitter able to improve the median value of recall up to 13%.

Keywords*: recommender system, identifier, normalization,* Stack Overflow

ABSTRAK

Situs tanya-jawab Stack Overflow telah sering digunakan sebagai acuan oleh *programmer*. Informasi atau solusi dalam proses pengembangan perangkat lunak dapat dicari dengan bantuan mesin pencari pada situs. Namun, perbedaan dalam gaya penulisan, terutama pada penulisan *identifier* program, sering menyebabkan rekomendasi (pencarian) menjadi tidak sesuai dengan kebutuhan *programmer*. Beberapa *programmer* menulis *identifier* dalam bentuk singkatan sementara yang lain tidak sehingga menurunkan kinerja rekomendasi. Penelitian ini mengadopsi Lingua::IdSplitter untuk menormalkan *identifier* pada data diskusi Stack Overflow. Proses normalisasi dilakukan dengan memisahkan *identifier* yang terdiri atas komposisi term serta memperluas singkatan yang ada pada *identifier* ke bentuk penuh. Hasil penelitian menunjukkan bahwa normalisasi *identifier* menggunakan Lingua::IdSplitter secara umum mampu meningkatkan nilai median *recall* hasil rekomendasi hingga 13%.

Kata kunci: sistem rekomendasi, *identifier*, normalisasi, Stack Overflow

PENDAHULUAN

Perusahaan perangkat lunak yang baik seharusnya dapat menyediakan *deliverable* dengan waktu yang singkat dan dengan kualitas yang baik pula. Untuk mencapai hal tersebut, dibutuhkan manajemen serta kerja sama yang baik dari setiap pemangku kepentingan terutama *programmer* yang menjadi tulang punggung terciptanya perangkat lunak. Dengan demikian, performa *programmer* yang baik merupakan keniscayaan dalam menyukseskan proyek perangkat lunak.

Dalam [1] disebutkan bahwa performa *programmer* dapat diukur berdasarkan kepribadian, kemampuan kognitif, serta tingkat kepercayaan terhadap nilai teoretis *(theoretical value belief)*. *Programmer* yang selalu mencari pembuktian kebenaran terhadap hasil kerjanya, tidak asal-asalan dalam menyediakan solusi dalam bentuk kode program, cenderung memiliki performa yang baik dalam jangka panjang. Hal tersebut memaksa *programmer*, terutama *programmer* pemula, untuk menginvestasikan waktunya dalam meningkatkan kemampuan terkait pemahaman algoritma, cara memprogram, informasi-informasi pendukung, atau hal-hal lain yang berkaitan dengan perangkat lunak yang akan dibangun. Mendapatkan informasi dari anggota tim atau teman kerja menjadi

salah satu carankya [2]. Namun, hal itu akan membebani anggota lain untuk juga meluangkan waktu yang pada akhirnya akan berimbas pada perlambatan waktu *delivery*. Guna menghindari hal tersebut, cara lain pun ditempuh. Seiring perkembangan web 2.0, informasi-informasi berguna terkait proses pengembangan perangkat lunak yang ditenagai oleh komunitas tersedia di internet, antara lain yang berbentuk situs tanya-jawab [3]. Media tersebut pada akhirnya banyak digunakan oleh pengembang perangkat lunak untuk mendapatkan solusi atas masalah-masalah yang dihadapi. Media tersebut dapat berupa sebagian dari platform media sosial seperti grup di Facebook atau situs khusus seperti Stack Overflow.

Stack Overflow merupakan situs tanya-jawab populer yang berfokus pada bidang pemrograman. Diskusi yang terjadi pada Stack Overflow adalah diskusi yang sangat aktif dan terpercaya. Dalam [3] disebutkan bahwa lebih dari 90% pertanyaan yang dilontarkan oleh penanya dijawab tidak lebih dari 12 menit oleh pengguna lainnya. Penanya relatif mendapatkan kepuasan atas jawaban tersebut serta kualitas jawaban yang juga selalu mendapat perhatian dari membernya. *Programmer* dapat mengakses informasi tersebut melalui kolom pencarian yang tersedia pada situs. Mengingat mesin pencari pada Stack Overflow juga diadopsi dari mesin pencarian teks pada umumnya, maka berbagai penyesuaian pun perlu dilakukan, sehingga hasil pencarian yang diberikan juga belum tentu sesuai dengan kebutuhan pengguna. Hal tersebut umumnya diakibatkan oleh sulitnya merumuskan *query* yang tepat sebagai masukan sistem pencarian, inkonsistensi penggunaan terminologi, serta berbagai kesulitan lainnya [4]. Oleh karena itu, berbagai penelitian dilakukan untuk menyempurnakan hasil pencarian tersebut.

Penelitian tentang sistem rekomendasi berbasis konteks [5] telah dilakukan. Penelitian tersebut dilatarbelakangi oleh ketidakmampuan mesin pencari dalam memberikan rekomendasi diskusi Stack Overflow yang baik ketika *query* yang diberikan berupa informasi *stack trace*. Informasi tersebut biasa muncul ketika terjadi eksepsi saat menjalankan program Java. Informasi tersebut diasumsikan oleh peneliti sebagai konteks permasalahan *programmer*. Dalam memberikan rekomendasi, sistem akan mengekstrak *stack trace* yang muncul untuk kemudian dicari strukturnya berdasarkan nama eksepsi, nama *method*, beserta referensinya. Struktur tersebut akan dijadikan sebagai acuan *query* yang akan dibangkitkan. Hasilnya, model rekomendasi yang dibangun memiliki performa yang lebih baik dibandingkan dengan model rekomendasi dengan *query* yang berbasis kata kunci maupun rekomendasi dari mesin pencari pada situs Stack Overflow sendiri. Namun, penelitian ini hanya akan memiliki hasil yang maksimal apabila data diskusi mengandung informasi *stack trace* pula.

Penelitian lain yang telah dilakukan adalah Seahawk [6]. Seahawk merupakan sistem rekomendasi data diskusi Stack Overflow yang mampu memberikan hasil rekomendasi dengan *query* yang dibangkitkan secara otomatis berdasarkan kode program yang sedang dibuka pada editor kode Eclipse (IDE). Kode program tersebut diasumsikan sebagai teks bahasa alami sehingga *query* yang dibangkitkan akan menghasilkan kumpulan kata kunci yang diperoleh dari proses tokenisasi dan beberapa praproses lain terhadap kode program. Pemeringkatan data diskusi dilakukan dengan memanfaatkan Apache Solr dengan model pembobotan TF-IDF. Pengujian dilakukan menggunakan kode program Java yang telah disiapkan sebelumnya. Hasilnya, Seahawk secara umum dapat merekomendasikan diskusi yang berguna bagi programmer. Namun, penelitian ini masih memiliki kekurangan. Hasil rekomendasi menjadi buruk ketika kata kunci bangkitan mengandung unsur singkatan seperti `strCmp`, `usrId`, atau lainnya. Padahal, pembentukan term yang baik menjadi kunci sukses perolehan informasi yang tepat dari literatur yang tersimpan [7].

