OPTIMASI VEKTOR BOBOT PADA *LEARNING VECTOR QUANTIZATION* MENGGUNAKAN *PARTICLE SWARM OPTIMIZATION* UNTUK KLASIFIKASI JENIS *ATTENTION DEFICIT HYPERACTIVITY DISORDER* (ADHD) PADA ANAK USIA DINI

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Windi Artha Setyowati
NIM: 135150207111081

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

OPTIMASI VEKTOR BOBOT PADA *LEARNING VECTOR QUANTIZATION* MENGGUNAKAN *PARTICLE SWARM OPTIMIZATION* UNTUK KLASIFIKASI JENIS *ATTENTION DEFICIT HYPERACTIVITY DISORDER* (ADHD) PADA ANAK USIA DINI

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Windi Artha Setyowati
NIM: 135150207111081

Skripsi ini telah diuji dan dinyatakan lulus pada
19 Januari 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I

Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D
NIP. 19720919 199702 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 19 Januari 2018

Windi Artha Setyowati

NIM: 135150207111081

iii

KATA PENGANTAR

Dengan menyebut nama Allah Yang Maha Pengasih lagi Maha Penyayang. Puji dan syukur atas kehadirat Allah SWT karena dengan rahmat dan karunia-Nya Penulis dapat menyelesaikan skripsi dengan judul "Optimasi Vektor Bobot Pada *Learning Vector Quantization* Menggunakan *Particle Swarm Optimization* untuk Klasifikasi Jenis *Attention Deficit Hyperactivity Disorder* (ADHD) Pada Anak Usia Dini".

Melalui kesempatan kali ini Penulis ingin menyampaikan terima kasih kepada semua pihak yang telah memberikan bantuan dan dukungan selama pengerjaan skripsi, yaitu:

1. Bapak Wayan Firdaus Mahmudy, S.Si, M. T, Ph.D selaku pembimbing skripsi yang telah membing dan mengarahkan penulis sehingga skripsi ini dapat terselesaikan.
2. Orang tua yang telah memberikan dukungan dan do'a sehingga penulis dapat menyelesaikan skripsi.
3. Raissa A., Dita R., K. Dewi, Fitri A., Ulfa L., Selly K. dan Ellina M. yang telah memberikan dukungan semangat, saran dan banyak bantuan selama pengerjaan skripsi sehingga penulis dapat menyelesaikan skripsi ini.
4. Civitas akademik Fakultas Ilmu Komputer yang telah memberikan arahan dan bantuan.
5. Pihak lain yang tidak dapat seluruhnya disebutkan oleh penulis, yang telah memberikan bantuan secara langsung maupun tidak langsung.

Malang, 19 Januari 2018

Penulis

setyowatiarthawindi@gmail.com

.

ABSTRAK

Attention Deficit Hyperactivity Disorder (ADHD) merupakan suatu gangguan perkembangan mental yang memiliki karakterisrik utama kesulitan penderitanya untuk memusatkan perhatian. Ciri-ciri ADHD sering muncul dan mulai dapat diamati pada anak usia 3 sampai 5 tahun atau ketika anak mulai belajar mengembangkan organ motorik. ADHD terdiri dari 3 jenis, yaitu*: inattention, hyperactivity*, dan *impulsivity.* Belum banyak masyarakat yang sadar dan tahu akan ADHD, maka dibutuhkan system untuk klasifikasi jenis ADHD.

Dengan mengamati gejala yang tampak, ADHD dapat diklasifikasikan menggunakan algoritme *Learning Vector quantiztion* (LVQ), namun pada penelitian sebelumnya, algoritme LVQ menghasilkan akurasi yang terbilang rendah. Untuk mengoptimalkan tingkat akurasi algortima LVQ, maka digunakan algoritme *Particle Swarm Optimization* (PSO) untuk mencari vektor bobot LVQ terbaik.

Dalam penelitian ini, dilakukan pengujian pada algoritme LVQ-PSO dan LVQ untuk mengetahui perbedaan hasil akurasi menggunakan data uji yang sama. Hasil pengujian menunjukkan rata-rata akurasi tertinggi algortima LVQ-PSO adalah 87,3% dengan waktu komputasi 84,6 detik dan akurasi terendah adalah 80% dengan waktu komputasi 84,2 detik. Algoritma LVQ memiliki nilai rata-rata tertinggi adalah 80,6% dengan waktu komputasi 4,8 detik dan akurasi terkecil adalah 74,5% dengan waktu komputasi 3,6 detik. Parameter-parameter PSO terbaik menghasilkan akurasi terbaik adalah adalah *Wmax* 0,6, *Wmin* 0,5, ukuran *swarm* 100, maksimal iterasi PSO 100, α 0,1, dan *dec α 0,1*. Dari hasil akurasi pengujian tersebut, maka dapat disimpulkan bahwa algoritme PSO dapat digunakan untuk mengoptimasi algortime LVQ meskipun membutuhkan waktu komputasi yang lebih lama.

Kata kunci: klasifikasi, optimasi, *attention deficit hyperactivity disorder, learning vector quantization, particle swarm optimization.*

ABSTRACT

Attention Deficit Hyperactivity Disorder (ADHD) is a mental development disorder that has the main characteristic of the sufferer's difficulties to focus attention. The characteristics of ADHD often appear in children aged 3 to 5 years or when children begin to learn to develop motor organs. ADHD consists of 3 types, namely: inattention, hyperactivity, and impulsivity. Not many people are aware of ADHD, then required a system for the classification type of ADHD.

By observing visible symptoms, ADHD can be classified using the Learning Vector quantiztion (LVQ) algorithm, but in previous studies, the LVQ algorithm yielded relatively low accuracy. To optimize the accuracy of the LVQ algorithm, the Particle Swarm Optimization (PSO) algorithm is used to find the best LVQ weight vector.

In this research, LVQ-PSO and LVQ are tested to know the difference of accuracy result using the same testing data. The test result shows that the highest accuracy of LVQ-PSO algorithm is 87,3% with computation time 84,6 seconds and lowest accuracy is 80% with computation time 84,2 seconds. LVQ algorithm has the highest average value is 80,6% with a computation time of 4,8 seconds and the smallest accuracy is 74,5% with a computation time of 3,6 seconds. The best parameters of PSO to produce the best accuracy are Wmax 0,6, Wmin 0.5, swarm size 100, maximum iteration PSO 100, α 0,1, and dec α 0,1. From the results of the test accuracy, it can be concluded that the PSO algortime can be used to optimize the LVQ algortime even though it takes longer computation time.

Keywords: classification, optimization, attention deficit hyperactivity disorder, learning vector quantization, particle swarm optimization.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Gangguan Pemusatan Perhatian dan Hiperaktif (GPPH) merupakan suatu gangguan perkembangan yang memiliki karakteristik atau ciri utama kesulitan penderita untuk memusatkan perhatian (Rohman & Fauzijah, 2008). Gangguan tersebut, biasa disebut *Attention Deficit Hyperactivity Disorder* (ADHD). Ciri-ciri ADHD sering muncul dan mulai dapat diamati pada anak usia 3 sampai 5 tahun dan ketika anak mulai belajar mengembangkan organ motorik. Jika tidak ditangani, ADHD dapat mempengaruhi perkembangan karena anak akan sulit berkomunikasi, sulit diam, sulit berkonsentrasi, sulit berinteraksi sosial, mengurangi pencapaian prestasi akademik (Novita & Siswati, 2010) bahkan memiliki kecenderungan melakukan bunuh diri ketika dewasa (Kompas, 2012). Agar penanganan ADHD dapat bekerja secara optimal, *treatment* sebaiknya disesuaikan dengan tipe ADHD yang diderita. Jenis-jenis ADHD dapat diketahui dengan memperhatikan gejala perilaku yang muncul (DSM IV). ADHD terdiri dari 3 jenis, yaitu*: inattention, hyperactivity*, dan *impulsivity*.

Penelitian untuk mengklasifikasikan jenis ADHD pernah dilakukan oleh Zainal Arifien pada tahun 2016 menggunakan algoritme dalam *Learning Vector Quantization* (LVQ), dengan nilai akurasi yang dihasilkan sebesar 70%. Nilai tersebut terbilang rendah jika dibandingkan dengan pengklasifikasian yang dilakukan dengan algoritme lain seperti: algoritme *Linear Discriminant Analysis* sebesar 90% (Nurmawati,2016), *Fuzzy K-Nearest Neighbour* mencapai akurasi 90% (Laxsmana, 2016), dan *Neighbor Weighted K-Nearest Neighbor* memiliki akurasi sebesar 95% (Fadila, 2016). Hasil akurasi penelitian-penelitian tersebut menunjukan perlu dilakukan perbaikan pada algoritme LVQ untuk menghasilkan nilai akurasi yang lebih baik untuk klasifikasi jenis ADHD pada anak usia dini.

Pengoptimalan suatu algoritme dapat dilakukan dengan menggabungkannya dengan algoritme lain. Novitasari, Cholissodin & Mahmudy (2016) menggunakan algoritme *Local Best* PSO untuk mengoptimasi *Support Vector Regression* (SVR) dan menunjukkan bahwa *Local Best* PSO-SVR menghasilkan nilai *error* paling rendah dibandingkan dengan PSO-SVR dan T-SVR. Rifqi (2016) mengoptimasi vektor bobot pada algoritme LVQ menggunakan algoritme genetika (GA) untuk klasifikasi tingkat penyakit stroke. Penelitiannya menghasilkan akurasi terbaik sebesar 93%. Tahun 2015, Ramanda meningkatan kerja algoritme *Neural Network* (NN) menggunakan *Particle Swarm Optimization* (PSO) untuk memprediksi kelahiran *premature* dengan kesimpulan bahwa algoritme NN-PSO memberikan nilai akurasi 0,40% lebih baik daripada algoritme NN.

Menurut Budianita dalam penelitiannya untuk pengklasifikasian status gizi anak pada tahun 2013, LVQ memiliki kelebihan selain mencari jarak terdekat, LVQ juga melalukan pelatihan terawasi dalam memperbaiki vektor bobot untuk memperkirakan keputusan pengklasifikasian. Penelitian Nurkozin (Budianita, 2013), membuktikan bahwa LVQ memberikan akurasi yang lebih baik dalam

membaca pola apabila dibandingkan dengan algoritme *Backpropagation*. Pada tahun 2015, Asriningtias dalam penelitiannya membuktikan jika PSO memiliki waktu yang lebih cepat mencapai konvergen daripada GA untuk *training* pada *neural network*. Pada tahun 2016, Caesar melakukan penelitian untuk mengoptimasi pakan ternak dan menyimpulkan bahwa ANN-PSO lebih baik daripada ANN-GA.

