PERBANDINGAN PERFORMA METODE PARALEL SVM DAN K-NN DALAM MENGKLASIFIKASI PASANGAN AKRONIM DAN KEPANJANGANNYA MENGGUNAKAN TEKNOLOGI BIG DATA SPARK

TUGAS AKHIR

Diajukan untuk melengkapi tugas-tugas dan
memenuhi syarat-syarat guna memperoleh gelar Sarjana Komputer

Oleh:

DENNY SYAPUTRA
1508107010036

JURUSAN INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SYIAH KUALA, BANDA ACEH
JANUARI, 2020

PENGESAHAN

PERBANDINGAN PERFORMA METODE PARALEL SVM DAN K-NN DALAM MENGKLASIFIKASI PASANGAN AKRONIM DAN KEPANJANGANNYA MENGGUNAKAN TEKNOLOGI BIG DATA SPARK

COMPARISON OF PERFORMANCE OF PARALLEL SVM AND K-NN METHODS IN CLASSIFICATION OF INDONESIAN ACRONYMS AND EXPANSIONS USING BIG DATA SPARK TECHNOLOGY

Oleh

Nama : Denny Syaputra
NPM : 1508107010036
Program Studi : S1 - Informatika

Menyetujui:

Pembimbing I,	Pembimbing II,
Prof. Dr. Taufik Fuadi Abidin, S.Si., M.Tech.	Ridha Ferdhiana, S.Si., M.Sc
NIP. 197010081994031002	NIP. 197302141998022001

Mengetahui:

Dekan Fakultas MIPA Universitas Syiah Kuala,	Ketua Jurusan Informatika FMIPA Universitas Syiah Kuala,
Dr. Teuku M. Iqbalsyah S.Si, M.Sc	Dr. Muhammad Subianto, S.Si., M.Si
NIP. 197110101997031003	NIP. 196812111994031005

Lulus Sidang Sarjana pada hari Selasa tanggal 21 Januari 2020

SURAT PERNYATAAN

Yang bertanda tangan dibawah ini:

1. Nama : Denny Syaputra
   NPM : 1508107010036
   Jurusan/Prodi : Informatika
   Status : Mahasiswa

2. Nama : Prof. Dr. Taufik Fuadi Abidin, S.Si., M.Tech.
   NIP : 197010081994031002
   Jurusan/Prodi : Informatika
   Status : Pembimbing I

3. Nama : Ridha Ferdhiana, S.Si., M.Sc
   NIP : 197302141998022001
   Jurusan/Prodi : Statistika
   Status : Pembimbing II

Dengan ini menyatakan hasil penelitian Tugas Akhir yang berjudul: Perbandingan Performa Metode Paralel SVM dan K-NN dalam Mengklasifikasi Pasangan Akronim dan Kepanjangannya menggunakan Teknologi Big Data SPARK, tidak dipublikasikan di *Electronic Thesis dan Dissertation* (ETD) hingga batas waktu 5 (lima) tahun, karena sedang dalam proses publikasi Jurnal / Prosiding.

Demikian surat pernyataan ini dibuat dengan sebenarnya untuk dapat dipergunakan seperlunya.

Daruss`alam, 21 Januari 2020

Yang membuat pernyataan,

Pembimbing I,	Pembimbing II,	Mahasiswa,
Prof. Dr. Taufik Fuadi Abidin, S.Si., M.Tech.	**Ridha Ferdhiana, S.Si., M.Sc**	**Denny Syaputra**
NIP. 197010081994031002	NIP. 197302141998022001	NPM. 1508107010036

Mengetahui:

Ketua Jurusan Informatika,	Koordinator TA,
Dr. Muhammad Subianto, S.Si, M.Si	Kurnia Saputra, S.T., M.Sc
NIP. 196812111994031005	NIP. 198003262014041001

PERNYATAAN BEBAS PLAGIASI

Saya yang bertanda tangan di bawah ini,

Nama lengkap	: Denny Syaputra
Tempat/Tanggal Lahir	: Pem. Tengah, 14 Juni 1998
NPM	: 1508107010036
Program Studi	: Informatika
Fakultas	: MIPA
Judul Tugas Akhir	: Perbandingan Performa Metode Paralel SVM dan K-NN dalam Mengklasifikasi Pasangan Akronim dan Kepanjangannya Menggunakan Teknologi Big Data SPARK.

Menyatakan dengan sesungguhnya bahwa Laporan Tugas Akhir saya dengan judul seperti di atas adalah **hasil karya saya sendiri** bersama dosen pembimbing dan **bebas plagiasi**.

Jika ternyata dikemudian hari terbukti bahwa Laporan Tugas Akhir merupakan hasil plagiasi, saya bersedia menerima sanksi yang berlaku di Universitas Syiah Kuala.

Banda Aceh, 21 Januari 2020
Yang menyatakan,

Denny Syaputra
NPM. 1508107010036

ABSTRAK

Data dalam jumlah yang besar dihasilkan setiap detiknya, hal ini membuat jumlah data bertambah secara eksponensial. Penambahan ini membawa masalah pada pemrosesan data karena akan membutuhkan waktu yang sangat lama apabila menggunakan satu mesin, untuk mengatasi hal tersebut maka proses komputasi bisa dilakukan secara paralel. Salah satu hal yang dapat diekstrak dari data teks adalah akronim dan kepanjangannya. Penelitian yang telah dilakukan pada ekstraksi pasangan akronim dan ekspansinya dari data teks masih menggunakan satu mesin. Penelitian ini berfokus pada melakukan pembangkitan pasangan-pasangan kandidat akronim dan ekspansinya serta fitur-fiturnya dari 100.000 dan 200.000 data artikel menggunakan Hadoop MapReduce dengan bahasa pemrograman Perl dan Java serta melakukan klasifikasi menggunakan Apache Spark untuk metode Paralel SVM dan K-NN. Hasil yang didapat dari proses pembangkitan fitur-fitur pasangan kandidat akronim dan ekspansinya adalah Perl dengan *library* Hadoop Streaming jauh mengungguli Java dalam melakukan pembangkitan fitur. Saat memproses 200.000 artikel, Perl dengan 39 server hanya membutuhkan waktu 2 jam 2 menit sementara Java membutuhkan waktu 22 jam 28 menit. Sedangkan selama proses klasifikasi, meskipun Paralel SVM memiliki nilai F-Measure yang sedikit lebih rendah dibandingkan K-NN, namun Paralel SVM membutuhkan waktu yang jauh lebih sedikit. Waktu yang dibutuhkan Paralel SVM untuk mengklasifikasi 119 juta pasangan kandidat akronim dan ekspansinya menggunakan 39 server adalah 1 menit 18 detik sedangkan K-NN membutuhkan 26 menit 42 detik.

Kata Kunci: *Apache Hadoop, Apache Spark, Spark MLlib, SVM, K-NN, IndoAcro*

ABSTRACT

Large amounts of data are generated every second, thus makes the amount of data increase exponentially. This addition brings a problem in data processing because it will require a very long time when using a single machine, then the use of parallel computing to overcome this. One of the things that can be extracted from text data is the acronym and its length. Research that has been done on the extraction of acronym pairs and their expansion of text data still uses a single machine. This research focuses on generating acronym candidate pairs and their expansion and features of 100,000 and 200,000 article data using Hadoop MapReduce with Perl and Java programming languages, and classifying using Apache Spark for the SVM and K-NN Parallel methods. The results obtained from the process of generating pairs of acronym candidate features and its expansion are Perl with the Hadoop Streaming library far outperforming Java in generating features. When processing 200,000 articles, Perl with 39 servers only takes 2 hours 2 minutes while Java takes 22 hours 28 minutes. Whereas during the classification process, although the F-Measure of the SVM Parallel is slightly lower than K-NN, but requires much less time. The time needed for SVM Parallel to classify 119 million pairs of acronym candidates and its expansion using 39 servers is 1 minute 18 seconds while K-NN requires 26 minutes 42 seconds.

Key Words: Apache Hadoop, Apache Spark, Spark MLlib, SVM, K-NN, IndoAcro

KATA PENGANTAR

Assalamu'alaikum Warahmatullah Wabarakatuh

Segala puji dan syukur kepada Allah SWT yang telah melimpahkan berkah dan hidayah-Nya sehingga penulis dapat menyelesaikan Laporan Tugas Akhir ini. Shalawat dan Salam penulis sanjung sajikan kepada Nabi Besar Muhammad SAW.

Laporan Tugas Akhir yang berjudul "**Perbandingan Performa Metode Paralel SVM dan K-NN dalam Mengklasifikasi Pasangan Akronim dan Kepanjangannya Menggunakan Teknologi Big Data SPARK**" ini telah dapat diselesaikan atas bantuan banyak pihak. Oleh karena itu, melalui tulisan ini penulis ingin mengucapkan terima kasih dan penghargaan sebesar-besarnya kepada :

1. Bapak Dr. Muhammad Subianto, M.Si., selaku Ketua Jurusan Informatika.
2. Bapak Kurnia Saputra, S.T., M.Sc., selaku Koordinator Tugas Akhir Jurusan Informatika Unsyiah.
3. Bapak Dr. Taufik Fuadi Abidin, S.Si., M.Tech., selaku Dosen Pembimbing I Tugas Akhir ini dan Ibu Ridha Ferdhiana, S.Si, M.Sc., selaku Dosen Pembimbing II Tugas Akhir ini.
4. Orang tua penulis yang telah membatu dan banyak memberikan dukungan secara spiritual, moral, dan material.
5. Rizka Puspitasari yang telah menemani dan mendukung penulis dalam masa-masa sulit selama menyelesaikan penelitian dan menulis Laporan Tugas Akhir ini.
6. Seluruh sahabat seperjuangan penulis di Jurusan Informatika Unsyiah yang telah memberi semangat untuk menyelesaikan Laporan Tugas Akhir ini.

Penulis menyadari bahwa dalam penulisan dan penyajian Laporan Tugas Akhir ini masih belum sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang konstruktif sebagai bahan pembelajaran dalam membuat tulisan-tulisan selanjutnya. Akhir kata semoga tulisan ini bermanfaat bagi banyak pihak dan berguna bagi ilmu pengetahuan.

Banda Aceh, Januari 2020

Denny Syaputra
NPM. 1508107010036

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1. LATAR BELAKANG

Setiap detiknya, data yang kompleks dan dalam jumlah yang sangat besar telah dihasilkan yang disebut dengan Big Data (Paul, 2013). Data-data tersebut dapat berupa *record*, gambar, video, abstrak, log, berkas teks, *web*, dan lain-lain (Chen et al., 2014). Hal ini menimbulkan tantangan dalam menggali informasi dari data tersebut (Gupta et al., 2015). Studi yang dilakukan oleh International Data Corporation (IDC) menemukan bahwa hanya 0,5% data yang dihasilkan secara global telah berhasil diolah (Priestley, 2015). Artinya, masih sangat banyak informasi yang belum berhasil tergali dari data-data yang tersedia.

Jumlah data yang sangat besar membuat proses komputasi terdistribusi harus dilakukan menggunakan sejumlah server yang beroperasi secara simultan agar waktu yang dibutuhkan dalam memproses data tidak terlalu lama (Dean and Ghemawat, 2004). *High Availability Distributed Object Oriented Platform* atau Hadoop adalah salah satu *platform* yang dapat digunakan untuk menganalisis dan memproses kumpulan data yang sangat besar. Hadoop terdiri dari 2 komponen utama, yaitu *Hadoop File System* (HDFS) dan MapReduce (Gupta et al., 2015). HDFS digunakan untuk menyimpan file dalam sistem Hadoop dan memungkinkan sistem untuk mempertahankan data jika terjadi kegagalan sistem. Cara kerja HDFS adalah dengan menyimpan data dalam beberapa bagian yang kemudian didistribusikan ke server-server yang ada dalam beberapa salinan, sehingga ketika kesalahan terjadi di salah satu server, data masih dapat diambil dari server yang lain (Bhosale and Gadekar, 2014). MapReduce merupakan komponen pada Hadoop yang mendistribusikan proses yang akan dijalankan pada Hadoop ke server-server yang tersedia untuk kemudian dikerjakan secara paralel (Bhosale and Gadekar, 2014).

Selain sistem komputasi, algoritma yang efisien juga menjadi faktor penting dalam memperoleh informasi dari data-data dalam jumlah besar. Beberapa algoritma klasifikasi ternama adalah C4.5 Decision Tree, Random Forest, Neural-Nework,

Support Vector Machine (SVM), Naive Bayes, Logistic Regression, dan K-Nearest Neighbour (K-NN) (Shafaque et al., 2014).

Salah satu permasalahan yang menarik untuk dibahas adalah pengolahan data teks yang berasal dari *web* untuk mendapatkan akronim dan kepanjangannya dalam Bahasa Indonesia (Wahyudi and Abidin, 2011). Beberapa kajian telah dilakukan untuk menentukan akronim dan kepanjangannya dari data teks dalam jumlah yang besar secara otomatis oleh komputer, diantaranya perbandingan algoritma klasifikasi Support Vector Machine (SVM) dan K-Nearest Neighbour (K-NN) serta menyempurnakan fitur yang pernah dikembangkan sebelumnya (Wahyudi and Abidin, 2011). Pada kajian tersebut, didapat hasil bahwa SVM dengan kernel *polynomial* memiliki nilai *F-Measure* lebih tinggi yaitu 99,5% dibandingkan dengan K-NN yaitu 99,2%. Namun, pada penelitian tersebut proses komputasi paralel hanya digunakan pada proses pembersihan data. Proses lainnya, seperti ekstraksi kandidat akronim, ekstraksi fitur, serta klasifikasi masih menggunakan server tunggal.

Penelitian selanjutnya (Abidin et al., 2018) memperbaharui proses komputasi penentuan fitur setiap kandidat pasangan akronim dan kepanjangannya menggunakan Hadoop. Hasil menunjukkan bahwa waktu proses yang dibutuhkan menurun secara signifikan bila dibandingkan dengan proses yang berjalan pada server tunggal. Waktu yang dibutuhkan untuk server tunggal untuk membersihkan 60.000 halaman web adalah 31,57 jam, sedangkan ekosistem Hadoop dengan 4 server hanya membutuhkan 11,37 jam. Namun, pada kajian tersebut, proses klasifikasi masih dilakukan menggunakan server tunggal.

Hal-hal di atas kemudian melatar belakangi penelitian ini. Penelitian ini bertujuan untuk membandingkan performa antara metode klasifikasi paralel SVM dan K-NN dalam mengklasifikasi pasangan akronim dan kepanjangannya menggunakan teknologi Big Data Hadoop. Penelitian ini juga bertujuan untuk menerapkan paradigma komputasi paralel pada proses klasifikasi karena model yang sebelumnya telah dibuat hanya dapat dijalankan pada server tunggal.

1.2. RUMUSAN MASALAH

Berdasarkan latar belakang yang telah dipaparkan sebelumnya, maka masalah yang dikaji pada tugas akhir ini adalah:

1. Belum dilakukan perbandingan performa antara metode klasifikasi paralel SVM dan K-NN dalam menentukan pasangan akronim dan kepanjangannya secara otomatis dari data teks.
2. Belum diterapkan pendekatan *MapReduce* dengan bahasa pemrograman Java dalam menentukan pasangan akronim dan kepanjangannya secara otomatis dari data teks.
3. Belum diterapkan proses komputasi yang terdistribusi pada proses klasifikasi pasangan akronim dan kepanjangannya secara otomatis dari data teks.

