DATA MINING

Model Self-Organizing Maps (SOMs)

Pertumbuhan data yang tersimpan dalam suatu *database* yang besar, telah jauh melebihi kemampuan manusia untuk bisa memahami sehingga diperlukan alat dan metode tepat yang mampu mentransformasikan sejumlah besar data dalam informasi yang berguna yang menopang keakuratan informasi itu sendiri. *Data mining* adalah salah satu metode yang digunakan untuk menganalisis data karena banyak memberikan ide yang menghasilkan hal-hal yang berbeda dari sebelumnya.

Neural network merupakan *tool* yang cukup penting untuk aplikasi-aplikasi *data mining*. "Kohonen Maps" atau "Self-Organizing Maps (SOMs)" adalah model *unsupervised* yang paling populer. SOMs adalah model *neural network* yang penting untuk mengurangi dimensi dan mengelompokkan data yang bisa membantu para peneliti mentransformasikan dan memvisualisasikan kemiripan antara pola-pola dalam klaster-klaster.

Buku ini memberikan contoh menggunakan beberapa studi kasus, bagaimana mengelompokkan data yang kompleks, multidimensi, mentransformasikan dan memvisualisasikan kemiripan antara pola-pola dalam klaster-klaster menggunakan Self-Organizing Maps (SOMs). Pembahasan dimulai dari *preprocessing data*, cara instalasi, dan menjalankan SOM (SOM *Toolbox Requirements*).

BINTANG SEMESTA MEDIA
Jl. Karangsari, Gg. Nakula, Sleman, Yogyakarta 55773
Telepon: (0274) 4550369 WA: 0858 6534 2317
Email: redaksibintangpustaka@gmail.com
Website: bintangpustaka.com

ISBN 978-623-8015-88-7
9 786238 015887

Sylvia Jane Annatje Sumarauw

DATA MINING Model Self-Organizing Maps (SOMs)

DATA MINING

Model Self-Organizing Maps (SOMs)

Sylvia Jane Annatje Sumarauw

DATA MINING MODEL
SELF-ORGANIZING MAPS (SOMs)

UNDANG-UNDANG REPUBLIK INDONESIA NOMOR 28 TAHUN 2014
TENTANG
HAK CIPTA
Lingkup Hak Cipta

Pasal 1 Ayat 1 :

1. Hak Cipta adalah hak eksklusif pencipta yang timbul secara otomatis berdasarkan prinsip deklaratif setelah suatu ciptaan diwujudkan dalam bentuk nyata tanpa mengurangi pembatasan sesuai dengan ketentuan peraturan perundang-undangan.

Ketentuan Pidana:
Pasal 113

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).
3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah).
4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah).

Pasal 114

Setiap Orang yang mengelola tempat perdagangan dalam segala bentuknya yang dengan sengaja dan mengetahui membiarkan penjualan dan/atau penggandaan barang hasil pelanggaran Hak Cipta dan/atau Hak Terkait di tempat perdagangan yang dikelolanya sebagaimana dimaksud dalam Pasal 10, dipidana dengan pidana denda paling banyak Rp100.000.000,00 (seratus juta rupiah).

Sylvia Jane Annatje Sumarauw

DATA MINING MODEL SELF-ORGANIZING MAPS (SOMs)

Diterbitkan Oleh
BINTANG SEMESTA MEDIA

***Data Mining* Model Self-Organizing Maps (SOMs)**

Penulis : Sylvia Jane Annatje Sumarauw
Penyelaras Aksara : Yosi Sulastri
Tata Letak : Ridwan Nur M
Desain Cover : Ridwan Nur M

Penerbit:
CV Bintang Semesta Media
Anggota IKAPI Nomor: 147/DIY/2021
Jl. Karangsari, Gang Nakula, RT 005, RW 031,
Sendangtirto, Berbah, Sleman, Yogyakarta 57773
Telp: 4358369. Hp: 085865342317
Facebook: Penerbit Bintang Madani
Instagram: @bintangpustaka
Website: www.bintangpustaka.com
Email: bintangsemestamedia@gmail.com
redaksibintangpustaka@gmail.com

Cetakan Pertama, Desember 2022
Bintang Semesta Media Yogyakarta
xiv + 123 hal : 15.5 x 23 cm
ISBN : 978-623-8015-88-7

Dicetak Oleh:
Percetakan Bintang 085865342319

PRAKATA

Segala sesuatu adalah dari Dia, oleh Dia, dan kepada Dia. Bagi Dialah kemuliaan sampai selama-lamanya. Segala puji syukur kepada Tuhan, karena hanya oleh kasih dan anugerah-Nya yang tidak terbatas penulis dapat menyelesaikan buku referensi ini dengan judul *Data Mining Model Self-Organizing Maps (SOMs)*, dan semua indah pada waktu-Nya.

Data mining adalah salah satu metode yang digunakan untuk menganalisis data, yang tersimpan dalam *database* yang besar yang mampu mentransformasikan sejumlah besar data ke dalam informasi yang berguna dan banyak memberikan ide untuk menghasilkan hal-hal yang berbeda dari sebelumnya. *Neural network* merupakan *tool* yang cukup penting untuk aplikasi-aplikasi *data mining*, dan "Kohonen Maps" atau "Self-Organizing Maps (SOMs), adalah model *unsupervised* yang paling populer. SOMs adalah model *neural network* yang penting untuk mengurangi dimensi dan mengelompokkan data yang bisa membantu para peneliti mentransformasikan dan memvisualisasikan kemiripan antara pola-pola dalam klaster-klaster.

Buku ini memberikan contoh menggunakan beberapa studi kasus, bagaimana mengelompokkan data yang kompleks, multidimensi, mentransformasikan dan memvisualisasikan kemiripan antara pola-pola dalam klaster-klaster menggunakan SelfOrganizing Maps (SOMs). Pembahasan dimulai dari *preprocessing* data, cara instalasi dan menjalankan SOM (SOM Toolbox *Requirements*), dan penerapan kasus pada beberapa dataset.

Pada kesempatan ini penulis menyampaikan terima kasih kepada semua pihak yang sudah membantu dalam penulisan buku ini. Semoga buku ini dapat dimanfaatkan dan bermanfaat terutama bagi peneliti khususnya *Data mining*, Neural Network lebih khusus lagi Self-Organizing Maps (SOMs). Karena buku ini masih jauh dari sempurna, maka kritik, komentar, dan saran dibutuhkan untuk penyempurnaan.

Manado, November 2022

Penulis

DAFTAR ISI

DAFTAR GAMBAR

BAB I

PENDAHULUAN

Perkembangan teknologi informasi dalam beberapa dekade terakhir memungkinkan semakin mudahnya untuk memperoleh data dan informasi dalam jumlah yang besar. Pertumbuhan data yang tersimpan dalam suatu *database* yang besar, telah jauh melebihi kemampuan manusia untuk bisa memahami sehingga diperlukan alat dan metode tepat yang mampu mentransformasikan sejumlah besar data ke dalam informasi yang berguna yang menopang keakuratan informasi itu sendiri. *Data mining* adalah salah satu metode yang digunakan untuk menganalisis data karena banyak memberikan ide untuk menghasilkan hal-hal yang berbeda dari sebelumnya. *Data mining* adalah proses menemukan pola dan tren yang berguna dalam himpunan data besar (Larose & Larose, 2014).

Dalam banyak aplikasi *data mining*, informasi yang bisa diperoleh dari input data sangat terbatas, dan sangat sering *dataset* memiliki struktur yang tidak memungkinkan untuk secara mudah mencari informasi yang ada di dalamnya. *Neural network* merupakan *tool* yang cukup penting untuk aplikasi-aplikasi *data mining* karena ampuh untuk mengatasi data yang beragam(Larose & Larose, 2014).

Neural network mencoba meniru neuron yang saling berhubungan di otak untuk membuat algoritma pembelajaran yang rumit untuk mengekstraksi pola dan mendeteksi tren (Alex Yu, 2022). Alex Yu (2022) mengemukakan *neural network* dapat dikategorikan sebagai "*supervised*" atau "*unsupervised*", bergantung kepada *output* yang diinginkan atau variabel dependen diketahui.

Model *unsupervised neural network* paling populer adalah "Kohonen Maps" atau "Self-Organizing Maps SOMs (Stevanovic et al., 2013). Model ini pertama kali diperkenalkan oleh Teuvo Kohonen pada tahun 1982 (Kohonen, 1982). SOMs adalah teknik pengelompokan yang mengidentifikasi kelompok dalam kumpulan data tanpa harus menggunakan teknik statistik tradisional (Melin et al., 2020). SOMs merupakan suatu algoritma kuantisasi vektor yang terorganisir secara topografis (Fix & Frezza-Buet, 2019). SOMs banyak digunakan dalam masalah *clustering* dan eksplorasi data di industri, keuangan, pengetahuan alam, dan tata bahasa (Kohonen, 2013). SOMs sangat cocok untuk analisis klaster karena terdapat pola tersembunyi di antara catatan dan bidang dicari (Larose & Larose, 2014).

Tujuan dari algoritma SOMs adalah untuk menghasilkan topologi yang mempertahankan pemetaan dari dimensi tinggi ruang ke ruang dimensi rendah dengan kata lain persekitaran dari ruang dimensi tinggi dapat direpresentasikan sebagai indeks persekitaran dalam dimensi yang rendah (Ma et al., 2022). SOMs menjamin bahwa sifat topologi dari ruang input dipertahankan, dan simpul persekitaran mengenali pola yang memiliki karakteristik serupa (García-Tejedor & Nogales, 2022). Algoritma SOMs membangun model-model sehingga semakin mirip model-model akan diasosiasikan dengan *node* yang dekat dengan *grid*, sedangkan model yang kurang mirip secara bertahap akan jatuh lebih jauh dari *grid* (Kohonen, 2013).

SOMs tidak membutuhkan informasi-informasi tambahan selain data input. Setiap *item* data input harus memilih pola yang paling

cocok elemen input, dan model ini, serta subset darinya tetangga di *grid*, harus dimodifikasi untuk pertandingan yang lebih baik (Kohonen, 2013). Inputan dibagi menjadi beberapa partisi yang tidak saling *overlap* yang pada dasarnya menggambarkan sebuah proses kompetisi di antara simpul-simpul pada *neural network*.

Untuk setiap pola pelatihan pada jaringan, simpul tunggal adalah yang terdekat dengan pola input, ditunjukkan dengan jarak pengukuran, memiliki arti bahwa pola pelatihan ditentukan oleh beratnya yang sesuai maupun berat simpul-simpul yang ada pada tetangganya. Dalam penggunaannya pada beberapa aplikasi, suatu pelatihan SOMs harus ditandai, khususnya menggunakan label pola yang memiliki kategori informasi yang menyertainya.

Rumusan masalah buku ini adalah bagaimana mengelompokkan data yang kompleks, multidimensi, mentransformasikan dan memvisualisasikan kemiripan antara pola-pola dalam klaster-klaster menggunakan Self-Organizing Maps (SOMs). Dimulai dari preprocessing data, bagaimana instal, dan menjalankan SOMs (SOMs Toolbox Requirements), dengan beberapa studi kasus.

BAB II

DATA MINING

Data mining pada tahun-tahun belakangan telah menarik perhatian sebagian besar industri informasi dalam kaitannya dengan sejumlah data yang besar dan kebutuhan untuk mengubah sebagian besar data menjadi informasi yang berguna. Hal tersebut terjadi karena sejalan dengan berkembangnya teknologi informasi maka data dalam jumlah yang besar semakin mudah diperoleh. Jumlah data yang luar biasa yang tersimpan dalam *database* yang besar dengan pertumbuhan yang cepat telah melampaui kemampuan manusia. *Data mining* dapat dipandang sebagai sebuah hasil dari evolusi yang alami pada teknologi informasi karena tanpa *data mining* pengumpulan data di dalam *database* yang besar hanya akan menjadi pusara data atau arsip data bersejarah yang jarang dikunjungi, atau dapat diartikan sebagai keadaan kaya data tapi sangat kekurangan informasi.

Data mining adalah ekstraksi informasi potensial yang sebelumnya tidak diketahui. *Data mining* merupakan proses eksplorasi otomatis dan penemuan pola dan informasi dari sejumlah besar data (Zadissa & Apweiler, 2022). *Data mining* menggunakan beberapa metode seperti statistik, kecerdasan buatan, sistem basis data, dan metode

pembelajaran mesin (Binu & Rajakumar, 2021). *Data mining* merupakan proses berulang yang ditentukan oleh kemajuan penemuan, baik melalui metode otomatis atau manual (Kantardzic, 2020). Dalam proses ini tidak terdapat gagasan atau ide yang ditentukan sebelumnya berkaitan dengan kesimpulan dari hasil gambaran data.

Tan et al. (2019) mengemukakan bahwa *data mining* digunakan untuk meningkatkan kinerja sistem tersebut dengan meningkatkan kualitas hasil pencarian berdasarkan relevansinya dengan masukan pertanyaan. Dengan demikian, tugas *data mining* adalah untuk mengekstrak pola dari data mentah atau menemukan pola yang menarik dari sejumlah data yang besar di mana data dapat disimpan di basis data, data *warehouse*, atau tempat penyimpanan lainnya.

Menurut Tan et al. (2019), *data mining* dapat dikategorikan menjadi dua bagian besar:

1. *Data mining* prediktif, yang menghasilkan model dari deskripsi suatu sistem berdasarkan himpunan data yang diberikan.
2. *Data mining* deskripsi, yang menghasilkan suatu informasi yang baru dan nontrivial berdasarkan data yang tersedia.

Tugas dari *data mining* berdasarkan pendapat Larose & dan Larose (2014)Larose & dan Larose (2014) adalah sebagai berikut.

1. Deskripsi merupakan menemukan cara untuk menggambarkan pola dan tren yang ada di dalam data. Untuk mendapatkan deskripsi berkualitas tinggi dapat dicapai dengan analisis eksplorasi data. Metode grafis dapat dipakai untuk mengeksplorasi data untuk mencari pola dan tren.
2. Estimasi. Dalam estimasi, nilai dari target variabel numerik diperkirakan atau didekati dengan menggunakan prediktor variabel numerik atau kategorial. Model dibangun menggunakan catatan "lengkap", yang memberikan nilai variabel target, sebaik prediktor. Kemudian, untuk

pengamatan baru, estimasi dari nilai variabel target yang dibuat berdasarkan nilai dari prediktor.

3. Prediksi. Prediksi mirip dengan klasifikasi dan estimasi kecuali untuk prediksi, hasilnya berada pada masa depan. Setiap metode dan teknik yang digunakan untuk klasifikasi dan estimasi dapat juga digunakan, dalam keadaan yang tepat, untuk prediksi.
4. Klasifikasi. Dalam klasifikasi, terdapat variabel kategoris target, seperti: sebagai braket pendapatan, yang, misalnya, dapat dipartisi menjadi tiga kelas atau kategori: berpenghasilan tinggi, berpenghasilan menengah, dan berpenghasilan rendah. Model *data mining* menguji satu himpunan *record* yang besar. Setiap *record* berisi informasi tentang target variabel serta satu set input atau variabel prediktor.
5. *Clustering*. *Clustering* mengacu pada pengelompokkan *record*, observasi, atau kasus ke dalam kelas-kelas objek serupa. *Cluster* adalah kumpulan *record* yang mirip satu sama lain, dan berbeda dengan *record* di *cluster* lain. Algoritma *clustering* dicari untuk mengelompokkan seluruh kumpulan data ke dalam subkelompok atau kelompok yang relatif homogen, di mana kesamaan *record* dalam *cluster* dimaksimalkan, dan kesamaan untuk *record* di luar *cluster* ini diminimalkan.
6. Asosiasi. Tugas asosiasi untuk *data mining* adalah tugas menemukan atribut mana yang "*go together*". Paling umum di dunia bisnis, yang dikenal sebagai analisis afinitas atau analisis keranjang pasar, tugas asosiasi berusaha mengungkap aturan untuk mengukur hubungan antara dua atau lebih atribut. Aturan asosiasi adalah dari bentuk "Jika anteseden maka konsekuen," bersama-sama dengan ukuran dukungan dan kepercayaan yang terkait dengan aturan.

Kantardzic (2020) mengemukakan langkah-langkah *data mining* berdasarkan prosedur eksperimental secara umum.

1. *State the problem*

   Sebagian besar studi pemodelan berbasis data dilakukan dalam domain aplikasi tertentu. Oleh karena itu, pengetahuan dan pengalaman khusus domain biasanya diperlukan untuk menghasilkan pernyataan masalah yang bermakna. Sayangnya, banyak studi aplikasi cenderung berfokus pada teknik *data mining* dengan mengorbankan pernyataan masalah yang jelas. Dalam langkah ini, seorang pemodel biasanya menentukan satu set variabel untuk ketergantungan yang tidak diketahui dan, jika mungkin, bentuk umum dari ketergantungan ini sebagai hipotesis awal. Di sana mungkin beberapa hipotesis dirumuskan untuk satu masalah pada tahap ini. Langkah pertama membutuhkan keahlian gabungan dari domain aplikasi dan model penambangan data. Dalam praktiknya, ini biasanya berarti interaksi yang erat antara ahli *data mining* dan ahli aplikasi. Dalam aplikasi *data mining* yang sukses, kerja sama ini tidak berhenti pada tahap awal; itu berlanjut selama seluruh proses *data mining*.

2. *Collect the data*

   Langkah ini berkaitan dengan bagaimana data dihasilkan dan dikumpulkan. Secara umum, ada dua kemungkinan yang berbeda. Yang pertama adalah ketika proses pembuatan data berada di bawah kendali seorang ahli (pemodel): pendekatan ini dikenal sebagai eksperimen yang dirancang. Kemungkinan kedua adalah ketika pakar tidak dapat memengaruhi proses pembuatan data: ini dikenal sebagai pendekatan observasional. Pengaturan pengamatan, yaitu pembuatan data acak, diasumsikan di sebagian besar aplikasi penambangan data. Biasanya, distribusi sampling benar-benar tidak diketahui setelah data dikumpulkan, atau sebagian dan secara implisit diberikan dalam prosedur

pengumpulan data. Akan tetapi, sangat penting untuk memahami bagaimana pengumpulan data memengaruhi distribusi teoretisnya karena pengetahuan apriori semacam itu dapat sangat berguna untuk pemodelan dan untuk interpretasi akhir hasil. Selain itu, penting juga untuk memastikan bahwa data yang digunakan untuk memperkirakan model dan data yang digunakan nanti untuk menguji dan menerapkan model berasal dari distribusi sampel yang sama, tidak diketahui. Jika hal ini tidak terjadi, model yang diestimasi tidak dapat berhasil digunakan sepenuhnya dalam aplikasi akhir dari hasil.

3. *Preprocess the data*

Dalam pengaturan observasional, data biasanya "dikumpulkan" dari *database* yang ada, gudang data, dan *data mart*. Pra-pemrosesan data biasanya mencakup setidaknya dua tugas umum:

a. Deteksi pencilan (dan penghapusan)
   Pencilan adalah nilai data yang tidak biasa yang tidak konsisten dengan sebagian besar pengamatan. Umumnya, pencilan dihasilkan dari kesalahan pengukuran dan kesalahan pengodean dan pencatatan. Kadang-kadang adalah nilai yang alami dan tidak normal. Sampel non-representatif tersebut dapat secara serius memengaruhi model yang dihasilkan nanti. Ada dua strategi untuk menangani pencilan:
   1) Mendeteksi dan akhirnya menghapus pencilan sebagai bagian dari fase pra-*processing*.
   2) Mengembangkan metode pemodelan yang kuat yang tidak sensitif terhadap pencilan.
b. Penskalaan, penyandian, dan pemilihan fitur
   Prapemrosesan data mencakup beberapa langkah seperti penskalaan variabel dan berbagai jenis pengodean.

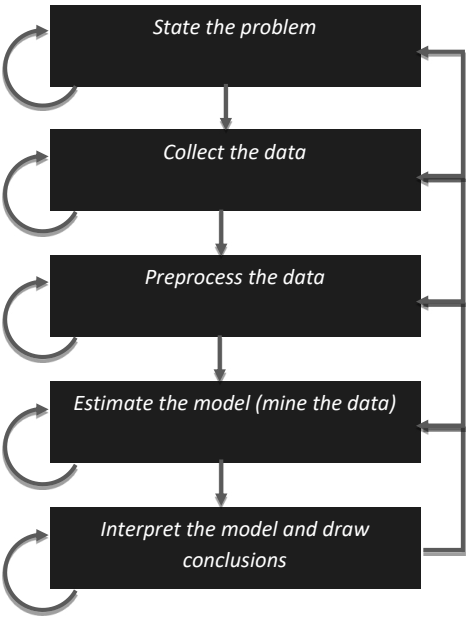
Misalnya, satu fitur dengan rentang [0, 1] dan fitur lainnya dengan rentang [–100, 1000] tidak akan memiliki bobot yang sama dalam teknik yang diterapkan; mereka juga akan memengaruhi hasil akhir *data mining* secara berbeda. Oleh karena itu, disarankan untuk menskalakannya dan membawa kedua fitur tersebut ke bobot yang sama untuk analisis lebih lanjut. Juga, metode pengodean khusus aplikasi biasanya mencapai pengurangan dimensi dengan menyediakan sejumlah kecil fitur informatif untuk pemodelan data berikutnya.

4. *Estimate the model (mine the data)*

Pemilihan dan implementasi teknik *data mining* yang tepat adalah tugas utama dalam fase ini. Proses ini tidak langsung. Biasanya, dalam praktik implementasinya didasarkan pada beberapa model dan memilih yang terbaik adalah tugas tambahan.

5. *Interpret the model and draw conclusions*

Dalam kebanyakan kasus, model penambangan data harus membantu dalam pengambilan keputusan. Oleh karena itu, model semacam itu perlu ditafsirkan agar bermanfaat karena manusia tidak mungkin mendasarkan keputusan mereka pada model "kotak hitam" yang kompleks. Perhatikan bahwa tujuan akurasi model dan akurasi interpretasinya agak bertentangan. Biasanya, model sederhana lebih mudah ditafsirkan, tetapi juga kurang akurat. Metode penambangan data modern diharapkan menghasilkan hasil yang sangat akurat menggunakan model berdimensi tinggi. Masalah menafsirkan model ini juga sangat penting, dianggap sebagai tugas yang terpisah, dengan teknik khusus untuk memvalidasi hasil. Seorang pengguna tidak ingin ratusan halaman hasil numerik. Dia tidak memahami mereka; dia tidak dapat meringkas, menafsirkan, dan menggunakannya untuk pengambilan keputusan yang sukses.



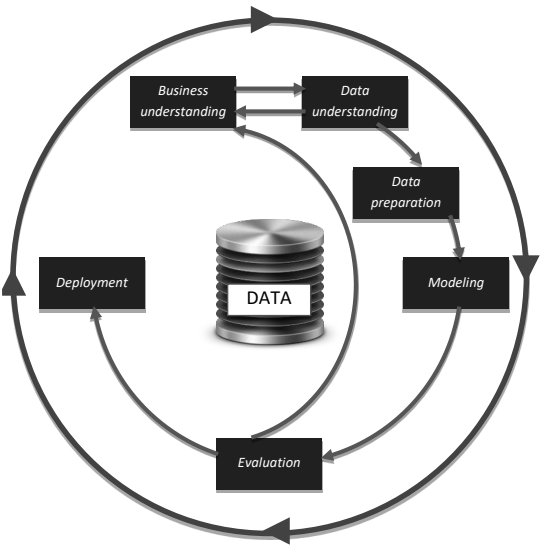
Gambar 2.1 Proses *Data Mining*

Ada kendala di beberapa perusahaan, karena inersia dan kompartementalisasi departemen, untuk mendekati *data mining* secara sembarangan, untuk menemukan kembali roda dan upaya duplikat (Larose & Larose, 2014). Berdasarkan hal tersebut jelas diperlukan adanya suatu standar, maka untuk memudahkan proses *data mining*. Pada tahun 1999 versi pertama dari CRoss-Industry Standard Process for *Data mining* (CRISP-DM), diperkenalkan (Chapman et al., 2000). Metodologi CRISP-DM menyediakan struktur pendekatan dalam merencanakan proyek *data mining* (Kantardzic, 2020). CRISP-DM berkaitan dengan proses dan tugas dan peran yang berbeda dalam proses tersebut (Martinez-Plumed et al., 2021). Sebagai metodologi CRIPS-DM mencakup deskripsi fase khas proyek, tugas yang terlibat dengan setiap fase, dan penjelasan tentang hubungan antara tugas-tugas tersebut. Di samping itu sebagai model proses, CRISP-DM memberikan gambaran tentang siklus hidup *data mining* (IBM Corporation, 2021).

Data Science Process Alliance (2021) mengemukakan beberapa keunggulan dari CRISP-DM:

1. *Common Sense*: Ilmuwan data secara alami mengikuti seperti proses CRISP-DM. Ketika orang diminta untuk melakukan proyek data sains tanpa arah manajemen proyek, mereka cenderung ke arah seperti CRISP metodologi dan dapat dengan mudah mengidentifikasi dengan fase CRISP-DM dan melakukan iterasi.
2. Siklus: CRISP-DM dapat mendukung sifat iteratif dari data sains, tetapi bagaimana sebenarnya melakukan iterasi tidak didefinisikan.
3. Dapat diadopsi: CRISP-DM dapat diimplementasikan tanpa banyak pelatihan, perubahan peran organisasi, atau kontroversi.
4. *Right Start*: Fokus awal pada Pemahaman Bisnis, yang sering diabaikan langkah, sangat membantu untuk menyelaraskan pekerjaan teknis dengan kebutuhan bisnis dan untuk menjauhkan ilmuwan data dari melompat ke masalah tanpa memahami tujuan bisnis dengan benar.
5. Fleksibel: Implementasi CRISP-DM yang longgar dapat menjadi fleksibel untuk memberikan banyak manfaat dari segi prinsip dan praktik tangkas. Dengan menerima bahwa proyek dimulai dengan tidak diketahui yang signifikan, pengguna dapat melalui siklus langkah-langkah, setiap kali mendapatkan pemahaman yang lebih dalam tentang data dan masalahnya. Pengetahuan empiris yang dipelajari dari siklus sebelumnya kemudian dapat dimasukkan ke dalam siklus berikutnya.

Tahapan CRISP-DM untuk *data mining* dapat dilihat pada gambar berikut:



Gambar 2.2 Proses *Data Mining* CRISP-DM.

Proses *data mining* adalah berbentuk lingkaran dengan memiliki beberapa tahap dan semua berputar kembali. Perputaran itu karena beberapa tahapan dari proses *data mining* dapat memberikan pemahaman baru yang memungkinkan beberapa tahapan sebelumnya dapat dikerjakan dengan lebih baik.

2.1 *Business Understanding*

Data memerankan peran yang penting dalam proses *data mining*. *Data miner* harus mengenal problem domain. Pada fase awal proses *data mining* yakni *business understanding* terfokus pada bagaimana memahami tujuan dan kebutuhan proyek dari sudut pandang bisnis, kemudian mengubah pengetahuan menjadi sebuah definisi permasalahan *data mining* dan sebuah rencana awal yang dirancang untuk mencapai tujuan. Terdapat empat yang harus dilakukan dalam proses *Business Understanding* berdasarkan Data Science Process Alliance (2021):

1. Determine business objectives
2. Assess situation
3. Determine project goals
4. Produce project plan

Selain itu, pada fase ini juga dilakukan penentuan persyaratan detail dan membuat formula untuk pemrosesan data atau menyiapkan strategi pemrosesan data awal (Santi Ika et al., 2017).