Dari beberapa penelitian sebelumnya, maka akan digunakan algoritme PSO untuk mengoptimasi algoritme LVQ untuk klasifikasi jenis ADHD pada anak usia dini.

1.2 Rumusan Masalah

Masalah yang dapat disimpulkan dari latar belakang yaitu:

1. Bagaimana cara menerapkan algoritme *Particle Swarm Optimization* (PSO) untuk mengoptimasi *Learning Vector Quantization* (LVQ) pada klasifikasi jenis ADHD pada anak usia dini?
2. Bagaimana tingkat akurasi algortime LVQ yang dioptimasi dengan algoritme PSO pada klasifikasi jenis ADHD pada anak usia dini?
3. Bagaimana tingkat akurasi algortime PSO-LVQ dibandingkan dengan algoritme LVQ untuk klasifikasi jenis ADHD pada anak usia dini?

1.3 Tujuan

Tujuan yang ingin dicapai dari penelitian ini adalah:

1. Menerapkan PSO untuk mengoptimasi LVQ untuk klasifikasi jenis ADHD pada anak usia dini.
2. Mengetahui tingkat akurasi algoritme LVQ yang dioptimasi dengan algotime PSO pada klasifikasi jenis ADHD pada anak usia dini.
3. Mengetahui perbandingan akurasi antara algoritme LVQ-PSO dengan LVQ untuk klasifikasi jenis ADHD pada anak usia dini.

1.4 Manfaat

Beberapa manfaat yang diperoleh melalui penilitian ini adalah penulis dapat menerapkan ilmu dan pengetahuan yang dimiliki untuk menyelesaikan suatu permasalahan dan mengetahui hasil suatu algoritme yang telah dioptimasi. Bagi dunia akademik, hasil penelitian ini dapat dijadikan pembanding atau acuan untuk penelitian selanjutnya.

1.5 Batasan Masalah

Agar penelitian ini tidak terlalu meluas diperlukan adanya batasan permasalahan. Batasan masalah yang akan digunakan adalah:

1. Data yang digunakan adalah data sekunder milik peneliti sebelumnya.
2. Algoritme yang digunakan adalah algoritme LVQ yang dioptimasi menggunakan PSO.

3. Hasil jenis klasifikasi ADHD terbagi menjadi empat, yaitu: *inattention*, *hyperactivity*, *impulsivity,* dan tidak ADHD.

1.6 Sistematika Pembahasan

Penulisan tugas akhir ini disusun dengan kerangka sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini berisi latar belakang penelitian, rumusan masalah, tujuan yang akan dicapai, manfaat penelitian, batasan masalah sebagai acuan peneitian, dan sistematika pembahasan.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini akan berisi penelitian-penelitian sebelumnya terkait algoritme dan teori pendukung yang akan digunakan pada penelitian ini.

BAB 3 METODOLOGI PENELITIAN

Bab ini berisi langkah-langkah yang dilakukan dalam penyelesaian masalah penelitian. Penelitian dimulai dari studi kepustakaan, analisis kebutuhan, pengumpulan data, perancangan, implementasi, pengujian, dan kesimpulan.

BAB 4 PERANCANGAN

Bab ini berisi formulasi permasalahan, perancangan algoritme, perhitungan manual, dan perancangan pengujian

BAB 5 IMPLEMENTASI

Bab ini membahas implementasi hasil perancangan algoritme.

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini berisi hasil pengujian dari implemetasi algoritme dan analisis dari hasil pengujian.

BAB 7 PENUTUP

Bab ini berisi kesimpulan dari hasil analisis.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Arifien, Fadila, Laxsmana, dan Nurmawati (2016) melakukan penelitian terkait anak ADHD. Masing-masing dari mereka membuat sebuah sistem untuk mendetekasi jenis ADHD pada anak usia dini menggunakan algoritme yang berbeda-beda. Mereka menggunakan total data latih dan data uji sebanyak 100 data. Fadila menggunakan algoritme *Neighbor Weighted K-Nearest Neighbour* (NWKNN), 80 data latih, 20 data uji, dan mencapai akurasi 95%. Laxsmana menggunakan algoritme *Fuzzy K-Nearest Neighbor* (FKNN), 80 data latih, 20 data uji, dan mendapatkan akurasi maksimum 90%. Nurmawati menggunakan algoritme *Linear Discriminant Analysis* dan mencapai akurasi 90%. Arifien menggunakan algoritme *Learning Vector Quantization* (LVQ) dan mendapatkan akurasi 70%.

Rifqi (2016) menggunakan algoritme genetika (GA) untuk mengoptimmasi vektor bobot pada LVQ untuk klasifikasi tingkat resiko penyakit stroke. Rifqi menggunakan data pasien sebanyak 200 data, dengan setiap data memiliki 5 fitur dan teklasifikasikan dalam 3 kelas. Hasil pengujian didapatkan akurasi terbaik sebesar 93%.

Nurmalahudin (2013) menggunakan algoritme *Particle Swarm Optimization* (PSO) sebagai algoritme belajar pada algoritme jarigan saraf tiruan. Penggabungan kedua algoritme tersebut digunakan untuk diujikan pada aplikasi prakiraan temperatur udara harian menggunakan model *time series*. Kesimpulan menunjukan bahwa menggunakan algoritme PSO sebagai algoritme pembelajaran menghasilkan kesalahan prakiraan 2,41% pada prakiraan temperatur udara minimum dan 3,81 % pada udara maximum. Sistem ini memiliki kelemahan, yaitu sistem komputasi yang lebih lama jika dibandingkan dengan algoritme belajar *Backpropagation* tapi algoritme belajar menggunakan PSO menghasilkan nilai error yang lebih kecil pada proses pelatihan dan proses pengujian menggunakan 50 particle dan 150 iterasi.

Ramanda (2015) melakukan penelitian untuk memprediksi kelahiran *premature* di RS Sumber Waras. Data yang digunakan adalah data pasien ginekologi sebanyak 500 data dengan setiap data memiliki 11 atribut. Ramanda meningkatkan kinerja *neural network* menggunakan PSO. Penelitiannya menyatakan bahwa algoritme ANN-PSO lebih baik daripada ANN. Akurasi dari ANN-PSO menghasilkan nilai akurasi 96,80 % dan algoritme ANN sebesar 96,40 %. PSO memiliki kinerja yag unggul untuk banyak masalah optimasi.

Budianita (2013) melakukan penelitian untuk mengklasifikasikan status gizi menggunakan LVQ3, yaitu penambahan parameter *window* dibandingkan LVQ1. Dalam penelitiannya, Budianita menggunakan 6 variabel. Dalam penelitiannya disebukan juga penelitian Nurkozin mengenai klasifikasi penyakit diabetes. Penelitian Nurkozin meyatakan LVQ memberikan akurasi yang lebih baik dalam membaca pola apabila dibandingkan dengan algoritme *Backpropagation.*

2.2 Attention Deficit Hyperactivity Disorder

Attention Deficit Hyperactivity Disorder (ADHD) atau dalam Bahasa Indonesia disebut dengan Gangguan Pemusatan Perhatian dan Hiperaktif (GPPH) merupakan gangguan perkembangan mental yang memiliki gejala utama sulit untuk memusatkan perhatian (Rohman & Fauzijah, 2008). ADHD pertama kali diutarakan seorang dokter, penulis buku pengobatan dan psikiatri, yaitu Dr. Heinrich Hoffman pada tahun 1845. Penyebab pasti ADHD masih belum diketahui (Tanoyo, 2013). Menurut buku *Diagnostic and Statistical Manual of Mental Disorder* edisi ke 4, biasanya gejala ADHD muncul dan dapat diamati sebelum usia 7 tahun (usia dini), meskipun ada beberapa yang muncul setelah umur tersebut dan ketika seorang anak memulai menggunakan organ motoriknya.

2.2.1 Jenis ADHD

ADHD terdiri dari tiga jenis, yaitu: *inattention*, *hyperactivity*, dan *impulsivity*.

Inattention atau secara umum dapat disebut kurang perhatian adalah gejala yang dapat diamati dalam bidang akademik atau ketika sedang bersosialisasi. Penderita seperti tidak mendengarkan ketika seseorang berbicara padanya, hasil pekerjaan berantakan, tidak memperhatikan detail atau petunjuk tertentu, berpindah dari suatu tugas ke tugas lain meskipun belum selesai sepenuhnya atau sering melupakan sesuatu yang baru didengar.

Hyperactivity atau sikap hiperaktif biasanya terlihat ketika penderita ADHD sulit untuk tidak bergerak, tidak dapat duduk dalam waktu lama atau ketika berbaris ketika upacara, berlari di dalam ruangan atau memanjat perabotan rumah ketika seharusnya tidak dilakukan. Sikap hiperaktif pada penderita ADHD usia dini sulit dibedakan dengan sikap aktif pada anak normal karena pada umumnya wajar apabila anak-anak banyak bermain dan berlari.

Impulsivity atau sikap impulsif biasanya dikaitkan dengan ketidaksabaran. Penderita sulit menunda suatu respon, misalnya menjawab suatu pertanyaan yang belum selesai ditanyakan, sulit untuk menunggu giliran, menyela pembicaraan lawan bicara, memegang barang milik orang lain atau mengganggu seseorang.

Gangguan ADHD lebih baik segera ditangani dengan tepat karena dapat mengganggu aktifitas sehari-hari penderitanya, baik bidang akademik, komunikasi, dan kesulitan bersosialisasi. Bukan hanya itu, menurut sebuah penelitian yang diterbitkan dalam Journal of Consulting and Clinical Psychology penderita gangguan ADHD memiliki kecendrungan menyakiti diri sendiri dan bahkan melakukan bunuh diri (Kompas, 2012).