1.3. TUJUAN PENELITIAN

Berdasarkan rumusan masalah yang telah dipaparkan sebelumnya, maka dapat dirumuskan tujuan dari tugas akhir ini adalah:

1. Membandingkan akurasi hasil klasifikasi serta waktu yang dibutuhkan antara metode klasifikasi paralel SVM dan K-NN dalam mengklasifikasi akronim dan kepanjangannya.
2. Membandingkan akurasi antara metode klasifikasi SVM yang berjalan pada satu mesin dan paralel SVM yang berjalan di atas Hadoop dengan jumlah server yang berbeda-beda dalam mengklasifikasi akronim dan kepanjangannya.
3. Menerapkan proses komputasi yang terdistribusi menggunakan Paralel SVM dan K-NN pada Apache Spark dalam mengklasifikasi pasangan akronim dan kepanjangannya.

1.4. MANFAAT PENELITIAN

Setelah penelitian ini dilakukan, didapatkan hasil dari perbandingan performa terbaik antara SVM dan K-NN dalam melakukan klasifikasi akronim dan kepanjangannya dalam Bahasa Indonesia sehingga model terbaik yang didapatkan dapat diimplementasikan pada *repository* IndoAcro untuk meningkatkan performa dari *repository* tersebut. Selain itu, program yang telah ditransformasikan ke bahasa pemrograman Java dengan *MapReduce* dapat diterapkan pada praproses *repository* IndoAcro sehingga dapat mengurangi waktu yang dibutuhkan dalam memproses dataset baru yang akan ditambahkan ke *repository*.

BAB II
TINJAUAN KEPUSTAKAAN

2.1. BIG DATA

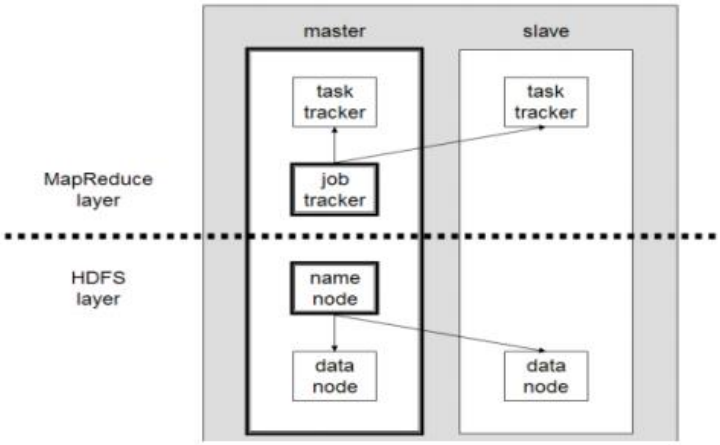
Pada masa sekarang, kita hidup di era *smart technologies* yang mewakili istilah "*ubiquitous computing*" (Ali et al., 2015). Setiap alat telah terhubung ke internet dan setiap detiknya menghasilkan data yang disebut dengan *Internet of Things* (IoT) (Ali et al., 2015). Data-data dalam jumlah banyak yang dihasilkan setiap saat membangun istilah baru untuk era di mana data berjumlah sangat besar yang disebut Big Data (Cecchinel et al., 2014). Istilah Big Data merujuk pada kumpulan data dengan ukuran yang sangat besar (*volume*), kompleksitas yang tinggi (*variability*), dan memiliki pertumbuhan yang sangat pesat (*velocity*) yang membuat kumpulan data tersebut tidak dapat diproses menggunakan mesin konvensional (Bhosale and Gadekar, 2014).

Big Data pada masa kini dapat berasal dari berbagai sumber dengan beragam format baik terstruktur maupun tidak (Ali et al., 2015). Format data dengan jumlah terbesar dalam Big Data adalah format teks (Krishnalal et al., 2010). Artikel berita sebagai salah satu penyumbang data teks pada era Big Data memiliki informasi yang sangat beragam didalamnya. Namun, jumlah yang sangat besar membuat waktu pengolahan data teks artikel berita sangat lama sehingga dibutuhkan teknologi yang dapat bekerja secara terdistribusi demi mengurangi waktu proses pengolahan data teks artikel berita tersebut (Krishnalal et al., 2010).

2.2. HADOOP

High Availability Distributed Object Oriented Platform (*Hadoop*) merupakan salah satu *platform* yang dapat digunakan untuk melakukan proses komputasi dari kumpulan data yang sangat besar di dalam kumpulan server-server yang bekerja secara terdistribusi (Bhosale and Gadekar, 2014). Hadoop (Apache Hadoop Software Foundation, 2018) merupakan *software open-source* yang ditulis dalam bahasa pemrograman Java yang diperkenalkan oleh perusahaan besar Apache (Rani and Rama, 2017). Terdapat dua komponen utama dalam Hadoop yaitu komponen penyimpanan data yaitu *Hadoop Distributed File System* (HDFS) dan komponen

proses yaitu MapReduce (Rani and Rama, 2017). Gambar 2.1 menunjukkan ilustrasi dari struktur komponen Hadoop.

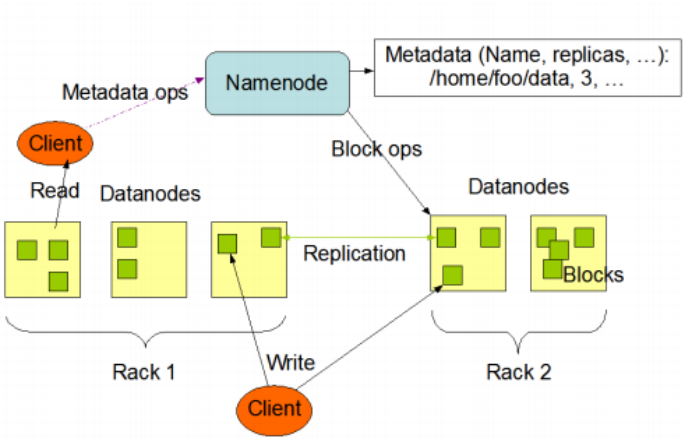


Gambar 2.1. Ilustrasi struktur komponen Hadoop (sumber: Rani and Rama, 2017)

2.2.1. *Hadoop Distributed File System*

Menurut Bhosale and Gadekar (2014), *Hadoop Distributed File System* (HDFS) merupakan salah satu komponen penting pada Hadoop yang berfungsi sebagai penyimpanan data. HDFS melakukan penyimpanan data dengan membagi data tersebut menjadi beberapa bagian yang disebut "blok*"* dan menyimpan blok-blok tersebut secara berulang di dalam server-server yang berbeda. Setiap blok yang disimpan oleh HDFS berukuran 64 MB atau 128 MB (Greeshma and Pradeepini, 2016). Selanjutnya, Bhosale and Gadekar mengatakan bahwa dengan menerapkan cara penyimpanan tersebut, HDFS dapat menyimpan data dalam jumlah besar serta bertahan dari kegagalan yang terjadi pada infrastruktur penyimpanan yang signifikan tanpa harus kehilangan data.

Menurut Pothuganti (Pothuganti, 2015) terdapat dua komponen yang masuk ke dalam HDFS, yaitu *Name Node* dan *Data Node*. *Name Node* merupakan sebuah server yang bertugas untuk mengatur metadata dan akses kontrol dari sistem *file*, sedangkan *Data Node* merupakan satu atau lebih server yang bertugas untuk menyimpan data. Setiap *data node* mengatur penyimpanan lokalnya sendiri dan salinan dari beberapa atau semua blok dalam *file system*. Ilustrasi dari arsitektur dari HDFS dapat dilihat pada Gambar 2.2.

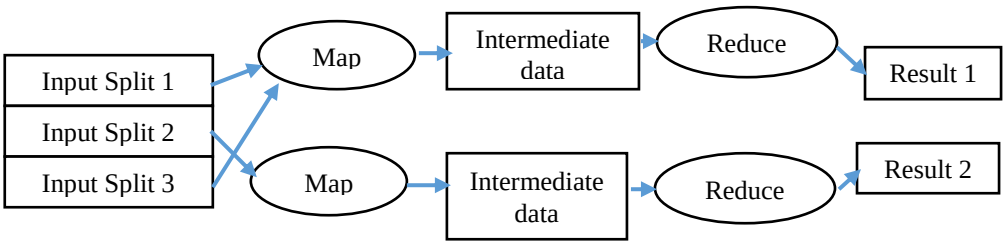


Gambar 2.2. Ilustrasi infrastruktur HDFS (sumber: Bhosale and Gadekar, 2014)

2.2.2. *MapReduce*

MapReduce merupakan paradigma pemrograman yang memungkinkan skalabilitas besar dijalankan pada ratusan atau ribuan server pada *cluster* Hadoop. Menurut Greeshma and Pradeepini (2016), *MapReduce* merupakan jantung dari Hadoop.

MapReduce terbagi menjadi dua buah proses, yaitu proses *map* dan proses *reduce.* Proses *map* merupakan proses yang bertugas untuk menerima input data dan memproses data tersebut menjadi pasangan *key-value*. Pasangan *key-value* yang disebut *intermediate data* itu kemudian diurutkan dan diberikan kepada proses kedua yang disebut proses *reduce*. Proses *reduce* kemudian menerima pasangan *key-value* sebagai masukan (input) untuk selanjutnya diproses untuk menghasilkan hasil akhir yang diinginkan (Greeshma and Pradeepini, 2016). Ilustrasi dari *MapReduce* dapat dilihat pada Gambar 2.3.



Gambar 2.3. Ilustrasi proses *MapReduce* (sumber: Greeshma and Pradeepini, 2016)

2.3. APACHE SPARK

Apache Spark (Apache Software Foundation, 2018) adalah sebuah *tool cluster computing* yang dapat digunakan untuk berbagai tujuan dalam *machine learning* yang sangat cepat dan dapat diandalkan (Shoro and Soomro, 2015). *Tool* ini mengombinasikan beberapa *library* diantaranya Spark SQL, Spark Streaming, MLlib, dan GraphX (Apache Software Foundation, 2018). Apache Spark awalnya dikembangkan oleh Matei Zaharia di Barkley's University California sebagai jawabannya atas peningkatan waktu komputasi yang dibutuhkan oleh MapReduce dalam memproses data yang semakin banyak. Spark kemudian diberikan dan dikelola oleh Apache Spark Foundation pada Juni 2013 (Pinto et al., 2017).

Spark merupakan sebuah *tool* yang cukup fleksibel yang menyediakan *Aplication Programming Interface* (API) dalam beberapa bahasa pemrograman, diantaranya Java, Scala, Python, dan R (Shoro and Soomro, 2015). Meskipun Spark dibangun di atas *Hadoop Distributed File System* (HDFS), namun Spark tidak terikat pada dua proses *Map* dan *Reduce* dari ekosistem Hadoop. Spark memungkinkan program yang ditulis untuk memuat data ke dalam memori *cluster* sehingga bisa di-*query* berulang kali. Hal ini membuat performa Spark dapat bekerja 100 kali lebih baik daripada *MapReduce* Hadoop (Ghatge, 2016).

2.4. INDOACRO

IndoAcro merupakan sebuah repositori dari akronim dan kepanjangannya dalam bahasa Indonesia yang dikumpulkan dan diekstrak secara otomatis menggunakan teknologi *machine learning* yang dikombinasikan dengan Big Data dalam praprosesnya (Abidin et al., 2018). IndoAcro awalnya diteliti dengan menggunakan K-NN sebagai metode klasifikasi (Wahyudi and Abidin, 2011). Penelitian tersebut berhasil mengklasifikasikan 89% akronim secara tepat. Namun, model yang didapat tidak dapat menentukan kandidat akronim dalam tulisan jika jumlah huruf kecil dalam akronim lebih dari 25% dari total huruf pembentuk akronim.

Kajian dilanjutkan dengan menerapkan teknologi *Big Data*, menambah fitur dan menggunakan metode SVM sebagai metode klasifikasinya. Namun, penggunaan teknologi *Big Data Hadoop* dalam kajian tersebut hanya dilakukan untuk membersihkan data web saja. Secara umum, proses masih menggunakan server

tunggal. Algoritma SVM berhasil meningkatkan performa menjadi 98.28%, meningkat 7.28% dari kajian sebelumnya.

Penelitian dilanjutkan kembali pada tahun 2018 dengan menerapkan teknologi *Big Data Hadoop* pada proses pembangkitan fitur (Abidin et al., 2018). Penelitian tersebut telah berhasil dalam mengekstrak kandidat akronim dan ekspansinya dan menghasilkan fitur menggunakan teknologi Hadoop. Fokus dari penelitian ini adalah untuk melihat peningkatan performa dari metode yang dilakukan pada server tunggal dengan metode yang menerapkan proses paralel. Hasil penelitian tersebut menunjukkan waktu yang dibutuhkan untuk mendapatkan hampir 52 juta pasang kandidat akronim dan kepanjangannya serta membangkitkan 8 fitur numerik yang digunakan untuk mengklasifikasi kandidat-kandidat pasangan akronim dan kepanjangan tersebut jauh lebih cepat menggunakan ekosistem Hadoop bila dibandingkan dengan menggunakan server tunggal. Hasil menunjukkan bahwa ekosistem Hadoop dapat meningkatkan kinerja proses dalam mengekstrak kandidat pasangan akronim dan kepanjangannya dari 60.000 file teks yang menghasilkan lebih dari 51 juta kandidat pasangan akronim dan kepanjangannya dari 31,57 jam bila menggunakan server tunggal menjadi 11,37 jam bila menggunakan proses yang terdistribusi ke 4 data node Hadoop.

2.5. KLASIFIKASI

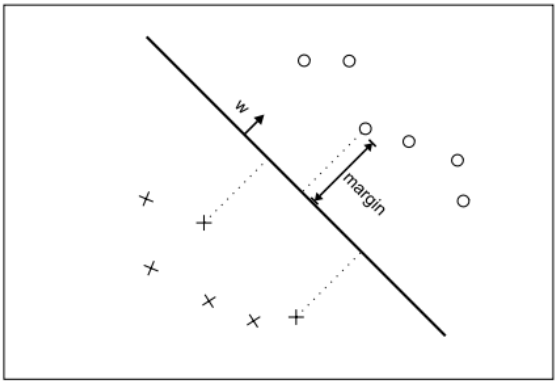
Menurut Han and Kamber (2006), klasifikasi adalah kegiatan membangun model atau *classifier* yang digunakan untuk memprediksi label atau kategori dari sebuah data yang belum diketahui kategorinya berdasarkan data-data relevan yang telah diketahui kategorinya. Kategori-kategori tersebut dapat berupa “aman” atau “tidak aman”, “ya” atau “tidak”, atau “negatif” atau “positif”. Kategori-kategori tersebut merupakan nilai diskrit yang tidak memiliki urutan tertentu. Sebagai contoh, nilai kategori 1, 2, atau 3 mungkin dapat mewakili kategori pengobatan tipe A, tipe B atau tipe C dalam kedoteran.