2.2 *Data Understanding*

Pada fase *data understanding* dimulai dengan pengumpulan data awal kemudian dilanjutkan dengan aktivitas untuk mengenal data, mengidentifikasi permasalahan kualitas data, menemukan pengertian awal dari data atau mendeteksi hal-hal yang menarik atas informasi yang tersembunyi dari data tersebut untuk membentuk hipotesis. Data Science Process Alliance (2021) menyebutkan terdapat empat hal yang harus dilakukan pada tahapan ini:

1. *Collect initial data*
2. *Describe data*
3. *Explore data*
4. *Verify data quality*

Tahapan ini sangat bergantung pada business *understanding* sehingga kedua proses ini perlu dilakukan silih berganti (Wendler & Gröttrup, 2021). Langkah ini sangat penting dalam menghindari masalah tak terduga selama fase berikutnya—*data preparation*– yang biasanya merupakan bagian terpanjang dari sebuah proyek (IBM Corporation, 2021).

2.3 *Data Preparation*

Fase *data preparation* atau persiapan data meliputi semua aktivitas untuk membentuk pengaturan dari data mentah awal ke data yang akan masuk ke dalam pemodelan. Persiapan data memilih kasus/

variabel untuk dianalisis dan memperbaiki *data field* jika diperlukan (Santi Ika et al., 2017).

Tugas data *preparation* tergantung kebutuhan, kadang-kadang ditunjukkan beberapa kali dan tidak terikat pada setiap urutan yang sudah ditentukan. Tugas pada fase *data preparation* di antaranya adalah:

1. *Select data*
2. *Clean data*
3. *Construct data*
4. *Integrate data*
5. *Format data* (Data Science Process Alliance, 2021)

2.4 *Modeling*

Pada fase ini, berbagai teknik pemodelan dipilih dan diaplikasikan dan parameter-parameter dikalibrasi menjadi nilai-nilai yang optimal. Biasanya terdapat beberapa teknik atau model untuk tipe permasalahan *data mining* yang sama. Setiap teknik atau model memiliki kebutuhan spesifik berdasarkan bentuk datanya.

Tugas dari proses *modeling* berdasarkan Data Science Process Alliance (2021)adalah

1. *Select modeling techniques*
2. *Generate test design*
3. *Build model*
4. *Assess model*

Jika proses *modeling* gagal atau kualitasnya tidak memuaskan, proses kembali ke *data preparation* dan mencoba menerapkan yang lain dan metode yang lebih canggih (Wendler & Gröttrup, 2021).

2.5 *Evaluation*

Pada tahap ini, telah terbentuk sebuah model (atau beberapa model) yang diharapkan memiliki kualitas yang tinggi dari perspektif analisis data. Sebelum berlanjut ke *deployment* akhir dari model

tersebut, penting untuk mengevaluasi dengan lebih saksama model dan memeriksa langkah-langkah yang dijalankan untuk membentuk dan memastikan model yang dihasilkan benar-benar mencapai tujuan bisnis. Tujuan utama adalah untuk memastikan jika terdapat beberapa isu bisnis penting yang belum cukup diperhitungkan. Pada akhir tahap ini, keputusan tentang penggunaan hasil *data mining* diharapkan dapat dicapai.

Berdasarkan Data Science Process Alliance (2021) terdapat tiga tugas *evaluation.*

1. *Evaluate results*
2. *Review process*
3. *Determine next steps*

Dua jenis hasil yang dihasilkan oleh *data mining* adalah:

1. Model akhir yang dipilih pada fase CRISP-DM sebelumnya.
2. Setiap kesimpulan atau inferensi yang diambil dari model itu sendiri maupun dari proses *data mining*. Ini dikenal sebagai temuan (IBM Corporation, 2021).

2.6 *Deployment*

Pembentukan model biasanya bukan merupakan akhir dari proyek. Apalagi jika tujuan dari model tersebut adalah untuk meningkatkan pengetahuan pada data, pengetahuan yang didapat, perlu diatur dan ditampilkan agar pelanggan atau pengguna dapat menggunakannya. Pada tahap *deployment* bisa mudah seperti menyusun laporan atau bisa rumit seperti mengimplementasikan proses *data mining* berulang pada perusahaan.

Data Science Process Alliance (2021) mengemukakan empat tugas dari *deployment*

1. *Plan deployment*
2. *Plan monitoring and maintenance*
3. *Produce final report*
4. *Review project*

Pada banyak kasus, penggunalah yang menjalankan langkah-langkah *deployment*, bukan para analis data, bahkan jika analis tersebut tidak akan menjalankan usaha *deployment*, bagi pelanggan sangat penting untuk memahami dari awal, tindakan apa yang perlu dilakukan agar model yang dihasilkan benar-benar bisa digunakan.

BAB III

ARTIFICIAL NEUTRAL NETWORK

Artificial Neural networks, atau biasanya hanya disebut *Neural networks* atau *Neural Nets* merupakan sistem komputasi yang terinspirasi oleh neuron di otak dan koneksi yang terjadi di antara neuron-neuron tersebut. Konsep ini pertama kali muncul dalam publikasi (Mcculloch & Pitts, 1943) dengan judul *A Logical Calculus of The Ideas Immanent in Nervous Activity* yang merupakan artikel pertama mengenai konsep *neurocomputing*. Beberapa tahun kemudian konsep *training Artificial Neural Networks* dengan nama Hebb Rule pertama kali diusulkan, berdasarkan *neurophysiologic nature* (Hebb, 1949).

Secara sederhana *Artificial Neural networks* terdiri dari elemen pemrosesan (disebut neuron) dan koneksi di antara mereka dengan koefisien (bobot) yang terikat pada koneksi (Shanmuganathan, 2016). Otak mempunyai sistem pemrosesan informasi yang sangat kompleks, nonlinear, dan paralel. Otak memiliki kemampuan untuk mengatur konstituen strukturalnya, yang dikenal sebagai neuron, sehingga dapat melakukan perhitungan tertentu berkali-kali lebih cepat daripada komputer digital tercepat yang ada saat ini (James & Eli-Chukwu, 2014). *Artificial Neural networks* mempelajari hubungan antara beban

dan suhu masa lalu, saat ini dan masa depan. ANN menginterpolasi antara data suhu dan beban dalam kumpulan data pelatihan untuk memberikan perkiraan beban (Okelola & Ayanlade, 2021).

ANN biasanya didefinisikan berdasarkan keempat parameter berikut.

1. Tipe dari neuron (atau *node* sebagai *neural network* yang mirip dengan *graph*).
2. Arsitektur koneksionis: Organisasi koneksi antara neuron digambarkan sebagai arsitektur. Hubungan antarneuron menentukan topologi *artificial neural network*s, yaitu terhubung sepenuhnya, terhubung sebagian atau parsial.
3. *Learning algorithm* adalah algoritma, yang melatih *networks.*
4. *Recall algorithm*: pengetahuan yang dipelajari diekstraksi dari *network* (Shanmuganathan, 2016).

Konsep *artificial neural networks* memiliki berbagai keunggulan. Sesuai dengan ini, *artificial neural networks* dapat digunakan untuk menemukan solusi dari masalah ketika metode klasik terbukti sulit atau sering gagal (Chakraverty & Jeswal, 2021).

Model *artificial neural network* adalah model dengan struktur fungsi yang fleksibel. Hal ini mengakibatkan model *artificial neural network* cepat berkembang dan telah banyak diaplikasikan pada berbagai bidang. Tiga hal yang sangat menentukan keandalan kinerja *neural network* adalah:

1. Pola rangkaian neuron-neuron dalam jaringan yang disebut dengan arsitektur jaringan.
2. Metode penentuan bobot (*weight*) pada hubungan, disebut pelatihan (*training*), pembelajaran (*learning*), atau algoritma.
3. Persamaan fungsi untuk mengolah masukan yang akan diterima oleh neuron yang disebut dengan fungsi aktivasi (Fausett, 1994).

Berikut ini adalah aplikasi kontemporer *artificial neural network* secara umum berdasar Shanmuganathan (2016):

1. Aproksimasi fungsi, ketika satu himpunan data disajikan.
2. Asosiasi pola.
3. Data *clustering*, kategorisasi, dan konseptualisasi.
4. Mempelajari parameter statistik.
5. Mengumpulkan pengetahuan melalui pelatihan.
6. "Mengekstrak" pengetahuan melalui analisis bobot koneksi.
7. Memasukkan pengetahuan dalam *neural network* untuk tujuan perkiraan pemikiran.

Beberapa hasil penelitian menyimpulkan bahwa kekuatan utama dalam metode ANN terletak pada kemampuannya untuk mengetahui hubungan linear yang melekat pada data, sedangkan model linear menggambarkan hubungan linear antara pengamatan saat ini dan waktu yang akan datang. ANN menggambarkan hubungan *nonlinear* antara keduanya (Samarasinghe, 2006).

Hal ini dapat digambarkan sebagai;

$$y_t = f\left(y_{t-1}, y_{t-2}, ...y_{t-p}\right) + \varepsilon_t \qquad (4.1)$$

dengan, $f\left(y_{t-1}, y_{t-2}, ...y_{t-p}\right)$ adalah fungsi *nonlinear* yang memetakan serangkaian pengamatan *nonlinear* masa lalu dengan hasil berikutnya. Fungsi ini adalah model *neural network*. Komponen terakhir persamaan yaitu ε_t adalah kesalahan yang sebagai menjadi variabel acak dengan rata-rata dan varians .

Nunes et al. (2017) mengemukakan ANN dapat dibagi menjadi tiga bagian yang disebut dengan *layer*:

1. *Input Layer*

   Lapisan ini bertanggung jawab untuk menerima informasi (data), sinyal, fitur, atau pengukuran dari lingkungan luar.

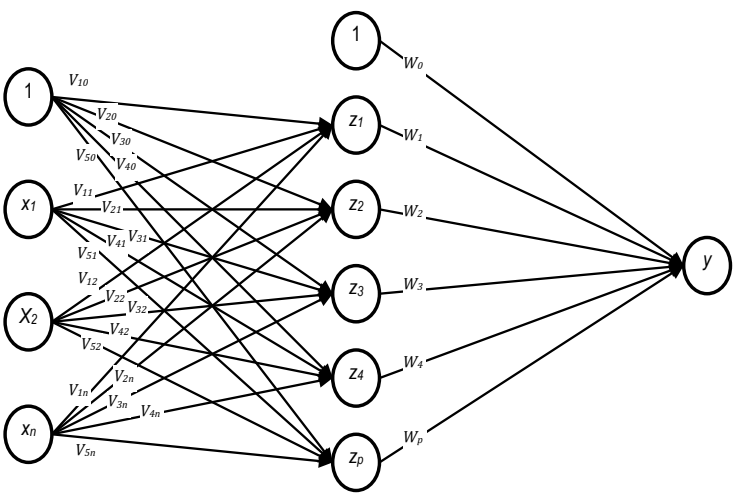
2. *Hidden, intermediate, or invisible layers*
   Lapisan ini terdiri dari neuron yang bertanggung jawab untuk mengekstraksi pola yang terkait dengan proses atau sistem yang dianalisis.
3. *Output Layer*
   Lapisan ini juga terdiri dari neuron, dan dengan demikian bertanggung jawab untuk memproduksi dan menyajikan keluaran jaringan akhir, yang dihasilkan dari pemrosesan dilakukan oleh neuron pada lapisan sebelumnya.

Struktur dari *neural network*s dapat dibagi menjadi beberapa kategori. Liu & Zhou (2020) mengemukakan terdapat empat kategori, yaitu:

1. *Feedforward neural networks*
   The feedforward neural network (FNN) merupakan *network architecture* dari artificial *neural network* yang pertama dan paling sederhana yang tersusun atas input *layer*, beberapa *hidden layers*, dan *output layer*. FNN memiliki *hierarchical structure* yang bersih yang terdiri atas *multiple layers* dari neuron, dan setiap *layer* hanya terhubung dengan *neighbor layers*.
2. *Convolutional neural network*s
   *Convolutional neural network*s (CNNs) adalah versi spesial dari FNN. FNN biasanya mencakup keseluruhan koneksi jaringan, sedangkan CNN lebih ke arah lokal.
3. *Recurrent neural network*s
   Recurrent neural network (RNN) tidak hanya menerima sinyal dan input dari neuron yang lain tetapi informasi historis. Mekanisme memori *recurrent neural network* (RNN) menolong model untuk melakukan proses data series secara efektif.

4. *Graph neural network*
   GNN didesain secara khusus untuk menangani *graph-structured* data seperti *social networks*, *molecular structures*, *knowledge graphs*, dan lain-lain.

Gambar 3.1 menunjukkan contoh arsitektur ANN 3-*layer*. Lapisan masukan mempunyai *n* buah *node*, yaitu: X_1, X_2, X_3, …X_n. Lapisan tersembunyi mempunyai *p* buah *node* yaitu Z_1, Z_2, Z_3,…, Z_p. Lapisan luaran (*output layer*) memiliki satu buah *node* yaitu *Y*. Bobot dari lapisan masukan ke lapisan tersembunyi dinyatakan dengan V_{pn} dengan *p* adalah node ke-*p* pada lapisan tersembunyi dan *n* adalah *node* ke *n* pada lapisan masukan, sedangkan V_{po} adalah bias yang masuk *node* ke-*p* pada lapisan tersembunyi. Bobot dari lapisan tersembunyi ke lapisan luaran dinyatakan dengan W_{mp} dengan *m* adalah *node* ke-*m* pada lapisan luaran dan *p* adalah *node* ke-*p* pada lapisan tersembunyi, sedangkan V_{mo} adalah bias yang masuk *node* ke-*m* pada lapisan luaran.



Gambar 3.1 Contoh Arsitektur ANN

3.1 *Training Processes and Properties of Learning*

Salah satu fitur yang paling relevan dari ANN adalah kemampuannya untuk belajar (*learning*) dari penyajian sampel (pola), yang mengekspresikan sistem perilaku (Nunes et al., 2017). Proses *learning* ini bergantung pada jenis hasil penelitian atau tujuan yang diharapkan oleh *networks*.

Proses *training* ANN dilakukan melalui proses *learning* berdasarkan data empiris. Oleh karena itu, proses pelatihan jaringan saraf terdiri dari penerapan diperlukan langkah-langkah terkoordinasi untuk menyetel *synaptic weights* dan ambang batas dari neuron, untuk menggeneralisasi solusi yang dihasilkan oleh *output*-nya (Nunes et al., 2017). Himpunan langkah-langkah terkoordinasi yang digunakan untuk melatih jaringan disebut *learning algorithm* atau algoritma pembelajaran. *Learning process* dapat dibedakan menjadi tiga model yaitu *supervised learning*, *unsupervised learning*, dan *reinforcement learning*.

1. *Supervised learning*

   Supervised learning merupakan rangkaian revisi model untuk mengurangi perbedaan antara *output* yang benar dan *output* dari model untuk input yang sama (Kim, 2017). Jika sebuah model dilatih dengan sempurna, itu akan menghasilkan *output* yang benar yang sesuai dengan input dari data pelatihan.

2. *Unsupervised learning*

   Unsupervised learning umumnya digunakan untuk menyelidiki karakteristik data dan pra-pemrosesan data (Kim, 2017). Dalam *unsupervised learning, network* disediakan dengan input tetapi tidak dengan *output* yang diinginkan, sistem itu kemudian harus memutuskan fitur apa yang akan digunakan untuk mengelompokkan data masukannya (Chakraverty & Mall, 2017).

3. *Reinforcement learning*

Algoritma pembelajaran yang digunakan pada reinforcement learning menyesuaikan parameter saraf internal yang mengandalkan informasi kualitatif atau kuantitatif apa pun diperoleh melalui interaksi dengan sistem (lingkungan) yang dipetakan, menggunakan informasi ini untuk mengevaluasi kinerja pembelajaran (Nunes et al., 2017).

3.2 Fungsi Aktivasi

Fungsi aktivasi merupakan fungsi yang digunakan pada jaringan saraf untuk mengaktifkan atau tidak mengaktifkan neuron (Budhiarti Nababan & Zarlis, 2015). Dalam metode *backpropagation*, fungsi aktivasi yang dipakai harus memenuhi beberapa syarat, yaitu kontinu, terdiferensiasi dengan mudah, dan merupakan fungsi yang tidak turun. Salah satu fungsi yang memenuhi ketiga syarat tersebut sehingga sering dipakai adalah fungsi *sigmoid biner* yang memiliki interval (0, 1) dan digunakan pada lapisan luaran, sesuai persamaan (4.2).

$$f(x) = \frac{1}{1+e^{-x}} \tag{4.2}$$

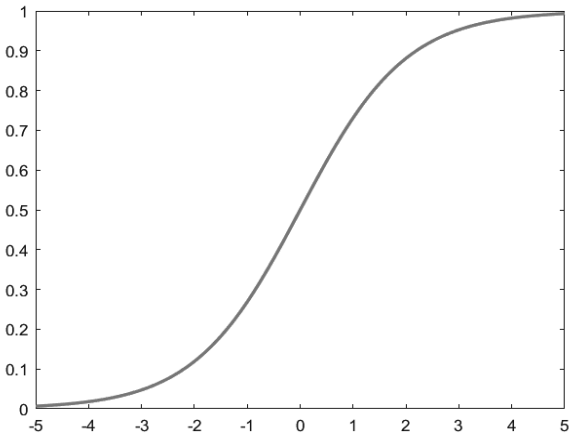
dengan $f'(x) = f(x)\left(1 - f(x)\right)$

Grafik fungsi persamaan (4.2) terlihat pada Gambar 2.4 (a). Fungsi lain yang sering dipakai adalah fungsi *sigmoid bipolar* yang bentuk fungsinya mirip dengan fungsi *sigmoid biner*, tapi dengan interval antara -1 (minus satu) sampai dengan 1 (satu) (persamaan 4.3).

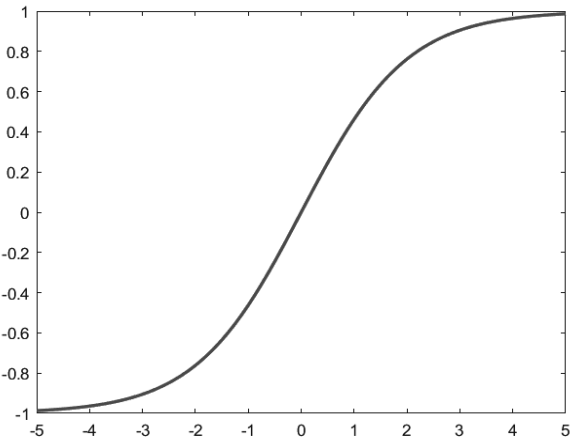
$$f(x) = \frac{2}{1+e^{x}} - 1 \tag{4.3}$$

dengan $f'(x) = \frac{\left(1 + f(x)\right) - \left(1 - f(x)\right)}{2}$

Grafik fungsi persamaan (3.2) terlihat pada Gambar 3.2 (b). Fungsi *sigmoid* memiliki nilai maksimum = 1 (satu). Pada pola yang targetnya lebih besar dari 1 (satu), pola masukan dan luaran harus terlebih dahulu ditransformasi sehingga semua polanya memiliki interval sama seperti fungsi *sigmoid* yang dipakai. Alternatif lain adalah menggunakan fungsi aktivasi *sigmoid* hanya pada lapisan yang bukan luaran *layer*. Pada lapisan luaran, fungsi aktivasi yang dipakai adalah fungsi identitas, $f(x) = x$.



(a)



(b)

Gambar 3.2 (a) Grafik Fungsi *Sigmoid* Biner (b) Bipolar (Fausett, 2004)

Fungsi lain yang mirip dengan *sigmoid bipolar* adalah fungsi *hyperbolic tangent* yang dirumuskan sebagai berikut:

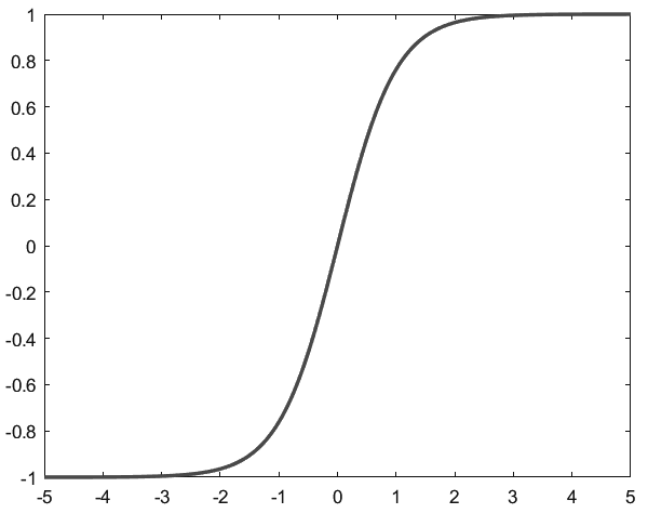
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.4}$$

atau

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{4.5}$$

dengan $f'(x) = \left[1 + f(x)\right]\left[1 - f(x)\right]$

Grafik fungsi persamaan (4.4) dan (4.5) adalah seperti pada Gambar 3.3.



Gambar 3.3 Grafik Fungsi Tansig

BAB V

KOHONEN SELF-ORGANIZING MAPS

Self-Organizing Maps (SOMs) atau yang sering disebut juga Kohonen Maps merupakan teknik visualisasi data yang diperkenalkan oleh Profesor dari Universitas Helsinki yang bernama Teuvo Kohonen dengan ide utama memproyeksikan data input n-dimensi ke dalam beberapa representasi yang dapat lebih dipahami secara visual, misalnya, dalam peta gambar 2D (Kantardzic, 2020). Kohonen Self-Organizing Maps adalah model *neural network* yang penting untuk mengurangi dimensi dan mengelompokkan data. Self-Organizing Maps merepresentasikan satu *set item* data yang berdimensi tinggi sebagai gambar dua dimensi terkuantisasi secara teratur yang dipetakan menjadi satu titik (simpul) di peta, dan jarak item dalam peta tersebut mencerminkan kesamaan antara *item* (Kohonen, 2014). SOMs juga diartikan sebagai *clustering* dan teknik visualisasi data berdasarkan *neural network viewpoint* (Tan et al., 2019). Tujuan dari *neural network* ini adalah untuk mentransfer semua objek data input dengan n atribut (n dimensi) ke *output* dengan cara objek-objek tersebut terkait satu sama lain (Melin et al., 2020). *Output* SOMs menekankan pada fitur yang menonjol dari data dan selanjutnya

mengarah pada pembentukan *cluster* secara otomatis dari item data yang serupa (Kantardzic, 2020). Dalam hal ini akan terbentuk sekumpulan *centroid* yang secara implisit mendefinisikan *cluster* dan setiap *cluster* terdiri dari beberapa *node* yang paling dekat dengan *centroid* tertentu.

SOMs didasarkan pada pembelajaran kompetitif, yaitu *node output* bersaing di antara mereka sendiri untuk menjadi *node* pemenang (neuron), satu-satunya *node* yang akan diaktifkan dengan pengamatan input tertentu (Larose & Larose, 2014). SOMs merupakan tipe pengelompokkan berbasis *centroid* dengan untuk menemukan satu set *centroids* (vektor referensi dalam terminologi SOMs) dan untuk menetapkan setiap objek dalam kumpulan data ke *centroid* yang memberikan perkiraan terbaik untuk itu objek (Tan et al., 2019).

SOMs dapat diterapkan dalam pemetaan data dan gambar (Santi Ika et al., 2017). SOMs dapat mempelajari serta mengambil struktur yang penting di dalam data. SOM dapat mempelajari data yang kompleks, multidimensi, dan mentransformasikannya menjadi suatu peta topologi dengan dimensi yang lebih sedikit, biasanya satu atau dua dimensi. Penggambaran dengan dimensi yang kecil ini membuatnya mudah untuk divisualisasikan yang akan membantu peneliti (*data miners*) memvisualisasikan klaster-klaster atau kemiripan di antara pola-pola.

Self-Organizing Maps (SOM) merupakan salah satu model *artificial neural network* dengan menggunakan metode pembelajaran tanpa *supervised* (Santi Ika et al., 2017). Pelatihan SOMs atau pembelajarannya sering dinamakan sebagai *unsupervised* karena target *output* yang berkaitan dengan setiap pola input di dalam SOMs tidak diketahui. Demikian juga selama proses pelatihannya, proses pola-pola input dalam SOMs dan pembelajaran *cluster* atau segmen data melalui penyesuaian bobot (*weights*).

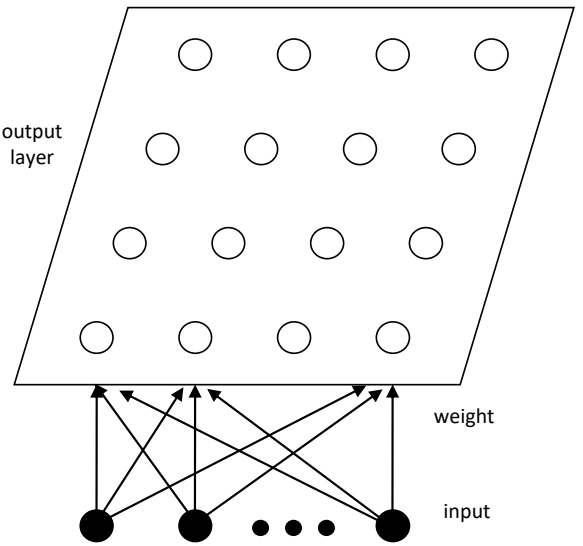
Larose dan Larose (2014) mengemukakan bahwa SOMs menunjukkan tiga proses karakteristik.

1. Kompetisi. Seperti disebutkan di atas, *node* keluaran bersaing untuk menghasilkan nilai terbaik untuk fungsi penilaian tertentu, paling sering jarak Euclidean. Dalam hal ini, simpul keluar yang memiliki jarak Euclidean terkecil antara input bidang dan bobot koneksi akan dinyatakan sebagai pemenang. Kemudian kita melihat secara rinci contoh bagaimana ini bekerja.
2. Kerja sama. *Node* pemenang kemudian menjadi pusat lingkungan neuron yang tereksitasi. Ini meniru perilaku neuron manusia, yang sensitif terhadap *output* neuron lain di sekitar mereka. Di SOM, semua *node* di lingkungan ini berbagi "kegembiraan" atau "hadiah" yang diperoleh oleh *node* pemenang, yaitu adaptasi. Jadi, meskipun *node* lapisan keluaran tidak terhubung langsung, mereka cenderung memiliki karakteristik yang sama karena parameter lingkungan ini.
3. Akomodasi. *Node* tetangga dari *node* pemenang berpartisipasi dalam adaptasi, yaitu dalam pembelajaran. Bobot *node* ini disesuaikan untuk lebih meningkatkan fungsi skor. Dengan kata lain, simpul-simpul ini akan memiliki peluang yang lebih besar untuk memenangkan persaingan lagi, untuk serangkaian nilai bidang yang serupa.