2.3 Klasifikasi

Klasifikasi adalah penyusunan bersistem dalam kelompok atau golongan menurut kaidah atau standar yang ditetapkan. Klasifikasi bisa juga diartikan sebagai pengelompokan berdasarkan tingkat kemiripan ke dalam kelompok tertentu yang telah ada. Dalam penelitian ini, kelompok *output* yang telah

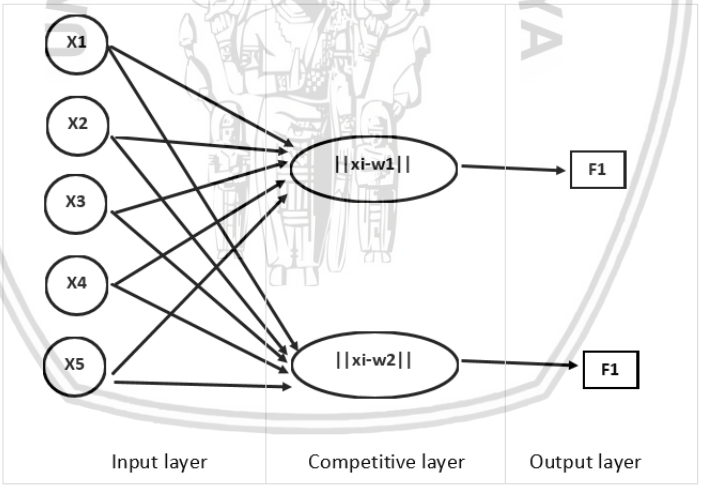
ditetapkan ada 4 jenis, yaitu: *inattention, hyperactivity, impulsivity* atau bukan ADHD.

2.4 Optimasi

Optimasi berarti membuat sesuatu menjadi optimal. Menurut KBBI, optimal berarti terbaik, tertinggi atau paling menguntungkan. Optimasi digunakan untuk mencari solusi yang dianggap paling baik, meskipun belum tentu yang terbaik dari sebuah permasalahan. Biasanya optimasi digunakan untuk mencari solusi dari permasalahan yang melibatkan data dengan jumlah banyak dan kompleks, permasalahan yang akan memerlukan waktu untuk proses komputasi. Ada beberapa algoritme yang bisa digunakan untuk masalah optimasi, antara lain: *Particle Swarm Optimization* (PSO), *Genethic Alghoritms* (GA) dan *Ant Colony Optimization* (ACO) (Mahmudy, 2015).

2.5 Learning Vector Quantization

Learning Vector Quantization adalah salah satu algoritme pembelajaran pada lapisan kompetitif jaringan saraf tiruan yang akan belajar otomatis untuk klasifikasi *input*an ke dalam kelas tertentu. Hasil klasifikasi akan bergantung pada kemiripan atau kedekatan vector. Jika ada sebuah *input*an yang hampir mirip dengan vektor kelas tertentu, maka akan dimasukkan ke dalam kelas tersebut.



Gambar 2.1 Struktur *Learning Vector Quantization*

Sumber: Arifien, 2016

2.5.1 Pelatihan Learning Vector Quantization

Pelatihan LVQ digunakan untuk mencari vektor bobot. Vektor bobot tersebut akan digunakan sebagai acuan untuk melakukan pengklasifikasian.

Algoritme pelatihan LVQ:

1. Inisialisasi nilai dari *learning rate* (α), pengurang *learning rate* (dec α), data latih, iterasi maksimal dan α minimal yang digunakan.

2. Untuk setiap *input* lakukan langkah a hingga d:
   a. Hitung jarak antara data *input* dengan vektor bobot untuk setiap kelas dengan Persamaan 2.1.

      $$D(x,w) = \left||Xi - Wj|\right| \quad (2.1)$$

   b. Tentukan jarak minimum dari hasil perhitungan jarak sehingga didapatkan keluaran C_j.

      C_j = Min D(x,w) (2.2)

   c. Perbaiki bobot wj dengan syarat:

      Apabila T = C

      $$Wj(baru) = Wj(lama) + \alpha[Xi - Wj(lama)] \quad (2.3)$$

      Apabila T ≠ C

      $$Wj(baru) = Wj(lama) - \alpha[Xi - Wj(lama)] \quad (2.4)$$

   d. Melakukan perbaikan terhadap nilai *learning rate* (α) dengan Persamaan 2.5.

      $$\alpha(\text{baru}) = \alpha(\text{lama}) \times \text{dec}\,\alpha \quad (2.5)$$

3. Berhenti ketika iterasi telah maksimum atau *learning rate* (α) telah minimum.

Keterangan:

1. Nilai α dan dec α yang digunakan adalah bilangan dari 0 sampai 1.
2. X_i merupakan *input* dengan indeks *i*=1 hingga *M*.
3. W_j merupakan vektor bobot dari kelas j.
4. *T* merupakan kelas target dari data latih yang digunakan sedangkan *C* merupakan kelas terdekat dari hasil perhitungan jarak *D(x,w)* (Arifien, et al., 2016).

2.6 *Swarm* Intelegence

2.6.1 Konsep dasar

Swarm intelegence atau kecerdasan berkelompok adalah konsep rekayasa dari kecerdasan buatan yang meniru cara kerja sistem alami dari suatu kelompok hewan, antara lain: *Artificial Bee Colony* (ABC) yang cara kerjanya mirip dengan sekelompok lebah*, Ant Colony Optimization* (ACO) yang mirip dengan cara kerja sekelompok semut dan *Particle Swarm Optimization* (PSO) yang cara kerjanya mirip sekelompok burung. Algoritme ini terdiri dari banyak partikel yang saling berkoordinasi (Cholissodin, 2016).

2.6.2 Particle Swarm Optimization

Particle Swarm Optimization pertama kali dicetuskan oleh Russell C. Eberhart dan James Kennedy tahun 1995, merupakan salah satu algoritme yang ada pada *swarm intelegence* (Kresna, 2015). Algoritme ini meniru cara kerja sekelompok

burung. PSO memiliki tiga komponen utama, yaitu: partikel atau individu, komponen kognitif dan sosial, dan kecepatan partikel. Pembelajaran partikel terdiri dari *cognitive learning* atau *pBest* (posisi terbaik yang pernah dicapai partikel) dan *social learning* atau *gBest* (posisi terbaik dari keseluruhan partikel dalam kelompok). Untuk menghitung kecepatan menggunakan *pBest* dan *gBest*, sedangkan untuk menghitung posisi selanjutnya menggunakan kecepatan (Cholissodin, 2016).

Karena kecepatan partikel dirasa berubah terlalu cepat, maka pada tahun 1998, dilakukan modifikasi untuk memperbarui kecepatan partikel. Perubahan tersebut adalah penambahan bobot inersia, sehingga secara matematis besar bobot inersia adalah seperti pada Persamaan 2.6.

$$\omega=\omega_{max}-\left(\frac{\omega_{max}-\omega_{min}}{i\ maksimal}\right) i \quad (2.6)$$

Keterangan:

1. ω_{max} = bobot inersia terbesar
2. ω_{min} = bobot inersia terkecil
3. i = iterasi

Algoritme PSO:

1. Inisialisasi banyak partikel, kecepatan awal partikel, posisi awal partikel, *w* (bobot), *c1, c2, r1, r2,* iterasi maksimal, *pBest* dan *gBest.*
2. Untuk setiap iterasi lakukan langkag a-f:
   a. *Update* kecepatan setiap partikel dengan Persamaan 2.7.
   $$V_{i,j}^{t+1} = \omega . V_{i,j}^{t} + c_1 . r_1(pBest_{i,j}^{t} - x_{i,j}^{t}) + c_2 . r_2(gBest_{g,j}^{t} - x_{i,j}^{t}) \quad (2.7)$$
   b. *Update* posisi menggunakan Persamaan 2.8.
   $$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1} \quad (2.8)$$
   c. Menghitung *fitness* partikel baru menggunakan Persamaan 2.11.
   d. *Update pBest*, *pBest* diambil dengan membandingkan antara *fitness pBest* lama dengan partikel paling baru. *PBest* dengan *fitness* lebih tinggi dijadikan *pBest* terbaru.
   e. Menghitung *fitness pBest* terbaru menggunakan Persamaan 2.11.
   f. *Update gBest*, *gBest* diambil dari *pBest* terbaru dengan nilai *fitness* paling besar
3. Pada ierasi terakhir, pilih partikel dengan nilai *fitness* terbesar sebagai partikel terbaik.

Keterangan:

1. *V* = kecepatan partikel (*velocity)*.
2. *i* = partikel ke-*i.*
3. *t* = iterasi ke-*t.*
4. *j* = dimensi/ posisi ke-*j* dari sebuah partikel.
5. *r1* dan *r2* = bilangan acak dengan nilai 0 sampai 1.
6. *c1* dan *c2* = bilangan acak dari 0 sampai 4.

2.6.3 *Fitness* Partikel PSO

Fitness atau kebugaran partikel dapat digunakan untuk mengukur tingkat keoptimalan suatu partikel. Semakin besar nilai *fitness* partikel, berarti solusi yang direpresentasikan partikel tersebut semakin optimal (Mahmudy, 2015). Untuk menghitung nilai *fitness,* terlebih dahuli menghitung *cost. Cost* atau penaliti adalah total ketidaksesuaian antara kelas data latih PSO hasil klasifikasi terhadap kelas data latih target (kelas sesungguhnya). Untuuk menghitung *cost* gunakan langkah 1 sampa 5 berikut:

1. Tentukan data latih.
2. Untuk setiap partikel lakukan langkah 3 sampai 4.
3. Untuk setiap data latih lakukan langkah *a* sampai *f*.
   a. Hitung jarak *euclidea*n antara data latih terhadap dimensi 0-44 partikel menggunakan Persamaan 2.1.
   b. Hitung jarak *euclidean* antara data latih terhadap terhadap dimensi 45-89 partikel menggunakan Persamaan 2.1.
   c. Hitung jarak *euclidean* antara data latih terhadap terhadap dimensi 90-134 partikel menggunakan Persamaan 2.1.
   d. Hitung jarak *euclidean* antara data latih terhadap terhadap dimensi 135-179 partikel menggunakan Persamaan 2.1.
   e. Cari kelas klasifikasi dengan cara mencari euclidean terkecil antara langkah *a* sampai *d* dengan Persamaan 2.2.
   f. Perbarui nilai cost dengan syarat :

      Apabila T = C

      $$cost = cost + 1 \quad (2.9)$$

      Apabila T ≠ C

      $$cost = cost + 0 \quad (2.10)$$

4. Hitung fitnes partikel dengan Persamaan 2.11.

$$fitness = \frac{data\ latih\ PSO - cost}{banyak\ \ data\ latih\ PSO} \quad (2.11)$$

5. Berhenti ketika didapatkan *fitness* untuk semua partikel.

2.7 Pengujian Learning Vector Quantization

Setelah pelatihan LVQ, dilakukan pengujian untuk mengetahui akurasi. Langkah yang dilakukan adalah:

1. Inisialisasi vektor bobot dan data uji.
2. Menghitung jarak *euclidean* antara data uji dan vektor bobot menggunakan Persamaan 2.1.
3. Menentukan kelas data uji dengan Persamaan 2.2
4. Menghitung akurasi menggunakan Persamaan 2.12.