Phips et al. (1996) mengatakan bahwa teori klasifikasi lahir dari ilmu filosofi, matematika, statistika, psikologi, *computer science*, linguistik, biologi, farmasi, serta bidang-bidang lainnya. Beberapa algoritma klasifikasi ternama diantaranya C4.5 *decision tree, random forest, neural-nework, Support Vector Machine* (SVM), Naive

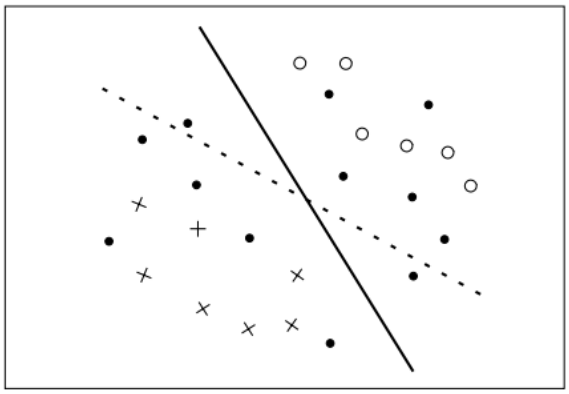
Bayes, Logistic Regression, dan K-NN (Shafaque et al., 2014). Dua metode klasifikasi yang dibandingkan pada tugas akhir ini adalah metode paralel SVM dan K-NN.

2.5.1. Paralel *Support Vector Machine* (SVM)

Support Vector Machine (SVM), diperkenalkan pertama kali untuk melakukan klasifikasi data teks oleh Joachims (Joachims, 1998). SVM merupakan salah satu metode pembelajaran terbimbing (*supervised learning*) yang sangat baik yang didasarkan pada prinsip minimalisasi risiko terstruktur (*structured risk minimization principle*) (Krishnalal et al., 2010). Metode ini merupakan metode yang berbasis vektor dengan tujuan untuk mencari garis pembatas (*hyperplane*) yang memisahkan dua kelas dengan memaksimalkan jarak antara setiap titik yang ada pada data pembelajaran (Paul, 2013). Gambar 2.5 dan 2.6 menunjukkan ilustrasi dari pemilihan *hyperplane* yang paling baik (Tong and Koller, 2001).



Gambar 2.4. Ilustrasi *hyperplane* yang baik (sumber: Tong and Koller, 2001)



Gambar 2.5. Ilustrasi pemilihan *hyperplane* yang baik antara 2 pilihan (sumber: Tong and Koller, 2001)

Selanjutnya, Tong and Koller mengatakan bahwa metode klasifikasi SVM membutuhkan data pembelajaran $(x_1 \dots x_n)$ yang merupakan vektor dalam ruang $X \subseteq R^n$ dengan label tiap data $\{y_1 \dots y_n\}$ di mana $y_i \, \epsilon \, \{-1, \ 1\}$. Setiap vektor atau data yang berada di satu sisi dari *hyperplane* berlabel -1, sedangkan vektor yang berada di sisi lain dari *hyperplane* berlabel 1. Menurut Han and Kamber (2006), garis *hyperplane* dapat ditulikan dengan fungsi:

$$\vec{w} \cdot \vec{x} + b = 0 \tag{2.1}$$

dimana $\vec{w}$ merupakan vektor tegak lurus dari *hyperplane* ke titik data, $\vec{x}$ merupakan vektor input dari data, dan *b* adalah bias dari fungsi garis. Setiap vektor yang berada di satu sisi *hyperplane* harus memenuhi kondisi yang dituliskan Formula 2.2.

$$\vec{w} \cdot \vec{x} + b > 1 \tag{2.2}$$

Sedangkan vektor yang berada di sisi lainnya harus memenuhi kondisi pada Formula 2.3.

$$\vec{w} \cdot \vec{x} + b < -1 \tag{2.3}$$

Support vector diambil dari data yang memiliki jarak paling dekat dengan batas margin *hyperplane* atau dapat dikatakan sebagai titik yang menjadi batas margin dari SVM. Jarak dari setiap titik ke garis *hyperplane* dapat dihitung dengan:

$$d = \frac{1}{\|w\|} \tag{2.4}$$

dengan $\|w\|$ merupakan fungsi jarak Euclidean dari *w*. Jarak antara *support vector* yang berada di salah satu sisi *hyperplane* dengan *support vector* yang berada di sisi lain *hyperplane* (margin optimal) adalah:

$$margin = \frac{2}{\|w\|} \tag{2.5}$$

Untuk memaksimalkan margin, nilai dari $\|w\|$ harus diminimalkan dengan subjek:

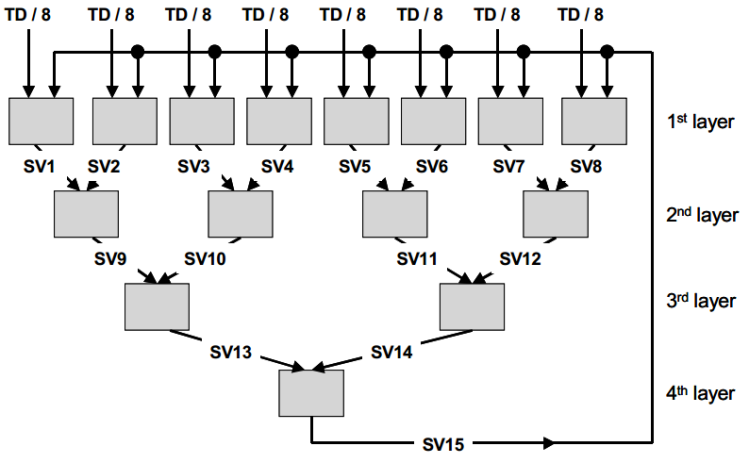
$$f(\vec{x}) = \begin{cases} 1 & jika\ \vec{w} \cdot \vec{x} + b \geq 1 \\ -1 & jika\ \vec{w} \cdot \vec{x} + b \leq -1 \end{cases} \quad (2.6)$$

Karakteristik dari SVM adalah fungsi pemisahannya dideskripsikan oleh *support vector* yang merupakan data terdekat ke *hyperplane*. Data selain *support vector* sebenarnya tidak berkontribusi dalam menentukan fungsi pemisah. Artinya, menghitung fungsi batas SVM dapat diartikan sebagai menemukan *support vector* dengan bobot yang sesuai untuk menggambar *hyperplane*.

Implementasi proses komputasi paralel pada algoritma SVM melahirkan istilah baru yaitu Paralel SVM. Alasan diimplementasikannya proses komputasi paralel pada SVM adalah masalah penggunaan memori serta waktu untuk menyelesaikan proses komputasi dalam menemukan *hyperplane* terbaik yang cukup lama (proses *training*) (ChanKrishnalal et al., 2007). Menurut Graf et al. (2004), salah satu metode yang dapat digunakan untuk mengimplementasikan proses komputasi paralel pada SVM disebut *Cascade* SVM. Metode *Cascade* SVM akan membagi data pembelajaran menjadi beberapa bagian yang kemudian didistribusikan ke server-server yang bekerja secara paralel. Setiap server kemudian akan menjalankan proses training SVM dengan menemukan *support vector*. Artinya, setiap server akan mengeliminasi data yang dianggap *non-support vector*. Selanjutnya, data digabungkan untuk dilakukan kembali eliminasi *non-support vector*.

Gambar 2.6 (Graf *et al*, 2004) merupakan ilustrasi dari paralel SVM. Proses eliminasi bekerja pada beberapa *layer*. Data-data yang tidak dieliminasi pada *layer* pertama kemudian setiap *support vector* dari 2 layer digabungkan untuk dijadikan data pembelajaran pada *layer* berikutnya, hingga tidak ada lagi set data yang dapat digabungkan. Kemudian, hasil dari layer terakhir akan dikembalikan ke layer pertama. Setiap SVM dari layer pertama akan menerima semua *support vector* dari layer terakhir sebagai masukan dan menguji fraksi dari vektor input tersebut untuk melihat apakah hasil dari setiap SVM pada layer pertama telah benar mengklasifikasikan data *support vector* dari layer terakhir sebagai data dari 2 kelas berbeda. Jika *support vector*

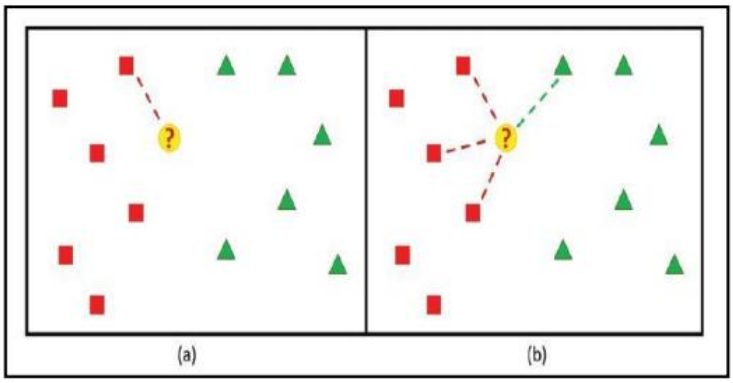
dari layer terakhir telah benar diklasifikasi berbeda, maka *Cascade* SVM akan berhenti. Jika tidak, maka Cascade SVM akan mengulang lagi proses dari layer pertama.



Gambar 2.6. Ilustrasi proses paralel SVM menurut Graf et al. (2004)

2.5.2. *K-Nearest Neighbor*

K-NN merupakan salah satu metode klasifikasi yang sederhana. Metode ini menggunakan seluruh data pembelajaran dalam mengklasifikasikan data baru dengan memberikan kelas kepada data baru berdasarkan kelas mayoritas dari *k* buah tetangga terdekat (Gou et al., 2012). Data pembelajaran pada metode K-NN dideskripsikan oleh *n* atribut (Han and Kamber, 2006). Setiap data pembelajaran tersebut kemudian direpresentasikan sebagai sebuah titik pada ruang dimensi *n*. Saat diberikan data yang kelasnya tidak diketahui, K-NN akan mencari ruang pola untuk *k* tetangga terdekat yang paling dekat dengan data tersebut. Gambar 2.8 (Imandoust and Bolandraftar, 2013) memperlihatkan penentuan kelas dari data baru berdasarkan data-data pembelajaran yang telah direpresentasikan kedalam ruang 2 dimensi.



Gambar 2.7. Ilustrasi penentuan kelas pada metode K-NN (sumber: Imandoust and Bolandraftar, 2013)

Jarak antara data baru dengan data pembelajaran didapat dengan melakukan perhitungan jarak atau *distance metrics* (Han and Kamber, 2006). Salah satu rumus perhitungan jarak yang dapat digunakan adalah rumus Euclidean distance. Jika terdapat data $X_1 = (x_{11}, x_{12}, x_{13}, ..., x_{1i})$ dan $X_2 = (x_{21}, x_{22}, x_{23}, ..., x_{2i})$, maka perhitungan jarak dengan Euclidean Distance adalah

$$dist(X1, X2) = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2} \quad (2.7)$$

Menurut Hechenbichler and Schliep (2005), setelah semua jarak dihitung, pembobotan juga perlu dilakukan untuk mendapatkan nilai yang relevan dan sesuai dengan jarak antara data baru dengan data-data yang sudah ada. Pembobotan akan memberi bobot yang lebih besar kepada data pembelajaran yang lebih dekat dengan data baru dan memberi bobot yang lebih kecil kepada data pembelajaran yang lebih jauh dengan data baru tersebut. Beberapa rumus yang dapat digunakan untuk melakukan pembobotan pada K-NN diantaranya *Cosine*, *Gaussian* dan *Invertion*. Pembobotan yang dipilih untuk penelitian ini adalah *Gaussian* yaitu:

$$\frac{1}{\sqrt{2\pi}} \exp(-\frac{d^2}{2}) \quad (2.8)$$

BAB III
METODE PENELITIAN

3.1. WAKTU DAN LOKASI PENELITIAN

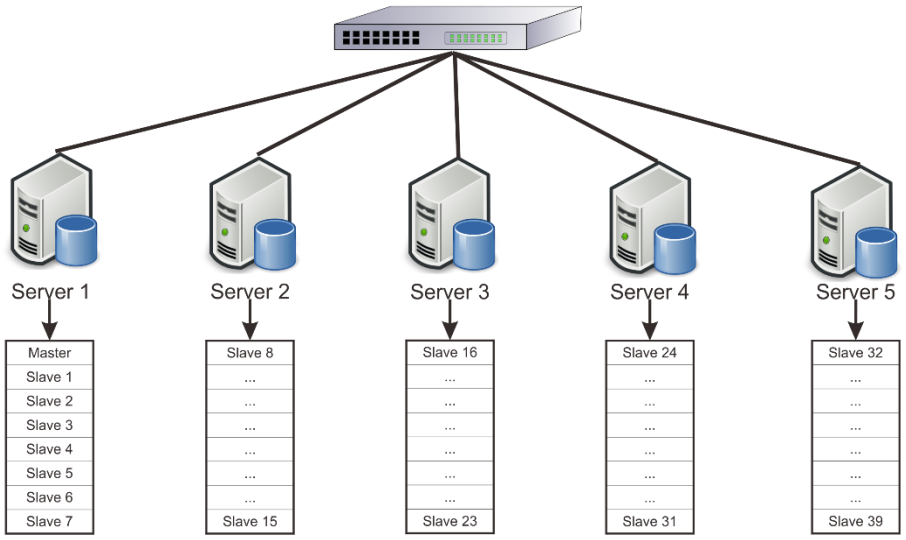
Penelitian ini dilakukan di Ruang Riset Grup DMIR, Gedung FMIPA Blok D lantai 2 Universitas Syiah Kuala. Waktu pelaksanaan penelitian ini adalah 8 bulan terhitung dari bulan April hingga Desember 2019.

3.2. ALAT DAN BAHAN

Alat dan bahan yang digunakan pada penelitian ini meliputi perangkat keras, perangkat lunak, dan data. Perangkat lunak yang digunakan adalah:

- Linux CentOS 7.0
- Linux Ubuntu 18.04 (SSH terkoneksi ke server Hadoop CentOS)
- Apache Hadoop 2.7.1
- Hortonworks HDP 3.1
- Hortonworks Ambari 2.7.4
- Apache Spark 2.1.2

Sedangkan perangkat keras yang digunakan pada penelitian ini adalah 1 unit Laptop Acer dengan RAM 6 GB, Intel® Core™ i5-5200U @ 2.20 GHz *Processor* dan *Harddisk* 500 GB. Ekosistem Hadoop dibangun dengan menggunakan 5 buah server fisik dengan spesifikasi RAM 128 GB, 32 core Processor dengan frekuensi 4 GHz, Hardisk 2 TB. Sebanyak 40 buah server virtual kemudian dibangun dengan menggunakan Proxmox VM dari 5 buah server fisik tersebut. Ilustrasi dari ekosistem yang digunakan ditunjukkan pada Gambar 3.1.