SOM bersifat *feedforward* dan *fully connected* (Larose & Larose, 2014). *Feedforward network* berarti tidak mengizinkan perulangan atau perputaran. Adapun *fully connected* berarti bahwa setiap *node* di lapisan tertentu terhubung ke setiap *node* di lapisan berikutnya, meskipun tidak ke *node* lain di lapisan yang sama.

Sebuah tipikal jaringan SOMs memiliki dua lapis simpul, lapisan input, dan lapisan *output* (kadang disebut dengan *Kohonen layer*).

Setiap simpul pada *layer* input terhubung sepenuhnya dengan simpul-simpul yang ada pada *layer output* dua dimensi. Gambar di bawah ini menunjukkan sebuah contoh jaringan SOMs dengan beberapa simpul input di dalam *layer* input dan sebuah *layer output* dua dimensi yang merupakan *array* bujur sangkar 4×4 dan terdiri dari 16 neuron.



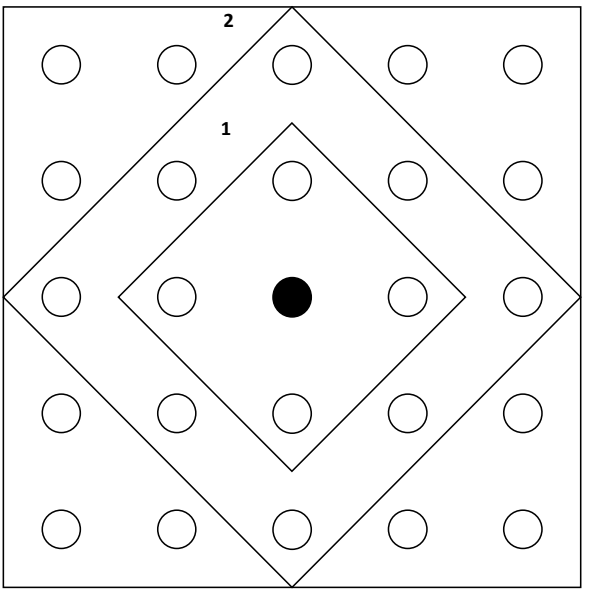
Gambar 4.1 Contoh Jaringan SOMs 4 x 4

Dapat juga digunakan *array hexagonal* atau *grid* dengan dimensi yang lebih tinggi lagi pada Kohonen *layer*. Jumlah simpul-simpul dalam *layer* input menghubungkan sejumlah variabel input dengan jumlah simpul *output* tergantung pada problem yang spesifik dan ditentukan sendiri oleh user. Biasanya jumlah neuron dalam *array* bujur sangkar cukup besar dan memungkinkan sejumlah *cluster* dapat dibentuk. Rekomendasi untuk jumlahnya sebesar sepuluh kali dimensi pola input (Kohonen, 1982).

Selama proses pelatihan, pola-pola input diberikan pada jaringan. Pada setiap langkah pelatihan ketika sebuah pola input yang dipilih secara acak dari pola-pola input yang ada, Setiap neuron pada *later output* akan menghitung seberapa mirip input tersebut dengan

bobotnya . Kemiripan sering diukur berdasarkan jarak antara dan . Ketika proses pelatihan berlangsung, neuron-neuron menyesuaikan bobotnya menurut hubungan topologi di dalam data input. Neuron dengan jarak minimum adalah pemenang dan bobot dari simpul pemenang tersebut dan juga simpul-simpul tetangganya dikuatkan atau diatur ke nilai yang lebih dekat dari pola input. Dengan kata lain pelatihan SOMs adalah *unsupervised* dan kompetitif dan pemenang akan menentukan semua strategi.

Sebuah konsep penentu dalam pelatihan SOMs adalah tetangga mengelilingi sebuah neuron pemenang , yang merupakan kumpulan dari semua simpul-simpul dengan jarak radial yang sama. Gambar di bawah ini adalah sebuah contoh simpul-simpul tetangga berupa Kohonen layer 5×5 pada radius 1 dan 2.



Gambar 4.2 Contoh Simpul Tetangga *Layer* 5 x 5

Komputasi aktual untuk menghasilkan himpunan terurut dari model SOMs sesuai pendapat (Kohonen, 2013) dapat diimplementasikan dengan salah satu dari jenis algoritma utama berikut:

1. Model dalam algoritma SOMs asli dihitung dengan proses pendekatan rekursif dan bertahap di mana item data input berada diterapkan pada algoritma satu per satu, dalam urutan periodik atau acak, untuk langkah sebanyak yang diperlukan untuk mencapai keadaan yang cukup stabil.
2. Dalam proses tipe *batch*, di sisi lain, semua item data input diterapkan ke algoritma sebagai satu *batch*, dan semua model diperbarui dalam satu operasi bersamaan. Proses *batch* ini biasanya perlu diulang beberapa hingga beberapa lusin kali, setelah itu model biasanya akan distabilkan dengan tepat. Bahkan waktu untuk mencapai keadaan yang mendekati stabil adalah urutan besarnya lebih pendek daripada dalam perhitungan bertahap.

Prosedur dasar dalam pelatihan SOMs adalah sebagai berikut:

1. Inisialisasi bobot ke nilai-nilai acak yang kecil dan ukuran tetangga cukup besar untuk meng-*cover* separuh simpul-simpul.
2. Pilih sebuah pola input x secara acak dari kumpulan pola-pola input dan masukkan ke jaringan.
3. Cari kesesuaian yang terbaik atau simpul "pemenang" dengan bobot vektor paling dekat ke vektor input yang sedang diamati menggunakan jarak vektor (*vector distance*).

Maka,

$$|| x\text{-}w_k || = || x\text{-}w_i || \quad (5.1)$$

dengan, ||. || merupakan jarak Euclidean (*Euclidean distance*)

4. Perbaharui bobot simpul-simpul pada tetangga k menggunakan *Kohonen Learning Rule*

   $w_i^{new} = w_i^{old} + \alpha h_{ik} \left(x - w_i \right)$ jika i di dalam N_k

   $w_i^{new} = w_i^{old}$ jika i tidak di dalam N_k

   dengan, adalah *learning rate* antara 0 dan 1 dan h_{ik} adalah *neighborhood* kernel *centered* pada simpul pemenang dan dapat ditentukan dengan bentuk Gaussian sebagai berikut:

   $$h_{ik} = exp\left[-\frac{\|r_i - r_k\|^2}{2\sigma^2}\right] \qquad (5.2)$$

   dengan, r_i dan r_k adalah posisi dari neuron dan pada SOMs *grid* dan adalah *neighborhood* radius.
5. Kurangi *learning rate* sedikit demi sedikit.
6. Ulangi langkah 1–5 beberapa kali dan kemudian kurangi ukuran dari *neighborhood*.
7. Ulangi terus sampai bobot menjadi stabil.

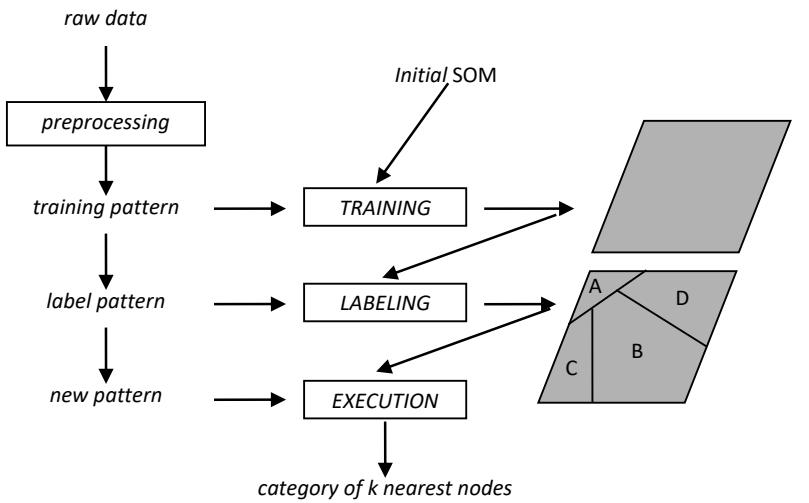
Semakin banyak siklus pelatihan (*epochs*), formasi klaster-klaster yang lebih baik akan ditemukan. Akhirnya, peta topologi adalah hasil optimasi (*fine-tuned*) dengan perbedaan ke arah yang lebih baik dari area klaster-klaster pada peta. Setelah pelatihan jaringan selesai, ia dapat digunakan sebagai sebuah alat visualisasi untuk memeriksa struktur data. Sekali klister-klaster diidentifikasi, neuron-neuron dalam peta dapat ditandai untuk mengindikasikan maknanya. Tugas untuk memberi makna (*meaning*) biasanya membutuhkan pengetahuan pada data dan keahlian pada bidang yang bersangkutan.

BAB V

METODOLOGI

5.1 Metodologi SOM

Metodologi SOMs berkaitan dengan visualisasi data multidimensi. Metodologi SOMs dapat terlihat pada gambar di bawah ini.



Gambar 5.1 Metodologi SOM

Berdasarkan gambar 5.1 dari data mentah yang diperoleh dilakukan *preprocessing* yang merupakan tahapan awal pada metodologi SOMs. Pada proses *preprocessing* dilakukan penanganan data-data yang salah dan normalisasi data input untuk memastikan semua data input memiliki nilai yang konsisten pada rentang nilai 0 s.d. 1. Langkah *Preprocessing* dapat dilakukan dengan berbagai macam cara tergantung pada bidang yang ditangani. Jika data mentah (*raw data*) telah mencerminkan suatu ruang input yang cukup memadai, pelatihan SOMs bisa segera dilakukan.

Dalam sebuah sistem SOM, sebuah peta biasanya adalah sebuah kumpulan simpul-simpul yang membentuk bidang bujur sangkar, beberapa SOMs menggunakan *hexagonal grid* (Kohonen, 1999). Semua input unit terhubung ke semua simpul pada peta, dan hubungan dari setiap input ke sebuah simpul direpresentasikan dengan bobot . Setiap input unit berhubungan ke satu input *field*, dan secara tipikal semua input unit digambarkan dengan nilai *binary* (0 atau 1), nilai *bipolar* (-1 atau +1), atau suatu nilai bilangan *real* (seperti 0 s.d. 1). Kumpulan nilai-nilai tersebut diasumsikan oleh *individual input unit* pada suatu siklus pelatihan tertentu dan ditunjukkan oleh sebuah vektor input , dengan merupakan suatu nilai spesifik dari input unit pada siklus . Keberhasilan pelatihan adalah jika pola-pola input beradaptasi dengan berbagai macam bobot pada setiap simpul pada peta. Pada setiap siklus pelatihan , sebuah sampel pelatihan dipilih secara acak. Kemudian setiap simpul dihitung jaraknya/kemiripannya terhadap input tersebut menggunakan rumusan pengukuran jarak/ kemiripan (seperti *Euclidean distance* atau cosinus sudut antara input dan vektor bobot simpul).

Bobot dari semua simpul pada tetangga (*neighborhood*) dengan jarak terkecil (simpul pemenang) kemudian diperbaharui (di-*update*) menggunakan *learning rule* (Clark & Ravishankar, 1990).

$$w_{ij}^{t+1} = w_{ij}^{t} + \propto (t)\left(x_j^t - w_{ij}^t\right) \qquad (6.1)$$

Parameter tambahan dan ukuran *neighborhood* dikurangi pada siklus selanjutnya mengikuti suatu fungsi parameter *adjustment* (Ritter et al., 1992).

Algoritma pelatihan saat ini dikerjakan pada *neighborhood region*. Suatu fungsi Gaussian *G (c,i,t)* digunakan, simpul-simpul yang dekat dengan simpul pemenang memiliki perubahan bobot lebih besar daripada yang lainnya (Kohonen, 1999).

$$w_{ij}^{t+1} = w_{ij}^{t} + \propto (t) G(c,i,t)\left(x_j^t - w_{ij}^t\right) \tag{6.2}$$

Fungsi *G (c,i,t)* didefinisikan dengan formula sebagai berikut:

$$G(c,i,t) = exp\left(\frac{-D(c,i)^2}{\sigma(t)^2}\right) \tag{6.3}$$

dengan, $\sigma(t)$ adalah sebuah parameter kontrol ukuran simpul-simpul tetangga yang memiliki substansi perubahan bobot, dan *D (c,i,)* adalah jarak *grid* antara simpul dan simpul n_i pemenang.

Di akhir fase pelatihan, sebuah SOMs akan ditemukan.

5.2 Metodologi Penandaan SOM

Peta yang dihasilkan pada setiap akhir fase pelatihan sering/ tidak bisa digunakan sampai setiap simpul pada peta ditandai. Sebuah metode penandaan SOMs (SOMs *labeling*) yang akan dibahas adalah *unsupervised* SOMs *labeling*. Metode umum SOMs *labeling* dilakukan dengan penandaan awal pola-pola *(prelabeled patterns*). Tentu saja, dalam banyak aplikasi, *prelabeled patterns* tidak mudah untuk dilakukan. Pada kenyataannya, jika *prelabeled patterns* tersedia, *supervised neural network* dapat juga diterapkan. Dalam pendekatan penandaan SOMs elemen vektor tersebut (yaitu fitur dari ruang input) yang paling relevan untuk pemetaan vektor input ke unit tertentu. Pada dasarnya dilakukan dengan menentukan kontribusi setiap elemen dalam vektor terhadap jarak Euclidean keseluruhan

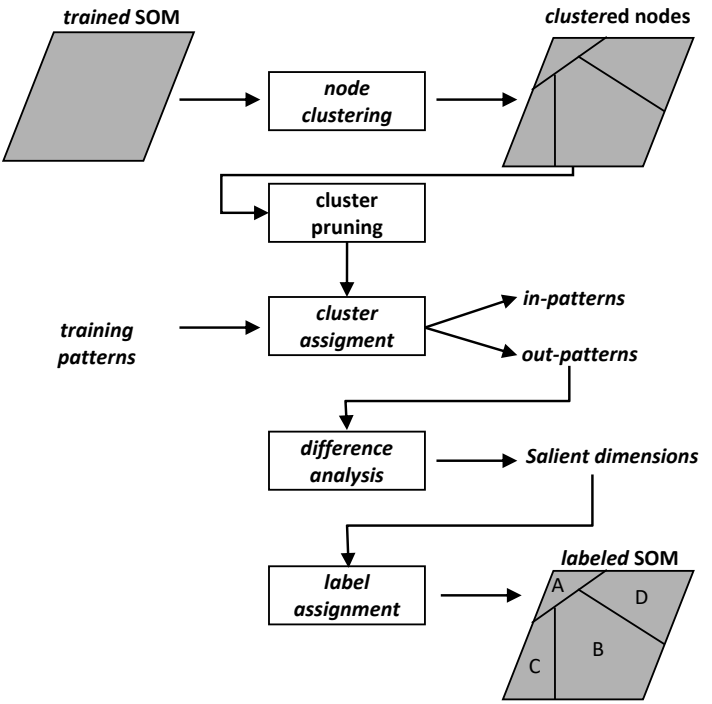
antara vektor input dan vektor bobot pemenang, yang membentuk dasar dari proses pelatihan SOMs (Rauber, 1999).

Model *unsupervised neural network* adalah opsi menarik karena tidak membutuhkan hasil yang diperoleh dari pola-pola pelatihan. Akan tetapi, model-model *unsupervised* ini menghendaki suatu penandaan pola untuk menandai resultan *neural network*, yang mengakibatkan penggunaan model-model ini pada waktu itu menjadi terbatas. Di balik kekurangannya, SOMs memiliki kemampuan untuk memperbaiki melalui suatu metode penandaan yang tidak menghendaki *prelabeled patterns* – tidak selama pelatihan, dan bahkan tidak juga selama penandaan.

Gagasan umum untuk metode *unsupervised* SOMs *labeling* terdiri dari lima langkah utama sebagai berikut:

1. Kelompokkan semua simpul-simpul yang memiliki kemiripan vektor bobot menggunakan suatu metode *clustering*.
2. Untuk setiap *cluster* dari simpul-simpul, hilangkan/buang simpul-simpul pencilan yang sangat berbeda dari *centroid*-nya.
3. Untuk setiap *cluster* dari simpul-simpul (setelah pencilan dibuang), klasifikasi kumpulan pola-pola pelatihan (sebelum dilabel), ambil salah satu yaitu pola yang masuk atau yang keluar tergantung pada apakah ia merupakan simpul paling dekat atau tidak di dalam peta dan di dalam *cluster*.
4. Berbasis pada kumpulan pola-pola yang masuk dan yang keluar dari suatu *cluster* yang diberikan, identifikasi dimensi-dimensi yang menonjol (*salient dimensions*).
5. Berdasarkan *salient dimensions*, tentukan sebuah tanda deskriptif (*descriptive label*) untuk setiap *cluster* dari simpul-simpul yang bisa memberikan arti/makna pada bidang yang diamati.

Metode *unsupervised* SOM dapat terlihat pada gambar di bawah ini.



Gambar 5.2 Metode *Unsupervised* SOM

Langkah 1. Mengelompokkan vektor-vektor bobot simpul

Dalam mengelompokkan simpul-simpul menjadi *cluster* dari referensi vektor bobot yang mirip, satu permasalahan utama adalah menentukan jumlah *cluster* yang sesuai/memadai. Permasalahan ini dapat diatasi dengan melakukan *hierarchical clustering* dari vektor-vektor bobot. Berbagai macam metode hierarki dapat digunakan dan pembahasan mendalam mengenai hal tersebut didiskusikan dalam beberapa buku mengenai *standard clustering* (Everitt, 1974; Hartigan, 1975; Späth, 1980; Xu & Wunsch, 2005)

Pada dasarnya ada dua tipe metode hierarki, yaitu *agglomerative* dan *divisive*. Dalam metode *agglomerative*, setiap vektor bobot simpul

dimulai sebagai sebuah individual *cluster*. Pada setiap langkah, dua *cluster* paling mirip digabung menjadi sebuah *single cluster* dan dihitung pusat *cluster* baru atau *centroid*-nya. Penggabungan klaster-klaster dilanjutkan sampai kualitas klaster-klaster memuaskan, artinya tidak ada dua *cluster* berbeda lagi yang dapat digabungkan yang diukur melalui suatu pengukuran kualitas *clustering*.

Dalam metode *divisive*, semua vektor bobot simpul dimulai sebagai sebuah individual *cluster*. Pada setiap siklus, sistem *partitioning* memilih *cluster* dengan variansi yang paling tinggi dan memecahnya menjadi dua. Vektor-vektor bobot dalam *cluster* dipecah dan dibagi ke dalam dua *cluster* yang baru. Pemecahan *cluster* dilanjutkan terus sampai kualitas pengelompokan memuaskan, yang diukur melalui suatu pengukuran kualitas *clustering*.

Kualitas (biasanya didasarkan pada variansi *cluster*) dari resultan pengelompokan dihitung setiap dua *cluster* digabungkan pada kasus metode *agglomerative*, atau ketika sebuah *cluster* dibagi menjadi dua pada metode *divisive*. Untuk metode *agglomerative*, peningkatan variansi di antara pola-pola relatif lebih besar sampai sebuah *cluster* memberi indikasi bahwa dua *cluster* benar-benar berbeda dan tidak dapat digabungkan. Ketika ini terjadi, *hierarchical clustering* dihentikan, dan jumlah *cluster* yang terbentuk merupakan estimasi yang baik untuk menentukan jumlah *cluster* yang sesuai. Contoh *agglomerative clustering* adalah K-means. Untuk metode *divisive*, pemecahan dihentikan ketika tidak ada lagi *cluster* yang dapat dipecah dengan penurunan yang signifikan terhadap variansi di antara pola-pola dalam klaster-klaster.

Langkah 2. Menghilangkan/membuang simpul-simpul pencilan

Simpul-simpul yang terlalu berbeda dari pusat *cluster* akan dihilangkan. Untuk melakukan hal tersebut, perlu dihitung *centroid* x_k dari setiap simpul *cluster* Γ^k sebagai berikut:

$$x_j^k = \frac{\sum_{n_i \epsilon \Gamma^k} w_{ij}}{\left|\Gamma^k\right|} \quad (6.1)$$

dengan $j = 1, 2, \ldots, D$

D merupakan dimensi data dan w_j adalah komponen ke-*j* dari vektor bobot referensi dari simpul n_i, satu dari simpul-simpul dalam Γ^k. Fungsi $|A|$ menghasilkan *cardinality* dari kumpulan A. Kemudian dihitung jarak d_i dari setiap simpul n_i dalam ke *centroid*-nya Γ^k sebagai berikut:

$$d_i = \sqrt{\sum_{j=1}^{D} \left(w_{ij} - x_j^k\right)^2} \quad (6.2)$$

Dengan jarak simpul masing-masing ke *centroid*-nya, maka dapat dihitung rata-rata (mean) μ_d^k dari semua jarak yang ada dan standar deviasi σ_d^k untuk setiap *cluster*. Menggunakan *z* = 1, sebuah simpul n_i dapat dipertahankan dalam *cluster* originalnya jika:

$$\mu_d^k - z \times \sigma_d^k < d_i < \mu_d^k + z \times \sigma_d^k \quad (6.3)$$

Simpul-simpul tersebut dengan jarak dari *centroid* yang berbeda dari rata-rata (*mean*) lebih dari satu standar deviasi dianggap sebagai pencilan dan dapat dikeluarkan dari *cluster*.

Langkah 3. Memisahkan pola-pola masuk (*In-patterns*) dan pola-pola keluar (*out-patterns)*

Ketika setiap simpul dalam peta ditentukan sebagai anggota sebuah *cluster*, *individual training patterns* digunakan kembali. Setiap pola-pola ini ditentukan sebagai anggota suatu *cluster* jika simpul tersebut memiliki jarak terdekat. Dasar dari penetapan pola pelatihan adalah membangun sebuah kumpulan *in-patterns* dan sebuah kumpulan *out-patterns* untuk setiap *cluster*. *In-patterns* adalah

pola-pola yang memang milik *cluster* yang telah diberikan, sedangkan *out-patterns* adalah semua pola-pola lain yang bukan milik *cluster* yang telah diberikan.

Langkah 4. Mengidentifikasi *salient dimension*

Langkah berikutnya adalah mengidentifikasi dimensi-dimensi dalam sebuah *cluster*, disebut sebagai *salient dimensions*, yang nilai-nilainya berbeda sekali dalam pandangan statistika dibandingkan dengan yang ada pada *cluster* yang lain. Untuk setiap *cluster*, dapat ditentukan jika nilai *mean* input di antara *in-patterns* untuk suatu dimensi lebih tinggi atau lebih rendah daripada nilai *mean* input di antara *out-patterns* yang berhubungan. Identifikasi *salient dimensions* dari setiap *cluster*, dapat dilakukan sebagai berikut:

1. untuk setiap dimensi v, hitung $\mu_{in}(k,v)$ dan $\mu_{out}(k,v)$ sebagai nilai *mean* input untuk kumpulan *In-patterns* $\Phi_{in}(k)$ dan *out-patterns* $\Phi_{out}(k)$;

2. $$\mu_{in}(k,v) = \frac{\sum_{p_i \epsilon \Phi_{in}(k)} x_{iv}}{\left|\Phi_{in}(k)\right|} \tag{6.4}$$

   $$\mu_{out}(k,v) = \frac{\sum_{p_i \epsilon \Phi_{out}(k)} x_{iv}}{\left|\Phi_{out}(k)\right|} \tag{6.5}$$

   dengan, p_i adalah pola pelatihan i dan x_{iv} adalah komponen ke- dari vektor input p_i;

3. hitung *difference factor* $df(k,v)$ dari setiap dimensi v

   $$df(k,v) = \frac{\mu_{in}(k,v) - \mu_{out}(k,v)}{\mu_{out}(k,v)} \tag{6.6}$$

4. hitung rata-rata (*mean*) *difference factor* $\mu_{df}(k)$ dan standar deviasi $\mu_{df}(k)$ untuk semua dimensi v; untuk menghindari kemungkinan kekacauan dalam indeks, diberikan formula untuk *mean* dan standar deviasi sebagai berikut:

$$\mu_{df}(k) = \frac{\sum_{v=1}^{D} df(k,v)}{D} \tag{6.7}$$

$$\sigma_{df}(k) = \left(\frac{\sum_{v=1}^{D} \left(df(k,v) - \mu_{df}(k,v) \right)^2}{D} \right)^{\frac{1}{2}} \tag{6.8}$$

Sebuah dimensi v adalah sebuah *salient dimension* untuk *cluster* Γ^k jika perbedaan *mean* paling sedikit satu standar deviasi; adalah jika

$$df(k,v) \leq \mu_{df}(k) - z \times \sigma_{df}(k) \tag{6.9}$$

atau

$$df(k,v) \geq \mu_{df}(k) + z \times \sigma_{df}(k) \tag{6.10}$$

dengan z = 1.

Langkah 5. Pembagian *descriptive labels*

Teridentifikasinya *salient dimensions* untuk setiap *cluster* dari simpul-simpul, secara manual memaknai kombinasi label dan memberikan deskripsi khusus sebagai *final cluster labels*. Langkah ini sudah menjadi ciri *supervised* pada banyak bidang aplikasi. Seorang *user* harus meneliti *salient dimensions* dan harus memberikan *descriptive label* yang dikehendaki.

BAB VI

SOM TOOLBOX

Salah satu paket perangkat lunak SOM yang tersebar luas adalah SOM Toolbox yang dikembangkan oleh Profesor Universitas Helsinki Teuvo Kohonen dengan timnya sejak tahun 1996. SOM Toolbox, berisi semua fungsi SOM utama dan alat grafis yang bagus dalam bentuk yang ringkas (Kohonen, 2014). SOM Toolbox kompatibel dengan MATLAB, sehingga dapat memanfaatkan semua fungsi yang ada di dalam MATLAB, termasuk grafik serbaguna dan program diagnostik. MATLAB merupakan *software* bahasa pemrograman dan komputasi numerik yang dikembangkan oleh MathWorks dengan berbagai keunggulan seperti manipulasi matriks, *plotting* fungsi dan data, implementasi algoritma, dan lain-lain.

SOM Toolbox dapat dengan bebas diperoleh pada *link* berikut:

- http://www.cis.hut.fi/projects/somtoolbox/documentation/
- http://www.cis.hut.fi/projects/somtoolbox/package/papers/techrep.pdft

Script dari SOMs Toolbox secara normal terdiri atas:

1. Definisi dari parameter-parameter.
2. *Loading* dari input data.
3. *Preprocessing* dari input data.
4. Pemanggilan *functions*.
5. *Plotting*.
6. Penyimpanan fungsi.