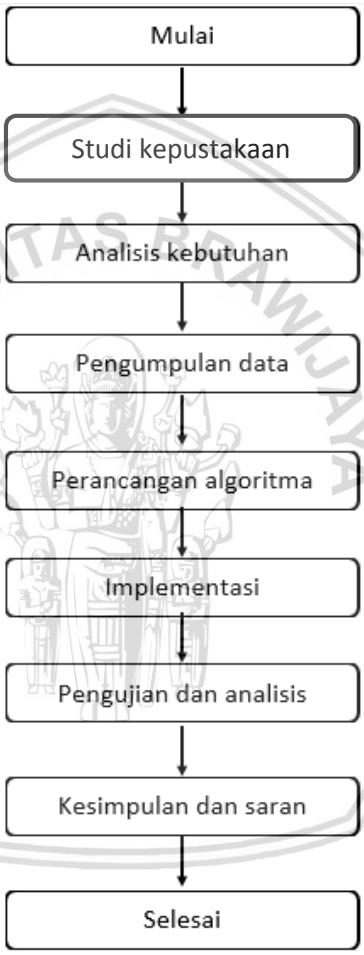
$$\text{Akurasi} = \frac{data\ uji\ sesuai\ target}{seluruh\ data\ uji} \text{ x } 100\% \qquad (2.12)$$

BAB 3 METODOLOGI

Bab ini berisi langkah-langkah yang dilakukan dalam penyelesaian masalah penelitian. Penelitian dimulai dari studi kepustakaan, analisis kebutuhan, pengumpulan data, perancangan, implementasi, pengujian dan kesimpulan.

3.1 Tahapan

Tahapan yang akan dilakukan untuk menyelasikan permasalahan ini dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram metodologi penelitian

3.2 Studi Kepustakaan

Studi kepustakaan dilakukan untuk memperdalam dasar teori dan ilmu yang telah dimiliki. Teori dapat diperoleh dari penelitian-penelitian sebelumnya yang berhubungan dengan permasalahan yang akan diteliti, seperti ADHD, LVQ, *swarm intelegence*, masalah optimasi, dan lain-lain. Sumber yang digunakan adalah paper, jurnal, buku dan media *online*.

3.3 Lingkungan Perancangan

Mengetahui lengkungan perancangan akan membantu proses implementasi dan untuk mengetahui apa saja yang dibutuhkan untuk mengimplementasi algoritme LVQ-PSO untuk mengklasifikasi jenis ADHD. Kebutuhan yang harus terpennnuhi untuk pengimplementasian meliputi:

1. Perangkat keras, yaitu:
   - Laptop dengan RAM 4 GB, processor Intel® Core™ i5-3337 CPU @1,80GHz (4 CPUs).
2. Perangkat lunak, yaitu:
   - NetBeans IDE 8.2.
   - Microsoft excel 2016.
3. Data sekunder dari penelitian Zainal Arifien (2016), klasifikasi ADHD berdasarkan gejala.

3.4 Pengumpulan Data

Data yang akan digunakan adalah data sekunder dari peneliti terdahulu, Zainal Arifien, pengklasifikasian jenis ADHD pada anak usia dini menggunakan LVQ. Data yang digunakan sebanyak 100 data dengan masing- masing data memiliki 45 atribut gejala. Data didapatkan dengan melakukan pengamatan di House of Fatima Child Center dan melakukan pengisian kuisioner seperti pda Tabel 3.1. Kuisioner dijawab dengan cara memilih salah satu opsi yang tersedia. Opsi akan bernilai 15 (untuk mewakili jawaban tidak pernah), 35 (untuk mawakili jawaban kadang- kadang), atau 50 (untuk mewakili jawaban kerap), nilai tersebut sesuai dengan pembobotan terhadap gejala ADHD yang diamati, semakin tinggi nilainya maka semakin sering suatu gejala muncul. Nilai pembobotan didapatkan dari seorang pakar pskologi dari House of Fatima Child Center Malang. Pertanyaan yang diajukan dapat dilihat pada Tabel 3.1.

Tabel 3.1 Kuisioner

No	Gejala	Opsi	Jawab
1	Anak kurang memperhatikan benda yang ditunjukkan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
2	Anak mudah mengalihkan pandangan pada sesuatu ketika dalam pembicaraan (masih terjadi kontak mata)	Kerap	
		Kadang-Kadang	
		Tidak pernah	
3	Anak tertinggal dalam mengikuti pembicaraan yang berlangsung	Kerap	
		Kadang-Kadang	
		Tidak pernah	

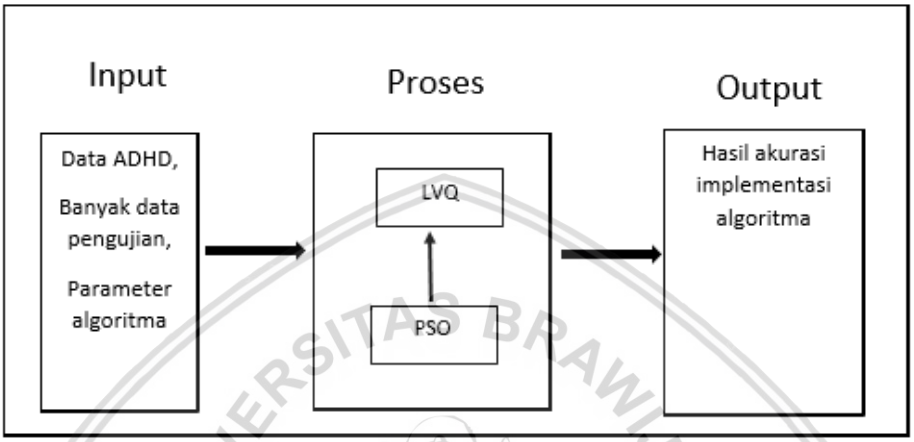
No	Gejala	Opsi	Jawab
4	Anak kehilangan mainan setelah dipakai	Kerap	
		Kadang-Kadang	
		Tidak pernah	
5	Anak melupakan aktivitas yang diajarkan dalam keseharian	Kerap	
		Kadang-Kadang	
		Tidak pernah	
6	Anak sulit menemukan tempat untuk mengambil atau mengembalikan benda	Kerap	
		Kadang-Kadang	
		Tidak pernah	
7	Perilaku yang muncul kurang sesuai dengan perintah yang diberikan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
8	Anak melakukan aktivitas secara acak (berlawanan dengan tahapan)	Kerap	
		Kadang-Kadang	
		Tidak pernah	
9	Anak melakukan permainan di luar dari instruksi	Kerap	
		Kadang-Kadang	
		Tidak pernah	
10	Anak meninggalkan mainannya ketika diminta untuk membereskan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
11	Anak tidak menjalankan perintah, meskipun telah diulang	Kerap	
		Kadang-Kadang	
		Tidak pernah	
12	Anak terus mempertahankan aktifitasnya ketika diberikan perintah	Kerap	
		Kadang-Kadang	
		Tidak pernah	
13	Anak memainkan alat permainan tidak sampai akhir	Kerap	
		Kadang-Kadang	
		Tidak pernah	
14	Anak tidak melanjutkan giliran bermain	Kerap	
		Kadang-Kadang	
		Tidak pernah	
15	Anak kurang mampu menjalankan permainan bersama hingga akhir	Kerap	
		Kadang-Kadang	
		Tidak pernah	
16	Anak menjawab sebelum pertanyaan selesai diberikan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
17	Anak menyela pembicaraan yang dilakukan orangtua atau saudara	Kerap	
		Kadang-Kadang	
		Tidak pernah	

No	Gejala	Opsi	Jawab
18	Anak berbicara diluar topik pembicaraan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
19	Anak ingin didahulukan dalam mendapatkan sesuatu	Kerap	
		Kadang-Kadang	
		Tidak pernah	
20	Anak sulit melakukan permainan secara bergantian	Kerap	
		Kadang-Kadang	
		Tidak pernah	
21	Anak menyerobot giliran orang lain	Kerap	
		Kadang-Kadang	
		Tidak pernah	
22	Anak merebut benda yang masih digunakan orang lain	Kerap	
		Kadang-Kadang	
		Tidak pernah	
23	Anak merengek dan menangis berlebihan untuk meminta sesuatu	Kerap	
		Kadang-Kadang	
		Tidak pernah	
24	Anak mendorong orang lain untuk mendapatkan yang diinginkan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
25	Anak bermain sendiri meskipun ada teman sebayanya	Kerap	
		Kadang-Kadang	
		Tidak pernah	
26	Anak nampak kurang membaur dengan teman ketika bermain	Kerap	
		Kadang-Kadang	
		Tidak pernah	
27	Anak menolak ajakan bergabung untuk bermain bersama	Kerap	
		Kadang-Kadang	
		Tidak pernah	
28	Anak tak acuh untuk menolong orang lain dalam hal sederhana	Kerap	
		Kadang-Kadang	
		Tidak pernah	
29	Anak mengabaikan orang yang meminta bantuan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
30	Anak pasif dalam kegiatan bermain bersama	Kerap	
		Kadang-Kadang	
		Tidak pernah	
31	Anak mudah meninggalkan tempat duduk ketika diajak berbicara	Kerap	
		Kadang-Kadang	
		Tidak pernah	