Gambar 3.1. Ilustrasi ekosistem server Hadoop

Spesifikasi untuk setiap *node* pada server yang digunakan adalah :

- Linux CentOS 7.0
- 16 GB RAM
- 8 Core Processor
- *Storage* 200 GB

Data yang digunakan berupa artikel bersih yang berisi URL, elemen judul dan isi artikel tanpa *tag-tag* HTML yang tidak diperlukan. Data artikel didapat dari penelitian sebelumnya pada tahun 2018 (Abidin *et al*., 2018). Artikel-artikel tersebut didapat dari beberapa situs berita yaitu Republika, Okezone, Detik, dan lain-lain dengan total data berjumlah 300,000 artikel. Selain itu, dalam membangun model dari metode klasifikasi paralel SVM dan K-NN digunakan 5,000 buah pasangan akronim dan ekspansi yang didapatkan dari penelitian sebelumnya (Abidin *et al.*, 2019). Data tersebut terdiri dari 2,469 pasangan benar dan 2,531 pasangan salah. Setiap pasangan tersebut dibagi menjadi 3 kelompok, yaitu akronim yang memiliki >= 75% huruf kapital (kelompok huruf kapital), 50-75% huruf kapital dan memiliki angka di akronim (kelompok gabungan huruf dan digit), dan yang memiliki <50% huruf kapital (kelompok *Syllables*). Detail distribusi dari data latih dapat dilihat pada Tabel 3.1. Kemudian untuk menguji model yang telah dibangun, digunakan 2,000 data pasangan akronim dan ekspansi yang terdiri dari 1,000 data benar dan 1,000 data salah. Data

tersebut dikumpulkan secara manual dengan mencari pasangan-pasangan akronim benar serta membuat pasangan yang salah dari pasangan-pasangan akronim dan ekspansi benar tersebut. Distribusi kelompok dari data uji dapat dilihat pada Tabel 3.2.

Tabel 3.1. Distribusi kelompok akronim pada data latih

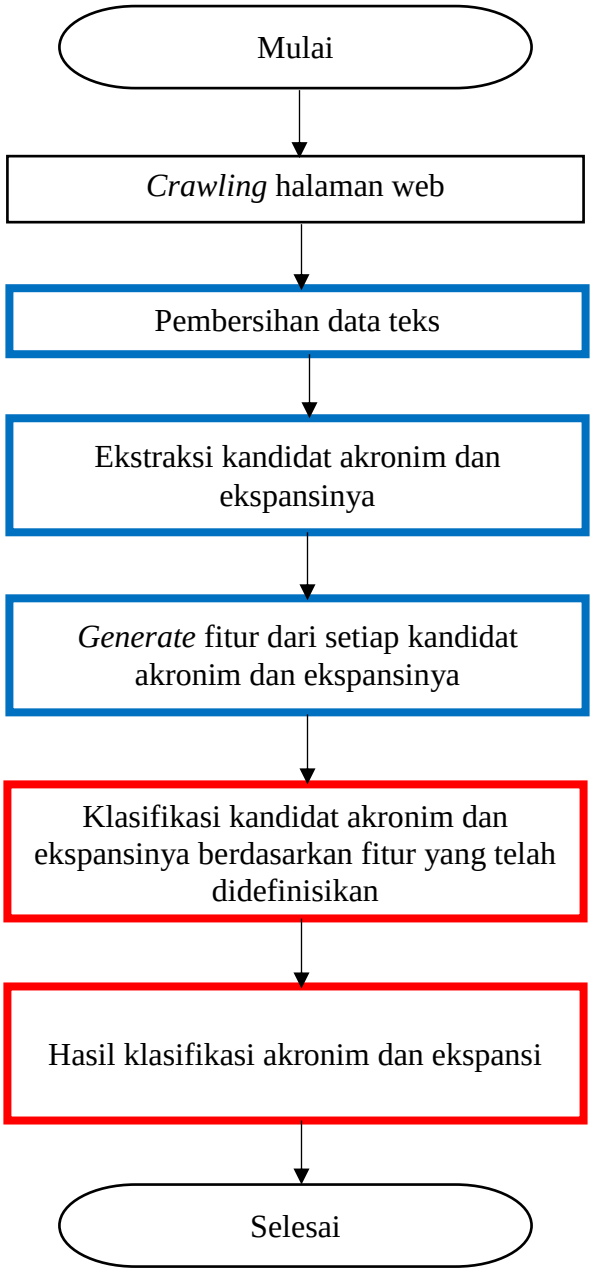
Jenis	**Jumlah**	
	Positif	**Negatif**
Kapital (>=75%)	1.729	1.344
Digit (50-75%)	10	15
Syllables (< 50%)	730	1.172
Total	**2.469**	**2.531**

Tabel 3.2. Distribusi kelompok akronim pada data uji

Jenis	**Jumlah**	
	Positif	**Negatif**
Kapital (>=75%)	569	549
Digit (50-75%)	13	13
Syllables (< 50%)	418	438
Total	**1.000**	**1.000**

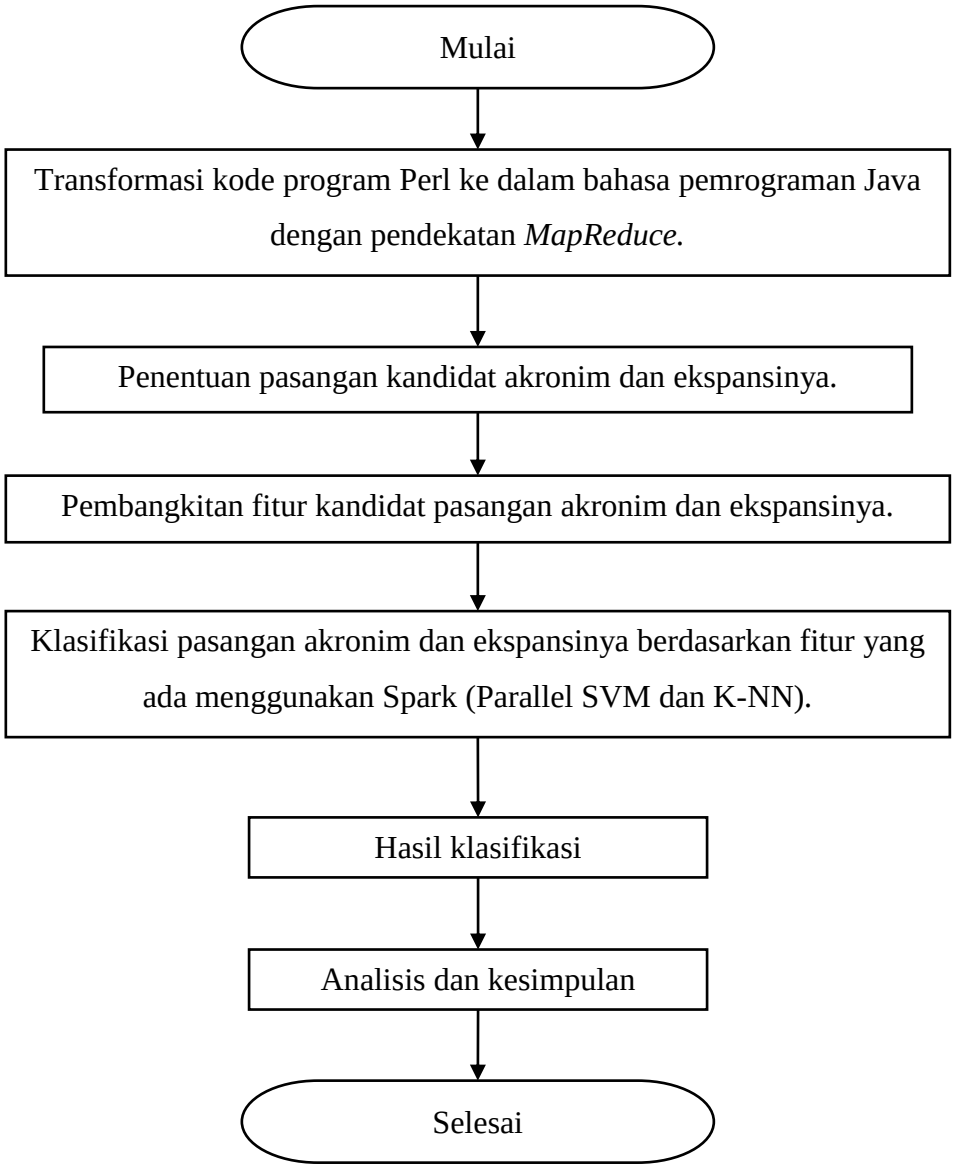
3.3. CARA KERJA

Penelitian ini melanjutkan kajian yang sebelumnya (Wahyudi and Abidin, 2011; Abidin et al., 2018; Abidin et al., 2020) dengan mengimplementasikan penggunaan Hadoop pada bagian-bagian yang belum diimplementasikan. Langkah kerja dari penelitian sebelumnya diperlihatkan pada Gambar 3.2.



Gambar 3.2. Diagram alur kerja penelitian sebelumnya

Bagian kerja yang berwarna biru dari alur kerja pada Gambar 3.1 di atas bermakna proses tersebut telah mengimplementasikan proses paralel atau telah menggunakan Hadoop. Namun, pengimplementasian proses paralel pada bagian-bagian tersebut masih menggunakan bahasa Perl, sedangkan bagian berwarna merah merupakan bagian yang dikerjakan dengan mengimplementasikan proses paralel ke dalam bagian tersebut. Alur kerja dari penelitian yang dikerjakan digambarkan pada Gambar 3.3.



Gambar 3.3. Diagram alur kerja

3.3.1. Ekstraksi kandidat akronim dan ekspansinya

Proses penentuan kandidat akronim pada penelitian yang telah dilakukan oleh Wahyudi and Abidin (2011) dan ditingkatkan performanya pada kajian di tahun 2018 (Abidin et al., 2018), melakukan ekstraksi kandidat akronim dengan menghitung rasio huruf kapital dalam kata. Jika rasio huruf kapital lebih dari 75%, maka kata tersebut termasuk ke dalam kandidat akronim. Selain itu, apabila rasio berada diantara 50-75% dan pada kata tersebut terdapat karakter angka (2-9), maka kata tersebut juga merupakan kandidat akronim. Terakhir, jika rasio kurang dari 75%, maka kata akan

diperiksa di kamus, jika kata tersebut tidak ditemukan di kamus, maka kata tersebut juga termasuk ke dalam kandidat akronim. Selain itu, kata tidak akan dianggap kandidat akronim.

Setelah kandidat akronim didapatkan, kandidat ekspansi dari kandidat-kandidat tersebut ditentukan. Penentuan pasangan ini dilakukan dengan mengambil beberapa kata di depan dan di belakang kandidat akronim, dengan ketentuan sebagai berikut:

$$n = \min(\sum K, \sum A + 2) \quad \text{..... (3.1)}$$

dimana K merupakan teks yang berada sebelum atau sesudah kandidat akronim, dan A merupakan panjang karakter kandidat akronim. Rumus 3.1 (Wahyudi and Abidin, 2011) tersebut digunakan untuk membatasi jumlah maksimum kata dalam kandidat kepanjangan sebuah akronim. Merujuk pada contoh kalimat "FMIPA fokus pada bidang ilmu sains dan merupakan singkatan dari Fakultas Matematika dan Ilmu Pengetahuan Alam". Maka kata FMIPA akan masuk ke dalam kandidat akronim yang memiliki rasio huruf kapital 100% dengan panjang karakter 5. Jumlah kata yang mengelilingi kata FMIPA ada 15 kata. Sehingga penentuan nilai *n* ditentukan dari n = min(15, 5+2) atau 7.

3.3.2. Pembangunan fitur

Algoritma yang digunakan untuk membangkitkan fitur-fitur pada proses ini didasarkan pada penelitian sebelumnya yang telah dilakukan oleh Wahyudi and Abidin (2011) dengan 5 fitur, yaitu:

1. Fitur pertama merupakan rasio antara jumlah karakter pada kandidat akronim dengan jumlah kata pada kandidat ekspansinya. Fitur ini akan menghasilkan nilai 1 jika jumlah huruf pada kandidat akronim dan jumlah kata pada kandidat ekspansi adalah sama, sedangkan untuk panjang yang tidak sama, fitur ini akan bernilai < 1.
2. Fitur kedua merepresentasikan rasio kata pada ekspansi yang huruf awalnya adalah huruf kapital. Fitur ini akan bernilai 1 jika huruf awal dari setiap kata pada

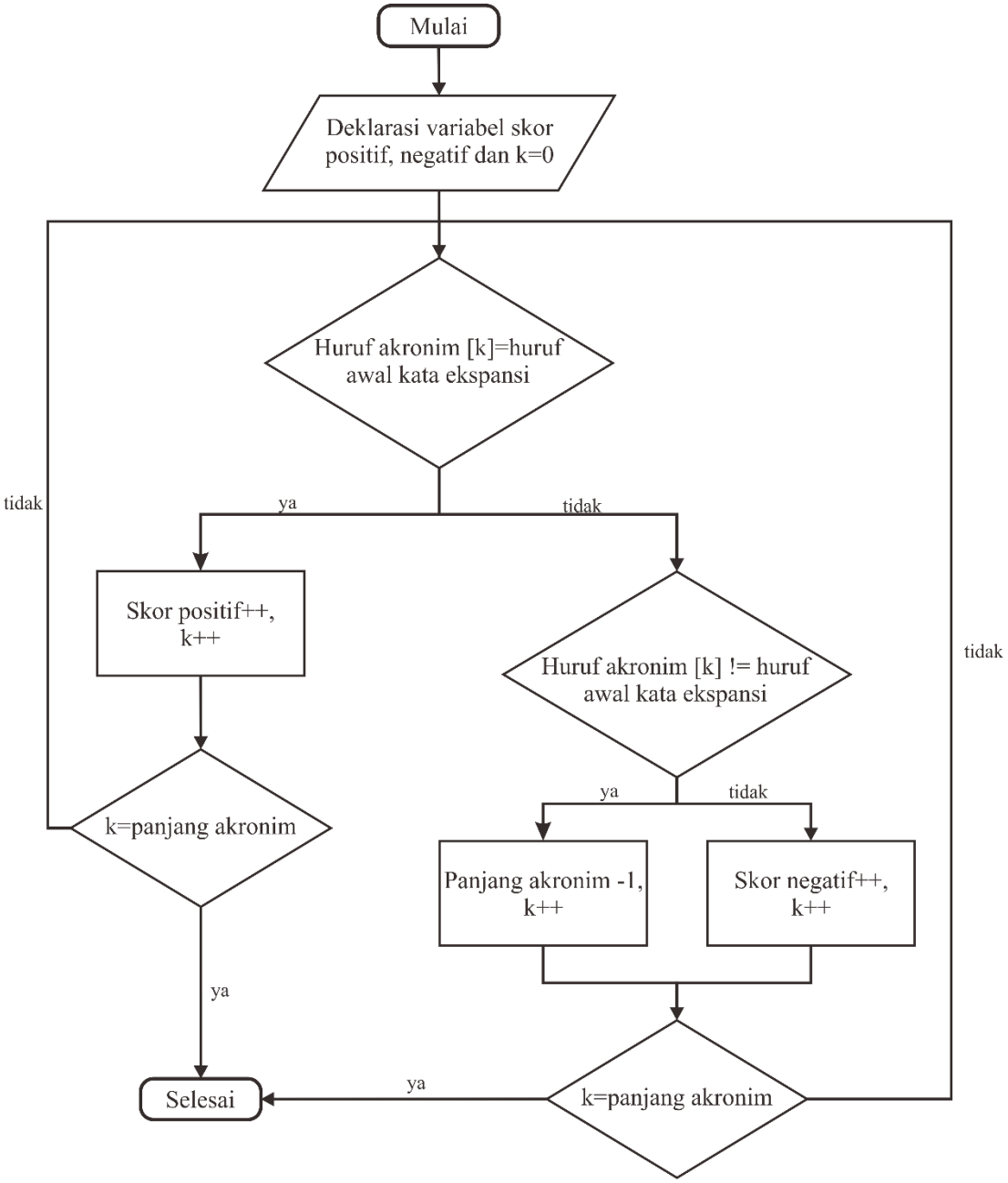
kandidat ekspansi yang tidak termasuk kata sambung atau kata depan merupakan huruf kapital.