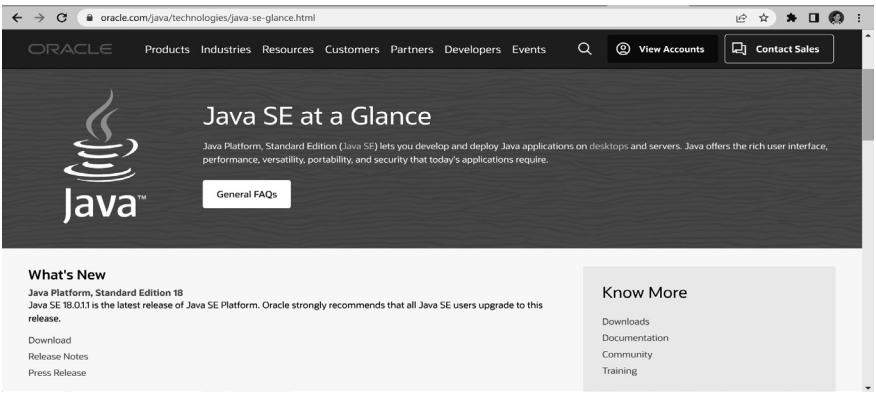
Fungsi sentral dalam perhitungan SOM berdasarkan pendapat Kohonen (2014) biasanya terdiri atas: inisialisasi model, pelatihan kasar para model, pelatihan model yang baik, dan fungsi yang digunakan dalam tampilan grafik SOM.

6.1 SOM Toolbox *Requirements*

Untuk menjalankan SOM Toolbox, beberapa hal yang dibutuhkan antara lain adalah sebagai berikut:

1. Paket SOM Toolbox berupa *file* kompresi SOMToolbox.tar.gz.
2. Java *Standard/Enterprise edition.*
3. Sistem operasi Linux, Unix ,atau Windows.
4. *Browser* seperti Google Chrome, Mozilla Firefox, Microsoft Edge atau Safari untuk membaca petunjuk-petunjuk dalam format html yang disertakan pada paket SOM Toolbox.
5. Editor dengan format "**.rtf**" atau ASCII untuk membuat/ mengedit *file* konfigurasi.
6. Matlab. SOM Toolbox dibangun menggunakan bahasa skrip MATLAB. Hal ini karena struktur dan matriks N-dimensi digunakan, membutuhkan setidaknya Matlab 5. Untuk SOM Toolbox 1.0, direkomendasikan Matlab 5.1.

6.2 Langkah-langkah penggunaan SOMs Toolbox



Gambar 6.1 Antarmuka Java

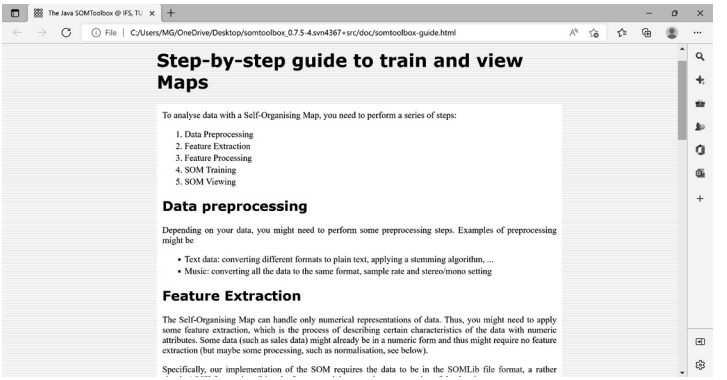
Berikut langkah-langkah dalam menggunakan SOM Toolbox.

1. Download Java Standard Edition (SE) Platform dari Sun Microsystem.
2. Ekstrak SOMToolbox.tar.gz menggunakan WinRar ke direktori yang dikehendaki (misal ke C:\SOMTollbox).
3. Setelah diekstrak, maka akan didapatkan *file* ekstraksi pada direktori C:\SOMToolbox sebagai berikut:

Name	Date modified	Type	Size
.settings	14/06/2022 10:42	File folder	
doc	14/06/2022 10:42	File folder	
lib	14/06/2022 10:43	File folder	
src	14/06/2022 10:43	File folder	
.classpath	23/11/2018 23:46	CLASSPATH File	5 KB
.project	04/01/2015 05:02	PROJECT File	1 KB
build	27/12/2014 06:47	XML Document	30 KB
ChangeLog	29/12/2009 19:21	File	58 KB
somtoolbox	15/09/2010 00:32	Windows Batch File	2 KB
somtoolbox	23/11/2018 23:44	Executable Jar File	6.364 KB
somtoolbox.sh	25/01/2012 21:23	SH File	5 KB
somviewer.prop	29/12/2009 19:21	PROP File	1 KB
somviewer.prop.Linux	29/12/2009 19:21	LINUX File	1 KB
somviewer.prop.Windows	29/12/2009 19:21	WINDOWS File	1 KB

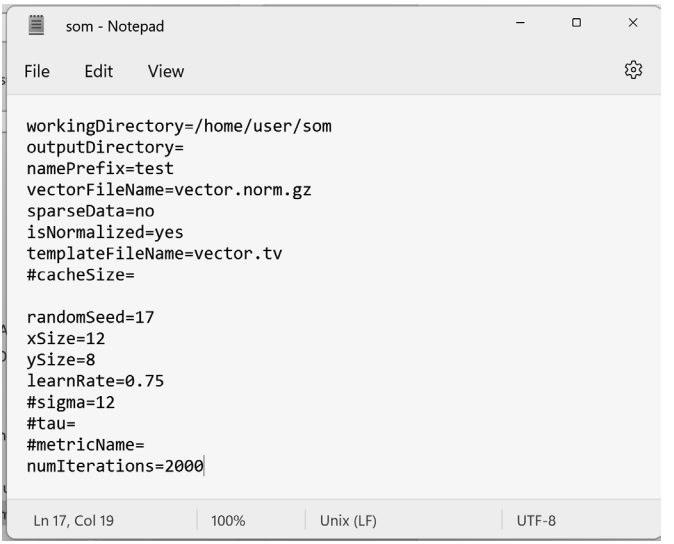
Gambar 6.2 *File* Ekstraksi pada Direktori C:\SOMToolbox

4. Cari *file* "**somtoolbox-howto.html**" dan pelajari petunjuk-petunjuk yang ada.

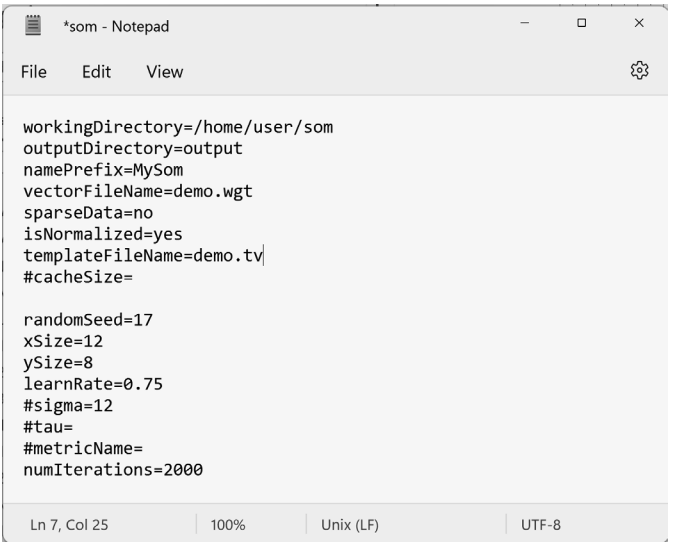


Gambar 6.3 Tampilan Petunjuk Penggunaan SOM Toolbox pada somtoolbox-howto.html

5. Cari *file* "**som.prop**" (terdapat pada folder "**..\SOMToolbox\doc\examples**"). Agar tidak kehilangan *file* aslinya, *copy file* "**som.prop**" dan berikan nama lain, misal "**som-original.prop**".
   Edit *file* "**som.prop**" menggunakan Notepad/WordPad.



Gambar 6.4 *File* som.prop Original



Gambar 6.5 *File* som.prop Original

Goal yang diharapkan adalah membuat properti untuk melakukan *training* pada *file demo* yang ada yaitu pada *file weight vector*-nya (GrowingSOM) dan melihat hasilnya (SOMViewer).

Karena sistem operasi yang digunakan adalah Windows, yang perlu diperhatikan adalah penulisan *directory path* dengan "\\" dan bukan "/" seperti pada sistem operasi linux.

6. Langkah selanjutnya adalah melakukan *training* SOM. Untuk sistem operasi windows *file script* yang digunakan adalah "**somtoolbox.bat**" yang ada pada folder SOMToolbox. Isi dari *file* "**somtoolbox.bat**" adalah sebagai berikut:

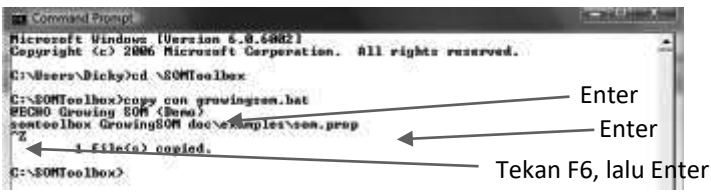
```
@echo off
rem Change BASE_DIR to directory where you installed
SOMtoolbox

set BASE_DIR=.
Set LIB_DIR=%BASE_DIR%\lib

IF EXIST %BASE_DIR%\somtoolbox.jar (
set CP=%BASE_DIR%\somtoolbox.jar
) ELSE (
set CP=%BASE_DIR%\bin
)
FOR %%f IN (%LIB_DIR%\*.jar) DO call :AddLib %%f

goto :StartApp

:AddLib
set CP=%CP%;%1
GOTO :EOF #acts like a return

:StartApp
IF "%1"=="" (
ECHO Runnable classes: SOMViewer
ECHO                   SOMViewer3D
ECHO                   UnitFileViewer
ECHO                   AttendeeMapper
ECHO                   SOMLibVectorNormalization
ECHO                   GrowingSOM
ECHO                   GHSOM
ECHO                   MnemonicSOM
ECHO                   TrajectoryOutputter
```

```
ECHO            HTMLOutputter
ECHO            LabelSOM
ECHO            SOMLibMapOutputter
ECHO            SOMLibZeroVectorRemover
ECHO            QualityMeasureComputer
ECHO            VectorFile2DatabaseImporter
GOTO :Ende
) ELSE (
set FQCLASS="asd"
if "%1"=="SOMViewer" set FQCLASS=apps.viewer.%* --appdir
%BASE_DIR%
if "%1"=="SOMViewer3D" set FQCLASS=apps.viewer3d.%*
--appdir %BASE_DIR%
if "%1"=="AttendeeMapper" set FQCLASS=apps.%*
if "%1"=="SOMLibVectorNormalization" set FQCLASS=data.%*
if "%1"=="SOMLibZeroVectorRemover" set FQCLASS=data.%*
if "%1"=="GrowingSOM" set FQCLASS=models.%*
if "%1"=="GHSOM" set FQCLASS=models.%*
if "%1"=="MnemonicSOM" set FQCLASS=models.%*
if "%1"=="TrajectoryOutputter" set FQCLASS=output.%*
if "%1"=="HTMLOutputter" set FQCLASS=output.%*
if "%1"=="LabelSOM" set FQCLASS=output.labeling.%*
if "%1"=="SOMLibMapOutputter" set FQCLASS=output.%*
if "%1"=="SOMLibDataInfoGenerator" set FQCLASS=util.%*
if "%1"=="QualityMeasureComputer" set FQCLASS=apps.%*
if "%1"=="VectorFile2DatabaseImporter" set
FQCLASS=database.%*
if "%FQCLASS%"=="asd" set FQCLASS=%*
)
```

```
rem ECHO executing java -Xmx1500M -cp %CP% at\ec3\
somtoolbox\%FQCLASS
@echo ON
java -Xmx1024M -cp %CP% at.ec3.somtoolbox.%FQCLASS%
:End
```

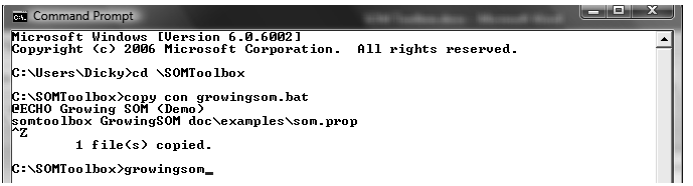
Modifikasi *file* "**somtoobox.bat**" tersebut tidak perlu dilakukan. *File* berekstensi "**.bat**" pada sistem operasi Windows dapat dieksekusi secara langsung. Namun karena membutuhkan opsi-opsi tambahan untuk eksekusinya, eksekusi tidak dapat dilakukan secara langsung melalui Windows Command Prompt. Eksekusi dilakukan dengan terlebih dahulu membuat *script* pada *file* berekstensi "**.bat**" agar setiap kali ingin mengeksekusinya, tidak perlu lagi menuliskan perintah yang panjang.

File tersebut diberi nama "**growingsom.bat**" dan dapat dibuat secara langsung dari *Windows Command Prompt* sebagai berikut:



Gambar 6.6 Tampilan *File* growingsom.bat pada Windows Command Prompt

Selanjutnya untuk melakukan proses *training*, yang diperlukan hanya mengeksekusi *file* "**growingsom.bat**" sebagai berikut:

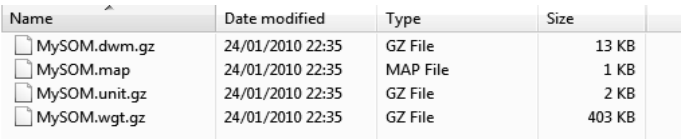


Gambar 6.7 Eksekusi *File* growingsom.bat

Tekan ENTER, dan proses *training* akan dijalankan.

```
C:\SOMToolbox>growingsom
Growing SOMs (Demo)

C:\SOMToolbox>somtoolbox GrowingSOM doc\
examples\som.prop
.....
      → Finished, took 0.03 seconds
25 Jan 10 20:29:43 at.ec3.somtoolbox.models.
GrowingSOM main
INFO: finishedGrowingSOM
C:\SOMToolbox>
```

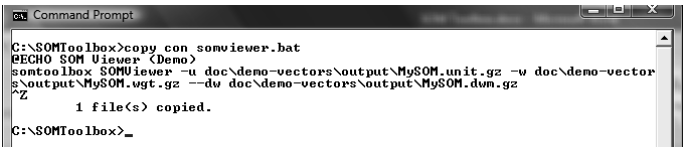
7. Keberhasilan SOMs *training* yang dilakukan dapat dilihat dari terbentuknya *file output* pada direktori yang telah ditentukan pada *file* properti "**som.prop**" yaitu pada folder "**output**". Folder "**output**" yang ditentukan pada "**som.prop**" relatif dan dalam hal ini folder tersebut dibuat dibuat di dalam folder "**C:\SOMToolbox\doc\demo-vectors\output**".
   Di dalam folder tersebut selama proses SOMs ***training*** dibuat empat buah *file* dengan nama depan (*name prefix*) yang telah ditentukan pada "**som.prop**". Keempat buah *file* tersebut adalah sebagai berikut:



Name	Date modified	Type	Size
MySOM.dwm.gz	24/01/2010 22:35	GZ File	13 KB
MySOM.map	24/01/2010 22:35	MAP File	1 KB
MySOM.unit.gz	24/01/2010 22:35	GZ File	2 KB
MySOM.wgt.gz	24/01/2010 22:35	GZ File	403 KB

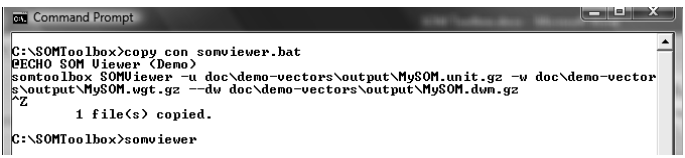
Gambar 6.8 Tampilan Empat Buah *File* dengan Nama Depan (*Name Prefix*)

8. Dengan terbentuknya 4 (empat) buah *file* tersebut, maka hasilnya dapat dilihat dengan menjalankan SOMViewer menggunakan perintah yang dalam hal ini perintah tersebut disimpan pada sebuah file berekstensi "**.bat**" yaitu "**somviewer.bat**" sebagai berikut:



Gambar 6.9 *File* somviewer.bat

Selanjutnya untuk menjalankan SOMs *Viewer*, langkah yang dilakukan adalah melakukan eksekusi *file* "s**omviewer.bat**" sebagai berikut:

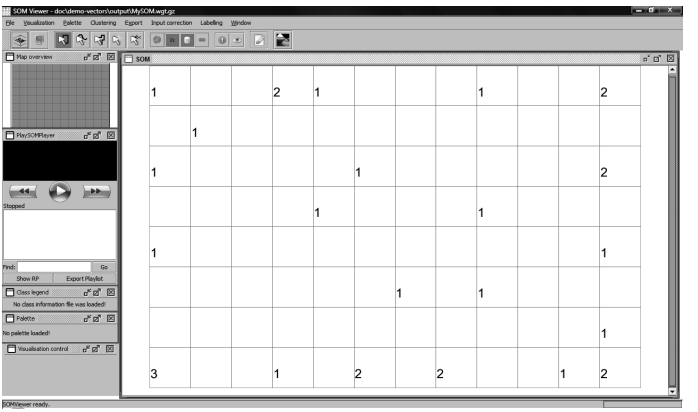


Gambar 6.10 Eksekusi *File* "somviewer.bat"

Tekan ENTER, dan proses inisialisasi sebelum menampilkan SOMViewer akan dijalankan sebagai berikut:

```
C:\SOMToolbox>somviewer
SOMViewer  (Demo)

C:\SOMToolbox>somtoolbox SOMViewer -u
doc\demo-vectors\output\MySOM.unit.gz -w d
oc\demo-vectors\output\MySOM.wgt.gz --dw
doc\demo-vectors\output\MySOM.dwm.gz
.....
        --> Finished, took 0.53 seconds
25 Jan 10 20:43:49 at.ec3.somtoolbox.apps.viewer.
SOMViewer createAndShowGUI
INFO: Initializing GUI ...
25 Jan 10 20:43:50 at.ec3.somtoolbox.apps.viewer.
SOMViewer createVisualizationMenu
INFO: SOMViewer ready.
```

Selanjutnya SOMViewer akan ditampilkan seperti gambar di bawah ini.



Gambar 6.11 Tampilan SOMViewer

BAB VII

PENERAPAN KASUS

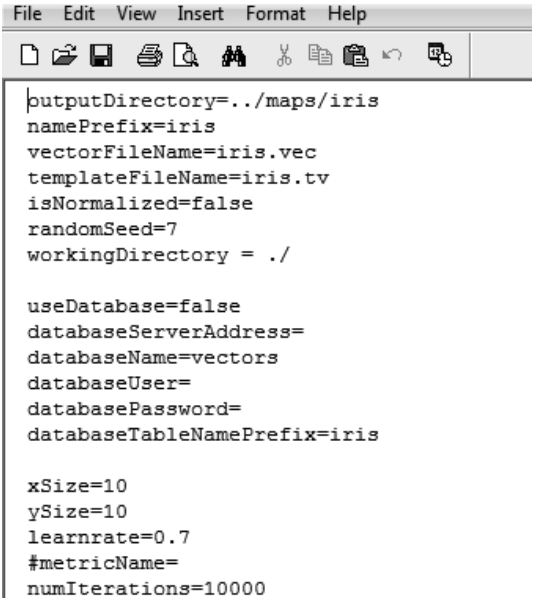
Pada bagian ini akan diberikan beberapa contoh penggunaan SOMs pada beberapa kasus yakni *Iris Dataset*, *Boston Housing Dataset*, dan *Artificial Dataset.*

7.1 *Iris Dataset*

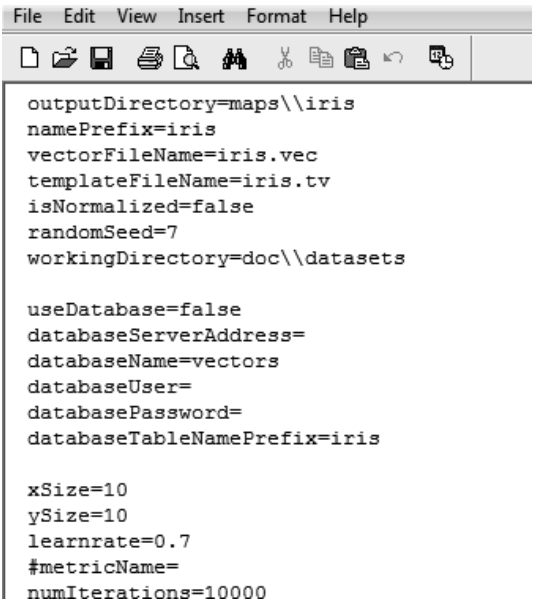
Iris dataset atau data Iris Fisher adalah kumpulan data multivariat yang pertama kali diperkenalkan oleh seorang ahli statistik dan biologi asal Inggris Ronald Fisher dalam makalahnya tahun 1936 yang berjudul *The Use of Multiple Measurements in Taxonomic Problems* sebagai contoh dari analisis diskriminan linear (Fisher, 1936). Kumpulan data ini terdiri atas 50 sampel dari masing-masing tiga spesies *iris* yaitu *iris setosa*, *iris virginica*, dan *iris versicolor*.

Berikut tahapan penerapan penggunaan **SOMToolbox** pada kasus *iris dataset*.

1. Cari *file* "**iris.prop**" (terdapat pada folder "..**SOMToolbox\doc\datasets**"). Agar tidak kehilangan *file* aslinya, *copy file* "**iris.prop**" dan berikan nama lain, misal "**iris-original.prop**". Edit *file* "**iris.prop**" menggunakan WordPad.

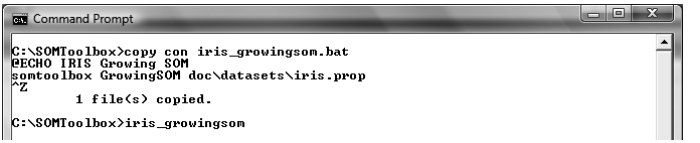


Gambar 7.1 *File* iris.prop Original



Gambar 7.2 *File* iris.prop Setelah Diedit

2. Perintah untuk *training iris dataset* disimpan dalam *file* "**iris_growingsom.bat**" sebagai berikut:

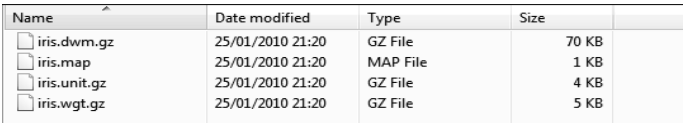


Gambar 7.3 *File* iris_growingsom.bat

Tekan ENTER, dan proses *training* akan dijalankan.

```
C:\SOMToolbox>iris_growingsom
IRIS Growing SOM

C:\SOMToolbox>somtoolbox GrowingSOM
doc\datasets\iris.prop
.....
: 25.01.10 - 21:20:39 CST, 0:00:00,03 r
emaining.
      --> Finished, took 0.12 seconds
25 Jan 10 21:20:39 at.ec3.somtoolbox.models.
GrowingSOM main
INFO: finishedGrowingSOM
C:\SOMToolbox>
```

3. Keberhasilan *training* yang dilakukan adalah terbentuknya *file output* pada direktori yang telah ditentukan pada *file* properti "**iris.prop**" yaitu pada folder "**maps\iris**".
   Folder "**output**" yang ditentukan pada "**iris.prop**" relatif dan dalam hal ini folder tersebut dibuat dibuat di dalam folder "C**:\SOMToolbox\doc\datasets\maps\iris**".
   Di dalam folder tersebut selama proses SOMs *training* dibuat empat buah *file* dengan nama depan (*name prefix*) yang telah ditentukan pada "**iris.prop**". Keempat buah *file* tersebut adalah sebagai berikut:



Name	Date modified	Type	Size
iris.dwm.gz	25/01/2010 21:20	GZ File	70 KB
iris.map	25/01/2010 21:20	MAP File	1 KB
iris.unit.gz	25/01/2010 21:20	GZ File	4 KB
iris.wgt.gz	25/01/2010 21:20	GZ File	5 KB

Gambar 7.4 Tampilan Empat Buah *File* dengan Nama Depan (*Name Prefix*) yang Telah Ditentukan pada "iris.prop".

4. Dengan terbentuknya empat buah *file* tersebut, maka hasil dapat dilihat dengan menjalankan SOMViewer menggunakan perintah yang dalam hal ini perintah tersebut akan disimpan pada sebuah *file* berekstensi "**.bat**" yaitu "**iris_somviewer.bat**" sebagai berikut:

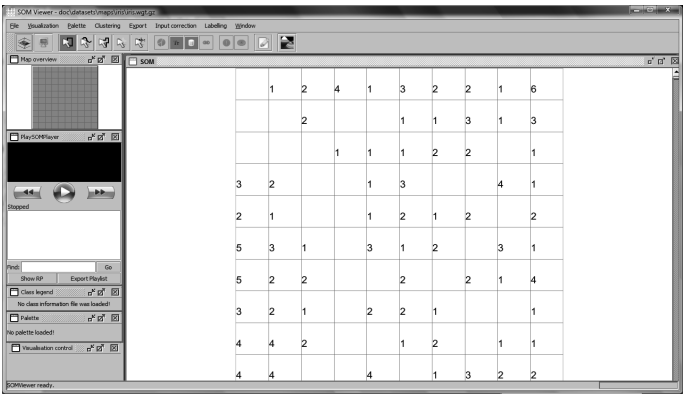
```
Command Prompt

C:\SOMToolbox>copy con iris_somviewer.bat
@ECHO Iris SOM Viewer
somtoolbox SOMViewer -u doc\datasets\maps\iris\iris.unit.gz -w doc\datasets\maps
\iris\iris.wgt.gz --dw doc\datasets\maps\iris\iris.wdm.gz
^Z
        1 file(s) copied.

C:\SOMToolbox>iris_growingsom
```

Gambar 7.5 *File* iris_somviewer.bat

Selanjutnya untuk menjalankan SOMs Viewer, yang perlu dilakukan adalah melakukan eksekusi *file* "**iris_somviewer.bat**" dan proses inisialisasi sebelum menampilkan SOMViewer akan dijalankan sebagai berikut:

```
C:\SOMToolbox>iris_somviewer
Iris SOMs Viewer

C:\SOMToolbox>somtoolbox SOMViewer -u
doc\datasets\maps\iris\iris.unit.gz -w doc
\datasets\maps\iris\iris.wgt.gz --dw doc\
datasets\maps\iris\iris.wdm.gz
.....
        --> Finished, took 0.71 seconds
25 Jan 10 21:39:10 at.ec3.somtoolbox.apps.viewer.
SOMViewer createAndShowGUI
INFO: Initializing GUI ...
25 Jan 10 21:39:10 at.ec3.somtoolbox.apps.viewer.
SOMViewer createVisualizationMenu
INFO: SOMViewer ready.
```

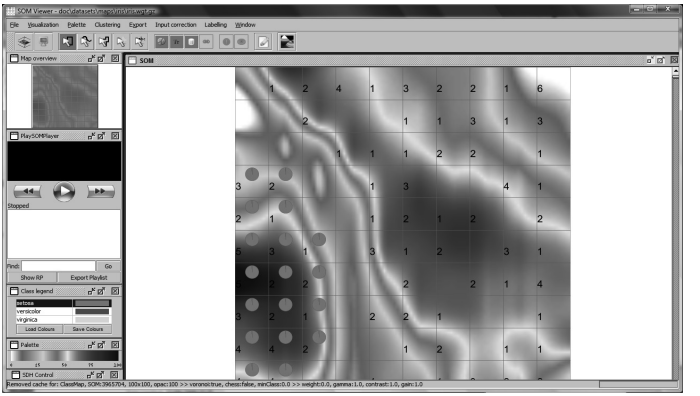
Selanjutnya SOMViewer akan ditampilkan seperti gambar di bawah ini.