No	Gejala	Opsi	Jawab
32	Anak beranjak dari tempat duduk ketika diberikan kegiatan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
33	Anak beranjak dari tempat duduk meskipun tanpa ada perintah	Kerap	
		Kadang-Kadang	
		Tidak pernah	
34	Anak berlari meskipun jarak yang dituju dekat	Kerap	
		Kadang-Kadang	
		Tidak pernah	
35	Anak memanjat kursi atau benda lain, bukan untuk mengambil benda yang tinggi	Kerap	
		Kadang-Kadang	
		Tidak pernah	
36	Anak masih tetap berjalan atau berlari pada tempat yang sama berulang kali	Kerap	
		Kadang-Kadang	
		Tidak pernah	
37	Anak berceloteh, namun bukan untuk berkomunikasi 2 arah	Kerap	
		Kadang-Kadang	
		Tidak pernah	
38	Anak berbicara banyak namun bukan untuk bertanya atau menjawab pertanyaan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
39	Anak aktif mengeluarkan kata-kata, namun tanpa arti dan tujuan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
40	Anak tidak menghiraukan larangan yang diberikan	Kerap	
		Kadang-Kadang	
		Tidak pernah	
41	Anak menjalankan aktifitas tanpa terarah	Kerap	
		Kadang-Kadang	
		Tidak pernah	
42	Anada tidak menghiraukan aturan yang ada	Kerap	
		Kadang-Kadang	
		Tidak pernah	
43	Anak menggoyangkan kaki ketika duduk	Kerap	
		Kadang-Kadang	
		Tidak pernah	
44	Anak sulit berhenti unuk mengetukkan jari pada meja atau benda lain	Kerap	
		Kadang-Kadang	
		Tidak pernah	
45	Anak tampak terburu-buru dalam beraktifitas	Kerap	
		Kadang-Kadang	
		Tidak pernah	

3.5 Perancangan Implementasi

Perancangan implementasi berisi tahapan implementasi algoritme. Dimulai dari membuat gambaran umum algoritme menggunakan diagram, detail perancangan dan perancangan pengujian. Gambaran umum tahap implementasi dapat dilihat pada Gambar 3.2.



Gambar 3.2 Diagram umum implementasi

3.6 Implementasi

Implementasi dilakukan sesuai dengan proses perancangan yang telah dibuat. Implementasi meliputi perhitungan menggunakan bahasa pemrograman Java.

3.7 Pengujian dan Analisis

Pengujian dilakukan untuk mengetahui apakah hasil implementasi sesuai dengan perancangan. Pengujian dilakukan untuk mengetahui tangkat akurasi kemudian dianalisis hasilnya.

3.8 Kesimpulan dan Saran

Kesimpulan didapatkan dari analisis yang telah dilakukan. Saran yang diberikan dapat digunakan oleh peneliti selanjutnya dan untuk pengembangan lebih lanjut.

BAB 4 PERANCANGAN

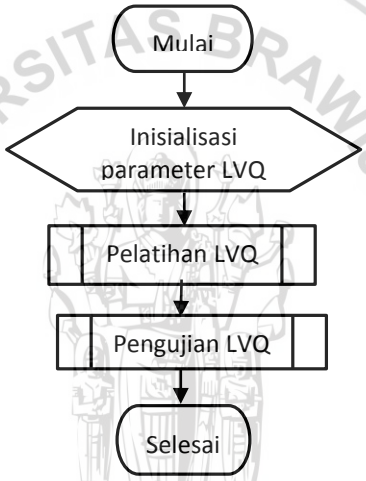
Pada bab empat berisi formulasi perancangan, perancangan algoritme, perhitungan manual, perancangan pengujian dan evaluasi.

4.1 Formulasi Permasalahan

Masalah utama pada penelitia ini adalah pengoptimalan algoritme LVQ menggunakan PSO. Algoritme PSO digunakan untuk mencari vektor bobot yang akan digunakan pada pelatihan LVQ.

4.1.1 Siklus Algoritme LVQ-PSO

Algortima LVQ terdiri dari 2 proses, yaitu: pelatihan LVQ dan pengujian LVQ. Algoritme PSO akan digunakan pada proses pelatihan LVQ. Siklus algoritme LVQ dapat dilihat pada Gambar 4.1.

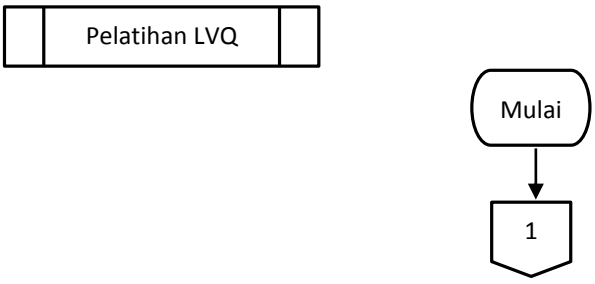


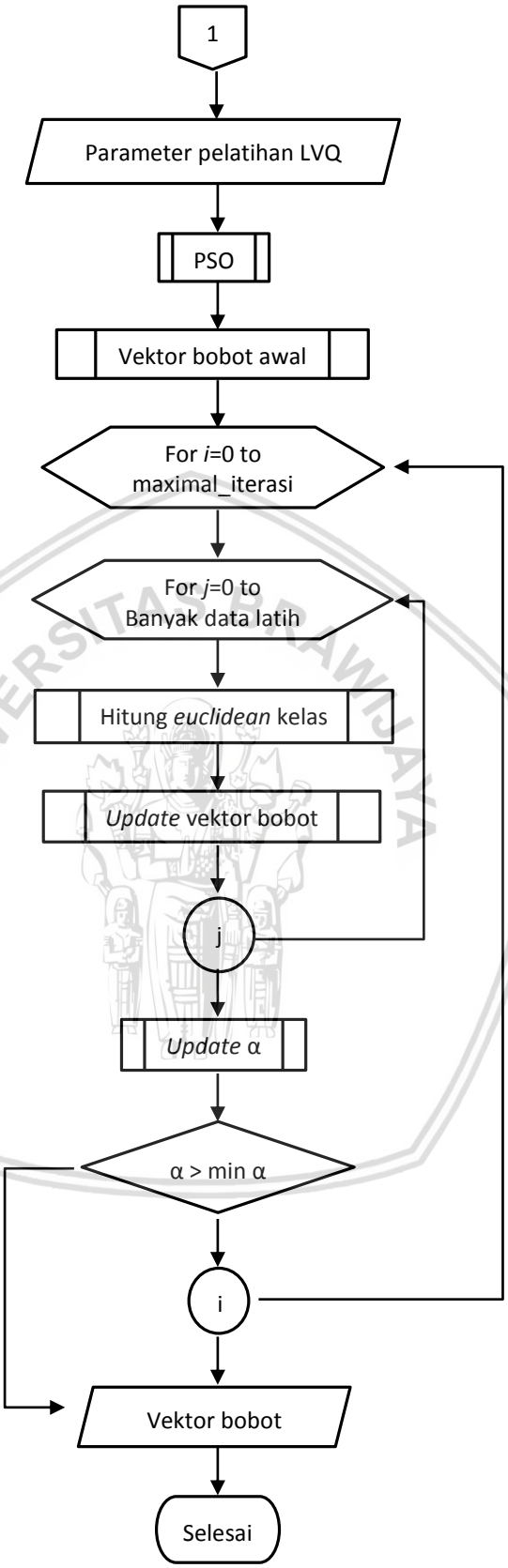
Gambar 4.1 Siklus algoritme LVQ

Parameter yang akan digunakan adalah iterasi, α, min α, dec α, jumlah data latih dan jumlah data uji.

4.1.1.1 Siklus Pelatihan LVQ

Siklus pelatihan LVQ dapat dilihat pada Gambar 4.2.



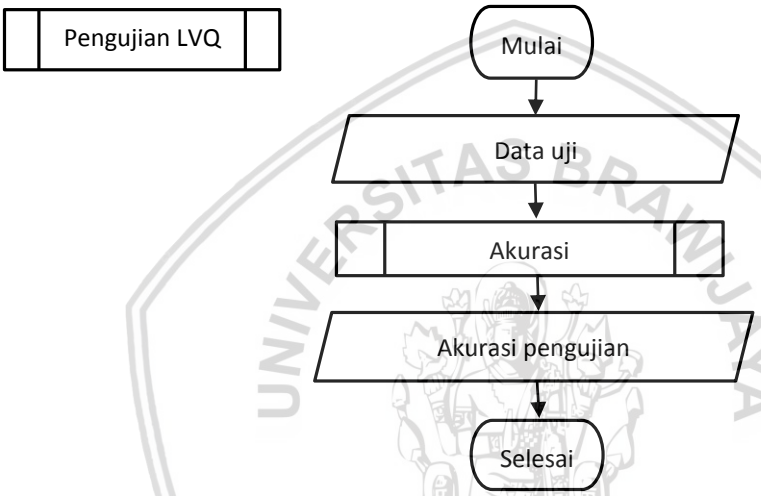


Gambar 4.2 Siklus pelatihan LVQ

Proses PSO merupakan proses yang pertama berjalan untuk mendapatkan partikel terbaik. Partikel terbaik PSO akan dijadikan vektor bobot awal pada proses vektor bobot awal. Lakukan perhitungan hitung *euclidean* kelas dan *update* vektor bobot pada sebanyak data latih. Setelah itu lakukan *update* α. Lakukan pengecekan, jika nilai α <= min α maka perulangan berhenti jika α > min α lakukan perulangan sampai sebanyak maksimal iterasi atau sampai nilai α <= min α (syarat berhenti terpenuhi). *Output* dari pelatihan LVQ adalah vektor bobot yang akan digunakan pada proses pengujian.

4.1.1.2 Siklus Pengujian LVQ

Siklus Pengujian LVQ dapat dilihat pada Gambar 4.3.