3. Fitur ketiga didapatkan dengan menilai keterkaitan antara huruf pada kandidat akronim dan kombinasi huruf pada kata dalam kandidat ekspansi. Algoritma yang digunakan untuk mendapatkan fitur ketiga dapat dilihat pada Gambar 3.4 (Wahyudi and Abidin, 2011).
4. Fitur keempat merepresentasikan hubungan antara huruf awal kandidat akronim dengan huruf awal kata pertama dalam kandidat ekspansi dan huruf terakhir kandidat akronim dengan huruf awal pada kata terakhir kandidat ekspansinya. Fitur ini menghasilkan nilai 1 untuk kandidat akronim yang huruf awalnya sama dengan huruf awal kata pertama pada kandidat ekspansinya dan huruf terakhir akronim tersebut sama dengan huruf awal kata terakhir kandidat ekspansinya.
5. Fitur kelima merupakan nilai dari 1 dikurangi dengan perbandingan antara jumlah kata pada kandidat ekspansi yang merupakan kata depan atau penghubung dengan jumlah kata ekspansi. Fitur ini akan bernilai rendah jika terdapat banyak kata penghubung atau kata depan pada kandidat ekspansi.

Pada penelitian di tahun 2017, 3 fitur tambahan diperkenalkan untuk memperbaiki kekurangan yang terdapat pada penelitian yang dilakukan pada tahun 2011.

6. Fitur keenam merupakan representasi dari rasio kemunculan karakter akronim dalam ekspansinya.
7. Fitur ketujuh berkaitan dengan fitur keenam. Fitur ini akan memiliki nilai 1 jika fitur keenam = 1 dan 0 jika fitur keenam < 1.
8. Fitur kedelapan dan terakhir merupakan nilai rata-rata dari tujuh fitur sebelumnya.

Hasil dari proses ini adalah fitur dari setiap kandidat akronim dan kandidat pasangan ekspansinya yang nantinya diklasifikasi menggunakan metode paralel SVM dan K-NN.



Gambar 3.4. Diagram alir penentuan fitur ketiga (sumber: Wahyudi and Abidin, 2011)

3.3.3. Klasifikasi

Terdapat dua metode klasifikasi yang digunakan dan dibandingkan pada penelitian ini, yaitu metode *Parallel Support Vector Machine* (SVM) dan *K-Nearest Neighbor* (K-NN). Proses klasifikasi untuk setiap metode dilakukan dengan menggunakan *tool* Apache Spark yang telah berjalan pada Hadoop YAVA. Data yang diproses pada tahap ini adalah fitur-fitur yang telah dihasilkan pada proses sebelumnya.

1. *Parallel SVM*

Pada metode *parallel* SVM, klasifikasi dilakukan dengan menggunakan kernel linear yang telah diimplementasikan pada *tool* Spark. Kemudian hasil klasifikasi dilihat dan dihitung akurasi klasifikasinya. Selain *parallel* SVM, klasifikasi dengan metode SVM pada server tunggal juga dilakukan guna melihat perbandingan antara SVM yang bekerja secara paralel dan SVM yang bekerja pada server tunggal.

2. K-NN

Parameter yang digunakan pada K-NN di penelitian ini adalah nilai K atau jumlah tetangga terdekat. Nilai *k* yang dipilih diantaranya 3, 5, 7, dan 9. Setiap nilai *k* diuji dengan menggunakan metode *K-Fold Cross Validation* menggunakan data latih.

3.3.4. Analisis

Metode analisa yang diterapkan pada penelitian ini adalah analisa kuantitatif. Analisa dilakukan dengan membandingkan hasil perhitungan nilai *F-Measure* serta waktu yang dibutuhkan dari setiap metode dan setiap parameter yang ada dalam mengklasifikasi kandidat pasangan akronim dan ekspansinya. Nilai *F-Measure* dihitung berdasarkan hasil klasifikasi yang dapat dilihat dari *confusion matrix* yang ditunjukkan pada Tabel 3.3 (Han and Kamber, 2006).

Tabel 3.3. *Confusion Matrix*

		Diprediksi sebagai	
		Positif	Negatif
Kelas sebenarnya	Positif	TP	FN
	Negatif	FP	TN

Formula untuk menghitung nilai *F-Measure* (F) adalah sebagai berikut:

$$F=2\times\frac{p\times r}{p+r} \tag{3.2}$$

Dimana *p* adalah *precision* dan *r* adalah *recall* dengan masing-masing dapat dihitung sebagai berikut:

$$p=\frac{TP}{TP+FP} \tag{3.3}$$

$$r=\frac{TP}{TP+FN} \tag{3.4}$$

BAB IV
HASIL DAN PEMBAHASAN

4.1. PERBEDAAN-PERBEDAAN PADA KODE PROGRAM PERL DAN JAVA

Proses transformasi kode program merupakan tahap pertama pada penelitian ini. Tahapan ini mencakup pengubahan kode program algoritma IndoAcro dalam melakukan *preprocessing* dari data artikel bersih hingga menjadi fitur-fitur yang telah dijelaskan pada **BAB III** serta pengujian untuk membuktikan apakah tahapan ini telah dilakukan dengan benar. Pengujian dilakukan dengan membandingkan kandidat serta fitur yang dihasilkan oleh kedua kode program (Perl dan Java).

Pengubahan kode program dimulai dengan membaca setiap baris dari kode program algoritma IndoAcro serta memahami maksud dari setiap baris tersebut. Hal tersebut bertujuan agar pada saat program Java yang baru dibuat, tidak mengalami kekeliruan dan perbedaan antara maksud dari baris-baris program. Hal ini disebabkan karena adanya beberapa perbedaan pada pemrograman Perl dan Java dalam memproses sesuatu. Beberapa perbedaan yang signifikan antara Perl dan Java diantaranya pada saat proses *Regular Expression* (*regex*) berlangsung. *Regex* merupakan templat yang menunjukkan apakah *string* yang diberikan cocok atau tidak dengan pola yang ada (Schwartz et al., 2008).

4.1.1. Perbedaan pada *regex match*

Proses *regex match* (menemukan *string* yang cocok dengan pola yang telah didefinisikan) pada perl dapat dilakukan hanya dengan satu baris kode program. Contoh *regex match* yang digunakan pada pemrograman Perl algoritma IndoAcro dapat dilihat pada Gambar 4.1.

```
$results[$m] =~ m/^(.*?),(.*?),(.*?)$/;
```

Gambar 4.1. Kode program bahasa Perl dalam melakukan *regex match*

Sedangkan pemrograman Java, beberapa langkah harus dilakukan untuk dapat melakukan proses *regex match*. Sebuah *method* baru kemudian didefinisikan untuk dapat melakukan proses *regex match* tanpa harus menuliskan beberapa baris kode program tersebut secara terus-menerus dengan tujuan menghemat baris dan membuat kode program lebih mudah diperiksa. Potongan kode program dari *method* yang dibuat dapat dilihat pada Gambar 4.2. *Method* tersebut akan mengembalikan nilai *true* jika pola yang dimasukkan ditemukan dalam *string* atau *false* jika pola yang dimasukkan tidak ditemukan. Parameter-parameter yang dibutuhkan oleh *method* tersebut adalah :

- *String* yang akan dicari polanya : *String*.
- Pola yang akan dicari : *String*.
- Modifier yang digunakan : *integer* (opsional).

```
private static boolean m(String in, String regex, int... modifier) {
        Pattern p = null;
        if (modifier.length > 0) {
            p = Pattern.compile(regex, modifier[0]);
        } else {
            p = Pattern.compile(regex);
        }

        Matcher m = p.matcher(in);
        return m.find();
}
```

Gambar 4.2. Kode program *method regex match* yang telah dibuat

4.1.2. Perbedaan pada *regex substitute*

Perbedaan lainnya antara bahasa Perl dan Java terletak pada *regex substitute*. *Substitute* merupakan *regex* yang dapat dijalankan untuk mengganti hanya bagian tertentu pada string yang cocok dengan pola yang diberikan (Schwartz et al., 2008). *Substitute* juga mengandung proses *match* di dalamnya, namun pada *substitute* bagian yang cocok dengan pola yang diberikan akan diganti dengan *string* baru yang dimasukkan. Bahasa Perl hanya membutuhkan satu baris kode program untuk dapat menjalankan *regex substitute*. Contoh penggunaan regex substitute pada Perl dapat dilihat pada Gambar 4.3

```
$acronym=~ s/^\s+|\s+$//g;
```

Gambar 4.3. Kode program bahasa Perl dalam melakukan *regex substitute*

Sedangkan pada bahasa Java, beberapa baris program harus dituliskan untuk dapat melakukan *regex substitute*. Sebuah *method* baru kemudian didefinisikan untuk melakukan *regex substitute* guna menghemat baris program serta mempermudah proses pemeriksaan kode jika terjadi kesalahan. Potongan kode program dari *method substitute* dapat dilihat pada Gambar 4.4. *Method* tersebut membutuhkan beberapa parameter untuk dapat dijalankan, diantaranya:

- *String* yang akan dicari polanya : *String*.
- Pola yang akan dicari : *String*.
- *String* yang akan menggantikan pola yang ditemukan : *String.*
- Modifier yang digunakan : *integer* (opsional).

```
private static String substitute(String in, String replaceWhat, String
replaceWith, int... modifier) {
        Pattern p = null;

        if (modifier.length > 0) {
            p = Pattern.compile(replaceWhat, modifier[0]);
        } else {
            p = Pattern.compile(replaceWhat);
        }
        Matcher m = p.matcher(in);
        return m.replaceAll(replaceWith);
}
```

Gambar 4.4. Kode program *method regex substitute* yang telah dibuat

4.1.3. Perbedaan saat menggunakan grup pada *regex substitute*

Perbedaan selanjutnya terdapat pada penggunaan grup pada saat melakukan *regex substitute*. Pada bahasa Perl, hanya dibutuhkan satu baris saat *regex substitute* dilakukan menggunakan grup. Grup pada *regex* merupakan sebuah cara untuk dapat menggunakan kembali bagian bagian yang telah ditemukan pada *regex* (Schwartz et al., 2008). Contoh penggunaan grup pada *regex substitute* bahasa Perl dapat dilihat pada Gambar 4.5. Penggunaan grup pada *regex* ditunjukkan oleh tanda kurung (**()**).

```
$prepros=~s/(..)(.+)/$1-$2/i;
```

Gambar 4.5. Contoh pengunaan grup pada regex substitute bahasa Perl

Sedangkan bahasa Java, beberapa langkah harus dilakukan untuk dapat melakukan proses *substitute* menggunakan grup pada *regex*. *String* harus melewati proses pencarian pola sebelum akhirnya grup yang sesuai dari *string* dapat digunakan. Contoh penggunaan grup pada bahasa Java dapat dilihat pada Gambar 4.6.

```
Pattern p = Pattern.compile("(..)(.+)", Pattern.CASE_INSENSITIVE);
Matcher m = p.matcher(prepros);
if (m.find()) {
      prepros = substitute(prepros, "(..)(.+)",m.group(1)+"-
      "+m.group(2), Pattern.CASE_INSENSITIVE);
}
```

Gambar 4.6. Contoh penggunaan grup *regex* pada bahasa Java

4.1.4. Perbedaan pada saat penentuan *N-Grams*

Selain perbedaan pada proses *regex*, perbedaan lain juga terdapat pada *library* yang digunakan untuk mendapatkan *N-Gram* dari sebuah kalimat. *N-Gram* merupakan elemen-elemen yang secara berurutan muncul pada teks. Elemen-elemen tersebut dapat berupa kata, karakter, maupun elemen-elemen yang muncul tepat setelah elemen lainnya pada teks (Sidorov et al., 2004). Pada bahasa Perl, *library* yang digunakan adalah Text::Ngrams. Cara penggunaan *library* tersebut dapat dilihat pada Gambar 4.7. *Library* tersebut nantinya akan mengembalikan 1 hingga n *gram* kata yang didapat dari kalimat yang dimasukkan, dimana n merupakan jumlah *gram* yang diinginkan. Sebuah fungsi baru kemudian dibuat untuk mengambil 2 hingga n *gram* yang dikembalikan oleh *library* yang digunakan.

```
sub n_gram{
      my ($n,$kata) = @_; #n adalah jumlah gram kata
      my $ngrams = Text::Ngrams->new( windowsize => $n, type => "word" );
      $ngrams->process_text($kata);
      my $result = $ngrams->to_string( orderby=>'none');
      $result=~ s/1-GRAMS.*?2-GRAMS/2-GRAMS/gs;
      $result=~ s/BEGIN\ OUTPUT.*?\n//g;
      $result=~ s/\d-GRAMS.*?\n//g;
      $result=~ s/FIRST\ N-GRAM.*?\-//gs;
      $result=~ s/\n+/\n/gs;
      $result=~ s/GRAM.*?\n//g;
      $result=~ s/END\ OUTPUT.*?\n//g;
      $result=~ s/\d//g;
      $result=~ s/\-.*?\n//g;
      $result=~ s/\t//g;
      $result=~ s/^(\s+)//g;
      $result=~ s/<NUMBER>//g;
      return $result;
}
```

Gambar 4.7. Pengimplementasian *library* Text::Ngrams pada Perl

Sedangkan pada Java, library yang digunakan adalah *library* Ngrams. *Library* tersebut akan mengembalikan `ArrayList<String>` dari n *gram* kalimat yang dimasukkan, dimana n merupakan jumlah *gram* diinginkan. Oleh karena algoritma IndoAcro mengharuskan jumlah *gram* yang dibangkitkan adalah 2 hingga n, maka perulangan harus dibuat untuk menghasilkan *gram-gram* yang diinginkan. Contoh penggunaan *library* tersebut dapat dilihat pada Gambar 4.8.

```
//fungsi n_gram
private static String n_gram(int m_maxlength, String n_kalkiri){
      StringBuilder tampung_gram= new StringBuilder();
      for (int ni=2; ni <= m_maxlength; ni++) {
            ArrayList<String> words =
                Ngrams.sanitiseToWords(n_kalkiri);
            ArrayList<String> ngrams = Ngrams.ngrams(words, ni);
            String[] gas = ngrams.toString().split("(,\\s+)");
            for (String ga : gas) {
                  tampung_gram.insert(0, ga + ",");
            }
      }
      tampung_gram                                                    =
          Optional.ofNullable(substitute(tampung_gram.toString(),
          "(\\[|])", "")).map(StringBuilder::new).orElse(null);
      return (tampung_gram == null ? "" : tampung_gram.toString());
}
```

Gambar 4.8. Pengimplementasian *library* Ngrams pada Java

Setelah semua bagian pada kode program algoritma IndoAcro dalam melakukan *preprocessing* selesai di transformasikan ke bahasa Java, kode kemudian diuji untuk melihat apakah transformasi berhasil. Pengujian dilakukan pada server tunggal dengan memproses 39 data artikel berita yang telah dibersihkan sehingga mengandung hanya URL, judul, serta isi dari artikel berita. Kandidat yang dihasilkan serta hasil klasifikasi yang didapat dari kedua kode program kemudian dibandingkan untuk melihat apakah kode hasil transformasi berhasil bekerja dengan baik. Klasifikasi pada pengujian ini dilakukan dengan menggunakan model SVM pada mesin tunggal yang telah ada. Hasil dari pengujian kode program hasil transformasi dapat dilihat pada Tabel 4.1. Hasil menunjukkan bahwa kode program yang telah ditransformasi telah berhasil menghasilkan data kandidat dengan jumlah yang sama (29.372 kandidat) serta tidak mengurangi nilai TP yang ada. Waktu yang dibutuhkan oleh Java dalam

menyelesaikan *preprocess* untuk 39 data artikel adalah 45 detik, sementara Perl adalah 75 detik.