Penelitian lain terkait sistem rekomendasi yaitu penelitian [8] yang menghasilkan JECO (Java Example Code). Penelitian ini mengusulkan sistem rekomendasi kode program pada data diskusi Stack Overflow dengan pemodelan topik LDA (Latent Dirichlet Allocation). *Query* yang digunakan berupa kata kunci yang dimasukkan secara manual oleh pengguna. Data diskusi dan *query* kemudian dicari topik beserta proporsinya menggunakan LDA. Berdasarkan topik tersebut, *query* dan data diskusi dihitung kesamaannya sehingga didapatkan hasil rekomendasi dengan memanfaatkan Cosine *similarity*. Hasilnya, JECO mampu merekomendasikan kode program Java ke *programmer* dengan rata-rata *precision* dan *recall* masing-masing sebesar 48% dan 58%. Namun, term singkatan juga tidak diperhatikan pada penelitian ini, baik pada *query* maupun data diskusi. Hal tersebut tentu akan mempengaruhi *recall* pada proses rekomendasi.

Unsur singkatan biasa muncul pada *identifier* kode program. Seorang programmer terkadang lebih suka menuliskan `string` dengan `str` dalam *identifier*. Sementara *programmer* lain lebih suka menulis *identifier* dalam bentuk lengkap dibandingkan dengan bentuk singkatan. Mengatasi hal tersebut, [9] mengembangkan Lingua::IdSplitter. Algoritma tersebut memiliki kemampuan untuk memisah *identifier* yang umumnya terdiri atas komposisi term serta mengekspansi term singkatan pada *identifier* menjadi term lengkap, baik yang menggunakan model penulisan *all lowercase* maupun *camel case*. Lingua::IdSplitter dikembangkan mengingat *identifier* merupakan salah satu sumber informasi yang cukup relevan untuk memahami sebuah program. Selain itu, *identifier* yang lengkap cenderung lebih memudahkan program untuk dipahami [10]. Pemisahan *identifier* oleh Lingua::IdSplitter dilakukan dalam dua tahap, yakni *hard split* dan *soft split*. *Hard split* dilakukan untuk memisahkan *identifier* berdasarkan karakter tertentu seperti tanda garis bawah *(underscore)* atau berdasarkan model penulisan *camel case*. Sedangkan *soft split* akan memisah *identifier* yang tidak terlalu terlihat tanda pemisahnya. Hal ini dilakukan dengan bantuan kamus kata untuk mendapatkan himpunan kandidat katanya. Kandidat kata tersebut selanjutnya akan diperingkat berdasarkan otomata yang dibangun berdasarkan himpunan kandidat kata. Himpunan kata dengan skor tertinggi akan menjadi keluaran Lingua::IdSplitter. Penelitian ini membandingkan algoritma yang diusulkan dengan algoritma-algoritma lain yang sejenis. Hasilnya, algoritma ini mampu menghasilkan term yang telah dipisah dan diekspansi dengan tingkat *f-measure* sebesar 90%, tidak jauh berbeda dengan hasil algoritma pembanding lainnya.

Penelitian ini mengadopsi Lingua::IdSplitter sebagai algoritma untuk menormalisasi *identifier* guna meningkatkan performa sistem rekomendasi data diskusi dari Stack Overflow yang pada penelitian-penelitian sebelumnya belum dilakukan. Dengan term yang baik, maka hasil rekomendasi akan menjadi lebih baik pula. Model data yang digunakan adalah model data Bag-of-Words (BOW) yang telah diboboti menggunakan TF-IDF. Model tersebut dipilih mengingat pemodelan topik LDA masih memiliki kekurangan karena tidak adanya proses seleksi atau pemilihan term penting [11] sehingga akan mengakibatkan ketidaksesuaian proporsi topik terhadap dokumen.

TINJAUAN PUSTAKA

Sistem Rekomendasi

Sistem rekomendasi merupakan metode yang mampu memberikan rekomendasi dengan memprediksi nilai sebuah item bagi pengguna untuk kemudian mempresentasikan item dengan nilai prediksi tertinggi sebagai hasil rekomendasi. Sistem ini awalnya merupakan sebuah mekanisme *information filtering*, yakni bertujuan menyaring informasi sebagai akibat membludaknya informasi di internet. Terdapat tiga pendekatan yang bisa digunakan dalam membangun sistem rekomendasi, yaitu rekomendasi berbasis konten, rekomendasi berbasis metode kolaboratif, dan rekomendasi berbasis metode hibrida [12].

Sistem rekomendasi dalam rekayasa perangkat lunak atau sering disebut *Recommendation Systems for Software Engineering* (RSSE) difungsikan untuk membantu pengembang dalam berbagai aktivitas, mulai dari sugesti informasi, penggunaan ulang kode program, hingga penulisan laporan bug yang efektif [13]. Dengan semakin kompleksnya proses pengembangan perangkat lunak, maka sistem semacam ini sangat dibutuhkan oleh para pengembang, terlebih untuk pengembang dengan tingkat pengalaman yang masih rendah.

Identifier pada Kode Program

Menurut [14], *identifier* dalam ranah pemrograman Java adalah rangkaian karakter Java dan digit Java dengan panjang tak hingga. Hal tersebut mengingat *identifier* umumnya terdiri atas gabungan lebih dari satu kata.Oleh karena itu, *programmer* sering menggunakan unsur singkatan dalam pembentukannya. Singkatan tersebut biasanya singkatan yang umum digunakan seperti HTML, URL, atau lainnya.

Adapun syarat pembentukan *identifier* yakni karakter pertama harus berupa karakter Java. Karakter Java sendiri merupakan karakter alfabet (non-simbol), yakni karakter-karakter yang biasa

digunakan untuk merangkai kata, baik yang terdapat dalam kumpulan karakter latin maupun non-latin. Sementara itu, digit Java merupakan angka 0–9, baik dalam bentuk karakter latin maupun non-latin. Sebagai tambahan, karakter pada Java didasarkan pada standar pengodean Unicode [14].

Identifier pada Java biasa digunakan untuk menamai elemen-elemen di dalam kode program, seperti *class*, variabel, atau *method* [15]. Secara umum, penulisannya mengacu pada konvensi yang dibuat oleh [16], yakni menggunakan aturan *camel-case* atau *underscore* seperti pada penulisan variabel `myWidth` dan `MAX_WIDTH`. Selain itu, penulisan *identifier* bersifat *case-sensitive*, membedakan huruf besar dan kecil, misalnya membedakan antara variabel `var2a` dan `var2A`. Namun, dalam penelitian ini, konsep sensitivitas karakter tersebut diabaikan. Langkah tersebut diambil mengingat perbedaan *case* tidak akan membedakan makna atau konsep *identifier*.

Praproses pada Sistem Rekomendasi

Praproses merupakan salah satu hal yang lumrah dilakukan sebelum sebuah data diproses lebih lanjut. Praproses ini digunakan untuk menghapus informasi-informasi yang tidak berguna ataupun menormalisasi data agar data menjadi bersih atau sesuai format serta siap untuk diproses. Mengingat data yang akan dibahas mengandung kode program, maka praproses yang dilakukan salah satunya harus mempertimbangkan pola penulisan kode program [17].