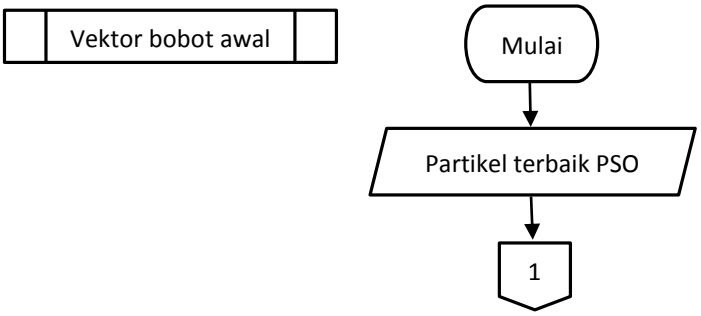


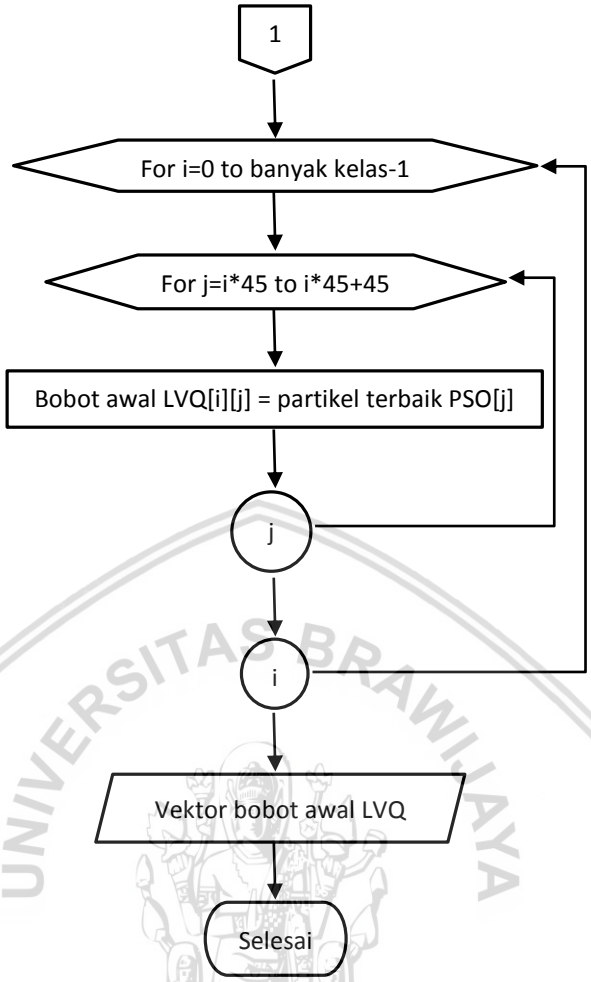
Gambar 4.3 Siklus pengujian LVQ

Pengujian LVQ dilakukan setelah pelatihan LVQ. Tujuan pengujian adalah untuk mengklasifiksasikan data *input* ke dalam kelas yang telah ditentukan. Data *input* diklasifikasikan dengan cara dicari jarak *euclidean* terkecil terhadap kelas-kelas klasifikasi. Parameter yang digunakan adalah vektor data uji (data *input*) dan vektor bobot acuan (didapat dari pelatihan LVQ).

4.1.1.3 Siklus Vektor Bobot Awal

Siklus vektor bobot awal dapat dilihat pada Gambar 4.4.



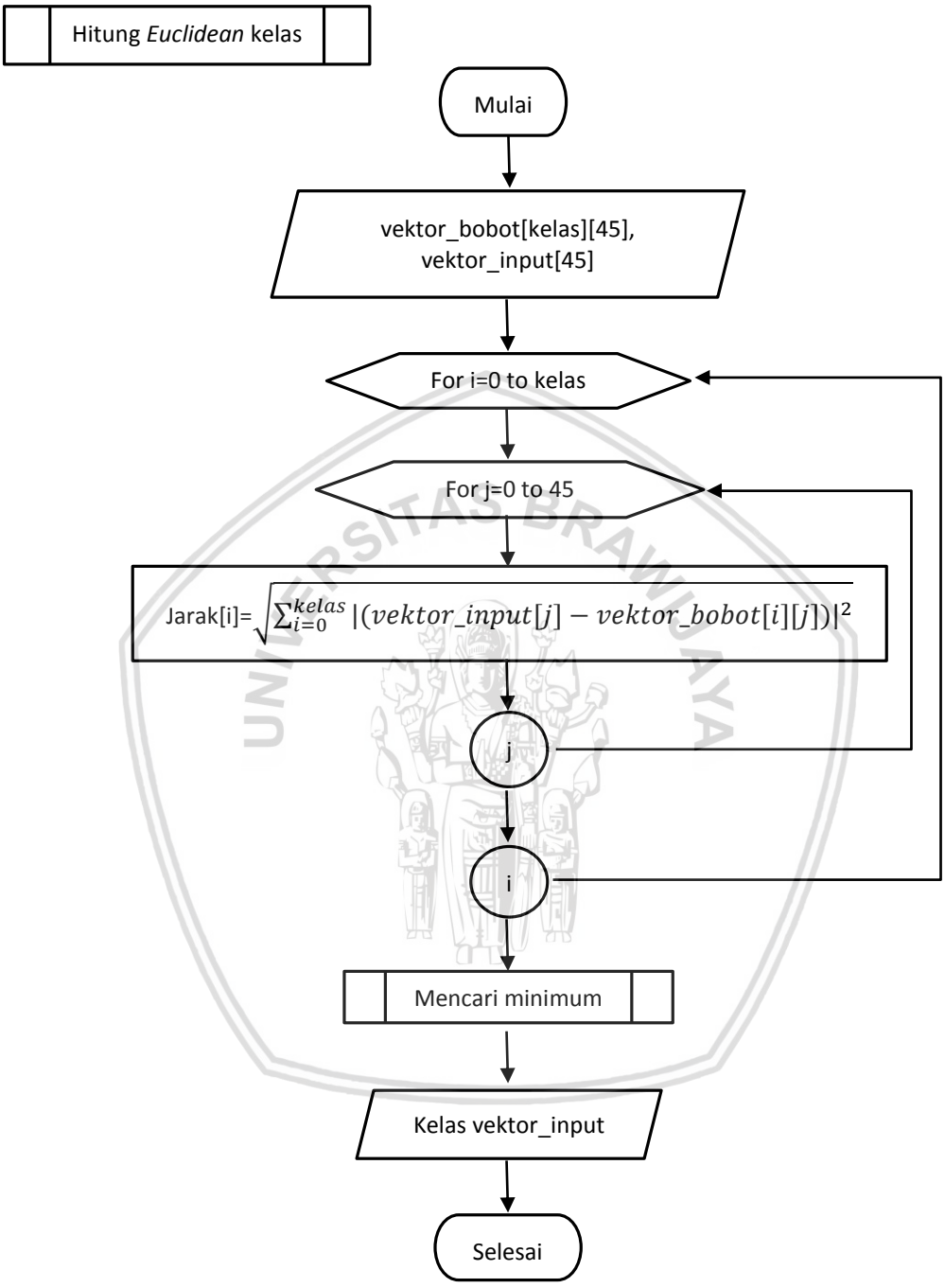


Gambar 4.4 Siklus vektor bobot awal

Proses vektor bobot awal bertujuan untuk merubah partikel terbaik PSO (panjang partikel PSO 1 x 180 dimensi) menjadi vektor bobot LVQ menjadi berukuran 4 x 45, ukuran tersebut berarti 45 data dari setiap kelas klasifikasi. Masukan pada proses ini adalah partikel terbaik partikel PSO, kemudian lakukan perulangan pada setiap 45 dimensi (perulangan parameter *j*) sebanyak kelas klasifikasi (perulangan parameter *i*). *Output* dari proses ini adalah vektor bobot awal untuk pelatihan LVQ.

4.1.1.4 Siklus Hitung *Euclidean* Kelas

Siklus hitung *euclidean* kelas dapat dilihat pada Gambar 4.5.

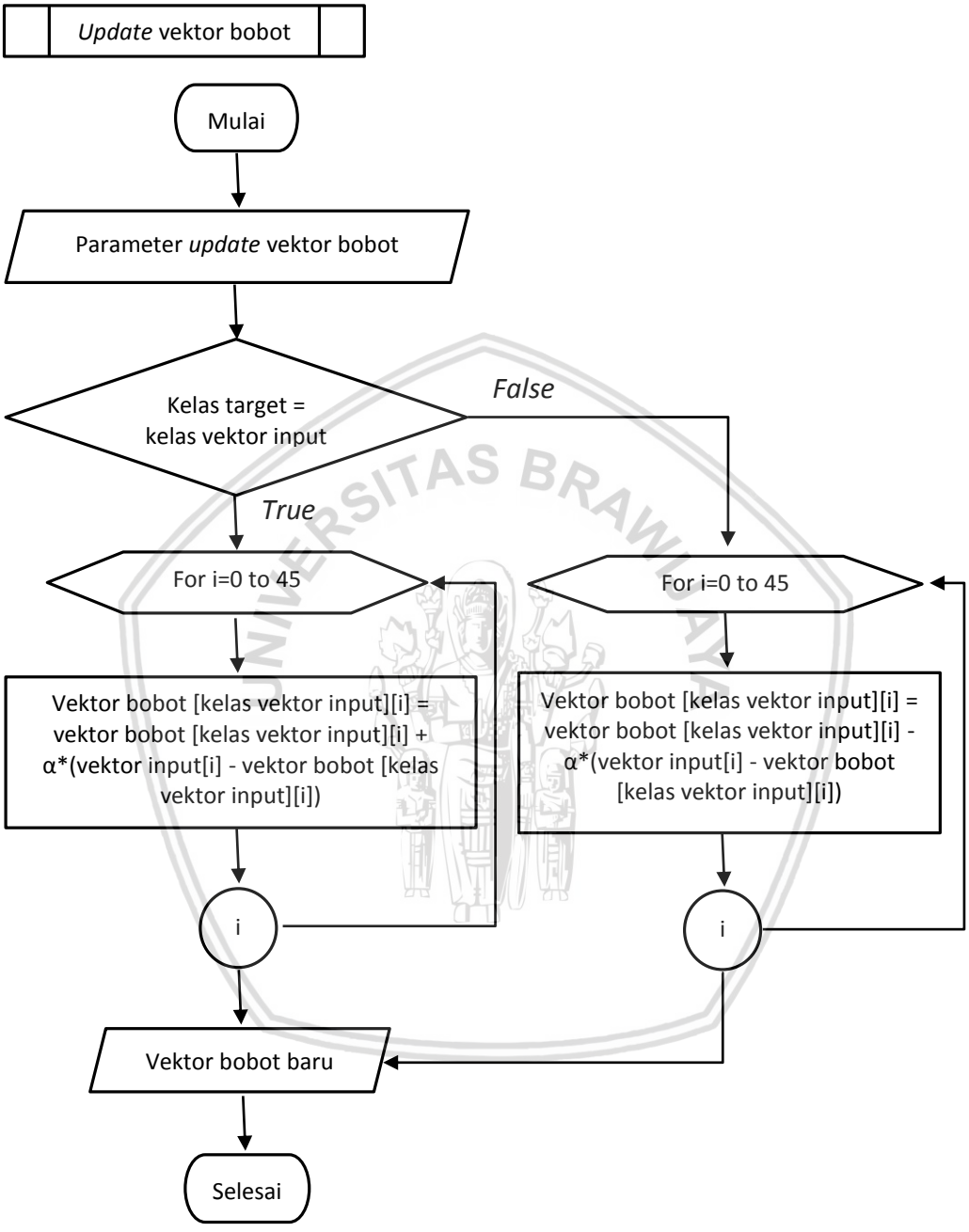


Gambar 4.5 Siklus hitung *Euclidean* kelas

Proses hitung *euclidean* kelas digunakan untuk menghitung jarak *euclidean* vektor *input* (data baru) terhadap kelas klasifikasi. *Input* yang digunakan adalah vektor bobot dan vektor yang akan diklasifikasikan. Vektor *input* dihitung jaraknya terhadap setiap vektor bobot. Kemudian dicari jarak terkecil terhadap setiap kelas menggunakan proses mencari minimum. *Output* dari proses ini adalah vektor *input* yang terklasifikasi ke dalam sebuah kelas.