Gambar 7.6 Tampilan SOMViewer *Iris Set Data*

Dengan memuat *input vector file* (iris.vec), *template vector file* (iris.tv), *data winner mapping file* (iris.dwm.gz), dan *class information file* (iris.clsinf), maka visualisasi dalam bentuk *Smoothed Data Histograms* (SDH) yang menunjukkan jarak

antara *input vector* dengan *weight vector* akan tampak seperti gambar di bawah ini.



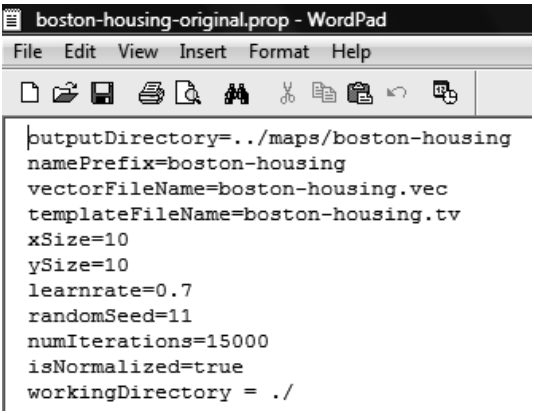
Gambar 7.7 *Smoothed Data Histograms* (SDH)

7.2 *Boston Housing Dataset*

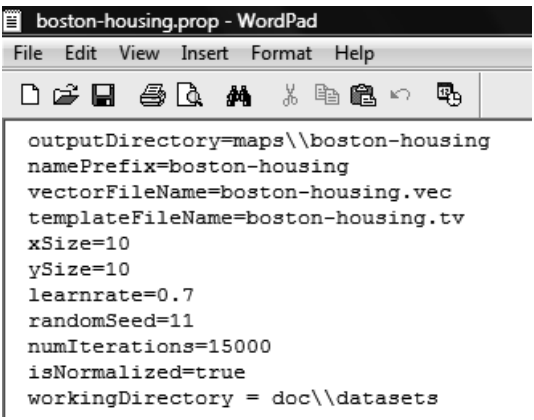
Boston Housing Dataset berisi mengenai informasi yang dikumpulkan oleh US Census Service mengenai perumahan di area Boston Mass. Jenis *dataset* ini pertama kali dipublikasikan oleh Harrison, D. and Rubinfeld, D.L. (Harrison & Rubinfeld, 1978) dengan judul *Hedonic Prices and The Demand for Clean Air*.

Tahapan penerapan penggunaan **SOMToolbox** pada kasus *Boston Housing Dataset* adalah sebagai berikut:

1. Cari *file* "**boston-housing.prop**" (terdapat pada folder "..**SOMToolbox\doc\datasets**"). Agar tidak kehilangan *file* aslinya, *copy file* "**boston-housing.prop**" dan berikan nama lain, misal "**boston-housing-original.prop**".
   Edit *file* "**boston-housing.prop**" menggunakan WordPad.

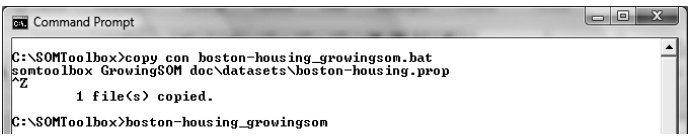


Gambar 7.8 *File* boston-housing.prop Original



Gambar 7.9 *File* boston-housing.prop Setelah Diedit

2. Perintah untuk *training boston housing dataset* disimpan dalam *file* "**boston-housing_growingsom.bat**" sebagai berikut:

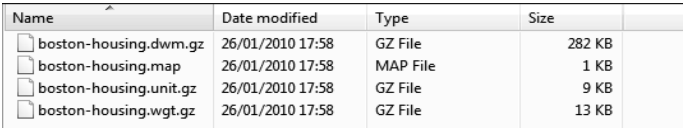


Gambar 7.10 *File* "boston-housing_growingsom.bat"

Tekan ENTER, dan proses *training* akan dijalankan.

```
C:\SOMToolbox>boston-housing_growingsom.
bat

C:\SOMToolbox>somtoolbox GrowingSOM doc\
datasets\boston-housing.prop
.....
        --> Finished, took 0.41 seconds
26 Jan 10 17:58:19 at.ec3.somtoolbox.models.
GrowingSOM main
INFO: finishedGrowingSOM
C:\SOMToolbox>
```

3. Keberhasilan *training* yang dilakukan terlihat dari terbentuknya *file output* pada direktori yang telah ditentukan pada *file* properti "**boston-housing.prop**" yaitu pada folder "**maps\boston-housing**".
   Folder "**output**" yang ditentukan pada "**boston-housing.prop**" relatif dan dalam hal ini folder tersebut dibuat di dalam folder "**C:\SOMToolbox\doc\datasets\maps\boston-housing**".
   Di dalam folder tersebut selama proses SOMs *training* dibuat empat buah *file* dengan nama depan (*name prefix*) yang telah ditentukan pada "**boston-housing.prop**". Keempat buah *file* tersebut adalah sebagai berikut:

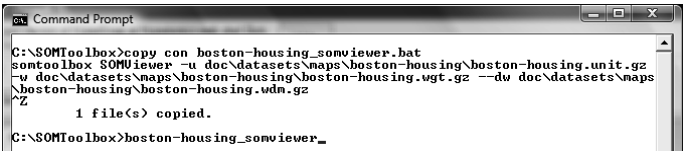


Name	Date modified	Type	Size
boston-housing.dwm.gz	26/01/2010 17:58	GZ File	282 KB
boston-housing.map	26/01/2010 17:58	MAP File	1 KB
boston-housing.unit.gz	26/01/2010 17:58	GZ File	9 KB
boston-housing.wgt.gz	26/01/2010 17:58	GZ File	13 KB

Gambar 7.11 Tampilan empat *File* dengan Nama Depan (*Name Prefix*) yang Telah Ditentukan pada "boston-housing.prop".

4. Dengan terbentuknya empat buah *file* tersebut, maka hasil dapat dilihat dengan menjalankan SOMViewer menggunakan

perintah yang dalam hal ini perintah tersebut akan disimpan pada sebuah *file* berekstensi "**.bat**" yaitu "**boston-housing_somviewer.bat**" sebagai berikut:

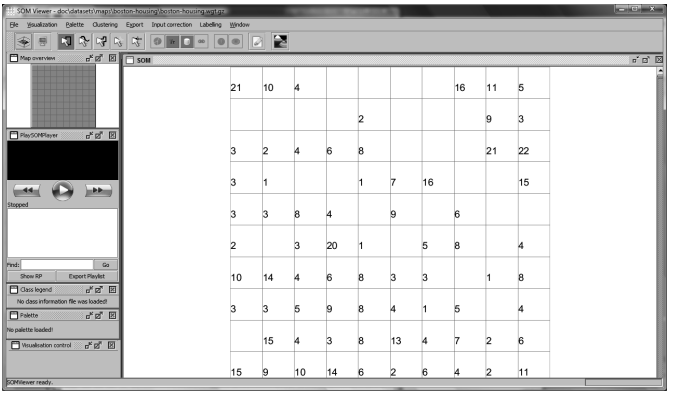


Gambar 7.12 *File* boston-housing_somviewer.bat

5. Selanjutnya untuk menjalankan SOMs Viewer, dilakukan dengan mengeksekusi *file* "**boston-housing_somviewer.bat**" dan proses inisialisasi sebelum menampilkan SOMViewer akan dijalankan sebagai berikut:

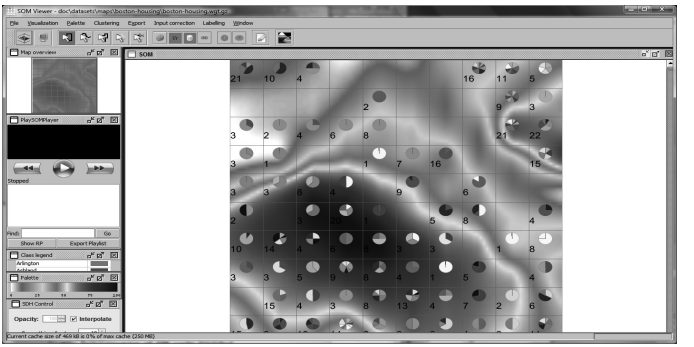
```
C:\SOMToolbox>boston-housing_somviewer

C:\SOMToolbox>somtoolbox SOMViewer -u
doc\datasets\maps\boston-housing\boston-ho
using.unit.gz -w doc\datasets\maps\boston-
housing\boston-housing.wgt.gz --dw doc
\datasets\maps\boston-housing\boston-
housing.wdm.gz
.....
        --> Finished, took 0.86 seconds
26 Jan 10 18:16:47 at.ec3.somtoolbox.apps.viewer.
SOMViewer createAndShowGUI
INFO: Initializing GUI ...
26 Jan 10 18:16:47 at.ec3.somtoolbox.apps.viewer.
SOMViewer createVisualizationMenu
INFO: SOMViewer ready.
```

Selanjutnya SOMViewer akan ditampilkan seperti gambar di bawah ini.



Gambar 7.13 Tampilan *Boston Housing Dataset* pada SOMViewer

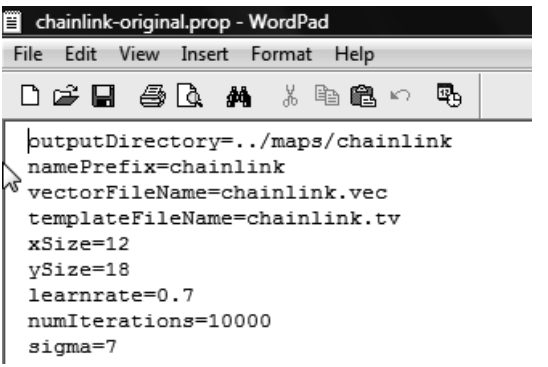
Dengan memuat *input vector file* (boston-housing.vec), *template vector file* (boston-housing.tv), *data winner mapping file* (boston-housing.dwm.gz), dan *class information file* (boston-housing.clsinf), maka visualisasi dalam bentuk *Smoothed Data Histograms* (SDH) yang menunjukkan jarak antara *input vector* dengan *weight vector* akan tampak seperti gambar di bawah ini.



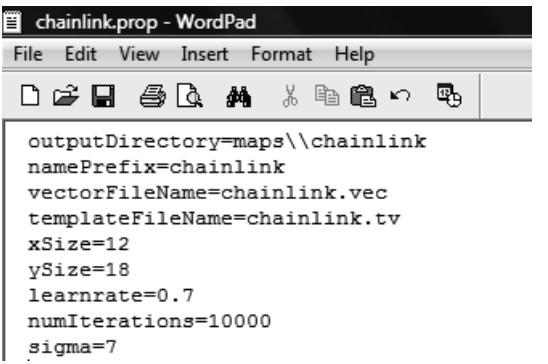
Gambar 7.14 *Smoothed Data Histograms* (SDH) *Boston Housing Dataset*

7.3 *Artificial Dataset*

1. Cari *file* "**chainlink.prop**" (terdapat pada folder "..**SOMToolbox\doc\datasets**"). Agar tidak kehilangan *file* aslinya, *copy file* "**chainlink.prop**" dan berikan nama lain, misal "**chainlink-original.prop**".
   Edit *file* "**chainlink.prop**" menggunakan WordPad.



Gambar 7.15 *File* chainlink.prop Original



Gambar 7.16 *File* chainlink.prop Setelah Diedit

2. Perintah untuk *training chain link dataset* disimpan dalam *file* "**chainlink_growingsom.bat**" sebagai berikut:

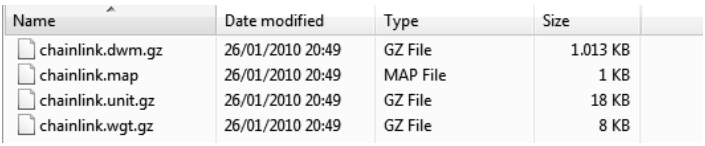
Gambar 7.17 *File* "chainlink_growingsom.bat"

Tekan ENTER, dan proses *training* akan dijalankan.

```
C:\SOMToolbox>chainlink_growingsom
C:\SOMToolbox>somtoolbox GrowingSOM doc\
datasets\chainlink.prop.
.....
        --> Finished, took 1.51 seconds
26 Jan 10 20:49:18 at.ec3.somtoolbox.models.
GrowingSOM main
INFO: finishedGrowingSOM
C:\SOMToolbox>
```

3. Keberhasilan *training* yang dilakukan adalah terbentuknya *file output* pada direktori yang telah ditentukan pada *file* properti "chainlink.prop" yaitu pada folder "maps\chainlink". Folder "**output**" yang ditentukan pada "chainlink.prop" relatif dan dalam hal ini folder tersebut dibuat di dalam folder "C:\SOMToolbox\doc\datasets\maps\chainlink".

   Di dalam folder tersebut selama proses SOMs *training* dibuat empat buah *file* dengan nama depan (*name prefix*) yang telah ditentukan pada "**chainlink.prop**". Keempat buah file tersebut adalah sebagai berikut:



Name	Date modified	Type	Size
chainlink.dwm.gz	26/01/2010 20:49	GZ File	1.013 KB
chainlink.map	26/01/2010 20:49	MAP File	1 KB
chainlink.unit.gz	26/01/2010 20:49	GZ File	18 KB
chainlink.wgt.gz	26/01/2010 20:49	GZ File	8 KB

Gambar 7.18 Tampilan Empat *File* dengan Nama Depan yang Telah Ditentukan pada "**chainlink.prop**"

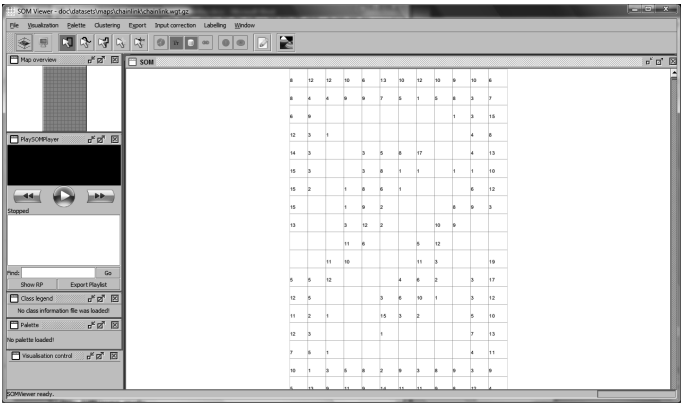
4. Dengan terbentuknya empat buah *file* tersebut, maka hasil dapat dilihat dengan menjalankan SOMViewer menggunakan perintah yang dalam hal ini perintah tersebut akan disimpan pada sebuah *file* berekstensi "**.bat**" yaitu "**chainlink_somviewer.bat**" sebagai berikut:

Gambar 7.19 *File* chainlink_somviewer.bat

Selanjutnya untuk menjalankan SOMs Viewer, hanya perlu untuk mengeksekusi *file* "**chainlink_somviewer.bat**" dan proses inisialisasi sebelum menampilkan SOMViewer akan dijalankan sebagai berikut:

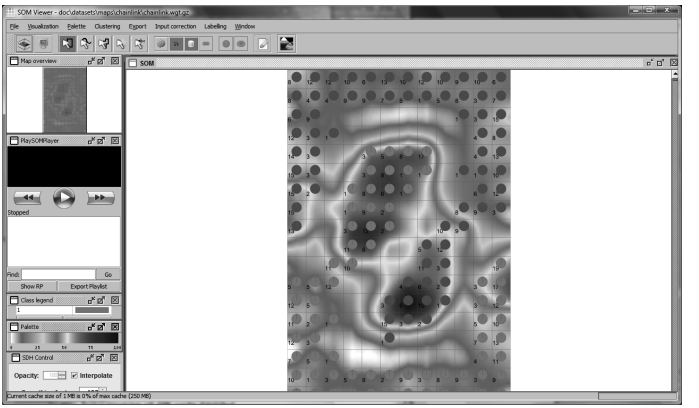
```
C:\SOMToolbox>chainlink_somviewer

C:\SOMToolbox>somtoolbox SOMViewer -u
doc\datasets\maps\chainlink\chainlink.unit
.gz -w doc\datasets\maps\chainlink\chainlink.
wgt.gz –dwdoc\datasets\maps\chainli
nk\chainlink.wdm.gz
.....
      --> Finished, took 1.15 seconds
26 Jan 10 20:53:37 at.ec3.somtoolbox.apps.viewer.
SOMViewer createAndShowGUI
INFO: Initializing GUI ...
26 Jan 10 20:53:37 at.ec3.somtoolbox.apps.viewer.
SOMViewer createVisualizationMenu
INFO: SOMViewer ready.
```

Selanjutnya SOMViewer akan ditampilkan seperti gambar di bawah ini.



Gambar 7.20 Tampilan *Artificial Dataset* pada SOMViewer

Dengan memuat *input vector file* (chainlink.vec), *template vector file* (chainlink.tv), *data winner mapping file* (chainlink.dwm.gz), dan *class information file* (chainlink.clsinf), maka visualisasi dalam bentuk *Smoothed Data Histograms* (SDH) yang menunjukkan jarak antara *input vector* dengan *weight vector* akan tampak seperti gambar di bawah ini.



Gambar 7.21 *Smoothed Data Histograms* (SDH) *Artificial Dataset*

7.4 Bekerja dengan *Text Data File*

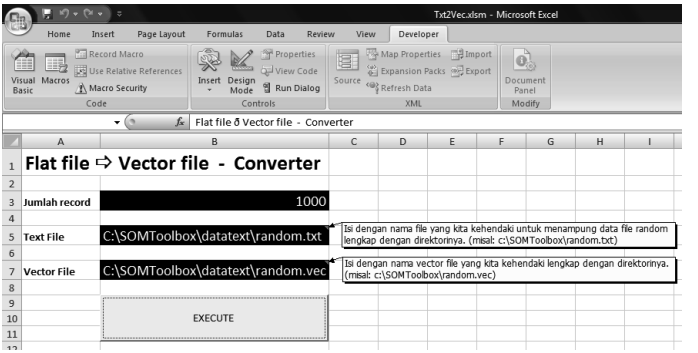
Pada bagian ini kasus yang akan diselesaikan apabila data yang dimiliki adalah *data text*, yang berarti harus diolah terlebih dahulu dan diubah menjadi data vektor yang dapat dikonsumsi oleh SOMs Toolbox. SOMs Toolbox membutuhkan beberapa *file* pendukung yang penting yaitu ***property file***, ***template vector file***, dan ***input vector file***.

1. Kasus

Sebagai contoh kasus akan digunakan data manusia meliputi: umur, tinggi badan, berat badan, dan jenis kelamin. Karena data tersebut tidak ada, sebagai simulasi data-data tersebut akan dibuat secara acak dengan ketentuan sebagai berikut:

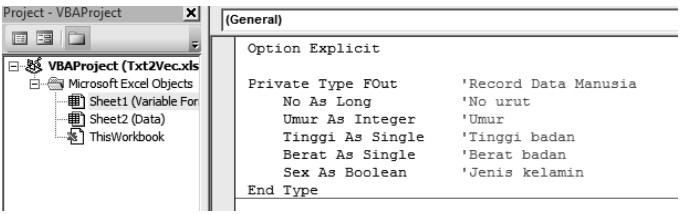
a. Umur memiliki kisaran antara 17 s.d. 65 tahun.
b. Tinggi badan memiliki kisaran antara 158 s.d. 180 cm.
c. Berat badan memiliki kisaran antara 40 s.d. 75 kg.
d. Jenis kelamin adalah pria (diasumsikan sama dengan *boolean - true*) dan wanita (*false*)

Kasus dan pemecahannya secara sederhana disimulasikan menggunakan aplikasi yang sangat populer yaitu MS Excel dan antarmuka untuk mendefinisikan variabel-variabelnya seperti pada gambar di bawah ini.



Gambar 7.22 Tampilan Ms Excel untuk Pendefinisian Variabel Kasus 1

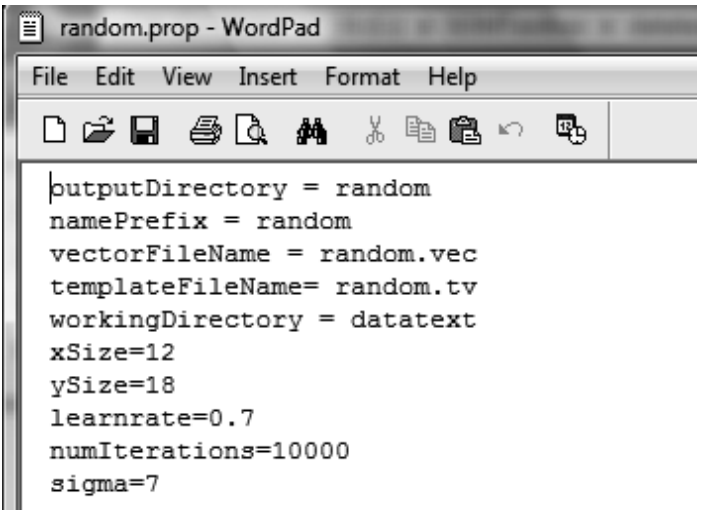
Deklarasi data-data tersebut pada *script program* (Visual Basic for Application/VBA) adalah sebagai berikut:



Gambar 7.23 Deklarasi Data Kasus 1 pada *Script Program* (Visual Basic for Application/VBA)

2. *Property File*

Property file seperti "**som.prop**" yang digunakan pada subbagian II, merupakan *file* yang dibutuhkan untuk menyimpan informasi dan digunakan selama proses SOMs *training*. Untuk kasus ini akan dibuat sebuah *property file* yang akan diberi nama "**random.prop**" yang isinya adalah sebagai berikut:



Gambar 7.24 *Property File* yang Akan Diberi Nama "random.prop"

3. *Template Vector File*

Template vector file pada dasarnya menggambarkan dimensi ruang fitur yang dimiliki dan dapat digunakan untuk menetapkan

label pada *cluster* dengan memilih pengidentifikasi dimensi yang paling khas untuk suatu *cluster*. Format *file* pada dasarnya sebagai berikut:

```
$TYPE template
$XDIM 7
$YDIM  <# of input vectors>
$VECDIM <# of features>
0 <name of feature 1> <df> <tf> <min_tf> <max_
tf> <mean_tf>
1 <name of feature 2> <df> <tf> <min_tf> <max_
tf> <mean_tf>
2 <name of feature 3> <df> <tf> <min_tf> <max_
tf> <mean_tf>
:
:
:
<# of features - 1> <name of feature n> <df>
<tf> <min_tf> <max_tf> <mean_tf>
```

(*Sumber: somlibFileFormatQuick.html pada paket SOMTollbox*)

TYPE adalah teks bebas yang merupakan pengenal *file*.

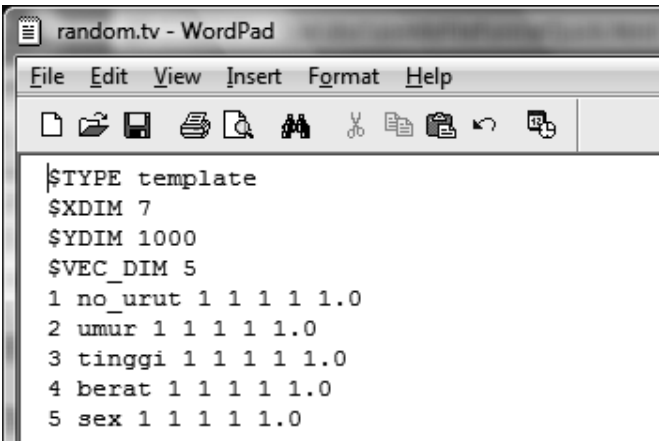
XDIM selalu diatur ke nilai 7 yang menunjukkan vektor kolom berisi 7 macam informasi.

YDIM biasanya diatur ke jumlah vektor yang akan dianalisis (pada kasus ini mengacak sampai 1000 data). Ini dapat digunakan dalam analisis teks untuk menghitung nilai-nilai berbobot seperti **tfxidf**.

VECDIM memberikan dimensi ruang fitur yang dimiliki (pada kasus ini ada lima yaitu: no urut, umur, tinggi badan, berat badan, dan jenis kelamin).

Setelah *header* terdapat baris <vecdim> untuk menjelaskan masing-masing dimensi, dengan kolom mewakili masing-masing, jumlah dimensi (sebuah *counter* dimulai dengan 0), nama dimensi (satu kata), frekuensi atribut dalam pengumpulan data, dokumen frekuensi atribut, serta minimal, maksimal, dan rata-rata frekuensi.

Pada kasus ini, *template vector file* yang dibuat selanjutnya diberi nama "**random.tv**" dan berisi informasi sebagai berikut:



Gambar 7.25 *File* random.tv

4. *Input Vector File* atau *Vector File*

Jumlah fitur dan masukan vektor adalah nilai-nilai integer dan elemen-elemen vektor tunggal dapat berupa nilai integer atau real >= 0.

```
$TYPE vec
$XDIM <# of input vectors>
$YDIM 1
$VECDIM <# of features>

<feat 1> <feat 2> ... <name of input vector 1>
<feat 1> <feat 2> ... <name of input vector 2>
```

TYPE pada baris pertama hanya merupakan teks bebas dalam bentuk tag (satu kata) yang memungkinkan untuk mengenali *file* vektor.

XDIM merupakan jumlah vektor input yang dimiliki (pada kasus ini = 1000).

YDIM untuk *file* vektor input selalu harus diatur pada urutan ke 1.

VECDIM memberikan dimensi ruang fitur yang dimiliki (pada kasus ini = 5).