Tokenisasi, Filter Simbol, dan Case Folding

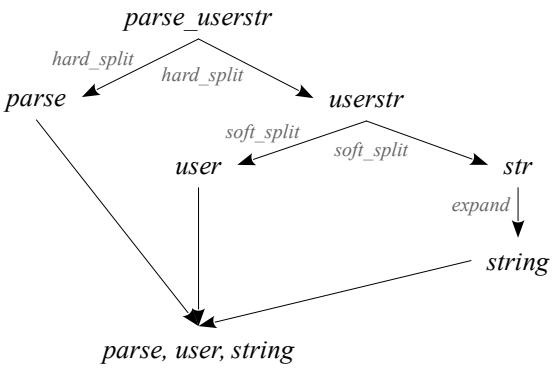
Jika diberikan kumpulan karakter dan pendefinisian unit dokumen, tokenisasi adalah proses pemotongan kumpulan karakter tersebut menjadi potongan-potongan yang disebut token. Gambar 1 merupakan contoh tokenisasi dengan pemisah berupa karakter spasi *(white space)*. Lebih dari itu, duplikasi token yang telah ditiadakan (token unik) sering disebut sebagai term pada bidang ilmu temu kembali informasi [4]. Mengiringi tokenisasi, penghapusan simbol atau data non-alfabet juga dilakukan. Pada kode program, simbol ini dapat berupa tanda plus (+), minus (-), modulus (%), atau simbol lain seperti @ (anotasi) [17]. Sebagai tambahan, pengubahan huruf besar ke huruf kecil *(case folding)* juga biasa dilakukan.

IN	even though backquote (') is not a standard quote character in	OUT (csv)	even, though, backquote, is, not, a, standard, quote, character, in

Gambar 1. Hasil tokenisasi, filter simbol, dan *case-folding* pada data teks

Normalisasi dengan Lingua::IdSplitter

Cara pemisahan *identifier* pada Lingua::IdSplitter dilakukan melalui dua tahap, yakni *hard split* dan *soft split*. *Hard split* dilakukan untuk memisahkan *identifier* berdasarkan karakter tertentu seperti tanda garis bawah *(underscore)* atau berdasarkan aturan penulisan *camel case*. Sementara itu, *soft split* akan memisahkan *identifier* yang tidak terlalu terlihat (tanda) pemisahnya [9]. Ilustrasi proses pemisahan dan ekspansi ini dapat dilihat pada Gambar 2.

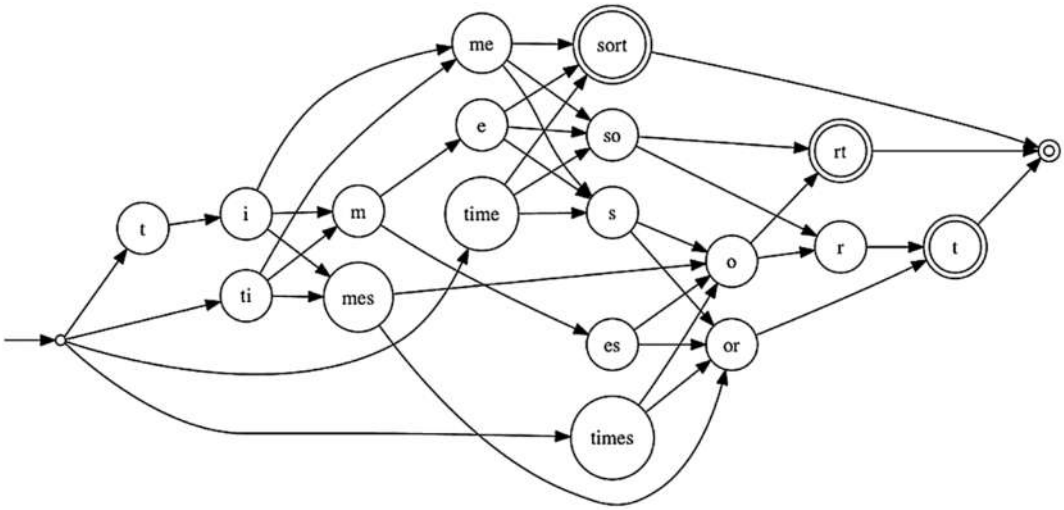


Gambar 2. Mekanisme pemisahan dan ekspansi *identifier* [9]

Soft split dilakukan salah satunya dengan bantuan kamus kata (akronim dan singkatan) untuk mendapatkan himpunan kandidat katanya. Apabila kandidat kata tersebut tidak ditemukan dalam kamus, maka Lingua::IdSplitter akan memotong *identifier* dengan jumlah karakter tertentu untuk kemudian dicek validitas katanya. Kandidat kata tersebut selanjutnya akan diperingkat berdasarkan otomata yang dibangun berdasarkan himpunan kandidat kata. Himpunan kata dengan skor tertinggi akan menjadi keluaran Lingua::IdSplitter. Ilustrasi hasil penskoran himpunan kandidat kata dapat dilihat pada Tabel 1. Sedangkan otomata yang dibangun dapat dilihat pada Gambar 3.

Tabel 1. Himpunan kandidat hasil *split* berdasarkan skor tertinggi [9]

Kandidat *Split*	Skor
`{time, sort}`	1,4400
`{ti, me, sort}`	0,1920
`{time, so, rt}`	0,1920
`{times, o, rt}`	0,1500
`{…}`	…



Gambar 3. Otomata untuk *identifier* `timesort` [9]

Otomata pada Gambar 3 digunakan untuk menghitung skor kandidat hasil pemisahan. Dengan melihat skor kandidat kata pada Tabel 1, didapatkan bahwa kandidat term `time` dan `sort` adalah term yang paling baik untuk menjadi hasil pemisahan atas *identifier* `timesort`.

Penghapusan Stop Word

Penghapusan *stop word* adalah proses penghapusan term yang tidak memiliki arti, tidak relevan, atau term yang sangat umum. Term yang diperoleh dari tahap tokenisasi dicek dalam suatu *stop list*, apabila sebuah kata masuk di dalam *stop list* maka kata tersebut tidak akan diproses lebih lanjut. *Stop list* tersimpan dalam suatu dokumen yang dimuat saat pemrosesan term. *Stop list* yang digunakan pada penelitian ini sebanyak 571 kata bahasa Inggris[1].

Stemming

Stemming dilakukan atas asumsi bahwa kata-kata yang memiliki akar kata yang sama memiliki makna yang serupa sehingga pengguna tidak keberatan untuk memperoleh dokumen-dokumen yang di dalamnya mengandung kata-kata dengan akar kata yang sama dengan *query* yang dimasukkan. Teknik *stemming* dapat dikategorikan menjadi tiga, berdasarkan aturan bahasa tertentu, berdasarkan kamus, atau berdasarkan teknik kemunculan bersama.

[1] Dikompilasi oleh Gerard Salton dan Chris Buckley untuk uji coba sistem SMART IR di Cornell University

Adapun *stemmer* yang paling umum digunakan terhadap dokumen berbahasa Inggris adalah Porter Stemmer yang berbasis aturan bahasa Inggris [4]. Meski masih memiliki kekurangan, metode *stemming* ini adalah metode yang cukup ringan dengan hasil yang cenderung lebih baik dibanding metode sejenis lainnya. Berkebalikan dengan hal tersebut, *stemming* berdasarkan kemunculan bersama akan sangat dipengaruhi oleh koleksi kata yang didapatkan dari dokumen. Oleh karena itu, kata yang tidak pernah muncul pada dokumen tidak akan bisa diperoleh akar katanya [18]. *Stemming* berdasarkan kamus tidak digunakan mengingat hasil berupa stem sudah mencukupi pada studi kasus yang diambil tanpa mengubah stem ke bentuk lema.