Tabel 4.1. Hasil klasifikasi kandidat yang dihasilkan kode program Perl dan Java (sumber: Abidin et al., 2019)

Bahasa	**Jumlah Kandidat**	**Matriks evaluasi**						**Waktu (detik)**
		TP	**FP**	**FN**	**P**	**R**	**F1**	
Perl	29.372	81	12	2	0,871	0,9759	0,9205	78
Java	29.372	81	10	2	0,8901	0,9759	0,9301	45

Pendekatan *pruning* juga dilakukan pada bagian ini. *Pruning* dilakukan dengan mengurangi n *gram* yang dibangkitkan dari kalimat. Pada algoritma asli IndoAcro, jumlah *gram* yang diambil adalah 2 hingga n buah *gram* dari kalimat. Sehingga apabila terdapat kalimat "FMIPA atau Fakultas Matematika dan Ilmu Pengetahuan alam, merupakan salah satu Fakultas di Universitas Syiah Kuala." dengan asumsi bahwa kandidat akronim dari kalimat tersebut adalah FMIPA, maka jumlah *gram* yang diambil sesuai formula 3.1 adalah:

$$n = \min\big(15, (5 + 2)\big)$$
$$n = \min(15,7)$$
$$n = 7$$

Artinya, jumlah *gram* yang akan dibangkitkan adalah 2 hingga 7 *gram*. Sehingga, jumlah pasangan kandidat yang didapatkan dari kalimat tersebut adalah 6 kandidat.

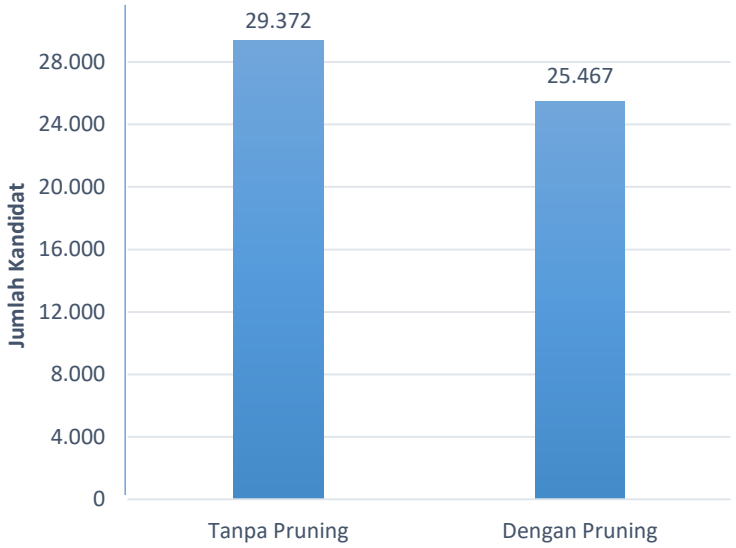
Ide dari *pruning* yang diimplementasikan pada bagian ini adalah mengurangi jumlah *gram* yang dibangkitkan dari kalimat yang mengandung kandidat akronim. Jumlah *gram* yang dibangkitkan tidak lagi dimulai dari 2 *gram*, namun dari jumlah karakter atau suku kata pada kandidat akronim dikurangi 1. Merujuk dari contoh di atas, maka jumlah *gram* yang diambil adalah 4 hingga 7 gram. Artinya, jumlah pasangan kandidat yang dibangkitkan dari kalimat di atas adalah 4 kandidat.

Pengujian pendekatan *pruning* yang diterapkan pada bagian pembangkitan kandidat dilakukan dengan menggunakan 39 data artikel yang sama dengan data pengujian jumlah kandidat sebelumnya pada bagian **4.1**. *Pruning* diimplementasikan

pada kode baru dengan Bahasa Java. Tabel perbandingan antara kandidat yang dihasilkan kode *pruning* dan tanpa *pruning* dapat dilihat pada Tabel 4.2. Hasil menunjukkan jumlah kandidat dapat dihemat sekitar 13%, yaitu dari 29.372 kandidat menjadi 25.467 kandidat (Abdin et al., 2019). Kandidat-kandidat yang didapatkan kemudian diklasifikasi menggunakan model SVM pada mesin tunggal yang telah ada untuk melihat dan membuktikan bahwa kandidat yang dihilangkan adalah kandidat-kandidat yang salah. Hasil klasifikasi menunjukkan bahwa kandidat yang dihemat adalah kandidat-kandidat yang masuk ke kategori TN (kandidat yang benar-benar salah).

Tabel 4.2. Efisiensi kode dengan *pruning* dan tanpa *pruning*
(sumber: Abidin et al., 2019)

Pruning	**Jumlah Kandidat**	**Matrix Evaluasi**				**Waktu (detik)**
		TP	**FP**	**FN**	**TN**	
Tanpa *pruning*	29.372	81	10	2	29.279	45
Dengan *pruning*	25.467	81	10	2	25.374	33



Gambar 4.9. Grafik perbandingan jumlah kandidat kode tanpa *pruning* dan dengan *pruning*

4.2. ANALISIS PERBANDINGAN WAKTU KOMPUTASI PERL DAN JAVA PADA HADOOP MAPREDUCE

Kode program Java yang telah selesai ditransformasikan dan telah dipastikan menghasilkan jumlah pasangan kandidat yang sama dengan kode program Perl kemudian dijalankan pada Hadoop dengan memanfaatkan MapReduce. Kedua kode program dijalankan pada Hadoop dengan menggunakan jumlah server yang berbeda-beda untuk setiap pengujian. Jumlah server yang digunakan pada pengujian waktu ini adalah 13, 26 dan 39 server. Pengujian dilakukan secara bergantian agar waktu yang dibutuhkan oleh satu pengujian tidak terpengaruh oleh pengujian lainnya. Selain itu, pengujian juga dilakukan dengan menggunakan jumlah data yang berbeda. Data artikel berjumlah 100.000 dan 200.000 digunakan untuk melihat perbedaan waktu yang dibutuhkan oleh kedua kode program untuk menyelesaikan *preprocess* dalam mengekstraksi data-data fitur dari data artikel bersih yang telah dimiliki. Jumlah pasangan kandidat yang dibangkitkan dari 100.000 data artikel adalah 51 juta pasangan, sedangkan pasangan kandidat yang dibangkitkan dari 200.000 data artikel adalah 119 juta pasang kandidat.

4.2.1. Pengujian dengan 100.000 Data Artikel

Kedua kode program dijalankan menggunakan Hadoop dengan memanfaatkan paradigma MapReduce. Perbedaan yang terdapat pada kedua program saat dijalankan adalah Java langsung mengimplementasikan MapReduce pada kodenya sehingga kode program dapat langsung dijalankan. Sedangkan Perl harus menggunakan *library* tambahan, yaitu Hadoop Streaming yang dapat menjadi jembatan antara MapReduce yang ada di Hadoop dengan kode program bukan Java yang dimiliki.

Hasil pengujian menunjukkan untuk 100.000 data artikel, pada 13, 26 dan 39 server, Perl dengan Hadoop Streaming jauh lebih cepat dibandingkan dengan Java yang mengimplementasikan MapReduce. Waktu yang dibutuhkan kode program Perl dalam memproses 100.000 data artikel dengan menggunakan 13 server adalah 2 jam 59 menit 39 detik, sedangkan Java membutuhkan waktu 33 jam 59 menit 22 detik. Menggunakan 39 server, waktu yang dibutuhkan Perl adalah 1 jam 8 menit 15 detik, sedangkan Java membutuhkan waktu 4 jam 44 menit 25 detik. Detail waktu yang

dibutuhkan dan grafik waktu dari kedua kode program dalam memproses 100.000 data artikel dapat dilihat pada Tabel 4.3 dan Gambar 4.10.

Tabel 4.3. Perbandingan waktu yang dibutuhkan Java MapReduce dengan Perl Hadoop Streaming dalam memproses 100.000 data artikel

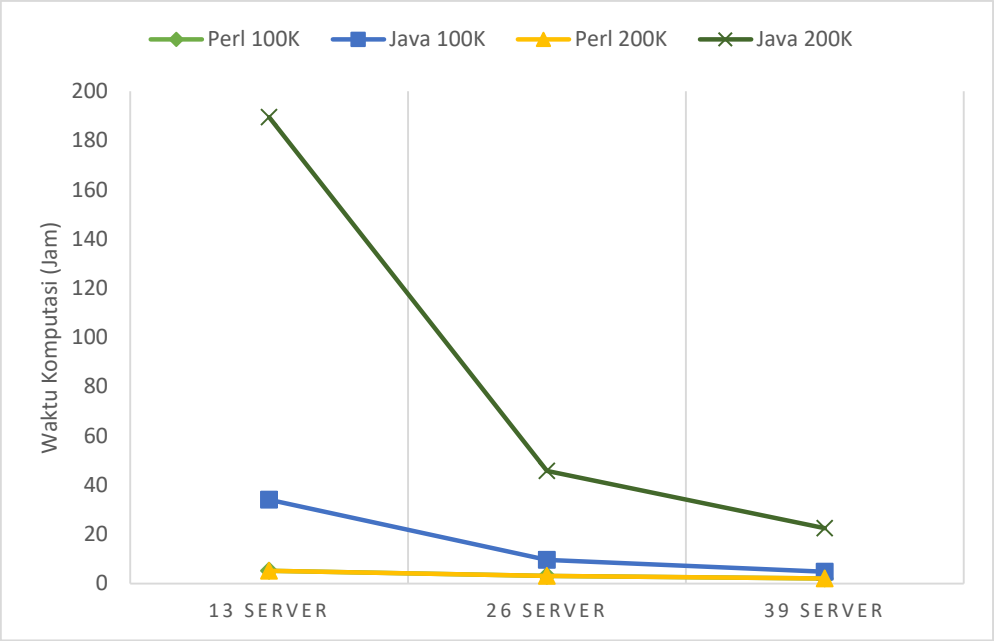
Jumlah Server	**Waktu yang dibutuhkan untuk menyelesaikan *preprocessing***	
	Perl	**Java**
13	2 jam 59 menit 39 detik	33 jam 59 menit 22 detik
26	1 jam 32 menit 2 detik	9 jam 37 menit 27 detik
39	1 jam 8 menit 15 detik	4 jam 44 menit 25 detik

4.2.2. Pengujian dengan 200.000 Data Artikel

Pengujian dilanjutkan dengan menggunakan 200.000 buah data artikel berita yang sudah dibersihkan. Pengujian ini juga dilakukan 3 kali dengan jumlah server yang berbeda-beda, yaitu 13, 26 dan 39 server. Hasil menunjukkan bahwa Perl masih unggul dalam memproses 200.000 artikel berita menjadi fitur-fitur dibandingkan Java. Waktu yang dibutuhkan Perl untuk memproses 200.000 artikel berita dengan menggunakan 13 server adalah 2 jam 59 menit 39 detik, sedangkan Java membutuhkan 5 hari 17 jam 19 menit dan 2 detik untuk dapat memproses 200.000 data artikel. Detail waktu dan grafik dapat dilihat pada Tabel 4.4 dan Gambar 4.10.

Tabel 4.4. Perbandingan waktu yang dibutuhkan Java MapReduce dengan Perl Hadoop Streaming dalam memproses 200.000 data artikel

Jumlah Server	**Waktu yang dibutuhkan untuk menyelesaikan *preprocessing***	
	Perl	**Java**
13	5 jam 14 menit 32 detik	189 jam 30 menit 39 detik
26	3 jam 7 menit 54 detik	45 jam 44 menit 41 detik
39	2 jam 2 menit 7 detik	22 jam 28 menit 30 detik



Gambar 4.10. Grafik perbedaan waktu yang dibutuhkan Perl dan Java dalam memproses 200.000 dan 100.000 data artikel

Berdasarkan 6 pengujian yang telah dilakukan, dapat diamati bahwa Perl membutuhkan waktu jauh lebih sedikit dibandingkan dengan Java dalam memproses data dalam jumlah besar. Lebih jauh, terdapat perbedaan pola penurunan waktu pada kedua kode program. Pola tersebut dapat dilihat pada pengujian dengan 100.000 dan 200.000 data pada 13, 26, dan 39 server.

Sebagai contoh, untuk memproses 100.000 data artikel pada 13 server, Java membutuhkan waktu 33 jam 59 menit, kemudian waktu yang dibutuhkan berkurang sebesar 24 jam 22 menit pada 26 server atau sebesar sebesar 71,68%. Selanjutnya apabila penambahan server digandakan menjadi 39 server, penurunan waktu yang terjadi adalah adalah 86,07%. Sedangkan pada Perl, untuk memproses 100.000 data pada 13 server dibutuhkan waktu 2 jam 59 menit, kemudian waktu yang berkurang sebesar 1 jam 27 menit pada 26 server atau sebesar 48,77%. Selanjutnya penurunan waktu yang dibutuhkan apabila 39 server digunakan adalah 62,01%. Hal ini sesuai dengan pernyataan yang ditulis oleh Pellakuri & Rao (2014) yaitu bertambahnya jumlah server, akan menambah jumlah proses *map* ataupun *reduce* yang dijalankan sehingga mengurangi waktu yang dibutuhkan untuk memproses data yang dibutuhkan.

4.3. ANALISIS PERFORMA METODE PARALEL SVM DAN K-NN

Metode klasifikasi yang digunakan pada penelitian ini adalah Paralel SVM dan K-NN yang dijalankan menggunakan SPARK. Model dari kedua metode klasifikasi dilatih menggunakan 5.000 pasangan akronim dan ekspansi yang telah dijelaskan pada **BAB III**. Pembangunan model dilakukan dengan menggunakan *K-Fold Validation* untuk mendapatkan model terbaik dari 10 iterasi. Setiap iterasi pada *K-Fold Validiation* menggunakan 4.500 data sebagai data latih dan 500 data sebagai data uji. Hasil dari proses pelatihan kedua metode klasifikasi menggunakan *K-Fold Validation* menunjukkan bahwa K-NN dengan K=3 memiliki *F-Measure* paling tinggi dengan 99,22%. Detail hasil pelatihan Paralel SVM dan K-NN menggunakan 5.000 data dapat dilihat pada Tabel 4.5.