4.1.1.5 Siklus *Update* Vektor Bobot

Siklus *Update* Vektor Bobot dapat dilihat pada Gambar 4.6.



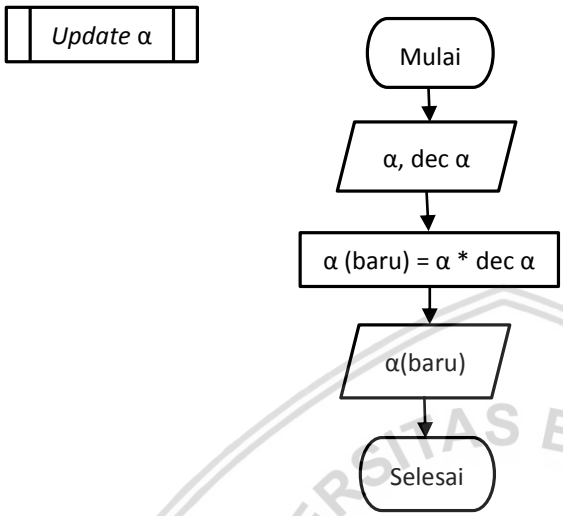
Gambar 4.6 Siklus *update* vektor bobot

Parameter yang digunakan pada proses ini adalah kelas target (kelas pada data), kelas vektor *input* (kelas hasil perhitungan *euclidean*), α dan vektor bobot. Apabila kelas target sesuai dengan kelas vektor *input*, maka *update* bobot menggunakan Persamaan 2.3. Bila kelas target tidak sama dengan kelas vektor *input*, maka vektor bobot di-*update* menggunakan Persamaan 2.4. *Update* bobot

dilakukan untuk setiap dimensi panjang data (parameter *i*). *Output* dari proses ini adalah vektor bobot baru.

4.1.1.6 Siklus *Update Learning Rate* (α)

Siklus *update* α dapat dilihat pada Gambar 4.7.

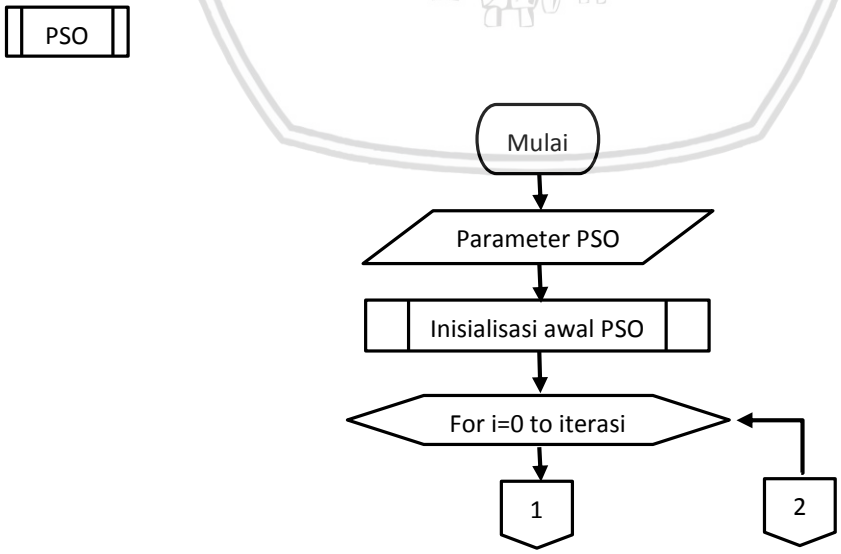


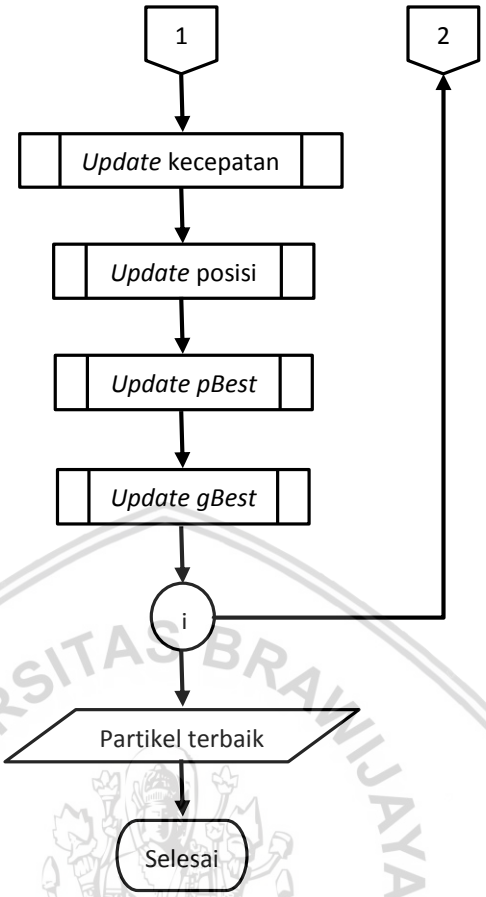
Gambar 4.7 Siklus *update* α

Proses *update* α digunakan untuk mendapatkan nilai α (baru). Nilai α(baru) didapat dengan mengalikan α(lama) dan dec α (pengurang α / pengali α).

4.1.2 Siklus Algoritme PSO

Algoritme PSO digunakan untuk mendapatkan partikel terbaik yang akan digunakan sebagai vektor bobot awal pada pelatihan LVQ. Siklus PSO dapat dilihat pada Gambar 4.8.



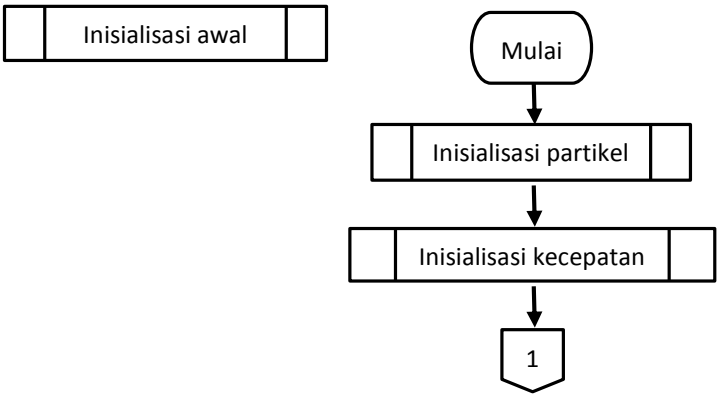


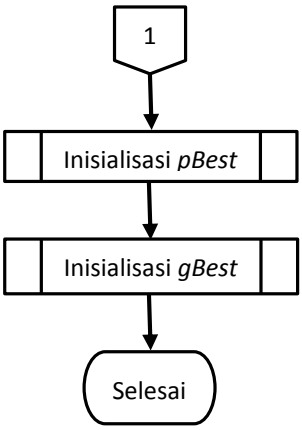
Gambar 4.8 Siklus Algoritme PSO

Parameter yang digunakan dalam algoritme PSO adalah ukuran *swarm* (banyak partikel), maksimal iterasi, c1, c2, r1, r2, jumlah data latih PSO, banyak partikel dan w. Setelah melakukan inisilisasi awal, lakukan perulangan sampai batas iterasi maksimal untuk proses *update* kecepatan, *update* posisi, *update pBest* dan *update gBest*. *Output* akhir adalah satu partikel terbaik.

4.1.2.1 Siklus Inisialisasi Awal

Siklus inisialisasi awal dapat dilihat pada Gambar 4.9.