Perekomendasian Berdasarkan Similaritas Data

Perekomendasian baru bisa dilakukan setelah praproses dilakukan. Dengan masukan berupa daftar term, kemudian term tersebut diboboti dengan metode pembobotan umum TF-IDF. Algoritma ini menggabungkan dua konsep penghitungan bobot, yaitu frekuensi kemunculan term *t* di dalam dokumen atau biasa disebut TF *(term frequency)* dan invers frekuensi dokumen, yaitu frekuensi dokumen yang mengandung term *t* atau biasa disebut IDF *(inverse document frequency)*. Bobot TF dan IDF kemudian digabungkan dengan cara dikalikan menggunakan persamaan (1) untuk mendapatkan bobot komposit berdasarkan pertimbangan kedua kriteria tersebut [4].

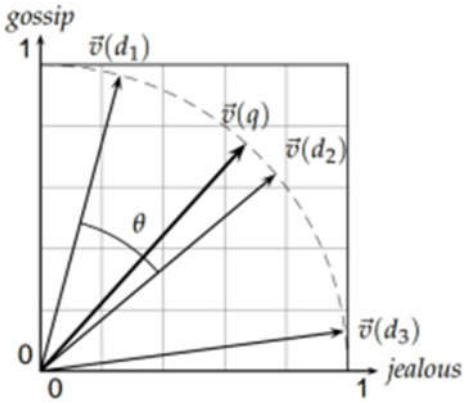
$$TF\text{-}IDF_{t,d} = TF_{t,d} \times IDF_t \quad (1)$$

dengan $TF_{t,d}$ adalah frekuensi kemunculan term *t* pada dokumen *d* dan IDF_t adalah invers frekuensi dokumen untuk term *t* yang dihitung menggunakan persamaan (2):

$$IDF_t = 1 + \log \frac{N}{DF_t + 1} \quad (2)$$

dengan N adalah total dokumen pada koleksi/korpus dan DF_t adalah jumlah dokumen yang mengandung term *t*.

Pembobotan setiap term yang telah dilakukan akan direpresentasikan dalam ruang vektor *(vector space model)* dalam bentuk matriks term-dokumen sehingga setiap dokumen akan berbentuk sebagai sebuah (fitur) vektor. Vektor-vektor tersebut yang kemudian dihitung kemiripannya dengan pendekatan Cosine *similarity*. Ilustrasi Cosine *similarity* dapat dilihat pada Gambar 4.



Gambar 4. Penentuan kesamaan dokumen (vektor) berdasarkan sudut yang terbentuk [4]

Melihat Gambar 4, semakin kecil sudut yang terbentuk antara dua vektor, maka semakin mirip kedua dokumen tersebut. Adapun persamaan yang digunakan adalah:

$$\text{sim}(d_1, d_2) = \frac{\overrightarrow{d_1} \cdot \overrightarrow{d_2}}{|\overrightarrow{d_1}||\overrightarrow{d_2}|} \quad (3)$$

dengan d_1 adalah vektor fitur dokumen ke-1 dan d_2 adalah vektor fitur dokumen ke-2.

Stack Overflow Sebagai Sumber Data Rekomendasi

Diskusi pada Stack Overflow umumnya berupa data teks. Sehingga, sangat memungkinkan untuk digunakan dalam membangun sistem rekomendasi yang berbasis teks. Pada Stack Overflow, pengguna umumnya mencantumkan potongan kode program pada pertanyaan ataupun jawaban. Ciri lainnya, pengguna juga terkadang mencantumkan gambar yang mampu mendukung pemahaman terhadap pertanyaan ataupun jawaban pengguna. Selain bertanya dan menjawab, pengguna Stack Overflow diberi fasilitas untuk melakukan *vote* terhadap pertanyaan maupun jawaban. Pertanyaan yang mendapat banyak *vote* bisa dikategorikan sebagai pertanyaan yang penting dan merupakan permasalahan yang sering terjadi. Sedangkan jawaban dengan status *accepted* menandakan jawaban tersebut merupakan jawaban yang sangat bisa dipertimbangkan sebagai solusi yang paling tepat. *Vote* dirupakan dengan ikon segitiga menghadap ke atas *(vote up)* dan ke bawah *(vote down)*.

Evaluasi Sistem Rekomendasi

Karena model sistem rekomendasi yang diteliti adalah rekomendasi berbasis konten, maka sistem dapat dievaluasi dengan menggunakan perhitungan *precision* dan *recall* [4] berdasarkan *query* yang dimasukkan. *Precision* merupakan rasio jumlah dokumen relevan yang dikembalikan terhadap jumlah seluruh hasil yang dikembalikan. Persamaannya adalah:

$$\text{Precision} = \frac{\#(\text{dokumen relevan yang dikembalikan})}{\#(\text{dokumen yang dikembalikan})} \quad (4)$$

Sedangkan *Recall* merupakan rasio jumlah dokumen relevan yang dikembalikan terhadap jumlah seluruh dokumen yang relevan. Persamaannya adalah:

$$\text{Recall} = \frac{\#(\text{dokumen relevan yang dikembalikan})}{\#(\text{dokumen yang relevan})} \quad (5)$$

Kedua pengukuran tersebut akan memiliki jangkauan nilai mulai 0 hingga 1 *(floating point)* atau sering pula dirupakan dalam bentuk persentase dengan jangkauan nilai mulai 0 hingga 100.

METODE

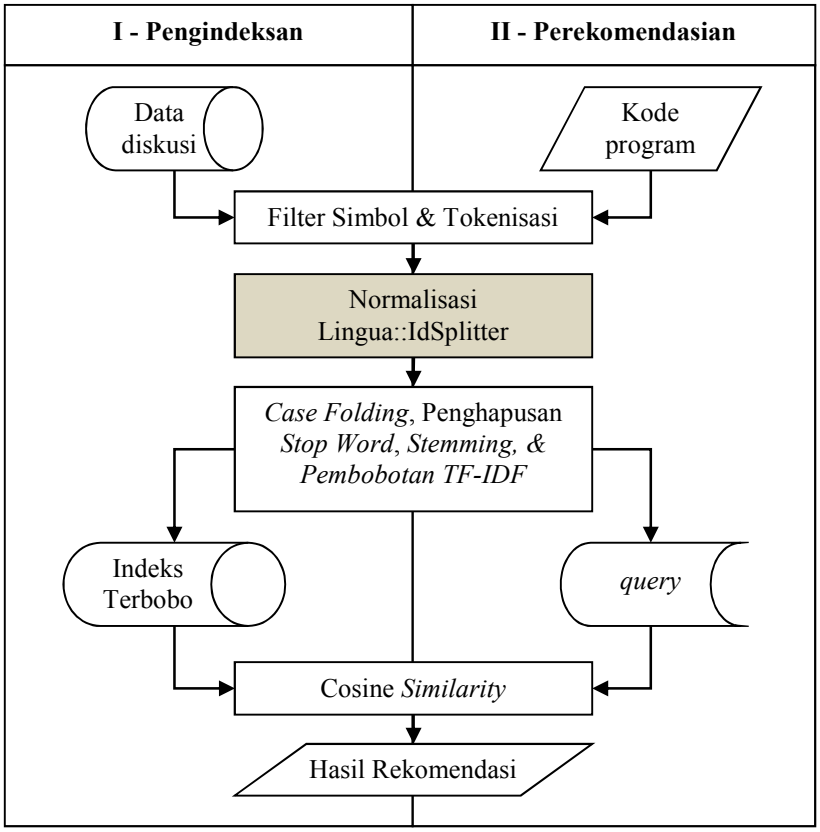
Penelitian ini menggunakan kerangka proses sebagaimana yang ditunjukkan pada Gambar 5. Rekomendasi didapatkan dari dua tahap yakni tahap pengindeksan yang hanya dijalankan sebanyak satu kali dan tahap perekomendasian data diskusi. Tahap pengindeksan harus dilakukan terlebih dahulu agar sistem rekomendasi dapat dijalankan.