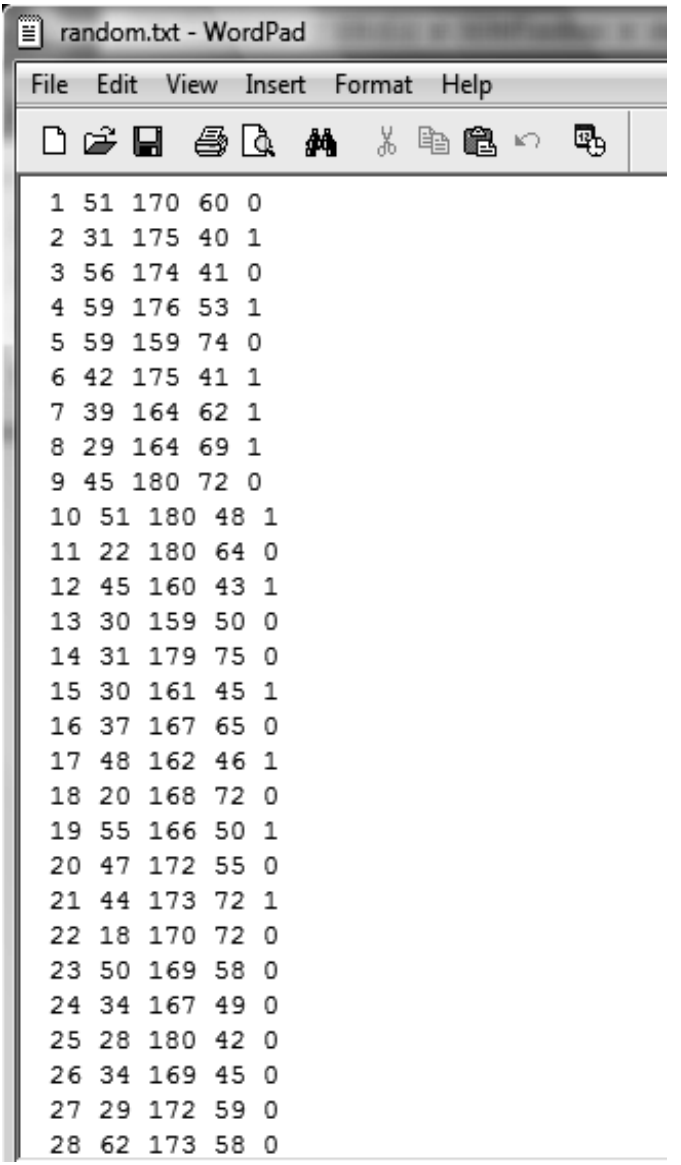
Setelah *header* di atas, diperoleh 1 baris untuk setiap vektor, daftar nilai-nilai atribut yang dipisahkan sebagai ruang bilangan *real*, diikuti dengan label atau nomor urut vektor, yang dapat berupa teks/id. Dengan demikian *file* vektor akan membentuk matriks dengan dimensi = **VECDIM+1** x **XDIM**. Membentuk *vector file* ini menggunakan VBA *script* adalah sebagai berikut:

```
Option Explicit

Private Type FOut     'Record Data Manusia
   No As Long         'No urut
   Umur As Integer    'Umur
   Tinggi As Single   'Tinggi badan
   Berat As Single    'Berat badan
   Sex As Boolean     'Jenis kelamin
End Type

Private Sub cmdExecute_Click()
   Dim FName_Text As String
   Dim FName_Vec As String
   Dim FileNumber As Integer
   Dim i As Long
```

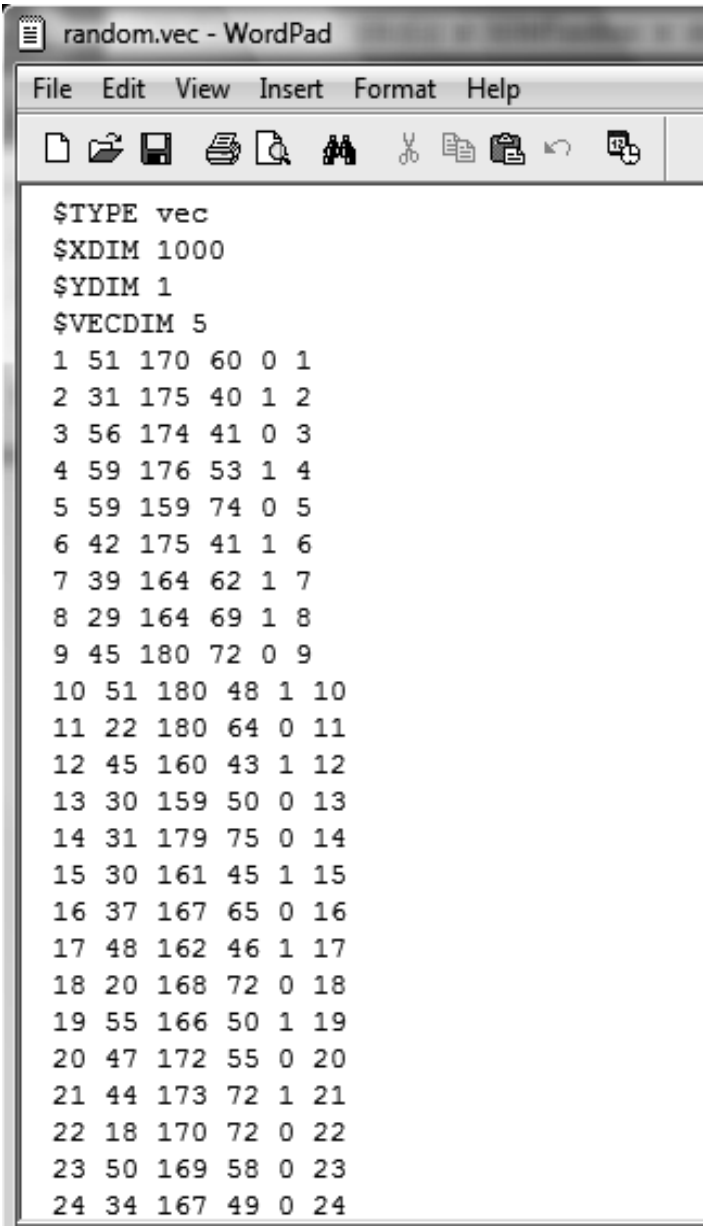
```
    Dim FO As FOut
    On Error GoTo ErrorHandler

    FileNumber = FreeFile
    FName_Text = Range("TextFile")
    FName_Vec = Range("VecFile")

  'Jika file "random.txt" atau "random.vec" sudah ada, maka
dihapus
    If Dir(FName_Text) <> "" Then Kill FName_Text
    If Dir(FName_Vec) <> "" Then Kill FName_Vec

    'Persiapan untuk menulis data random
    Open FName_Vec For Output As #2
    Print #2, "$TYPE vec"
    Print #2, "$XDIM " + Str(Range("JmlRec"))
    Print #2, "$YDIM 1"
    Print #2, "$VECDIM 5"

    Open FName_Text For Output As #1
    For i = 1 To Range("JmlRec")
       FO.No = i
       FO.Umur = MyRand(17, 65)
       FO.Tinggi = MyRand(158, 180)
       FO.Berat = MyRand(40, 75)
       FO.Sex = MyRand(0, 1)
       Print #1, RString(FO)
       Print #2, RString(FO) + " " + Trim(Str(i)) + " "
    Next i
    Close #1
    Close #2
GoTo Dne
ErrorHandler:
```

```
    MsgBox "Masih ada yang salah mbak...!", vbOKOnly,
"Pesan kesalahan"
Dne:
   'Do nothing
End Sub

Private Function RString(A As FOut) As String
   RString = Trim(Str(A.No)) + " " + Trim(Str(A.Umur)) + _
    " " + Trim(Str(A.Tinggi)) + " " + Trim(Str(A.Berat)) + _
    " " + Trim(Str(Abs(Int(A.Sex))))
End Function

Public Function MyRand(ByVal Low As Long, ByVal High
As Long) As Long
  MyRand = Int((High - Low + 1) * Rnd) + Low
End Function
```

Data *text file* yang dihasilkan dari *script program* tersebut adalah seperti pada gambar di bawah ini (kiri) dan *vector file* adalah seperti pada gambar di sebelah kanan.



Gambar 7.26 *File* random.txt



Gambar 7.27 *File* random.vec

BAB VIII

SOM *Training*

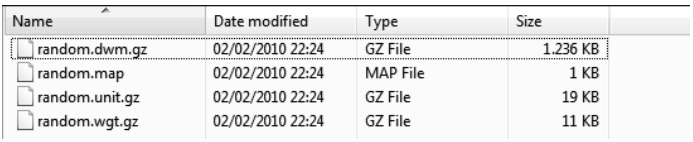
Setelah siap untuk melakukan SOMs *training*, dan perintahnya disimpan dalam *file* "**random_growingsom.bat**" sebagai beikut:

Gambar 8.1 *File* "random_growingsom.bat"

Tekan ENTER, dan proses *training* akan dijalankan.

```
C:\SOMToolbox>random_growingsom

C:\SOMToolbox>somtoolbox GrowingSOM
datatext\random.prop
.....
      --> Finished, took 1.6 seconds
02 Feb 10 22:24:40 at.ec3.somtoolbox.models.
GrowingSOM main
INFO: finishedGrowingSOM
C:\SOMToolbox>
```

1. Keberhasilan *training* yang dilakukan adalah terbentuknya *file output* pada direktori yang telah ditentukan pada *file* properti "chainlink.prop" yaitu pada folder "random". Folder "**random**" yang ditentukan pada "**random.prop**" relatif dan dalam hal ini folder tersebut dibuat dibuat di dalam folder "C:\SOMToolbox\datatext\random". Di dalam folder tersebut selama proses SOMs *training* dibuat empat buah *file* dengan nama depan (*name prefix*) yang telah ditentukan pada "**chainlink.prop**". Keempat buah *file* tersebut adalah sebagai berikut:



Name	Date modified	Type	Size
random.dwm.gz	02/02/2010 22:24	GZ File	1.236 KB
random.map	02/02/2010 22:24	MAP File	1 KB
random.unit.gz	02/02/2010 22:24	GZ File	19 KB
random.wgt.gz	02/02/2010 22:24	GZ File	11 KB

Gambar 8.2 Tampilan Empat *file* dengan Nama Depan *(Name Prefix)* yang Telah Ditentukan pada *"chainlink.prop"*.

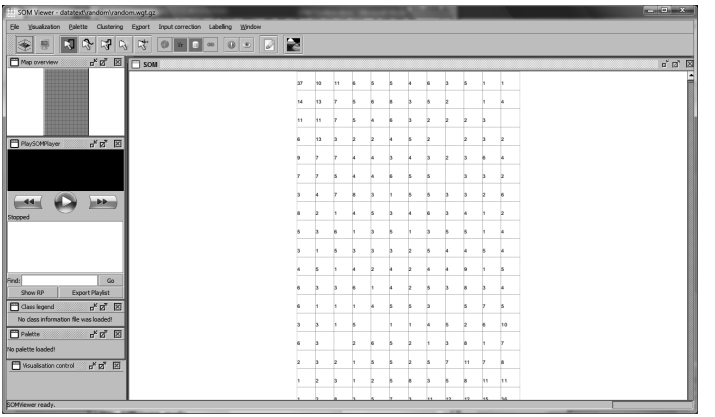
2. Dengan terbentuknya empat buah *file* tersebut, maka hasil dapat dilihat dengan menjalankan SOMViewer menggunakan perintah yang dalam hal ini perintah tersebut akan disimpan pada sebuah *file* berekstensi "**.bat**" yaitu "**random_somviewer.bat**" sebagai berikut:

Gambar 8.3 *File* "random_somviewer.bat"

Selanjutnya untuk menjalankan SOMs Viewer, *file* "**random_somviewer.bat**" dieksekusi dan proses inisialisasi sebelum menampilkan SOMViewer akan dijalankan sebagai berikut:

```
C:\SOMToolbox>random_somviewer

C:\SOMToolbox>somtoolbox SOMViewer -u
datatext\random\random.unit.gz -w datatext
\random\random.wgt.gz --dw datatext\
random\random.dwm.gz
.....
        --> Finished, took 1.39 seconds
02 Feb 10 22:35:50 at.ec3.somtoolbox.apps.
viewer.SOMViewer createAndShowGUI
INFO: Initializing GUI ...
02 Feb 10 22:35:50 at.ec3.somtoolbox.apps.
viewer.SOMViewer createVisualizationMe
nu
INFO: SOMViewer ready.
```

Selanjutnya SOMViewer akan ditampilkan seperti gambar di bawah ini.



Gambar 8.4 SOMViewer Proses *Training*

BAB IX

SOMLib *Digital Library*

IfS Department of Software Technology (1999) mengemukakan SOMLib *Digital Library* merupakan suatu pengembangan sistem perpustakaan digital yang mendukung penjelajahan koleksi dokumen yang intuitif dan ramah pengguna. SOMLib *Digital Library* didasarkan pada Self-Organizing Map (Kohonen, 1995). Perpustakaan digital yang dimaksud secara fisik dapat berupa informasi apa saja (html, *email*, dokumen terformat (pdf, ps, doc, xls, ppt, dan lain-lain.

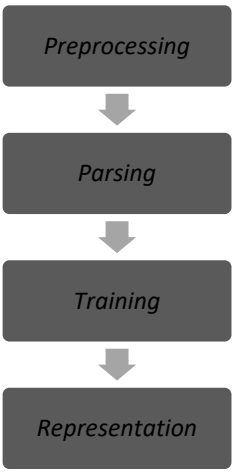
Metodologi yang akan disampaikan di sini adalah cara membangun sebuah sistem perpustakaan digital yang tidak hanya mengoleksi informasi-informasi tersebut, tetapi merupakan sistem yang mampu memberikan kemudahan untuk mengakses atau mencari informasi tersebut. Dengan kata lain sistem tersebut mampu untuk mengetahui fitur-fitur yang ada pada setiap informasi dan mengelompokkan setiap informasi yang memiliki kemiripan fitur.

SOMLib yang digunakan memiliki berbagai modul yang diperlukan untuk mengombinasikan informasi-informasi tersebut satu demi satu. Sebagai data digunakan kumpulan koleksi demo yang bisa diunduh pada:

http://www.ifs.tuwien.ac.at/~andi/somlib/download/download/democollection.tar.gz

Demo tersebut terdiri dari 50 dokumen yang informasinya cukup singkat, tetapi cukup mewakili untuk digunakan sebagai sampel data.

Alur proses pengolahan data dalam aplikasi SOMLib ini adalah sebagai berikut:

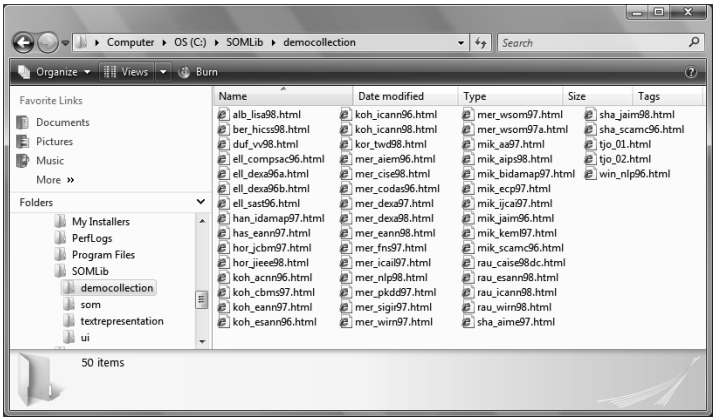


Gambar 9.1 Alur Proses Pengolahan Data dalam Aplikasi SOMLib

9.1 *Preprocessing*

Langkah ini digunakan untuk mendapatkan pengelompokan dokumen teks berdasarkan topik di perpustakaan. Beberapa langkah *preprocessing* dapat dilakukan untuk meningkatkan kualitas data itu sendiri. Beberapa langkah-langkah ini, seperti penghapusan format instruksi, bahasa yang benar-benar independen, dll. Jika dokumen dalam ASCII yang sudah bersih, langkah ini dapat dilewati dan melanjutkan ke proses *parsing*.

Pada bagian ini dokumen dalam format HTML yang diekstrak dari *file* "democollection.tar.gz" dan disimpan di dalam direktori "C:\SOMLib\democollection".



Gambar 9.2 Dokumen Format HTML yang Diekstrak dari *File* "democollection.tar.gz"

Format penghapusan informasi untuk membuat SOM mengklaster dokumen menurut isi, bukan oleh *markup-tag*, maka format kata kunci perlu dihapus dari dokumen. Pada *file* HTML dibutuhkan beberapa program yang dapat membersihkan *file* dari semua HTML-tag. Ini dapat dilakukan dengan menggunakan beberapa cara sebagai berikut:

1. Menggunakan modul *preprocess* Java somlib.ui.Bookmarks.
2. Menggunakan *script* yang tersedia yaitu html2txt konverter. *Script* tersebut merupakan *Linux script* dan dijalankan pada sistem operasi *Unix like* seperti *distro linux*, *bsd*, atau *unix*.
3. Alternatif lain adalah membuat *script* sendiri untuk menghilangkan HTML-tag. Cara ini lebih rumit karena harus membuat logika pemrograman yang benar dan bisa memakan waktu berhari-hari.

9.1.1 Preprocessing dengan Java somlib.ui.Bookmarks

Modul *preprocess* "Bookmarks" dipanggil dengan Java somlib.ui.Bookmarks. Pada prinsipnya modul ini bekerja dengan mengambil informasi dari sebuah *file bookmark* yang bisa merupakan format *bookmark* dari sebuah *browser*. *Bookmark* ini harus diisi dengan informasi penanda-*file* yang diberikan (dan sesuai dengan mode), semua *link*

akan diunduh ke dalam *buffer*-direktori (di *Intermediate*) dan kemudian dipecah ke *plain-text* (di dalam direktori yang harus ditetapkan sebagai *output*. *File* yang ditandai ini merupakan *file* input yang tentu saja adalah merupakan *file* HTML.

Pada *preprocessing* Java somlib.ui.Bookmarks ini ada empat mode yang berbeda yang tersedia:

1. ***Clear***: direktori akan dibersihkan dari semua *file* dan *file* baru yang diunduh yang akan disimpan di sana.
2. ***Overwrite***: ada *file* dengan nama yang sama, unduh *file* yang ditimpa, yang lain tetap seperti apa adanya.
3. ***Add new***: tidak ada *file* yang dihapus, tidak ada yang ditimpa, hanya *file* baru ditambahkan.
4. ***Use present***: penunjuk-*file* yang tidak digunakan (dengan demikian parameter tidak akan ditetapkan), hanya dalam *buffer*-direktori *file* yang sudah ada digunakan.

Sebagai opsi tambahan, sebuah deskripsi *file* vektor dapat ditulis. Hal ini bisa dilakukan dengan mendefinisikan sebuah direktori di "VecDes" dalam "Options"-blok.

Untuk menjalankan somlib.ui.Bookmarks, jalankan perintah sebagai berikut.

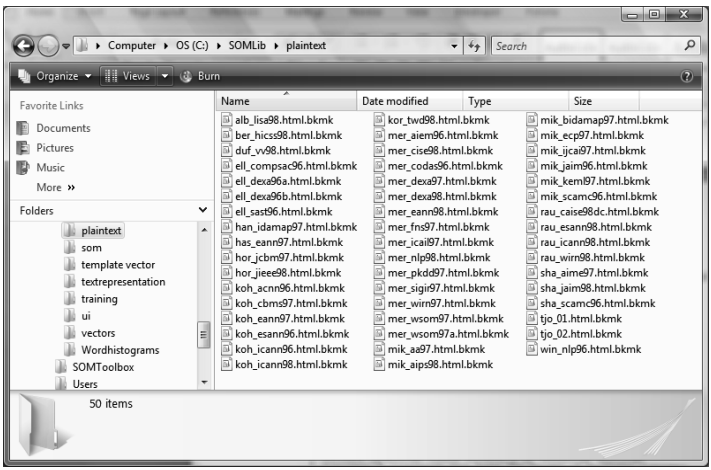
```
C:\>java somlib.ui.Bookmarks
```

Tujuan *preprocessing* ini adalah membuat data kumpulan dokumen dalam format HTML dijadikan *file plaintext* dengan menghapus Informasi untuk membuat SOM mengklaster dokumen menurut isi, bukan oleh *markup-tag*, maka format kata kunci perlu dihapus dari dokumen.

Untuk memulai *preprocessing* "Bookmarks", langkah-langkah yang dilakukan adalah sebagai berikut:

- Isi kolom "Buffer Directory" dengan "C:\SOMLib\democollection", yang merupakan lokasi koleksi *file* HTML berada.
- Isi kolom "Output Plaintext Directory" dengan "C:\SOMLib\plaintext", yang nantinya akan digunakan sebagai tempat *file* hasil *preprocessing*. *File* ini nantinya akan memiliki nama sama dengan *file* HTML asli, tetapi diberikan tambahan ekstensi ".bkmk".
- Isi kolom *option* pada pilihan "Mode" dengan "*use present*" sebab *file* yang akan diproses telah ada dan diletakkan pada "Buffer Directory".
- Isi kolom *debug option* "Verbosity" dengan nilai sama dengan 2.
- Isi kolom *debug option* "Logfile" dengan "C:\SOMLib\log\log".

Hasil pemrosesan pada folder "C:\SOMLib\plaintext" akan tampak sebagai berikut.



Gambar 9.3 Hasil Pemrosesan pada Folder "C:\SOMLib\plaintext"

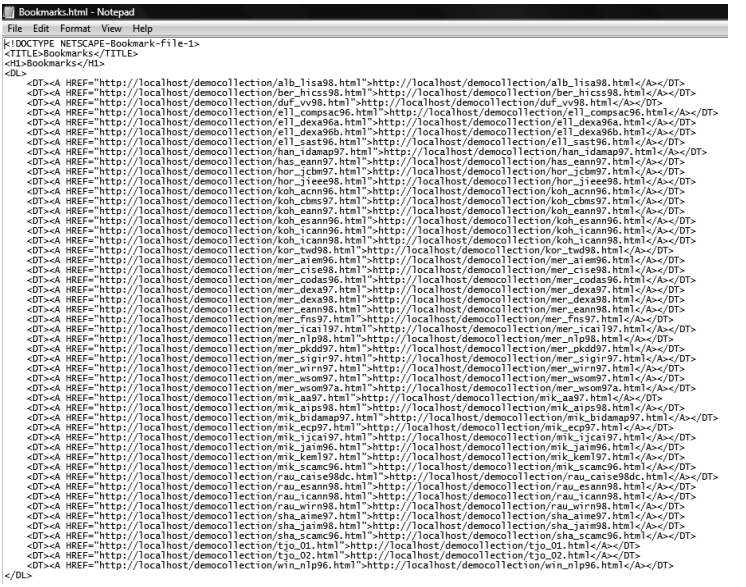
Jika file yang akan diproses berada pada lokasi remot (misal di internet atau pada sebuah *sharing folder* pada jaringan), sebelum memulai Java somlib.ui.Bookmarks, sebuah *file bookmarks* harus dibuat. Pembahasan tambahan ini hanya untuk meyakinkan bahwa istilah "*bookmark*" sebagai fitur "java somlib.ui.Bookmarks" penggunaannya cukup luas.

Sebagai contoh lokasi remot adalah *web server local* pada komputer, dan koleksi *file* html diletakkan pada folder "C:\inetpub\wwwroot\democollection", dengan"C:\inetpub\wwwroot" merupakan *root directory web server local*.

Langkah tambahan yang perlu dilakukan adalah sebagai berikut:

- Isi kolom "Buffer Directory" dengan direktori yang pada awalnya tentu saja masih kosong.
- Isi kolom input "Bookmark File" dengan sebuah *file bookmarks* (bisa dengan seperti *netscape bookmark*).
- Isi kolom *option* pada pilihan "Mode" selain "*use present*". Hal ini dilakukan karena *file* yang akan diproses belum dimiliki nanti selama proses akan diunduh pada "Buffer Directory".

Untuk kasus lokasi remot seperti tersebut di atas, contoh sebuah *file bookmark* yang telah dibuat pada lokasi "C:\SOMLib\Bookmarks.html" adalah sebagai berikut:



Gambar 9.4 Contoh Sebuah *File Bookmark* yang Telah Dibuat pada Lokasi C:\SOMLib\Bookmarks.html

Sebelum menjalankan java somlib.ui.Bookmarks untuk lokasi remote pada sistem operasi Windows, maka aplikasi "*wget for windows*" yang bisa diunduh pada alamat "http://downloads.sourceforge.net/gnuwin32/wget-1.11.4-1-setup.exe" harus telah terinstal, dan tambahan *path directory* harus diarahkan pada lokasi *file* "wget.exe" (Biasanya pada folder "C:\Program Files\GnuWin32\bin".

9.1.2 *Preprocessing* dengan *Script* html2txt

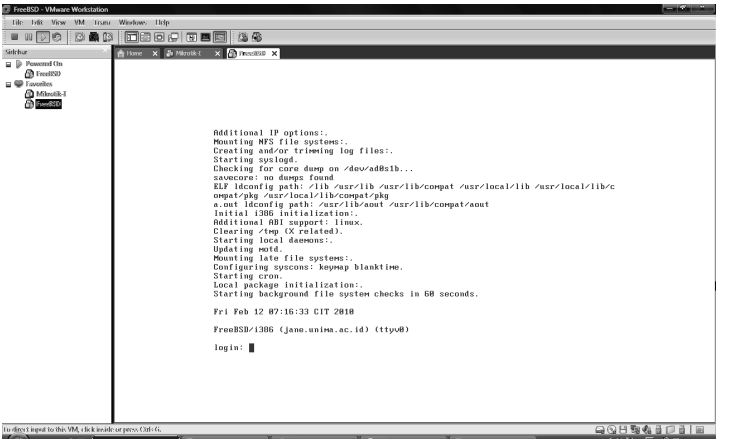
Pada *file* HTML dibutuhkan beberapa program yang dapat membersihkan *file* dari semua HTML-tag. Ini dapat dilakukan menggunakan *script* yang tersedia yaitu html2txt konverter. *Script* tersebut merupakan *Linux script* dan tidak dapat digunakan pada lingkungan sistem operasi Windows.

Alternatif lain adalah membuat *script* sendiri untuk menghilangkan HTML-tag. Ini akan menjadi lebih rumit karena harus membuat logika pemrograman yang benar dan bisa memakan waktu berhari-hari.

Pilihan yang digunakan adalah menginstal *Virtual Machine* (VM) yaitu VMWare pada sistem operasi Windows. Sistem operasi yang diinstal pada VM adalah FreeBSD versi 7.2 dengan konfigurasi minimal dan menambahkan *package linux compatibility*, *perl*, *wget*, *bash shell*.

Pada dasarnya ide yang akan diterapkan adalah melakukan *preprocessing* dengan menggunakan skrip-skrip yang telah tersedia pada FreeBSD, dan proses selanjutnya pada Windows. Oleh karena itu, pada instalasi FreeBSD tidak menginstal Java (JDK6u) maupun Windows *environment*.

FreeBSD telah diinstal pada VMWare yang memakan waktu hanya sekitar 10 menit dan hasil instalasi tampak seperti pada gambar di bawah ini.



Gambar 9.5. FreeBSD yang Telah Diinstal pada VMWare

Terlebih dahulu dipersiapkan *file* yang dibutuhkan dalam *preprocessing* yaitu SOMLIB.tar.gz, democollection.tar.gz dan html2txt. Selanjutnya ekstrak *file* kompresi "SOMLIB.tar.gz" dan pindahkan *file* hasil ekstraksi "SOMLIB.tar.gz" ke folder kerja yang diinginkan. Kemudian pindahkan ke folder "/usr/somlib". Setelah itu buat folder "/usr/somlib/democollection" yang akan digunakan untuk menampung *file* ekstraksi "democollecton.tar.gz". Dilanjutkan dengan ekstrak *file* "democollection.tar.gz" ke folder "/

usr/somlib/democollection". Buat folder "/usr/somlib/program" untuk meletakkan *file script* "html2txt". *File* "html2txt" adalah *file executable*, dan harus dipastikan bahwa *executable permission* diizinkan. Jika tidak, perlu tambahkan *permission*-nya.

Jika *file* yang diperlukan telah tersedia, tahapan *preprocessing* siap untuk dilakukan. Namun terlebih dahulu dipersiapkan direktori untuk menyimpan *file* hasil *preprocessing*. Di sini *file* hasil *preprocessing* tersebut akan disimpan dalam folder "/usr/somlib/files_cleaned".