Tabel 4.5. Hasil nilai F-Measure terbaik dari metode Paralel SVM dan K-NN selama pelatihan menggunakan *K-Fold Cross Validation*

No	Metode	Iterasi Ke	*F-Measure*
1	K-NN K=3	5	99,22%
2	K-NN K=5	7	98,74%
3	K-NN K=7	7	98,74%
4	K-NN K=9	7	98,74%
5	Paralel SVM	9	99,16%

Model dari iterasi terbaik yang didapat menggunakan *K-Fold Validation* dengan K=10 untuk kedua metode kemudian disimpan. Model tersebut selanjutnya diuji menggunakan 2.000 data uji yang telah dijelaskan pada BAB III. Berdasarkan pengujian, didapatkan bahwa K-NN dengan K=9 memiliki *F-Measure* yang sedikit lebih unggul dibandingkan dengan K-NN dengan K=3, 5, 7 serta paralel SVM. Namun, nilai *F-Measure* tertinggi dipegang oleh model SVM Light dari penelitian sebelumnya dengan angka 97,93%. Detail dari akurasi yang didapatkan dari pengujian menggunakan 2.000 data uji dapat dilihat pada Tabel 4.6.

Tabel 4.6. Hasil pengujian model menggunakan 2,000 data uji

Metode	***Precision***	***Recall***	***F-Measure***
SVM Light (*Single*)	99,08%	96,80%	97,93%
Paralel SVM	99,15%	93,20%	96,08%
K-NN K=3	99,78%	92,00%	95,73%
K-NN K=5	99,68%	93,40%	96,44%
K-NN K=7	99,68%	93,80%	96,65%
K-NN K=9	99,68%	94,10%	96,81%

Setelah model terbaik dari kedua metode didapatkan, proses kemudian dilanjutkan dengan melakukan klasifikasi terhadap kandidat-kandidat pasangan akronim dan ekspansi yang didapat dari hasil *MapReduce* pada tahapan 4.2.1 dan 4.2.2. Metode paralel SVM dan K-NN dengan K=9 digunakan untuk mengklasifikasi kandidat pasangan akronim dan ekspansinya yang dihasilkan dari 100.000 dan 200.000 data artikel menggunakan pendekatan *pruning* dan tanpa *pruning*. Analisa juga dilakukan terhadap jumlah server yang berbeda dalam melakukan klasifikasi menggunakan kedua metode tadi. Jumlah server yang dibandingkan kecepatan waktunya dalam mengklasifikasi pasangan-pasangan kandidat akronim dan ekspansinya adalah 13, 26, dan 39 server. Hasil menunjukkan bahwa paralel SVM jauh mengungguli K-NN dalam melakukan klasifikasi kandidat-kandidat yang dimiliki. Waktu yang dibutuhkan SVM untuk mengklasifikasi pasang kandidat yang dihasilkan oleh 200.000 artikel tanpa menggunakan pendekatan *pruning* serta menggunakan 39 server adalah adalah 60,07 detik atau sekitar 1 menit. Sedangkan K-NN dengan K=9 menggunakan parameter yang sama membutuhkan waktu 2,3 jam. Selain itu paralel SVM juga mengungguli SVM dengan server tunggal dalam mengklasifikasi data yang telah disebutkan sebelumnya. Namun, SVM dengan server tunggal membutuhkan waktu yang lebih singkat dalam mengklasifikasi kandidat akronim dan kepanjangannya dibandingkan dengan metode K-NN menggunakan Spark. Detail waktu yang dibutuhkan oleh paralel SVM, K-NN dengan K=9 dan SVM dengan server tunggal dapat dilihat pada Tabel 4.7, Tabel 4.8 dan Tabel 4.9. Namun, penambahan jumlah server pada K-NN tidak selalu mengurangi waktu komputasi. Waktu yang dibutuhkan justru bertambah apabila jumlah server ditambahkan. Grafik perbedaan waktu untuk metode Paralel SVM dan K-NN pada Spark dapat dilihat pada Gambar 4.11 dan 4.12.

Tabel 4.7. Detail waktu yang dibutuhkan SVM dalam mengklasifikasi kandidat pasangan akronim dan ekspansinya

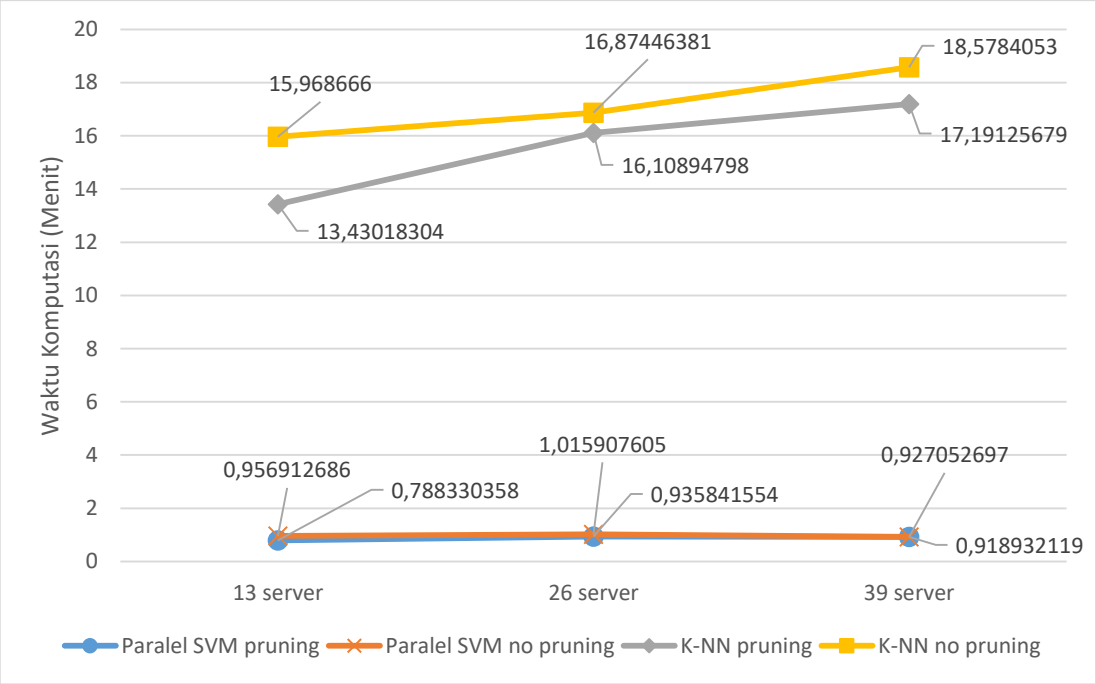
Jumlah Server	**Waktu Klasifikasi (detik)**			
	100.000		**200.000**	
	Tanpa *Pruning*	***Pruning***	**Tanpa *Pruning***	***Pruning***
13	48,46	46,39	85,21	82,35
26	48,46	46,39	79,66	68,93
39	50,47	49,58	77,90	75,61

Tabel 4.8. Detail waktu yang dibutuhkan K-NN dengan K=9 dalam mengklasifikasi kandidat pasangan akronim dan ekspansinya

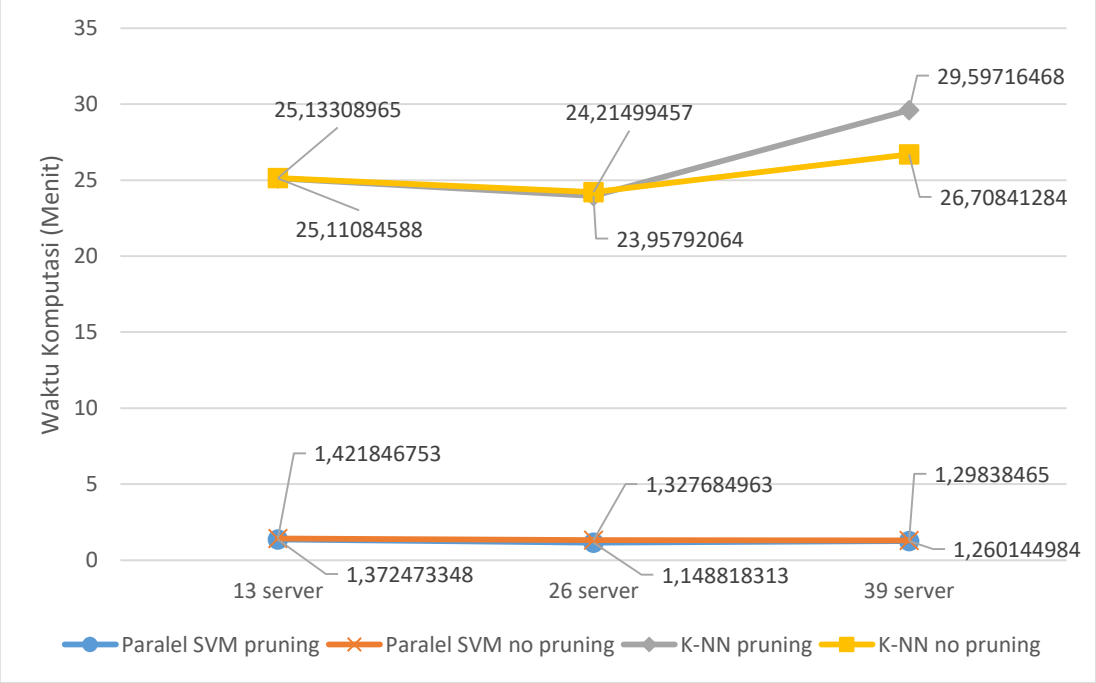
Jumlah Server	**Waktu Klasifikasi (detik)**			
	100.000		**200.000**	
	Tanpa *Pruning*	***Pruning***	**Tanpa *Pruning***	***Pruning***
13	958,12	805,81	1.507,99	1.506,65
26	1.012,46	966,53	1.452,90	1.437,48
39	1.114,70	1.031,47	1.602,50	1.775,83

Tabel 4.9. Detail waktu yang dibutuhkan SVM Light (server tunggal) dalam mengklasifikasi kandidat pasangan akronim dan ekspansinya

Jumlah Kandidat	**Sumber Kandidat**	**Waktu Klasifikasi (detik)**
51.784.649	100.000 tanpa *pruning*	1.490
45.330.209	100.000 *pruning*	1.264
119.645.044	200.000 tanpa *pruning*	3.258
107.360.576	200.000 *pruning*	2.959

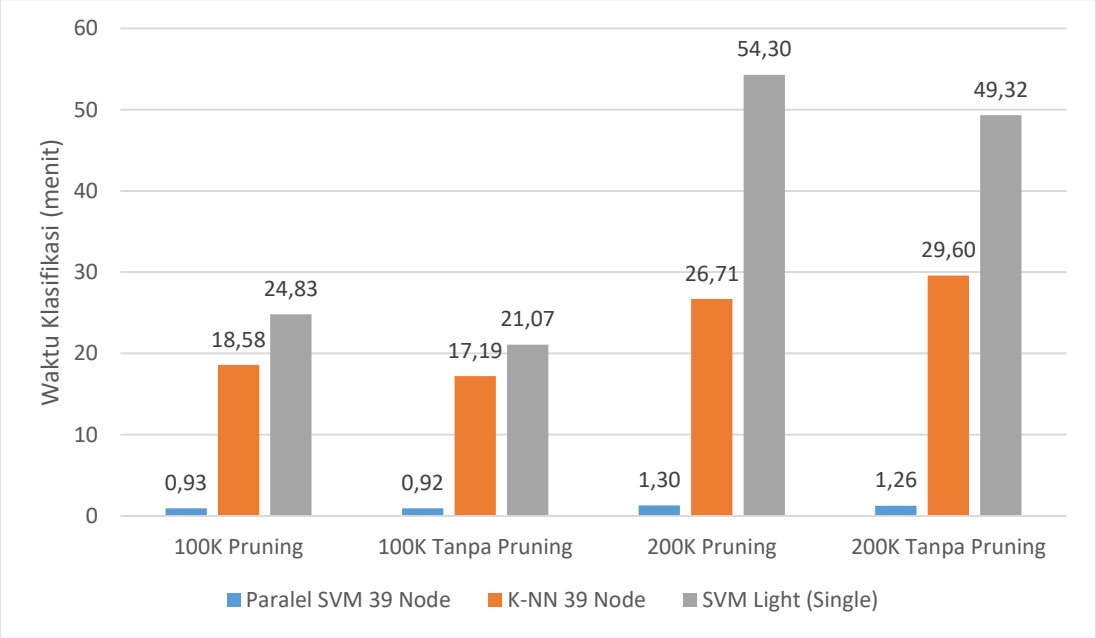


Gambar 4.11. Performansi Paralel SVM dan K-NN dalam mengklasifikasi kandidat yang dihasilkan dari 100.000 artikel menggunakan pendekatan *pruning* dan tanpa *pruning*



Gambar 4.12. Performansi Paralel SVM dan K-NN dalam mengklasifikasi kandidat yang dihasilkan dari 200.000 artikel menggunakan pendekatan *pruning* dan tanpa *pruning*

Berdasarkan proses klasifikasi yang telah dilakukan menggunakan metode paralel SVM dan K-NN menggunakan Spark, dapat diamati bahwa waktu yang dibutuhkan oleh metode SVM jauh lebih sedikit dibandingkan dengan K-NN pada Spark. Namun, K-NN pada Spark masih mengungguli SVM Light pada server tunggal dalam melakukan klasifikasi. Grafik berbedaan waktu klasifikasi setiap metode ditunjukkan pada gambar 4.13.



Gambar 4.13. Performansi metode Paralel SVM, K-NN dan SVM Light menggunakan 100.000 dan 200.000 data dengan dan tanpa pendekatan *pruning*

BAB V
KESIMPULAN DAN SARAN

5.1. KESIMPULAN

Berdasarkan penelitian dan analisis dari percobaan-percobaan yang telah dilakukan, dapat diambil kesimpulan bahwa:

1. Implementasikan pendekatan *pruning* dalam melakukan pembangkitan pasangan kandidat akronim dan ekspansinya pada tahap *preprocessing* algoritma IndoAcro telah berhasil dilakukan. Efisiensi pendekatan *pruning* ini mengurangi jumlah pasangan kandidat akronim sebesar 13% tanpa menghilangkan pasangan akronim yang benar sesuai dengan hasil temuan pada Abidin *et al,*. 2019.
2. Bahasa pemrograman Perl dengan menggunakan *library* Hadoop Streaming dapat mengungguli bahasa pemrograman Java secara signifikan dalam melakukan *preprocessing* pada algoritma IndoAcro.
3. Penambahan server dapat mengurangi waktu komputasi pada tahap *preprocessing* baik menggunakan bahasa pemrograman Perl maupun Java.
4. Perbedaan waktu komputasi pada tahap *preprocessing* yang sangat signifikan terlihat pada bahasa pemrograman Java untuk memproses 200.000 data. Waktu yang dibutuhkan berkurang drastis sebesar 88,14% dari 187,5 jam menggunakan 13 server ke 22,47 jam menggunakan 39 server.
5. Metode K-NN pada Spark dengan nilai K=9 memiliki akurasi tertinggi dibandingkan dengan nilai K=3, K=5 atau K=7 dan Paralel SVM selama tahap pengujian menggunakan 2.000 data uji. Namun, SVM Light pada server tunggal masih mengungguli kedua metode paralel selama tahap pengujian.
6. Metode Paralel SVM dapat mengungguli K-NN pada Spark dengan cukup signifikan dalam mengklasifikasi pasangan kandidat akronim dan ekspansinya baik yang berasal dari 100.000 artikel maupun 200.000 artikel.
7. Penambahan server pada klasifikasi menggunakan metode paralel SVM atau K-NN tidak selalu dapat mengurangi waktu yang dibutuhkan untuk melakukan klasifikasi.