Data Masukan

Masukan pada penelitian ini terbagi menjadi dua jenis, yakni masukan untuk tahap pengindeksan dan masukan untuk tahap perekomendasian. Masukan untuk tahap pengindeksan berupa teks diskusi (pertanyaan dan jawaban) yang diambil dari Stack Overflow. Data diambil dengan bantuan Stack Exchange Data Explorer[2]. Data yang diambil adalah data dengan label *(tag)* Java yang berjumlah 153 data. Data tersebut terdiri atas 34 data tentang topik *sorting*, 34 data tentang topik *database*, 32 data tentang topik *text file*, 34 data tentang topik *graphics*, dan 19 data tentang topik *thread*. Spesifikasi data ini adalah spesifikasi data yang digunakan oleh [8].

Adapun masukan untuk tahap perekomendasian merupakan potongan kode program `(*.java)` yang dimasukkan oleh pengguna. Kode program yang diambil sebanyak 32 buah dan didapatkan secara acak dari GitHub[3] dengan spesifikasi: 8 program tentang *thread synchronization*, 8 program tentang *accessing* MySQL, 8 program tentang *bubble sort*, serta 8 program tentang *draw rectangle*.

[2] *Tools* tersedia di: http://data.stackexchange.com/stackoverflow/query/new
[3] Lihat http://github.com



Gambar 5. Desain sistem rekomendasi dengan normalisasi Lingua::IdSplitter

Filter Simbol

Data masukan yang mengandung simbol non-alfabet seperti titik koma (;), tanda tanya (?), dan lain-lain akan dihapus sehingga yang tersisa hanya karakter alfabet. Sedangkan tanda titik (.) akan diubah menjadi spasi karena tanda titik digunakan oleh Java untuk memisahkan objek atau *class* dengan membernya. Selain itu, karena data diskusi berupa data HTML, maka *tag-tag* HTML pada diskusi juga akan dihilangkan. Selain mengandung *tag* HTML, dokumen web terkadang juga mengandung simbol yang dirupakan dalam bentuk entitas HTML seperti `<` yang ketika ditampilkan di peramban web akan muncul sebagai simbol *kurang dari* (<). Entitas HTML tersebut juga akan dihapus sebelum data diproses lebih lanjut. Gambar yang disertakan pada diskusi juga akan dihapus mengingat tag `<img>` pada data hanya memuat lokasi gambar saja pada server. Dengan kata lain, dokumen HTML akan diubah bentuknya menjadi teks *plain*.

Tokenisasi

Data yang telah difilter kemudian ditokenisasi sehingga akan didapatkan daftar term. Model tokenisasi yang digunakan adalah *word token*, yakni pemisahan term berdasarkan karakter spasi *(white space)*.

Normalisasi Term dengan Lingua::IdSplitter dan *Case Folding*

Term-term hasil tokenisasi akan dinormalisasi dengan Lingua::IdSplitter. Normalisasi ini akan mengubah *identifier* yang umumnya mengandung gabungan kata serta mengekspansi singkatan menjadi kata lengkap. Term yang memiliki variasi huruf besar dan huruf kecil selanjutnya akan diubah semuanya ke huruf kecil. Hal ini dimaksudkan untuk menyamaratakan semua karakter ke bentuk huruf kecil. Sebagai contoh, *identifier* `BufferedImage` akan dinormalisasi menjadi `buffered` dan `image`, `createText` akan dinormalisasi menjadi `create` dan `text`, serta `fillRect` akan dinormalisasi menjadi `fill` dan `rectangle`.

Mengingat Lingua::IdSplitter merupakan modul Perl, maka pada penelitian ini, Lingua::IdSplitter akan difungsikan sebagai program eksternal yang akan dieksekusi pada aplikasi berbasis Java yang dikembangkan dengan bantuan objek `Runtime` dan `Process` sehingga pada akhirnya akan dihasilkan ≥ 1 term setiap normalisasi dijalankan.

Penghapusan Stop Word

Daftar term yang telah diperoleh selanjutnya difilter kembali untuk menghilangkan term-term yang merupakan term umum *(stop word)* seperti term *the*, *a*, *or*, dan lain-lain sebagaimana yang muncul pada daftar *stop list* yang digunakan.

Stemming

Daftar term yang telah difilter dengan *stop word* selanjutnya akan diubah lagi bentuknya dengan proses *stemming* untuk mengurangi variasi kata yang memiliki akar kata (stem) yang sama. Sebagai contoh, term *following* akan berubah menjadi *follow*, *buffered* akan menjadi *buffer*, *image* akan menjadi *imag*, *graphics* akan menjadi *graphic*, serta *rectangle* akan menjadi *rectangl* dengan memanfaatkan Porter *Stemmer*.

Pembobotan TF-IDF

Term yang telah didapatkan selanjutnya dihitung frekuensi kemunculannya serta diboboti dengan menggunakan TF-IDF berdasarkan persamaan (1). Dengan demikian, akan didapatkan daftar term unik beserta bobotnya terhadap data diskusi. Hasil pembobotan ini akan menghasilkan fitur vektor untuk tiap-tiap data diskusi.

Cosine *Similarity*

Penghitungan Cosine *similarity* digunakan untuk membandingkan antara *query* yang dimasukkan dengan data diskusi yang sudah tersedia dalam bentuk indeks terbobot (vektor). Selanjutnya hasil pembandingan akan diperingkat berdasarkan tingkat kemiripannya menggunakan persamaan (3).

Evaluasi Hasil

Evaluasi pada penelitian ini dilakukan dengan tujuan untuk mengetahui kualitas perbaikan metode rekomendasi yang memanfaatkan proses normalisasi *identifier* menggunakan Lingua::IdSplitter melalui penghitungan nilai *precision* dan *recall* berdasarkan persamaan (4) dan (5). Nilai-nilai tersebut akan dibandingkan dengan nilai *precision* maupun *recall* yang dihasilkan oleh sistem rekomendasi yang tidak memanfaatkan proses normalisasi *identifier*. Mengingat data yang digunakan berasal dari [8], maka hasil penelitian ini juga akan dilihat keselarasannya dengan penelitian tersebut. Selain itu, guna mengetahui kemampuan Lingua::IdSplitter dalam meningkatkan performa rekomendasi, akan digunakan pula data diskusi alternatif sejumlah 80 buah (20 diskusi/topik) dengan *identifier* singkatan yang muncul pada data dengan proporsi 60:20 (ada singkatan : tidak ada singkatan). *Identifier* singkatan muncul terutama sebagai *identifier-identifier* yang merupakan term kunci topik.