Sebelum melakukan *preprocessing* sebaiknya terlebih dahulu *file* tersebut sebelum dilakukan *preprocessing*, dan nanti akan dibandingkan setelah pada *file* tersebut dilakukan *preprocessing*.

Ketika akan melakukan *preprocessing*, pastikan *file* berada pada direktori "/usr/somlib/democollection" untuk memudahkan mengeksekusi *script*. FreeBSD menggunakan csh sebagai *default shell*, dan pesan kesalahan tersebut terjadi karena permasalahan PATH. Agar lebih mudah proses dapat dilakukan dengan beralih saja ke *bash shell*.

Sebaiknya proses diulangi beberapa kali untuk benar-benar menghilangkan tag HTML yang bersarang. Dapat juga menggunakan *pipe* (|) melalui perintah dua kali, sebelum akhirnya mendapatkan *file* yang bersih dan hanya berupa teks ASCII. Hasil dapat dilihat pada *file* "alb_lisa98.html" setelah dilakukan *preprocessing*.

Proses pembentukan *plain text* kini telah selesai, dan hasilnya akan diproses lebih lanjut menggunakan SOMLib Java Package pada sistem operasi Windows. Terlebih dahulu *file* tersebut harus disimpan pada *removable media* atau bisa diunggah pada FTP *server*. Keduanya (FreeBSD dan Windows) secara fisik adalah komputer yang sama, tetapi secara logika mereka adalah dua buah mesin yang berbeda.

9.2 Text Representation

Agar dokumen secara otomatis dapat diatur menurut isi, sebuah representasi numerik dari berbagai dokumen dalam koleksi perlu dibuat. Perwakilan ini diciptakan oleh pengindeksan teks lengkap dokumen. Daftar semua kata yang terdapat dalam koleksi dokumen, yang disebut *vector template*, perlu diciptakan.

Untuk mendapatkan representasi isi yang lebih baik, kata-kata dapat dikurangi misalnya dengan menghilangkan akhiran yang paling umum, seperti '-ed', '-ing' atau jamak (yaitu trailing) '-s'. Harap diperhatikan, bahwa proses pembuatan vektor ini, tidak dilakukan secara manual tetapi harus memungkinkan untuk pemrosesan secara otomatis dari beragam koleksi dokumen.

Sebagai hasil dari proses pembuatan daftar kata, yaitu "*template vector*" umumnya terdiri dari sejumlah kata yang sangat besar (sekitar 100.000 kata-kata, tergantung pada koleksi dokumen), daftar ini harus dipangkas untuk membuat himpunan bagian yang relevan untuk isi yang berbasis pada klasifikasi dokumen.

Hal ini dapat dicapai dengan menghapus semua kata yang muncul dalam jumlah yang sangat besar (katakanlah lebih dari 90%) dari dokumen, dan kata-kata ini tidak memberikan kontribusi yang spesifik terhadap isi dokumen. Hal ini biasanya termasuk kata-kata yang sering digunakan seperti *tanda berhenti*, *seperti artikel*, *kata ganti*, dll. (*the*, *is*, *are*, *he*, *she*), tetapi juga koleksi kata-kata yang sangat spesifik seperti *komputer* dalam sebuah koleksi dokumen hanya meliputi topik-topik yang berkaitan dengan komputer. Umumnya, jumlah kata yang dihapus oleh kriteria ini agak kecil (~ 10–50).

Lebih jauh lagi, dilakukan penghapusan semua kata yang muncul dalam dokumen hanya sedikit, karena hanya akan memberikan perbedaan yang sangat kecil di antara masing-masing dokumen. Hal ini tidak berguna ketika yang diinginkan adalah mengatur dokumen keseluruhan, yakni mengelompokkannya sesuai dengan materi.

Dengan menghapus kata-kata yang kurang dari, katakanlah, tiga dokumen, kesalahan ejaan juga dapat disingkirkan. Pengurangan ini biasanya mengurangi daftar kata cukup banyak, sehingga daftar yang ada menjadi sekitar 5.000–15.000 kata.

Daftar kata yang dihasilkan adalah *template vector* yang telah dipangkas untuk masing-masing koleksi dokumen.

Pada langkah kedua, sebuah deskripsi vektor mengikuti *template vector* untuk masing-masing koleksi dibuat untuk setiap dokumen. Pada dasarnya, setiap dokumen dijelaskan oleh kata-kata yang terjadi di dalamnya. Representasi dokumen dapat disederhanakan menjadi indikasi biner dari kenyataan apakah kata tersebut ada atau tidak, yang mengarah ke vektor sesuai representasi dari 0 atau 1. Model ini biasanya disebut sebagai representasi biner.

Representasi yang lebih canggih dapat diperoleh dengan menghitung jumlah kejadian dari setiap kata dalam *template vector* pada setiap dokumen, menghasilkan kata-histogram perwakilan untuk setiap dokumen. Representasi ini juga disebut sebagai representasi frekuensi istilah (tf).

Berbagai langkah yang berbeda dapat digunakan untuk menentukan pentingnya setiap kata dalam setiap dokumen berdasarkan representasi dokumen frekuensi istilah. Pilihan yang paling umum untuk melakukan ini adalah apa yang disebut representasi *term frequency* dikalikan *inverse document frequency* (tf x idf), dalam hal ini pentingnya setiap kata dinilai berdasarkan jumlah dokumen yang ada. Sebuah kata adalah lebih penting, semakin sering itu terjadi dalam satu dokumen dan semakin sering itu terjadi pada koleksi seperti itu. Sangat sering kata-kata, yang muncul dalam hampir semua dokumen (tapi terlalu jarang untuk dihapus selama proses pengurangan vektor) dengan demikian menerima bobot yang sangat rendah.

Dalam bentuk yang sederhana, representasi tf x IDF untuk setiap kata dapat diperoleh dengan membagi frekuensi istilah kata dalam

suatu dokumen oleh frekuensi, yaitu jumlah kata-kata ini muncul dalam dokumen. Namun, saat ini sebagian besar sistem menggunakan versi representasi tf x IDF yang sederhana, pembobotan setiap kata dikalikan dengan frekuensi istilah dengan logaritma alami (N/df), dengan N adalah jumlah dokumen dalam koleksi dan df adalah frekuensi dokumen, yaitu jumlah dokumen di mana setiap kata tersebut muncul.

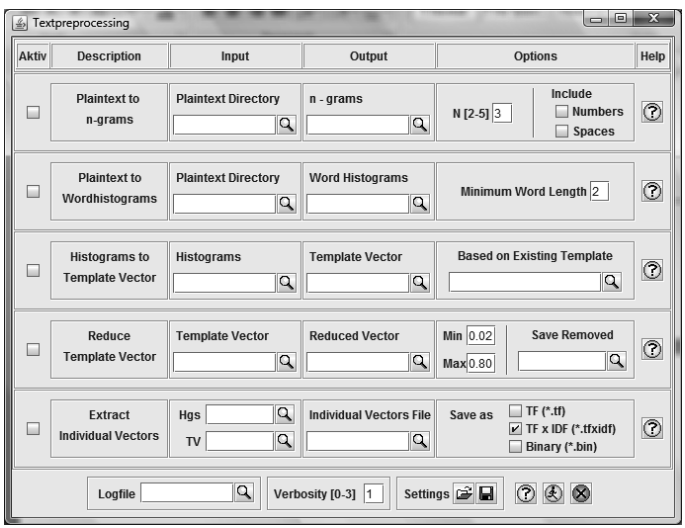
9.3 *Parsing*

Proses *parsing* fitur akan menciptakan vektor yang menggambarkan isi dari dokumen. Di sini akan digunakan program ekstraksi fitur dari paket Java SOMLib untuk mendapatkan fitur vektor.

Buka Windows *command prompt*, dan somlib.ui.Parser Java *parser* akan mulai dieksekusi.

```
C:\Users>cd \

C:\>java somlib.ui.Parser
_
```

Maka sebuah jendela akan tampil seperti pada gambar di bawah ini.



Gambar 9.6 Hasil Eksekusi somlib.ui.Parser Java Parser

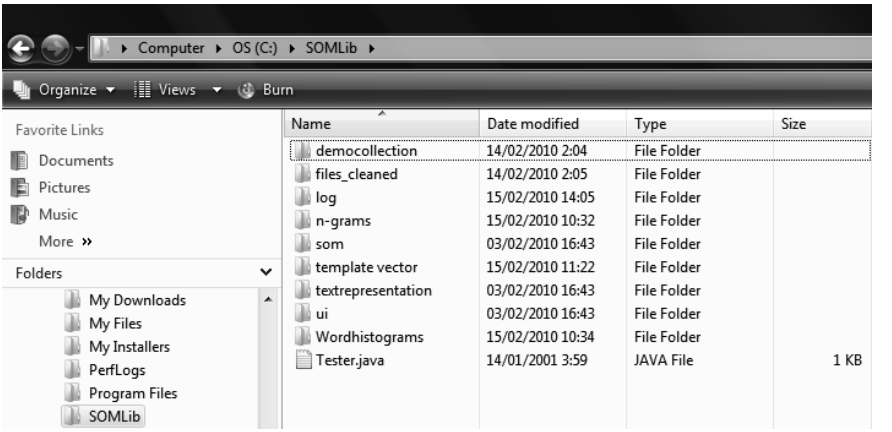
somlib.ui.Parser Java Parser terdiri dari lima modul. Modul-modul ini dapat diaktifkan dengan memilih tombol *check button* di sebelah kiri dari setiap modul.

Pada jendela somlib.ui.Parser ada enam kolom yang masing-masing memiliki fungsi sebagai berikut:

- Kolom 1 (*Aktiv*), digunakan untuk mengaktifkan modul yang akan diproses.
- Kolom 2 (*Description*), hanya merupakan deskripsi singkat tentang modul.
- Kolom 3 (*Input*), merupakan variabel input yang harus diisi jika modul akan diaktifkan dan inputan merupakan *variable string* yang menunjukkan lokasi direktori/*file*. Direktori/*file* ini *text processing* dijalankan akan diambil informasinya, tetapi tidak akan mengalami perubahan.
- Kolom 4 (*Output*), merupakan *variable output* yang harus diisi jika modul akan diaktifkan dan merupakan *variable string* yang menunjukkan lokasi direktori/*file* hasil pemrosesan nantinya. Jelas bahwa direktori yang tadinya kosong akan terisi dengan *file* hasil pemrosesan atau *file* yang sudah ada akan mengalami perubahan.
- Kolom 5 (*Option*)
- Kolom 6 (*Help*), merupakan tombol yang jika diklik akan menampilkan informasi yang lebih rinci tentang modul dan penggunaannya.

Pada baris bawah dari somlib.ui.Parser terdapat kumpulan variabel *logfile* dan *verbosity* sebagai parameter *processing* dan tombol-tombol untuk bantuan, memuat/menyimpan konfigurasi, menjalankan *text processing* (*Run*) dan tombol untuk keluar program. Parameter *verbosity* yang diatur sama dengan 2 sesuai yang dianjurkan pada "Creating a SOMLib Digital Library A Step-by-Step Guide".

Plain text hasil *preprocessing* untuk menghilangkan HTML *tag* yang sebelumnya telah diproses pada FreeBSD telah tersedia. Hal lain yang perlu dipersiapkan adalah direktori dan *file* untuk hasil *text processing*. Di sini berturut-turut dibuat direktori baru di bawah folder "C:\SOMLib" yaitu: "n-grams", "Wordhistograms", "template vektor" dan "log" seperti pada gambar di bawah ini.



Gambar 9.7 n-grams, Wordhistograms, *Template Vector* dan Log

Pada setiap proses *parsing* isikan inputan *log file* dengan "C:\SOMLib\log\log", yang berguna untuk menyimpan informasi selama aktivitas somlib.ui.Parser dijalankan. Ini akan bermanfaat untuk mengetahui penyebab jika terjadi suatu kegagalan.

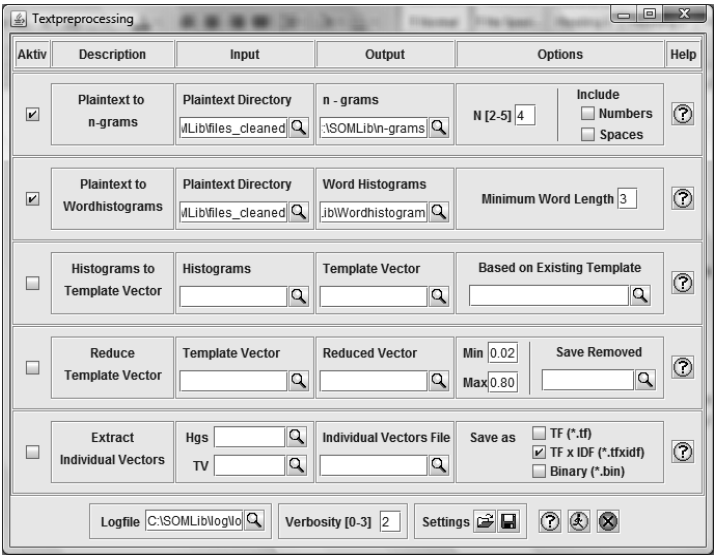
9.3.1 *Parsing* Tahap 1

Dua modul pertama memiliki fungsi yang sama: *plain-text-file* yang diterjemahkan ke *histogram-file*. "Plaintext untuk Wordhistograms"' mengambil *file* dalam "Direktori plaintext" dan transfer mereka ke direktori "Word Histogram". *File* baru adalah Hashtables serial yang mengandung kata-kata tunggal dan jumlah pemunculan mereka. Dari sini suatu "Minimum Word Length" dapat didefinisikan.

Setelah semuanya siap, modul yang pertama dan kedua akan dijalankan. Langkah-langkahnya adalah sebagai berikut:

- Pada modul "Plaintext to n-grams", isi input "Plaintext Directory" dengan "C:\SOMLib\files_cleaned" yang merupakan lokasi data *plain text*.
- Masih pada modul "Plaintext to n-grams" isi *output* "n-grams" *directory* dengan "C:\SOMLib\n-grams" yang merupakan lokasi di mana *file* hasil *text processing* nantinya akan diletakkan.
- *Option* untuk modul "Plaintext to n-grams" ada dua. *Option* yang pertama merupakan parameter untuk membuat *hashtable* (**N**) yang dapat diisi angka integer 2–5. Di sini nilai yang diberikan adalah 4. Nilai ini akan memberikan konsekuensi penulisan "Plaintext", akan diterjemahkan menjadi: *plai*, *lain*, *aint*, *inte*, *ntex*, *text* menggunakan n-grams (untuk n=4). *Option* kedua akan memberitahukan program apakah angka atau spasi akan diproses. Karena hanya mempedulikan kata saja, *option* ini tidak dipilih.
- Pada modul "Plaintext to Wordhistograms" isi input "Plaintext Directory" dengan "C:\SOMLib\files_cleaned" yang merupakan lokasi data *plaintext* yang dimiliki.
- Masih pada modul "Plaintext to Wordhistograms" isi *output* "Word Histograms" *directory* dengan "C:\SOMLib\ Wordhistograms" yang merupakan lokasi dimana *file* hasil *text processing* nantinya akan diletakkan.
- *Option* untuk modul "Plaintext to Wordhistograms" adalah "Minimum Word Length" yang diatur sama dengan tiga sesuai dengan yang direkomendasikan pada "Creating a SOMLib Digital Library A Step-by-Step Guide"
- Pada kolom paling kiri, aktifkan modul "Plaintext to n-grams" dan "Plaintext to Wordhistograms"

Hasil akhir pada jendela somlib.ui.Parser akan tampak seperti gambar di bawah ini.



Gambar 9.8 Hasil Akhir pada Jendela somlib.ui.Parser pada *Parsing* 1

Untuk memudahkan agar dapat menjalankan ulang somlib. ui.Parser dengan konfigurasi seperti tersebut di atas, konfigurasi tersebut disimpan pada sebuah *file* yang diberi nama "setting_12".

Setelah tahapan tersebut dilakukan *text processing* modul 1 dan 2 siap untuk dijalankan.

Hasil pemrosesan pada folder "C:\SOMLib\Wordhistograms" maupun pada folder "C:\SOMLib\n-grams". Untuk selanjutnya histogram yang digunakan adalah hasil pemrosesan yang terdapat pada folder "C:\SOMLib\Wordhistograms".

9.3.2 *Parsing* Tahap 2

Pada tahap ini histogram akan diterjemahkan ke *template vector* yang merupakan penggabungan *file* yang dibuat di langkah terakhir menjadi sebuah *file* tunggal "Template Vector".

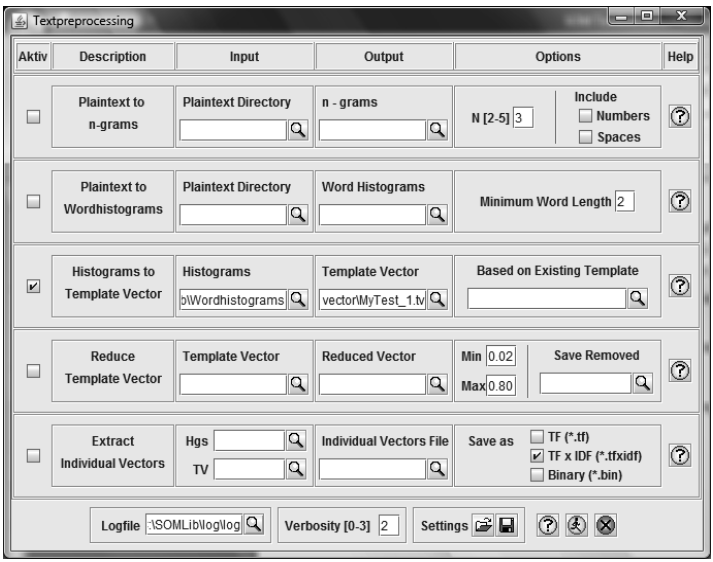
Langkah-langkahnya adalah sebagai berikut:

- Pada modul "Histograms to Template Vector", isi input "Histograms" direktori dengan "C:\SOMLib\

Wordhistograms" yang merupakan lokasi data histogram pada *parsing* tahap 1.

- Masih pada modul "Histograms to Template Vector" isi *output* "Template Vector" direktori dengan "C:\SOMLib\MyTest_1.tv" yang merupakan *file template vector* yang merupakan hasil penggabungan dari penerjemahan *file* histogram.
- *Option* untuk modul "Histograms to Template Vector" yaitu "Based on existing Template" dibiarkan tetap kosong.
- Pada kolom paling kiri, pastikan hanya modul "Histograms to Template Vector" yang diaktifkan.

Hasil akhir pada jendela somlib.ui.Parser akan tampak seperti gambar di bawah ini.



Gambar 9.9 Hasil Akhir pada Jendela somlib.ui.Parser pada *Parsing* 2

Untuk memudahkan agar dapat menjalankan ulang somlib.ui.Parser dengan konfigurasi seperti tersebut di atas, konfigurasi tersebut disimpan pada sebuah *file* yang diberi nama "setting_3". Setelah hal tersebut *text processing* modul 3 telah siap dilakukan.

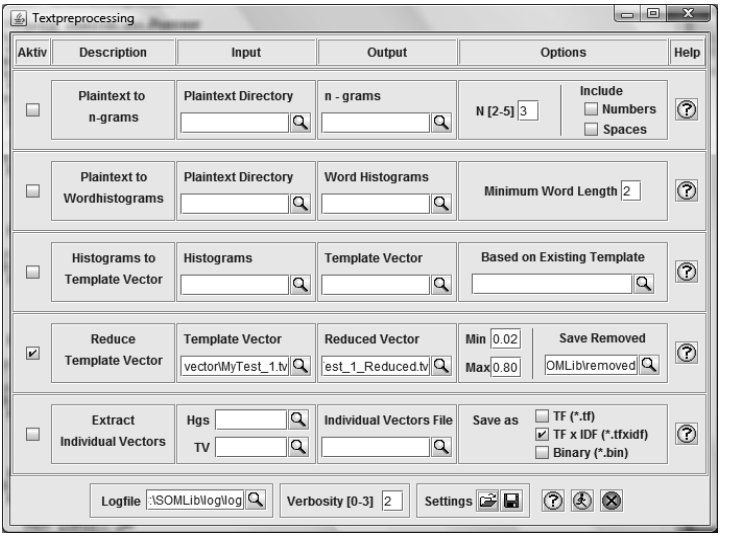
9.3.3 *Parsing* Tahap 3

Pada tahap ini "Template Vector" akan dikurangi menjadi "Reduced Vector" dengan menetapkan minimum dan maksimum pemunculan dari sebuah kata (atau n-gram masing-masing kata). Angka-angka tersebut merupakan persen dengan nilai antara 0 dan 1, dalam hal ini tanda titik menunjukkan titik desimal (misal 0.02–0.8).

Langkah-langkahnya adalah sebagai berikut:

- Pada modul "Reduced Template Vector", isi input "Template Vector" direktori dengan "C:\SOMLib\template vector\ MyTest_1.tv" yang merupakan lokasi *file template vector* pada *parsing* tahap 2.
- Masih pada modul "Reduced Template Vector" isi *output* "Reduced Vector" *file* dengan "C:\SOMLib\MyTest_1_ Reduced.tv" yang merupakan *file template vector* yang merupakan hasil penggabungan dari penterjemahan *file* histogram.
- Option untuk modul "Reduced Template Vector" ada dua. *Option* pertama sebagai isian untuk mendefinisikan nilai minimum dan maksimum pemunculan dari sebuah kata. Disini 0.02 ditentukan sebagai batas minimum dan 0.8 sebagai batas maksimum. *Option* kedua jika diinginkan dapat digunakan untuk menetapkan lokasi dari kata-kata yang dihapus. *File* selanjutnya akan simpan pada lokasi "C:\ SOMLib\removed".
- Pada kolom paling kiri, pastikan hanya modul "Reduced Template Vector" yang diaktifkan.

Hasil akhir pada jendela somlib.ui.Parser akan tampak seperti gambar di bawah ini.



Gambar 9.10 Hasil Akhir pada Jendela somlib.ui.Parser pada *parsing* 3

Untuk memudahkan agar dapat menjalankan ulang somlib. ui.Parser dengan konfigurasi seperti tersebut di atas, konfigurasi tersebut disimpan pada sebuah *file* yang diberi nama "setting_4".

Text processing modul 4 telah siap untuk dilakukan.

9.3.4 *Parsing* Tahap 4

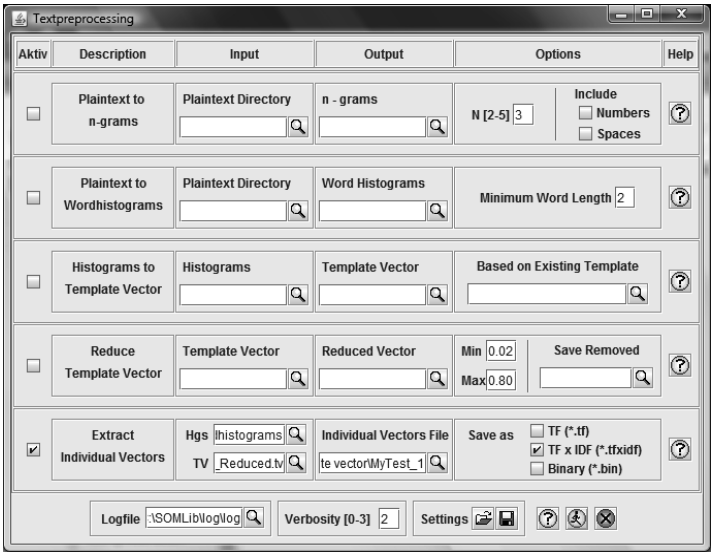
Pada tahap ini akan dilakukan ekstraksi individual *vectors* yakni "Reduced Template Vector" dan berbagai histogram *file* harus disesuaikan. Hasilnya adalah *vector file* yang merupakan sebuah *file* berekstensi ".tf", "*.bin" atau "*.tfxidf".

Langkah-langkahnya adalah sebagai berikut:

- Pada modul "Extract Individual Vectors", isi input "Hgs" direktori dengan "C:\SOMLib\Wordhistograms" yang merupakan lokasi *file* histogram pada parsing tahap 1.
- Isi input "TV" *file* dengan "C:\SOMLib\template vector\ MyTest_1_Reduced.tv" yang merupakan *file template vector* pada *parsing* tahap 3.

- Masih pada modul "Extract Individual Vectors" isi *output* "*Individual Vectors File*" dengan "C:\SOMLib\template vector\MyTest_1" yang merupakan *file vector output* hasil *text processing*.
- *Option* untuk modul "Extract Individual Vectors" adalah pilihan ekstensi untuk *file output*. Di sini ekstensi yang dipilih adalah "*.tfxidf" sesuai *default*.
- Pada kolom paling kiri, pastikan hanya modul "Extract Individual Vectors" yang diaktifkan.

Hasil akhir pada jendela somlib.ui.Parser akan tampak seperti gambar di bawah ini.



Gambar 9.11 Hasil Akhir pada Jendela somlib.ui.Parser pada *Parsing* 4

Untuk memudahkan agar dapat menjalankan ulang somlib. ui.Parser dengan konfigurasi seperti tersebut di atas, konfigurasi tersebut disimpan pada sebuah *file* yang diberi nama "setting_5". Setelah itu *text processing modu*l 5 telah siap djalankan.

9.4 SOM Modul *(Training)*

Penggunaan SOM modul ini menggunakan perintah *command line*. Ada beberapa langkah yang perlu dilakukan dan dalam setiap langkah *verbosity* dapat diatur menggunakan nilai 0–3. Dalam percobaan ini menggunakan nilai *verbosity* sama dengan 2.

9.4.1 Langkah 1 - *Preprocessing*

Syntax perintah untuk melakukan *preprocessing* adalah sebagai berikut.

```
java somlib.som.preprocess.Vec2Vec -i SOM-Input-
Vector_in
[-o SOM-Input-Vector_out] [-n normalize [t/f]] [-v
verbosity]
```

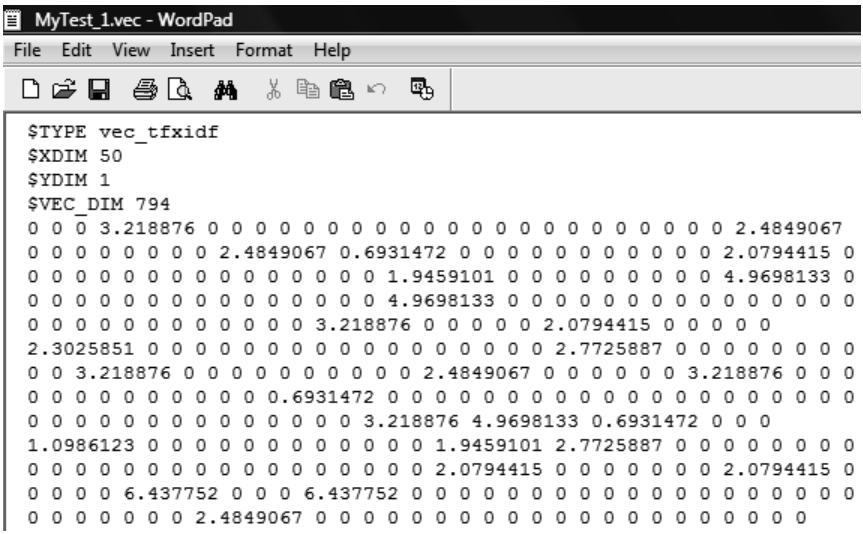
Vector output nantinya akan ditampung dalam sebuah sebuah folder yaitu "C:\SOMLib\vector". Untuk itu maka perlu agar folder tersebut dibuat terlebih dahulu.

```
C:\>mkdir SOMLib\vectors

C:\
```

Selanjutnya perintah *preprocessing* bisa dilakukan.

```
C:\>java somlib.som.preprocess.Vec2Vec -i "SOMLib\template
vector\MyTest_1.tfxid

f" -o SOMLib\vectors\MyTest_1.vec -n f -v 2
    start: initIV.
    end: initIV.
    start converting ...
    done converting.
    start writing to SOMLib\vectors\MyTest_1.vec ...
    writing finished.

C:\>
```

Sebagai hasil dari *preprocessing* ini adalah terbentuknya *file* "MyTest_1.vec, yang jika dilihat menggunakan Wordpad akan tampak sebagai berikut.