5.2. SARAN

Saran penelitian selanjutnya adalah agar implementasi metode klasifikasi Paralel SVM dapat dilakukan dengan menggunakan kernel selain Linear serta pembobotan pada metode K-NN menggunakan *library* MLlib pada Apache Spark. Perbandingan kode program dalam melakukan *preprocess* dengan Bahasa Pemrograman Python juga dapat dilakukan. Penggunaan jumlah server yang lebih bervariasi juga dapat dilakukan guna melihat tren yang ada apabila variasi jumlah server lebih banyak.

DAFTAR KEPUSTAKAAN

Abidin, T. F., Adriman, R., & Ferdhiana, R. (2018). Performance Analysis of Machine Learning Algorithm for Extracting Acronyms and Expansions on Large Number of Web Pages Using Apache Hadoop Parallel Processing. *In Press of Proc. ICITISEE*. Yogyakarta: IEEE.

Abidin, T. F., Ferdhiana, R., Syaputra, D., & Putera, T. W. (2019). A Distributed Approach to Determine Acronym and Expansion Pairs using Apache Spark. *International Conference on Cybernetics and Computational Intelligence*. Banda Aceh: IEEE.

Abidin, T. F., Mahazir, A., Subianto, M., Munadi, K., & Ferdhiana, R. (2020). Recognizing Indonesian Acronym and Expansion Pairs with Supervised Learning and MapReduce. *Information 2020, 11(4)*, 210-212.

Ali, Z. H., Ali, H. A., & Badawy, M. M. (2015). Internet of Things (IoT): Definitions, Challenges, and Recent Research. *International Journal of Computer Applications*, 37-47.

Apache Hadoop Software Foundation. (2018). *Apache Hadoop*. Retrieved from Apache Hadoop: https://hadoop.apache.org/

Apache Software Foundation. (2018). Diambil kembali dari Apache Spark: https://spark.apache.org/

Bhosale, H. S., & Gadekar, D. P. (2014). A Review Paper on Big Data and Hadoop. *International Journal of Scientific and Research Publications*, 1-7.

Cecchinel, C., Jimenez, M., Mosser, S., & Riveill, M. (2014). An Architecture to Support the Collection of Big Data. *SERVICES '14 Proceedings of the 2014 IEEE World Congress on Services* (hal. 442-449). Anchorage: IEEE.

Chang, E. Y., Zhu, K., Wang, H., & Bai, H. (2007). PSVM: Parallelizing Support Vector Machines. *Advances in Neural Information Processing Systems 20* (hal. 1-8). Vancouver: Curran Associates, Inc.

Chen, M., Mao, S., & Liu, Y. (2014). Big Data: A Survey,. *Mobile Netwoks and Aplications, 19(2)*, 171-209.

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA*, 137-150.

Ghatge, D. D. (2016). Review: Apache Spark and Big Data Analytics for Solving Real World Problems. *International Journal of Computer Science Trends and Technology*, 301-304.

Gou, J., Du, L., Zhang, Y., & Xiong, T. (2012). A New Distance-weighted k-nearest Neighbor Classifier. *Journal of Information & Computational Science* , 1429–1436.

Graf, H. P., Cosatto, E., Bottou, L., Durdanovic, I., & Vapnik, V. (2004). Parallel Support Vector Machines: The Cascade SVM. *Neural Information Processing Systems 17* (hal. 521-528). Vancouver: The MIT Press.

Greeshma, L., & Pradeepini, G. (2016). Big Data Analytics with Apache Hadoop MapReduce Framework. *Indian Journal of Science and Technology*, 1-5.

Gupta, P., Kumar, P., & Gopal, G. (2015). Sentiment Analysis on Hadoop with Hadoop Streaming. *International Journal of Computer Applications, 121(11)*, 4-8.

Han, J., & Kamber, M. (2006). *Data Mining: Concepts and Techniques 2nd Edition.* San Francisco: Morgan Kaufmann.

Hechenbichler, K., & Schliep, K. (2005). Weighted k-Nearest-Neighbor Techniques and Ordinal Classification. *Discussion paper // Sonderforschungsbereich 386 der Ludwig-Maximilians-Universität München*, 1-16.

Imandoust, S. B., & Bolandraftar, M. (2013). Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events: Theoretical Background. *S B Imandoust et al. Int. Journal of Engineering Research and Applications* , 605-610.

Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. *10th European Conference on Machine Learning* (hal. 137-142). Chemnitz: Springer-Verlag London.

Krishnalal, G., Rengarajan, S. B., & Srinivasagan, K. G. (2010). A New Text Mining Approach Based on HMM-SVM for Web. *International Journal of Computer Applications*, 98-104.

Paul, A. (2013). Support Vector Machine in Apache Spark. *International Journal of Advanced Research, 4(8)*, 76-80.

Pellakuri, V., & Rao, D. (2014). Hadoop Mapreduce Framework in Big Data Analytics. *International Journal of Computer Trends and Technology Vol. 3*, 115-119.

Phips, A., Larry, H., & Geert, D. S. (1996). *Clustering and Classification.* Singapore: World Scientific.

Pinto, V. F., Kini, S., & Dsouza, I. M. (2017). A Review Document on Apache Spark for Big Data Analytics with Case Studies. *International Journal of Computer Science Trends and Technology*, 70-73.

Pothuganti, A. (2015). Big Data Analytics: Hadoop-Map Reduce & NoSQL Databases. *International Journal of Computer Science and Information Technologies*, 522-527.

Priestley, T. (2015, Juli 16). *The 3 Elements The Internet Of Things Needs To Fulfil Real Value*. Retrieved from Forbes: https://www.forbes.com/sites/theopriestley/2015/07/16/the-3-elements-the-internet-of-things-needs-to-fulfil-real-value/#4129497c4005

Rani, C. S., & Rama, B. (2017). MapReduce with Hadoop for Simplified Analysis of Big Data. *International Journal of Advanced Research in Computer Science*, 853-856.

Schwartz, R. L., Phoenix, T., & Foy, B. d. (2008). *Learning Perl.* Sebastopol: O'Reilly Media.

Shafaque, U., Thakare, P. D., Ghonge, M. M., & Sarode, M. V. (2014). Algorithm and Approaches to Handle Big Data. *International Journal of Computer Applications*, 18-22.

Shoro, A. G., & Soomro, T. R. (2015). Big Data Analysis: Apache Spark Perspective. *Global Journal of Computer Science and Technology, 15(1)*, 6-14.

Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., & Chanona-Hernández, L. (2004). Syntactic N-grams as machine learning features for natural language processing. *Expert Systems with Applications, 41(3)*, 853-860.

Tong, S., & Koller, D. (2001). Support Vector Machine Active Learning with Applications to Text Classification. *Journal of Machine Learning Research*, 45-66.

Wahyudi, J., & Abidin, T. (2011). Penentuan Secara Otomatis Akronim dan Ekspansinya. *SNATIKA*, (hal. 114-118). Malang.

Yu, H., Yang, J., & Han, J. (2003). Classifying Large Data Sets Using SVMs with Hierarchical. *KDD*. Washington: ACM.

LAMPIRAN

Lampiran 1. Nilai F-Measure pada proses pelatihan metode SVM menggunakan *K-Fold Cross Validation*

Iterasi	Jumlah Data Latih	Jumlah Data Uji	Total	*F-Measure*
1	4,500	500	5,000	97,29%
2	4,500	500	5,000	98,11%
3	4,500	500	5,000	97,80%
4	4,500	500	5,000	97,69%
5	4,500	500	5,000	98,05%
6	4,500	500	5,000	98,01%
7	4,500	500	5,000	97,90%
8	4,500	500	5,000	97,96%
9	4,500	500	5,000	99,16%
10	4,500	500	5,000	98,60%

Lampiran 2. Hasil akurasi proses pelatihan metode paralel K-NN dengan K=3, K=5, K=7, dan K=9 menggunakan *K-Fold Cross Validation* K=10

1. Proses pelatihan K-NN pada Spark dengan K=3

Iterasi	Jumlah Data Latih	Jumlah Data Uji	Total	*F-Measure*
1	4,500	500	5,000	98,57%
2	4,500	500	5,000	98,74%
3	4,500	500	5,000	97,99%
4	4,500	500	5,000	97,26%
5	4,500	500	5,000	99,22%
6	4,500	500	5,000	98,44%
7	4,500	500	5,000	99,16%
8	4,500	500	5,000	97,55%
9	4,500	500	5,000	98,95%
10	4,500	500	5,000	98,81%

2. Proses pelatihan K-NN pada Spark dengan K=5

Iterasi	Jumlah Data Latih	Jumlah Data Uji	Total	*F-Measure*
1	4,500	500	5,000	98,02%
2	4,500	500	5,000	98,73%
3	4,500	500	5,000	97,99%
4	4,500	500	5,000	97,48%
5	4,500	500	5,000	98,64%
6	4,500	500	5,000	98,22%
7	4,500	500	5,000	98,74%
8	4,500	500	5,000	98,17%
9	4,500	500	5,000	98,74%
10	4,500	500	5,000	98,22%

3. Proses pelatihan K-NN pada Spark dengan K=7

Iterasi	Jumlah Data Latih	Jumlah Data Uji	Total	*F-Measure*
1	4,500	500	5,000	98,02%
2	4,500	500	5,000	98,74%
3	4,500	500	5,000	97,80%
4	4,500	500	5,000	97,06%
5	4,500	500	5,000	98,44%
6	4,500	500	5,000	98,00%
7	4,500	500	5,000	98,74%
8	4,500	500	5,000	97,96%
9	4,500	500	5,000	98,32%
10	4,500	500	5,000	98,41%

4. Proses pelatihan K-NN pada Spark dengan K=9

Iterasi	Jumlah Data Latih	Jumlah Data Uji	Total	*F-Measure*
1	4,500	500	5,000	98,03%
2	4,500	500	5,000	98,74%
3	4,500	500	5,000	97,99%
4	4,500	500	5,000	97,06%
5	4,500	500	5,000	98,44%
6	4,500	500	5,000	98,00%
7	4,500	500	5,000	98,74%
8	4,500	500	5,000	97,96%
9	4,500	500	5,000	98,53%
10	4,500	500	5,000	98,41%

Lampiran 3. Proses dan hasil percobaan *preprocessing* pada Hadoop

1. Proses MapReduce yang dijalankan menggunakan 39 server

```
19/10/16 23:48:48 INFO mapreduce.Job:  map 0% reduce 0%
19/10/16 23:49:55 INFO mapreduce.Job:  map 1% reduce 0%
19/10/16 23:49:56 INFO mapreduce.Job:  map 2% reduce 0%
19/10/16 23:49:57 INFO mapreduce.Job:  map 3% reduce 0%
19/10/16 23:50:56 INFO mapreduce.Job:  map 4% reduce 0%
19/10/16 23:51:29 INFO mapreduce.Job:  map 5% reduce 0%
19/10/16 23:52:26 INFO mapreduce.Job:  map 6% reduce 0%
19/10/16 23:53:27 INFO mapreduce.Job:  map 7% reduce 0%
19/10/16 23:54:27 INFO mapreduce.Job:  map 8% reduce 0%
19/10/16 23:55:27 INFO mapreduce.Job:  map 9% reduce 0%
19/10/16 23:56:26 INFO mapreduce.Job:  map 10% reduce 0%
19/10/16 23:57:27 INFO mapreduce.Job:  map 11% reduce 0%
19/10/16 23:58:26 INFO mapreduce.Job:  map 12% reduce 0%
19/10/16 23:58:59 INFO mapreduce.Job:  map 13% reduce 0%
19/10/16 23:59:59 INFO mapreduce.Job:  map 14% reduce 0%
19/10/17 00:00:58 INFO mapreduce.Job:  map 15% reduce 0%
19/10/17 00:02:00 INFO mapreduce.Job:  map 16% reduce 0%
19/10/17 00:02:58 INFO mapreduce.Job:  map 17% reduce 0%
19/10/17 00:03:58 INFO mapreduce.Job:  map 18% reduce 0%
```

2. Informasi setelah proses MapReduce selesai

```
19/10/17 01:38:42 INFO mapreduce.Job: Job job_1571148728216_0017 completed
 successfully
19/10/17 01:38:42 INFO mapreduce.Job: Counters: 33
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=9229808
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=313855071
                HDFS: Number of bytes written=19918653816
                HDFS: Number of read operations=273
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=78
        Job Counters
                Launched map tasks=39
                Data-local map tasks=31
                Rack-local map tasks=8
                Total time spent by all maps in occupied slots (ms)=432390
```

3. Hasil *preprocess* dengan MapReduce

```
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>Anfaton
i M Bachir Kisworo:+:(0,(8,[0,1,2,3,4,5,6,7],[0.91829583405449,1,0.25,0,1,0.2,0,0.481185119150641]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>orang i
tu adalah Anfatoni:+:(0,(8,[0,1,2,3,4,5,6,7],[1,0.75,0.25,0,0.5,0.4,0,0.414285714285714]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>Bachir
Kisworo Darsono Rowi:+:(0,(8,[0,1,2,3,4,5,6,7],[0.91829583405449,1,0.25,0,1,0.2,0,0.481185119150641]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>Rowi Ha
ry Sarkum Wasitah:+:(0,(8,[0,1,2,3,4,5,6,7],[0.91829583405449,1,0.25,0,1,0.6,0,0.538327976293498]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>M Bachi
r Kisworo Darsono:+:(0,(8,[0,1,2,3,4,5,6,7],[0.91829583405449,1,0.25,0,1,0.2,0,0.481185119150641]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>Kisworo
 Darsono Rowi Hary:+:(0,(8,[0,1,2,3,4,5,6,7],[0.91829583405449,1,0.25,0,1,0.4,0,0.50975654772207]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>Ahmad D
aldiri:+:(0,(8,[0,1,2,3,4,5,6,7],[1,1,-0.75,0,1,0,0,0.321428571428571]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>Suparto
 dan Ahmad:+:(0,(8,[0,1,2,3,4,5,6,7],[1,1,-0.25,0,0.666666666666667,0.2,0,0.373809523809524]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Yusuf=>Suparto
 dan Ahmad Daldiri:+:(0,(8,[0,1,2,3,4,5,6,7],[0.970950594454669,1,0.25,0,0.75,0.2,0,0.452992942064953])
)
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Suparto=>Sarku
m Wasitah:+:(0,(8,[0,1,2,3,4,5,6,7],[0.970950594454669,1,-0.5,0.5,1,0.571428571428571,0,0.5060541665547
49]))
http://news.okezone.com/read/2008/02/20/1/85298/8-mantan-anggota-dprd-banyumas-ditahan:+:Suparto=>Wasit
ah Yusuf:+:(0,(8,[0,1,2,3,4,5,6,7],[0.970950594454669,1,-0.833333333333333,0,1,0.285714285714286,0,0.34
6190220976517]))
```