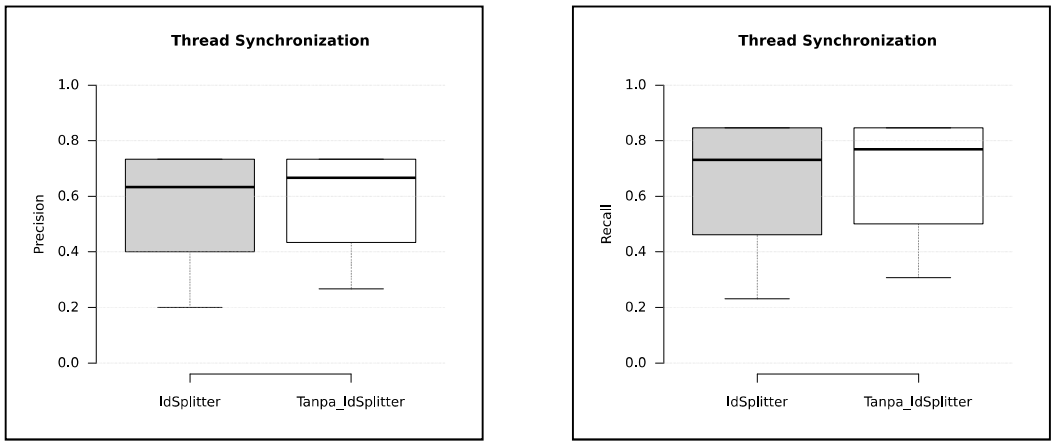
Data uji *(query)* masukan pada sistem ini adalah kode program Java yang berjumlah 32 buah sebagaimana yang telah dipaparkan sebelumnya. Setiap kode program yang dimasukkan, diharapkan mampu menghasilkan rekomendasi diskusi sesuai topik dari kode program. Hasil rekomendasi yang akan dihitung performanya adalah 15 data dengan nilai bobot dokumen tertinggi pertama sebagaimana yang dilakukan oleh [6].

HASIL DAN PEMBAHASAN

Pengujian dengan Kode Program Terkait *Thread Synchronization*

Berdasarkan grafik pada Gambar 6 (a), dapat dilihat bahwa nilai *precision* untuk data yang dinormalisasi dengan Lingua::IdSplitter memiliki jangkauan antara 0,2 hingga 0,73 dengan median

0,63 sementara data yang tidak dinormalisasi memiliki jangkauan antara 0,27 hingga 0,73 dengan median 0,67. Sedangkan apabila dilihat dari nilai *recall*, sebagaimana tampak pada Gambar 6 (b), diketahui bahwa *recall* yang diperoleh ketika data dinormalisasi yakni antara 0,23 hingga 0,85 dengan median 0,73 sementara data yang tidak dinormalisasi yakni antara 0,31 hingga 0,85 dengan median 0,77. Dengan demikian, dapat disimpulkan bahwa proses normalisasi tidak dapat meningkatkan nilai *precision* maupun *recall* untuk *query* dengan topik *thread synchronization*.



(a) *Precision* (b) *Recall*

Gambar 6. Perbandingan performa untuk topik *thread synchronization*

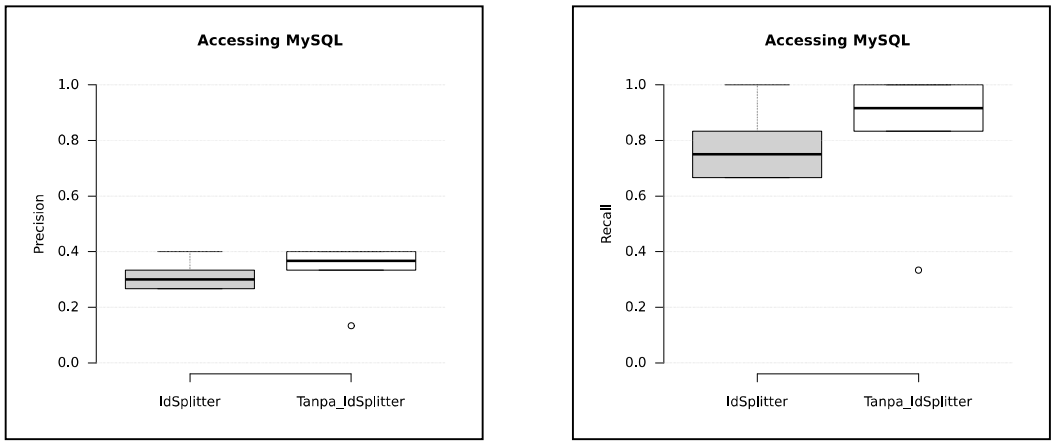
Hal ini disebabkan oleh tidak ditemukannya *identifier* `synchronize` yang disingkat menjadi `sync` pada data diskusi sebagaimana yang lumrah digunakan oleh *programmer* dalam menyingkat *identifier*. Oleh karena itu, normalisasi (ekspansi) term tidak berdampak terhadap kasus *query* dengan topik *thread synchronization* ini. Selain itu, sinkronisasi *thread* juga umum menggunakan konsep *exception* dengan jenis `InterruptedException`. Pemisahan *identifier* tersebut menjadi `Interrupted` dan `Exception` justru menurunkan skor dokumen terkait *thread synchronization* karena term `exception` lumrah ditemukan pada topik-topik selainnya.

Pengujian dengan Kode Program Terkait *Accessing* MySQL

Berdasarkan grafik pada Gambar 7 (a), dapat dilihat bahwa nilai *precision* untuk data yang dinormalisasi dengan Lingua::IdSplitter memiliki jangkauan antara 0,27 hingga 0,4 dengan median 0,3 sementara data yang tidak dinormalisasi memiliki jangkauan antara 0,33 hingga 0,4 dengan median 0,37. Sementara apabila dilihat dari nilai *recall*, sebagaimana tampak pada Gambar 7 (b), diketahui bahwa *recall* yang diperoleh ketika data dinormalisasi yakni antara 0,23 hingga 1 dengan median 0,75 sementara data yang tidak dinormalisasi yakni antara 0,83 hingga 1 dengan median 0,92. Dengan demikian, dapat disimpulkan bahwa proses normalisasi juga tidak dapat meningkatkan nilai *precision* maupun *recall* untuk *query* dengan topik *accessing* MySQL. Hal ini antara lain disebabkan oleh hasil ekspansi *identifier* `jdbc` menjadi `j`, `database`, dan `c`. Perluasan tersebut justru membuat direkomendasikannya data-data diskusi lain yang mengandung term `database` meski tidak memiliki term `jdbc` yang menjadi ciri umum bahasan tentang pengaksesan MySQL. Selain itu, pemisahan *identifier* `mysql` menjadi `my` dan `sql` juga turut menurunkan performa rekomendasi. Pemisahan tersebut justru mengakibatkan hilangnya term penting `mysql` dalam mewakili data diskusi terkait *accessing* MySQL.