Gambar 9.12 Hasil dari *Preprocessing*

9.4.2 Langkah 2 - *Training*

Syntax perintah untuk melakukan *training* adalah sebagai berikut.

```
java somlib.som.Training -t SOM-Input-Vector_in
[-d SOM-Input-Vector-Description_in] -i iterations
-n init_neighborhood [-g neighborhood_name] -l
init_learnrate
[-h learnrate_name] -x XDim -y YDim -f SOM-Map-
Description_out -c SOM-Weight-Vector_out -r SOM-
Node-Description_out
[-s seed for map-init] [-u SOM-Training-File_out]
[-v verbosity]
```

Ini adalah bagian terpenting dari SOM-modul dan pasti yang paling membuang waktu. Menggunakan aplikasi ini yang SOM Map dapat dilatih dengan Vektor Pelatihan disimpan dalam *file vector input* SOM-Input-Vector_in dengan Vector Input Deskripsi.

```
SOM-Input-Vector-Description_in.
```

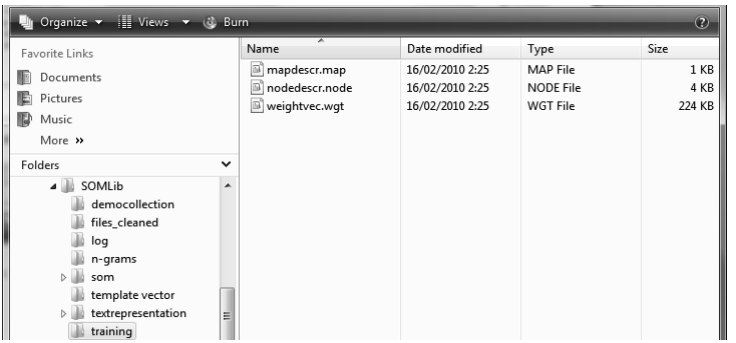
Hasil *training* nantinya akan ditampung dalam sebuah sebuah folder yaitu "C:\SOMLib\training". Untuk itu maka perlu agar folder tersebut dibuat terlebih dahulu.

```
C:\>mkdir SOMLib\training

C:\
```

Selanjutnya perintah *training* bisa dilakukan.

```
C:\>java   somlib.som.Training   -t   "SOMLib\template
vector\MyTest_1.tfxidf" -i 5000
 -n 3.5 -l 0.8 -x 8 -y 3 -f SOMLib\training\mapdescr.
map -c SOMLib\training\weig
htvec.wgt -r SOMLib\training\nodedescr.node -s 20
  Training:
  Map created.
    Training started...
      5000 iterations
    Training finished in 4s.
      Map: storing SOM Weigth Vectors
      Map: storing SOM Unit Descriptions

C:\>
```

Sebagai hasil dari proses *training* ini adalah terbentuknya *file* "mapdescr.map", "weightvec.wgt" dan "nodedescr.node" pada folder "C:\SOMLib\training".



Gambar 9.13 *File* mapdescr.map, weightvec.wgt dan nodedescr.node

9.4.3 Langkah 3 - *Mapping*

Syntax perintah untuk melakukan *mapping* adalah sebagai berikut.

```
java somlib.som.Mapping -m SOM-Map-Description_in
-i
SOM-Input-Vector_in [-d SOM-Input-Vector-
Description_in]
[-f SOM-Map-Description_out] [-c SOM-Weight-Vector_
out]
[-r SOM-Node-Description_out] [-p SOM-Quantisation-
Error-Map_out]
[-n SOM-Mapping-File_out] [-v verbosity]
```

Dalam *mapping*, sebuah *map* yang sebelumnya telah dilatih dimasukkan pada parameter `SOM-Map-Description_in`. Vector file input dapat ditemukan pada `SOM-Input-Vector_in` - dan dijelaskan di `SOM-Input-Vector-Description_in`. Deskripsi *File* SOM *map* yang baru ditulis pada `SOM-Map-Description_out`, SOM *Weight Vector* ditulis pada `SOM-Weight-Vector_out`, deskripsi *node* dari SOM ditulis pada `SOM-Node-Description_out` dan *file* SOM *mapping* ditulis pada `SOM-Mapping-File_out`. Menggunakan perintah ini berulang kali maka *file output* yang lama akan ditimpa. File SOM *Weightvector* tidak akan ditulis dalam kasus ini, karena tetap tidak berubah dalam langkah ini.

Hasil proses *mapping* nantinya akan ditampung dalam sebuah sebuah folder yaitu "C:\SOMLib\mapping". Untuk itu maka perlu agar folder tersebut dibuat terlebih dahulu.

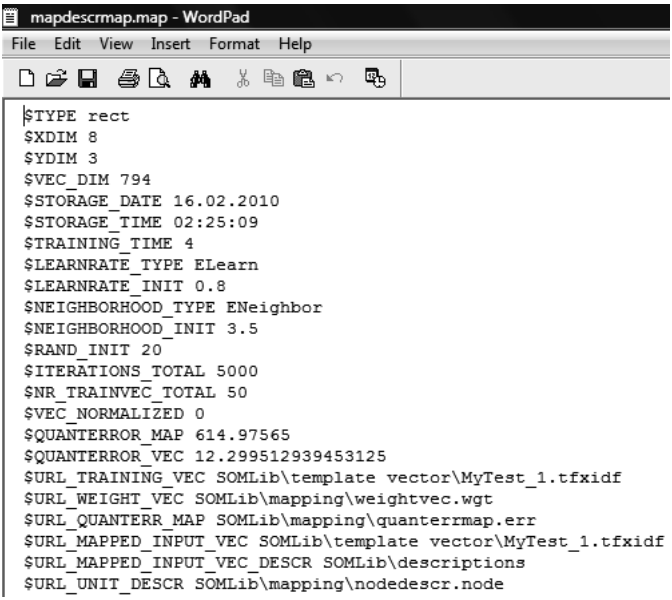
```
C:\>mkdir SOMLib\mapping

C:\
```

Selanjutnya perintah *mapping* bisa dilakukan.

```
C:\>java somlib.som.Mapping -m SOMLib\training\mapdescr.map -i
"SOMLib\template
vector\MyTest_1.tfxidf"  -d  SOMLib\descriptions  -f  SOMLib\
mapping\mapdescrmap.map
 -c SOMLib\mapping\weightvec.wgt -p SOMLib\mapping\quanterrmap.
err -r SOMLib\map
ping\nodedescr.node -n SOMLib\mapping\mapping.mpn
  Map created.
  Mapping:
    ... vectors from file SOMLib\template vector\MyTest_1.tfxidf
    writing map-description
      Map: writing SOM Map Description File
      Map: storing SOM Weigth Vectors
      Map: storing SOM Unit Descriptions
      Map: writing SOM Quantization Error Map
      Map: writing Mapping File

C:\>
```

Sebagai hasil dari proses *mapping* ini adalah terbentuknya *file* "mapdescrmap.map", "weightvec.wgt", "quanterrmap.err", "nodedescr.node" dan "mapping.mpn" pada folder "C:\SOMLib\mapping".

File "mapdescrmap.map" jika dilihat menggunakan Wordpad akan tampak sebagai berikut.



Gambar 9.14 *File* mapdescrmap.map Jika Dilihat Menggunakan Wordpad

9.4.4 Langkah 4 - *Labelling*

Map labelling berarti digunakan untuk memilih dimensi yang paling penting dari sebuah *node*, sebagai objek yang dipetakan di *node* ini dijelaskan oleh karakteristik ini. Ada dua cara pelabelan yaitu "Nwords" dan "LabelSOM"

Syntax perintah untuk melakukan Nword *labelling* adalah sebagai berikut.

```
java somlib.som.labelling.NWords -i SOM-Map-
Description_in –w
SOM-Template-Vector_in [-o SOM-Node-Description_
out]
[-l SOM-Label-File_out] -n number_of_labels [-v
verbosity]
```

Selanjutnya perintah NWord *Labelling* bisa dilakukan.

```
C:\>java   somlib.som.labelling.NWords   -i   SOMLib\
mapping\mapdescrmap.map -w "SOMLi
b\template  vector\MyTest_1.tv"  -l  SOMLib\mapping\
label.lbl -n 4
    NWords initialized.

C:\>
```

Syntax perintah untuk melakukan LabelSOM *labelling* adalah sebagai berikut.

```
java somlib.som.labelling.LabelSOM -i SOM-Map-
Description_in –w
SOM-Template-Vector_in [-o SOM-Node-Description_
out]
[-l SOM-Label-File_out] -m minimum weight of label
-n
maximum number of labels [-v verbosity]
```

Selanjutnya perintah NWord *Labelling* bisa dilakukan

```
C:\>java  somlib.som.labelling.LabelSOM  -i  SOMLib\
mapping\mapdescrmap.map -l SOML
ib\mapping\label1.lbl  -w  "SOMLib\template  vector\
MyTest_1.tv" -m 0.85 -n 5
    LabelSOM initialized.

C:\>
```

Sebagai hasil dari proses pelabelan ini adalah terbentuknya *file* "label.lbl" dan "label1.lbl" pada folder "C:\SOMLib\mapping".

9.4.5. Langkah 5 - *Postprocess*

Pada langkah *postprocess* ini deskripsi akan ditulis dalam format HTML.

Syntax perintah untuk melakukan *postprocess* ini adalah sebagai berikut.

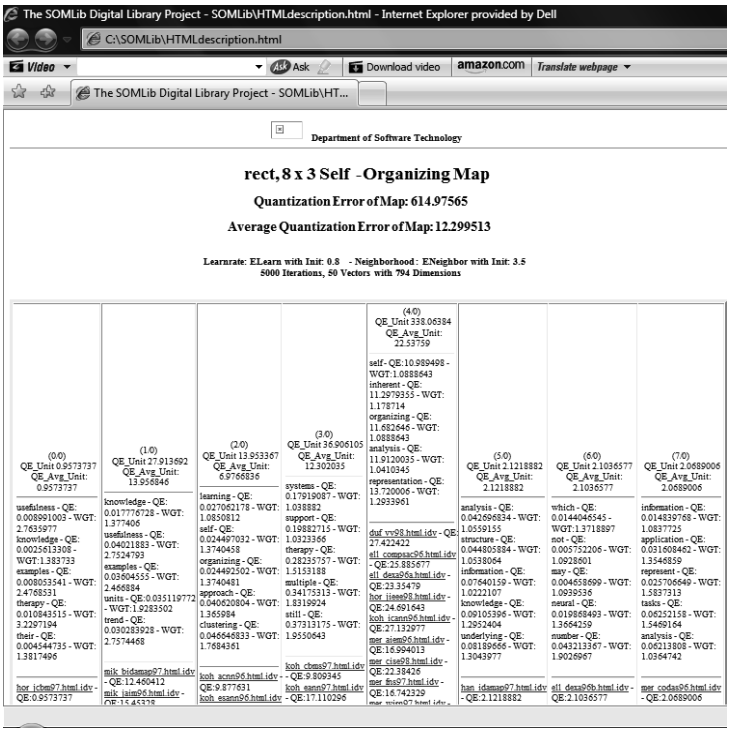
```
java somlib.som.postprocess.HTMLDescr -i SOM-Map-
Description_in
-o html_out [-r html-relative Path] [-e cut #
extensions [0-2]]
[-p show General Parameters [t/f] ]
[-q show Quantisation Error of SOM [t/f] ]
[-a show Average Quantisation Error of SOM [t/f] ]
[-t show ID of Unit [t/f] ]
[-u show Quantisation Error of Unit [t/f] ]
[-b show Average Quantisation Error of Unit [t/f] ]
[-l show Labels [t/f] ] [-m show Label Quantisation
Error [t/f] ]
[-w show Label Weight [t/f] ] [-s show Mapped
Vectors [t/f] ]
[-d show Vectors Distance [t/f] ] [-v verbosity]
```

Selanjutnya perintah *postprocess* bisa dilakukan.

```
C:\>java  somlib.som.postprocess.HTMLDescr  -i  SOMLib\
mapping\mapdescrmap.map -o S
OMLib\HTMLdescription.html -p t -q t -a t -t t -u t -b t
-l t -m t -w t -s t -d
t
  HTMLDescr:
    writing HTML-description to SOMLib\HTMLdescription.
html

C:\>
```

Sebagai hasil dari proses *postprocess* ini adalah terbentuknya *file* "HTMLdescription.html" pada folder "C:\SOMLib".

File "HTMLdescription.html" ini karena merupakan *file* HTML, maka bisa membukanya melalui *browser* dan akan tampak sebagai berikut.



Gambar 9.15 *File* HTMLdescription.html Dibuka dengan *Browser*

9.4.6 Membuat Deskripsi untuk LibViewer

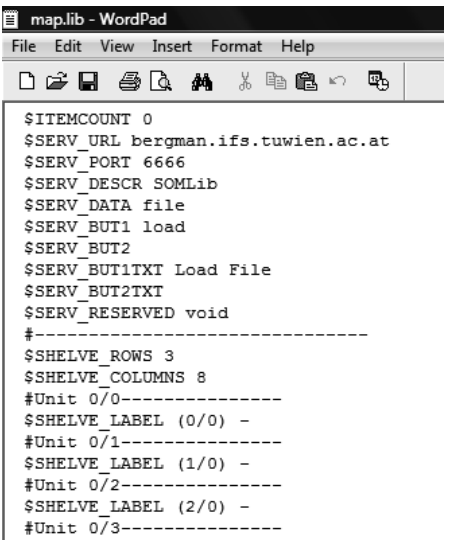
Syntax perintah untuk membuat deskripsi ini adalah sebagai berikut.

```
java somlib.som.postprocess.LibViewer -i SOM-Map-
Description_in
-o LibViewer_out [-l imageLocation]
[-s SOM-Vector-Description-Suffix] [-e cut #
extensions [0-2]]
[-v verbosity]
```

Selanjutnya perintah untuk membuat deskripsi ini bisa dilakukan.

```
C:\>java somlib.som.postprocess.LibViewer -i SOMLib\
training\mapdescr.map -o SOM
Lib\map.lib
  LibViewer:
    writing LibViewer-description to SOMLib\map.lib

C:\>
```

Sebagai hasil dari proses untuk membuat deskripsi ini adalah terbentuknya *file* "map.lib" pada folder "C:\SOMLib", yang jika dilihat menggunakan Wordpad akan tampak sebagai berikut.



Gambar 9.16 *File* map.lib Jika Dibuka dengan Menggunakan Wordpad

BAB X

SIMPULAN

- *Neural network* merupakan *tool* yang cukup penting untuk aplikasi-aplikasi *data mining*, karena ampuh untuk mengatasi data yang beragam dan besar.
- *Neural network* mencoba meniru neuron yang saling berhubungan di otak untuk membuat algoritma pembelajaran yang rumit untuk mengekstraksi pola dan mendeteksi tren.
- *Neural network* dapat dikategorikan sebagai "*supervised*" atau "*unsupervised*", bergantung kepada *output* yang diinginkan atau variabel yang diketahui.
- Model *unsupervised neural network* paling populer adalah "Kohonen Maps" atau "Self-Organizing Maps (SOMs)".
- SOMs banyak digunakan pada masalah *clustering* dan eksplorasi data di industri, keuangan, pengetahuan alam, dan tata bahasa.
- SOMs adalah teknik pengelompokkan yang mengidentifikasi kelompok dalam kumpulan data tanpa harus menggunakan teknik statistik tradisional.
- SOMs merupakan suatu algoritma kuantisasi vektor yang terorganisir secara topografis.

- SOMs sangat cocok untuk analisis klaster karena terdapat pola tersembunyi di antara catatan dan bidang dicari.
- Tujuan dari algoritma SOMs adalah untuk menghasilkan topologi yang mempertahankan pemetaan dari dimensi tinggi ruang ke ruang dimensi rendah dengan kata lain persekitaran dari ruang dimensi tinggi dapat direpresentasikan sebagai indeks persekitaran dalam dimensi yang rendah.
- Algoritma SOMs membangun model-model sehingga semakin mirip model-model akan diasosiasikan dengan simpul-simpul yang dekat dengan *grid*, sedangkan model yang kurang mirip secara bertahap akan jatuh lebih jauh dari *grid*.
- Salah satu paket perangkat lunak SOM yang tersebar luas adalah SOM Toolbox yang berisi semua fungsi SOM utama dan alat grafis yang bagus dalam bentuk yang ringkas.

DAFTAR PUSTAKA

Alex Yu, C. H. 2022. *Data Mining and Exploration*. CRC Press. https://doi.org/10.1201/9781003153658

Binu, D., & Rajakumar, B. R. 2021. *Artificial Intelligence in Data Mining* (D. Binu & B. R. Rajakumar, Eds.). Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-820601-0.00005-7

Budhiarti Nababan, E., & Zarlis, M. 2015. "Analisis Fungsi Aktivasi Sigmoid Biner dan Sigmoid Bipolar dalam Algoritma Backpropagation pada Prediksi Kemampuan Siswa". *Jurnal Teknovasi*, *02*(1), 103–116. https://doi.org/http://dx.doi.org/10.55445/teknovasi.v2i1.45

Chakraverty, S., & Jeswal, S. K. 2021. *World Scientific Solving Algebraic Equations*.

Chakraverty, S., & Mall, S. 2017. *Artificial Neural Networks for Engineers and Scientists: Solving Ordinary Differential Equations* (1st ed.). CRC Press. https://doi.org/https://doi.org/10.1201/9781315155265

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Rüdiger Wirth. 2000. *CRISP-DM 1.0 Step-by-Step Data Mining Guide*. Daimler Chrysler.

Clark, D. M., & Ravishankar, K. 1990. "Acquisition and Decay Rates in Synaptically Coded Memory". *Neural Network*, *3*(5), 525–533. https://doi.org/https://doi.org/10.1016/0893-6080(90)90003-4

Data Science Process Alliance. 2021. *Evaluating Crisp- DM for Data Science*. www.DataScience-PM.com

Fausett, L. v. 1994. *Fundamentals of Neural Networks Architechtures, Algorithms, aIons*. New Jersey: Prentice-Hall.

Fisher, R. A. 1936. "The Use of Multiple Measurements In Taxonomic Problems I. Discriminant Bunctions". *Annals of Eugenics*, *7*(2). https://doi.org/10.1111/j.1469-1809.1936.tb02137.x

Fix, J., & Frezza-Buet, H. 2019. *Advances in Intelligent Systems and Computing 976 Advances in Self-Organizing Maps, Learning Vector Quantization, Clustering and Data Visualization*. http://www.springer.com/series/11156

García-Tejedor, Á. J., & Nogal"s, A. 2022. "An Open-Source Python Library for Self-Organizing-Maps". *Software Impacts*, *12*. https://doi.org/10.1016/j.simpa.2022.100280

Harrison, D., & Rubinfeld, D. L. 1978. "Hedonic Prices and The Demand for Clean Air". *Journal of Environmental Economics and Management*, *5*(1), 81–102. https://doi.org/https://doi.org/10.1016/0095-0696(78)90006-2

Hebb, D. O. 1949. The Organization of Behavior: In *John Wiley*. New York: Wiley.

IBM Corporation. 2021. *IBM SPSS modeler CRISP-DM guide to modeler*.

James, U. I., & Eli-Chukwu, N. 2014. "Artificial Neural Network Based Short Term Load Forecasting in a Power System Network in Nigeria". *International Journal of Smart Home*, *8*(3), 145–150. https://doi.org/10.14257/ijsh.2014.8.3.13

Kantardzic, M. 2020. *Data Mining Concepts, Models, Methods, and Algorithms* (3rd ed.). New Jersey: John Wiley & Sons, Inc.

Kim, P. 2017. "MATLAB Deep Learning". In *MATLAB Deep Learning*. Apress. https://doi.org/10.1007/978-1-4842-2845-6

Kohonen, T. 1982. "Self-Organized Formation of Topologically Correct Feature Maps". *Biol. Cybern*, *43*, 59–69. https://doi.org/https://doi.org/10.1007/BF00337288

Kohonen, T. 1999. "Fast Evolutionary Learning with Batch-Type Self-Organizing Maps". In *Neural Processing Letters* (Vol. 9). https://doi.org/https://doi.org/10.1023/A:1018681526204

Kohonen, T. 2013. "Essentials of The Self-Organizing Map". *Neural Networks*, *37*, 52–65. https://doi.org/10.1016/j.neunet.2012.09.018

Kohonen, T. 2014. *Self-Organizing Map Poverty Map*. http://docs.unigrafia.fi/publications/kohonen_teuvo/index.html

Larose, D. T., & Larose, C. D. 2014. *Discovering Knowledge In Data An Introduction to Data Mining Second Edition Wiley Series on Methods and Applications in Data Mining*. New Jersey: Wiley.

Liu, Z., & Zhou, J. 2020. *Introduction to Graph Neural Networks*. Morgan & Claypool. https://doi.org/10.2200/S00980ED1V01Y202001AIM045

Ma, X., Kirby, M., & Peterson, C. 2022. "Self-Organizing Mappings on The Flag Manifold with Applications to Hyper-Spectral Image Data Analysis". *Neural Computing and Applications*, *34*(1), 39–49. https://doi.org/10.1007/s00521-020-05579-y

Martinez-Plumed, F., Contreras-Ochando, L., Ferri, C., Hernandez-Orallo, J., Kull, M., Lachiche, N., Ramirez-Quintana, M. J., & Flach, P. 2021. "CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories". *IEEE Transactions on Knowledge and Data Engineering*, *33*(8), 3048–3061. https://doi.org/10.1109/TKDE.2019.2962680

Mcculloch, W. S., & Pitts, W. 1943. "A Logical Calculus of The Ideas Immanent in Nervous Activity". *Bulletin of Mathematical Biophysics*, *5*, 115–133. https://doi.org/https://doi.org/10.1007/BF02478259

Melin, P., Monica, J. C., Sanchez, D., & Castillo, O. 2020. "Analysis of Spatial Spread Relationships of Coronavirus (COVID-19) Pandemic in the World using Self Organizing Maps". *Chaos, Solitons and Fractals*, *138*. https://doi.org/10.1016/j.chaos.2020.109917

Nunes, I., Danilo, S. ·, Spatti, H., Andrade, R., Luisa, F., Liboni, H. B., Franco, S., & Alves, R. 2017. *Artificial Neural Networks A Practical Course*. Springer Cham. https://doi.org/https://doi.org/10.1007/978-3-319-43162-8

Okelola, M. O., & Ayanlade, S. 2021. *An Artificial Neural Network Approach to Short-Term Load An Artificial Neural Network Approach to Short-Term Load Forecasting for Nigerian Electrical Power Network*. *April 2022*. https://doi.org/10.17605/OSF.IO/539DV

Rauber, A. 1999. *LabelSOM: On the Labeling of Self-Organizing Maps*. http://www.ifs.tuwien.ac.at/~andi

Ritter, H., Martinetz, T., & Addison-Wesley, K. S. 1992. *Neural Computation and Self-Organizing Maps-An Introduction*. New York: Addison-Wesley.

Santi Ika, M., Narendra, A. A. N., & Sudarma, M. 2017. "Mapping Patterns Achievement Based on CRISP-DM and Self Organizing Maps (SOM) Methods". *International Journal of Engineering and Emerging Technology*, *2*(1). https://doi.org/https://doi.org/10.24843/IJEET.2017.v02.i01

Shanmuganathan, S. 2016. *Artificial Neural Network Modelling* (S. Shanmuganathan & S. Samarasinghe, Eds.). Springer Cham. https://doi.org/https://doi.org/10.1007/978-3-319-28495-8

Stevanovic, D., Vlajic, N., & An, A. 2013. "Detection of Malicious and Non-Malicious Website Visitors Using Unsupervised Neural Network Learning". *Applied Soft Computing Journal*, *13*(1), 698–708. https://doi.org/10.1016/j.asoc.2012.08.028

Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. 2019. *Introduction to Data Mining* (2nd ed.). Pearson Education Limited.

Wendler, T., & Gröttrup, S. 2021. "Data Mining with SPSS Modeler". In *Data Mining with SPSS Modeler*. Springer International Publishing. https://doi.org/10.1007/978-3-030-54338-9

Zadissa, A., & Apweiler, R. 2022. "Data Mining, Quality, and Management in the Life Sciences". In *Methods in Molecular Biology* (Vol. 2449). https://doi.org/10.1007/978-1-0716-2095-3_1

IfS Department of Software Technology. 1999. *The SOMLib Digital Library Project*. Vienna: Vienna University of Technology.

Kohonen, T. 1995. *Self-Organizing Maps*. Springer-Verlag.

TENTANG PENULIS

Nama lengkap Dr. Sylvia Jane Annatje Sumarauw, M.Si., M.Kom. Lahir di Manado pada 27 Oktober 1962. Pada tahun 1986 lulus S-1 Jurusan Matematika FKIE IKIP Manado, sekarang Fakultas Matematika, Ilmu Pengetahuan Alam, dan Kebumian (FMIPAK) Universitas Negeri Manado (Unima). Sejak maret 1987 diangkat menjadi dosen pada jurusan tersebut, pada awalnya untuk mata kuliah yang berhubungan dengan Komputasi dan Statistika. Kemudian menjadi dosen mata kuliah Algoritma dan Pemrograman, *Data mining* dan *Data Warehouse*, Sistem Manajemen Basis Data dan Kecerdasan Buatan. Sejak tahun 2017 juga mengajar pada jurusan Teknik Informatika, Fakultas Teknik Unima, dengan mata kuliah Teori Bahasa Otomata dan Kecerdasan Buatan. Pada tahun 1994 lulus S-2 bidang Kependudukan UGM dengan gelar M.Si. Mendapat gelar Magister Komputer (M.Kom.) bidang Ilmu Komputer Universitas Gadjah Mada (UGM) pada tahun 2006 dan pada tahun 2016 mendapat gelar Doktor bidang Ilmu Komputer pada FMIPA UGM. Aktif melakukan penelitian di bidang Soft Computing, *Data mining*, *Machine Learning*, dan *Timeseries Forecasting*. Bisa dihubungi melalui alamat email: sylviasumarauw@unima.ac.id dan janesumarauw@gmail.com.