Normalisasi (pemisahan) dapat memberikan peningkatan hasil rekomendasi untuk *query* kode program `org.embulk.output.MySQLOutputPlugin.java` yang berisi tentang kode program untuk melakukan *insert* ke basis data MySQL secara massal. Tanpa normalisasi, sistem menjadikan *identifier* seperti `batchInsert`, `mysqlBatchInsert`, dan `newBatchInsert` sebagai term penting *query*. Secara logika, term-term tersebut memang tepat mewakili *query*. Namun, hal tersebut tidak didukung oleh sampel data diskusi, tidak ada satupun topik data diskusi yang membahas tentang proses *insert* secara massal. Jika demikian, setidaknya sistem bisa merekomendasikan data diskusi

tentang proses *insert* pada MySQL secara umum. Akan tetapi, kenyataannya tidak. Efek tersebut dapat dilihat pada pencilan nilai *precision* pada titik 0,13 dan *recall* pada titik 0,33 untuk proses rekomendasi yang tidak menggunakan normalisasi Lingua::IdSplitter.

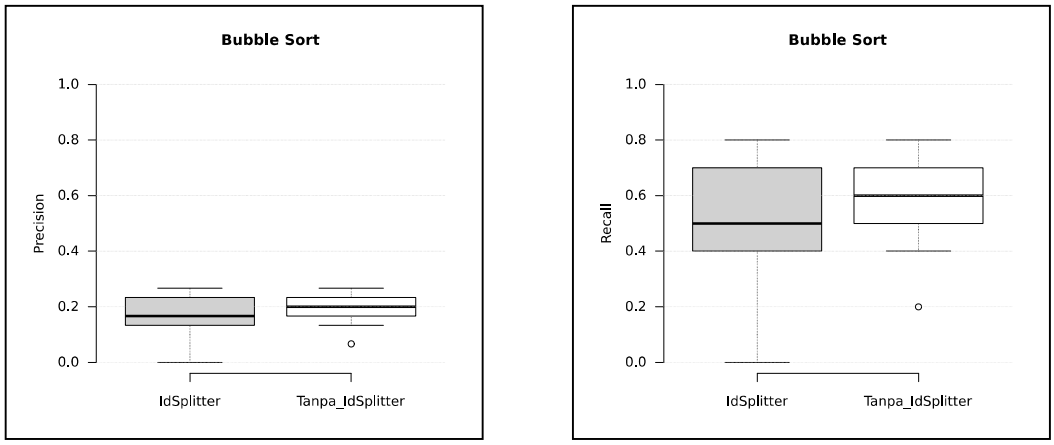


(a) *Precision* (b) *Recall*

Gambar 7. Perbandingan performa untuk topik *accessing* MySQL

Pengujian dengan Kode Program Terkait *Bubble Sort*

Grafik pada Gambar 8 (a) menunjukkan bahwa nilai *precision* untuk data yang dinormalisasi dengan Lingua::IdSplitter memiliki jangkauan antara 0 hingga 0,27 dengan median 0,17 sementara data yang tidak dinormalisasi memiliki jangkauan antara 0,13 hingga 0,27 dengan median 0,2. Sementara apabila dilihat dari nilai *recall*, sebagaimana tampak pada Gambar 8 (b), diketahui bahwa *recall* yang diperoleh ketika data dinormalisasi yakni antara 0 hingga 0,8 dengan median 0,5 sementara data yang tidak dinormalisasi yakni antara 0,4 hingga 0,8 dengan median 0,6. Dapat disimpulkan bahwa proses normalisasi juga tidak dapat meningkatkan nilai *precision* maupun *recall* untuk *query* dengan topik *bubble sort*. Hal ini antara lain disebabkan oleh hasil pemisahan *identifier* `BubbleSort` menjadi `bubble` dan `sort`. Pemisahan tersebut justru membuat meningkatnya skor data diskusi tentang *sorting* yang lain, yang menggunakan metode selain *bubble sort*. Ekspansi pada topik ini juga tidak terlalu berpengaruh mengingat *identifier* `bubble` maupun `sort` jarang dirupakan dalam bentuk singkat.

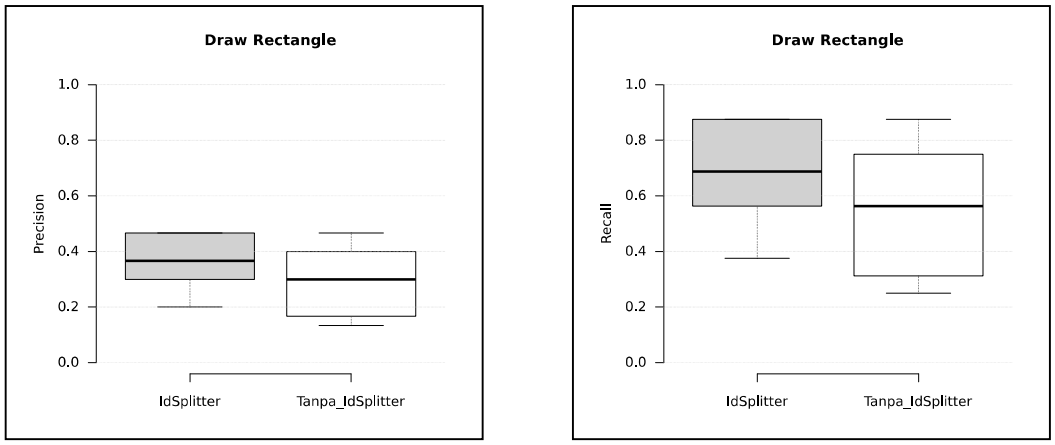


(a) *Precision* (b) *Recall*

Gambar 8. Perbandingan performa untuk topik *bubble sort*

Pengujian dengan Kode Program Terkait *Draw Rectangle*

Berdasarkan grafik pada Gambar 9 (a), dapat dilihat bahwa nilai *precision* untuk data yang dinormalisasi dengan Lingua::IdSplitter memiliki jangkauan antara 0,2 hingga 0,47 dengan median 0,37 sementara data yang tidak dinormalisasi memiliki jangkauan antara 0,13 hingga 0,47 dengan median 0,3.



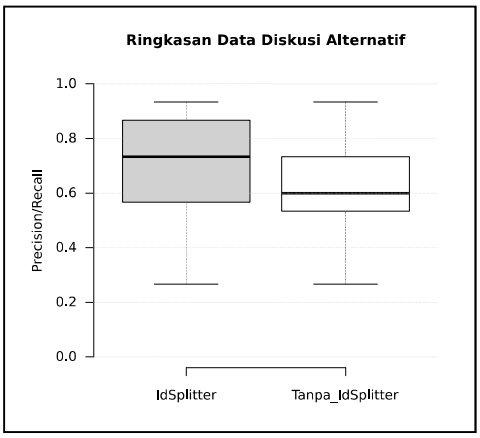
(a) *Precision* (b) *Recall*

Gambar 9. Perbandingan performa untuk topik *draw rectangle*

Sementara apabila dilihat dari nilai *recall*, sebagaimana tampak pada Gambar 9 (b), diketahui bahwa *recall* yang diperoleh ketika data dinormalisasi yakni antara 0,38 hingga 0,88 dengan median 0,69 sementara data yang tidak dinormalisasi yakni antara 0,25 hingga 0,88 dengan median 0,56. Dengan demikian, dapat disimpulkan bahwa proses normalisasi dapat meningkatkan nilai median *precision* maupun *recall* masing-masing sebesar 7% dan 13% untuk *query* dengan topik *draw rectangle*. Hal ini antara lain disebabkan oleh pemisahan *identifier* `DrawFeature` menjadi `Draw` dan `Feature`, `DrawRectangle` menjadi `Draw` dan `Rectangle`, serta `DrawRectangleOptions` menjadi `Draw`, `Rectangle`, dan `Options`. *Identifier* dalam bentuk terpisah lebih banyak ditemui pada data diskusi. Oleh karena itu, proses normalisasi menjadi penting. Selain itu, pemisahan dan ekspansi *identifier* `fillRect` menjadi `fill` dan `Rectangle` serta `drawRect` menjadi `draw` dan `Rectangle` memberikan dampak positif terhadap hasil rekomendasi, mengingat teks diskusi yang ada lebih banyak menggunakan kata lengkap dibandingkan singkatan.

Uji Coba Menggunakan Data Alternatif

Hasil uji coba menggunakan data alternatif dengan dominasi *identifier* yang mengandung singkatan dapat dilihat pada Gambar 10. Pengujian ini menghasilkan nilai-nilai *precision* dan *recall* yang sama, sehingga representasinya hanya ditampilkan dalam satu grafik saja.



Gambar 10. Ringkasan perbandingan uji coba data alternatif

Berdasarkan grafik pada Gambar 10, dapat dilihat bahwa nilai *precision* dan *recall* baik untuk data yang dinormalisasi dengan Lingua::IdSplitter maupun tidak memiliki jangkauan yang sama antara 0,27 hingga 0,93. Akan tetapi, nilai median keduanya berbeda. Nilai median untuk data yang dinormalisasi mampu mencapai 0,73. Sedangkan nilai median untuk data yang tidak dinormalisasi hanya mampu mencapai 0,6. Dengan demikian, untuk data alternatif ini, proses normalisasi mampu meningkatkan nilai *precision* dan *recall* sebesar 13%.

Ringkasan Hasil Uji Coba

Berdasarkan hasil pengujian yang telah dijelaskan untuk tiap-tiap topik, diketahui bahwa proses normalisasi dengan Lingua::IdSplitter hanya berpengaruh positif terhadap rekomendasi *query* dengan topik *draw rectangle*. Hal ini disebabkan oleh keberadaan *identifier* `rectangle` pada data diskusi maupun *query*, baik berupa singkatan maupun term lengkap. Sedangkan berdasarkan uji coba terhadap data alternatif yang telah dipilih, diketahui bahwa normalisasi *identifier* menggunakan Lingua::IdSplitter juga berpengaruh positif terhadap hasil rekomendasi dilihat dari nilai median yang dihasilkan. Hal tersebut juga dipengaruhi oleh keberadaan *identifier* pada data diskusi yang ditulis dalam bentuk singkatan.

Adapun apabila dilihat keselarasannya dengan penelitian [8] yang menggunakan model topik dalam merekomendasikan data diskusi, penggunaan model bobot TF-IDF masih cukup bisa diandalkan dengan melihat nilai median *recall* umum yang bisa dihasilkan oleh sistem pada data uji yang sama, yakni sebesar 0,67. Namun, perbaikan masih perlu dilakukan mengingat nilai median *precision* umum yang dihasilkan hanya mampu mencapai 0,33. Hal ini diakibatkan oleh ketidaktepatan pemilihan *identifier-identifier* yang layak untuk dinormalisasi, baik terkait kasus ketidaktepatan pemisahan komposisi term maupun pengekspansian term penuh dari term singkatan sebagaimana yang telah dijelaskan sebelumnya.

KESIMPULAN

Berdasarkan hasil penelitian, dapat disimpulkan bahwa:

1. Proses normalisasi *identifier* menggunakan Lingua::IdSplitter mampu meningkatkan nilai median *precision* sebesar 7% dan nilai median *recall* sebesar 13% khusus untuk data diskusi dengan topik *draw rectangle* pada data uji awal.
2. Proses normalisasi *identifier* juga secara umum mampu meningkatkan nilai median *precision* maupun *recall* sebesar 13% pada data uji alternatif.

UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih kepada Kementerian Riset, Teknologi, dan Pendidikan Tinggi RI (sebelumnya: Dikti) atas dukungan finansialnya dalam bentuk Beasiswa Unggulan.

DAFTAR PUSTAKA

[1] Cegielski, C. G. and Hall, D. J., 2006. What Makes A Good Programmer? *Communications of the ACM*, 49(10), pp.73-75.
[2] LaToza, T. D., Venolia, G., and DeLine, R., 2006. Maintaining Mental Models: A Study of Developer Work Habits. *Proceedings of the 28th international conference on Software engineering*, *ACM*, pp.492-501.
[3] Anderson, A., Huttenlocher, D., Kleinberg, J., and Leskovec, J., 2012. Discovering Value from Community Activity on Focused Question Answering Sites: A Case Study of Stack Overflow. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, *ACM*, pp.850-858.
[4] Manning, C. D., Raghavan, P., and Schütze, H., 2008. *Introduction to Information Retrieval*. Cambridge: Cambridge University Press.

[5] Cordeiro, J., Antunes, B., and Gomes, P., 2012. Context-Based Recommendation to Support Problem Solving in Software Development. *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop*, *IEEE*, pp.85-89.
[6] Ponzanelli, L., Bacchelli, A., and Lanza, M., 2013. Leveraging Crowd Knowledge for Software Comprehension and Development. *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference*, *IEEE*, pp.57-66.
[7] Krauthammer, M. and Nenadic, G., 2004. Term identification in the biomedical literature. *Journal of biomedical informatics, 37*(6), pp.512-526.
[8] Arwan, A., Rochimah, S., and Akbar, R. J., 2015. Source Code Retrieval on StackOverflow Using LDA. *Information and Communication Technology (ICoICT), 2015 3rd International Conference*, *IEEE*, pp.295-299.
[9] Carvalho, N. R., Almeida, J. J., Henriques, P. R., and Varanda, M. J., 2015. From Source Code Identifiers to Natural Language Terms. *Journal of Systems and Software, 100*, pp.117-128.
[10] Lawrie, D., Morrell, C., Feild, H., and Binkley, D., 2007. Effective identifier names for comprehension and memory. *Innovations in Systems and Software Engineering, 3*(4), pp.303-318.
[11] Hu, D. J., 2009. Latent Dirichlet Allocation for Text, Images, and Music. *University of California, San Diego*.
[12] Adomavicius, G. and Tuzhilin, A., 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *Knowledge and Data Engineering, IEEE Transaction, 17*(6), pp.734-749.
[13] Robillard, M. P., Walker, R. J., and Zimmermann, T., 2010. Recommendation Systems for Software Engineering. *Software, IEEE, 27*(4), pp.80-86.
[14] Gosling, J., Joy, B., Steele, G., Bracha, G., and Buckley, A., 2015. *Chapter 3. Lexical Structure*, *(online)*, (https://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html, diakses 20 Juli 2016).
[15] Vohra, D., Baesens, B., Backiel, A. and Vanden Broucke, S., 2015. *Beginning Java Programming: The Object-oriented Approach*. John Wiley & Sons.
[16] Sun Microsystems, 1997. *Java Code Conventions*. Sun Microsystems, Inc.
[17] Mahmoud, A. and Niu, N., 2011. Source Code Indexing for Automated Tracing. *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, *ACM*, pp.3-9.
[18] Jivani, A. G., 2011. A Comparative Study of Stemming Algorithms. *Int. J. Comp. Tech. Appl, 2*(6), pp.1930-1938.