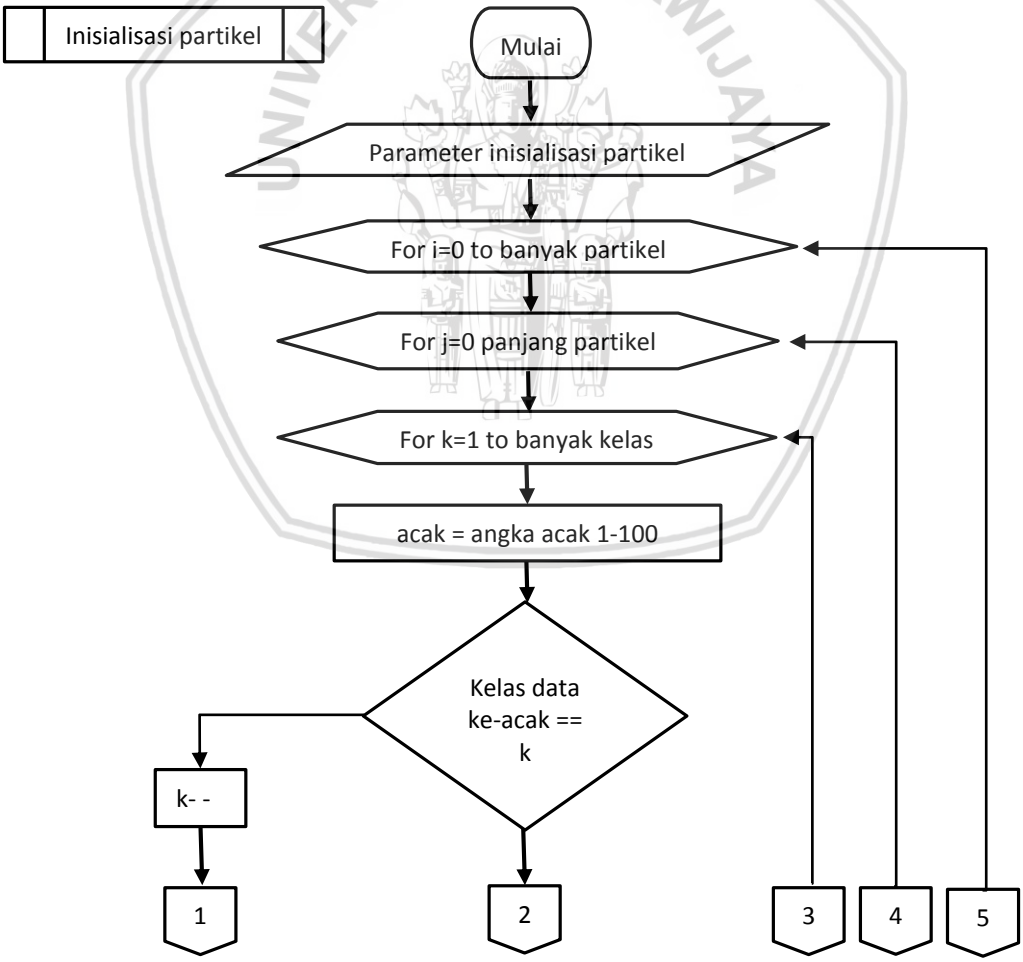


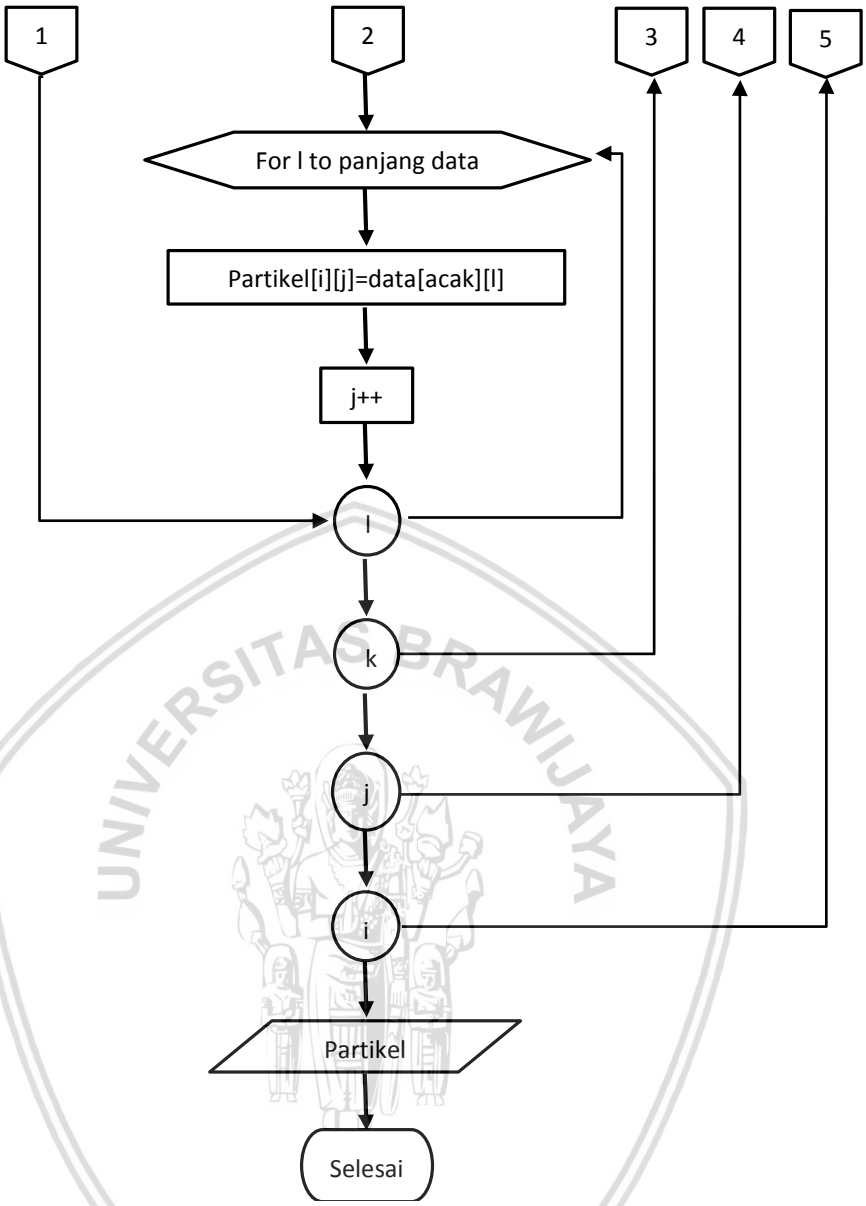
Gambar 4.9 Siklus inisialisasi awal

Pada proses ini yang diinisialisasi adalah partikel, kecepatan, *pBest*, dan *gBest*.

4.1.2.2 Siklus Inisialisasi Partikel

Siklus inisialisasi partikel dapat dilihat pada Gambar 4.10.





Gambar 4.10 Siklus inisialisasi partikel

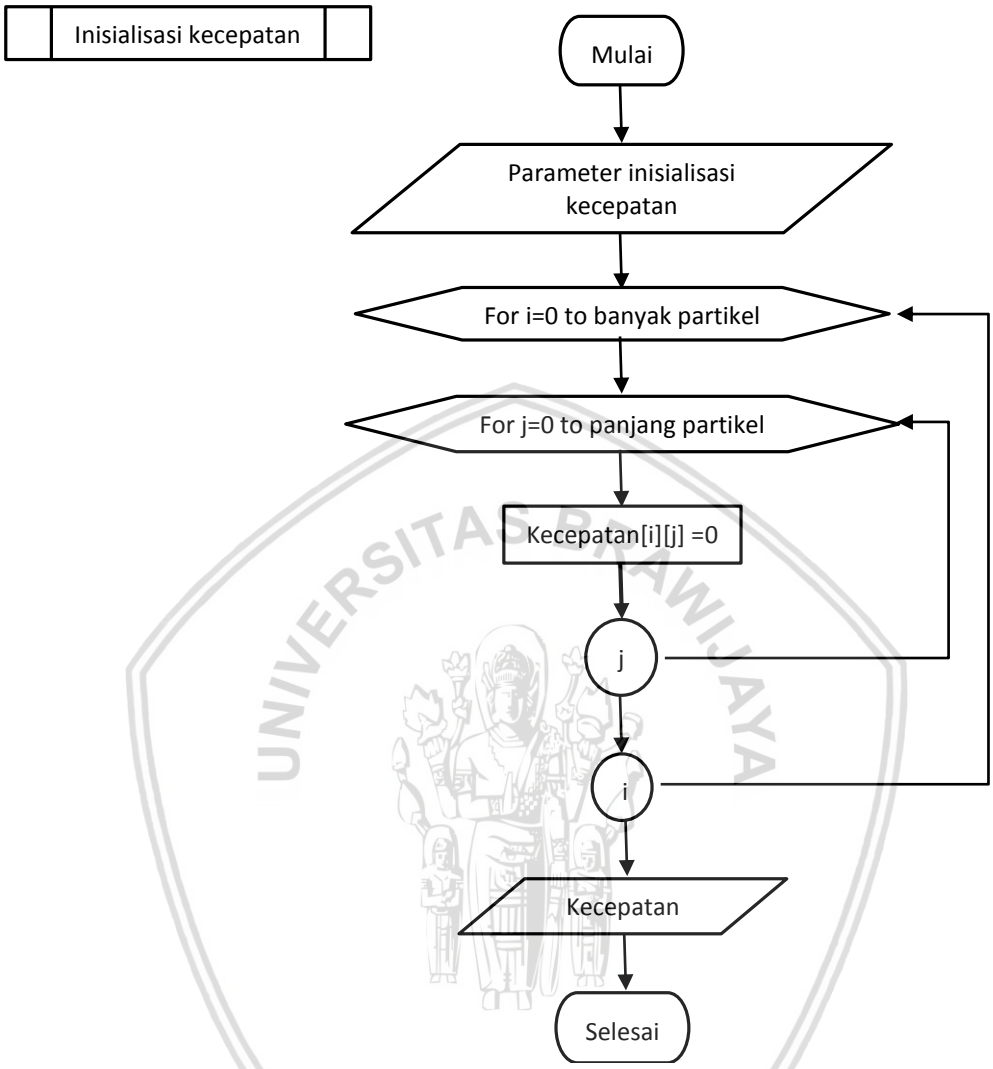
Proses inisialisasi partikel digunakan untuk membentuk partikel sebanyak ukuran *swarm* yang telah ditentukan. Sebuah partikel akan mewakili data dari 4 kelas klasifikasi, sehingga partikel akan memiliki panjang 45 x 4 (180) dimensi.

Parameter *i* digunakan sebagai parameter perulangan banyak partikel (ukuran *swarm*), parameter *j* digunakan untuk perulangan panjang partikel (180 dimensi), parameter *k* digunakan perulangan sebanyak kelas klasifikasi (4) dan parameter *l* digunakan untuk perulangan panjang data setiap kelas (45). Parameter *acak* digunakan untuk memilih baris secara acak dari seluruh data (100 data).

Untuk setiap partikel, akan dipilih baris data secara acak, bila kelas klasifikasi data tersebut sama dengan nilai *k*, maka data akan dimasukkan ke dalam partikel, jika berbeda maka akan dilakukan lagi (pengurangan nilai *k*) pemilihan baris secara acak sampai mendapatkan kelas klasifiksai data sama dengan nilai *k*. Dengan cara seperti ini, maka setiap partikel akan dapat mewakili 4 kelas klasifikasi.

4.1.2.3 Siklus Inisialisasi Kecepatan

Siklus inisialisasi Kecepatan dapat dilihat pada Gambar 4.11.

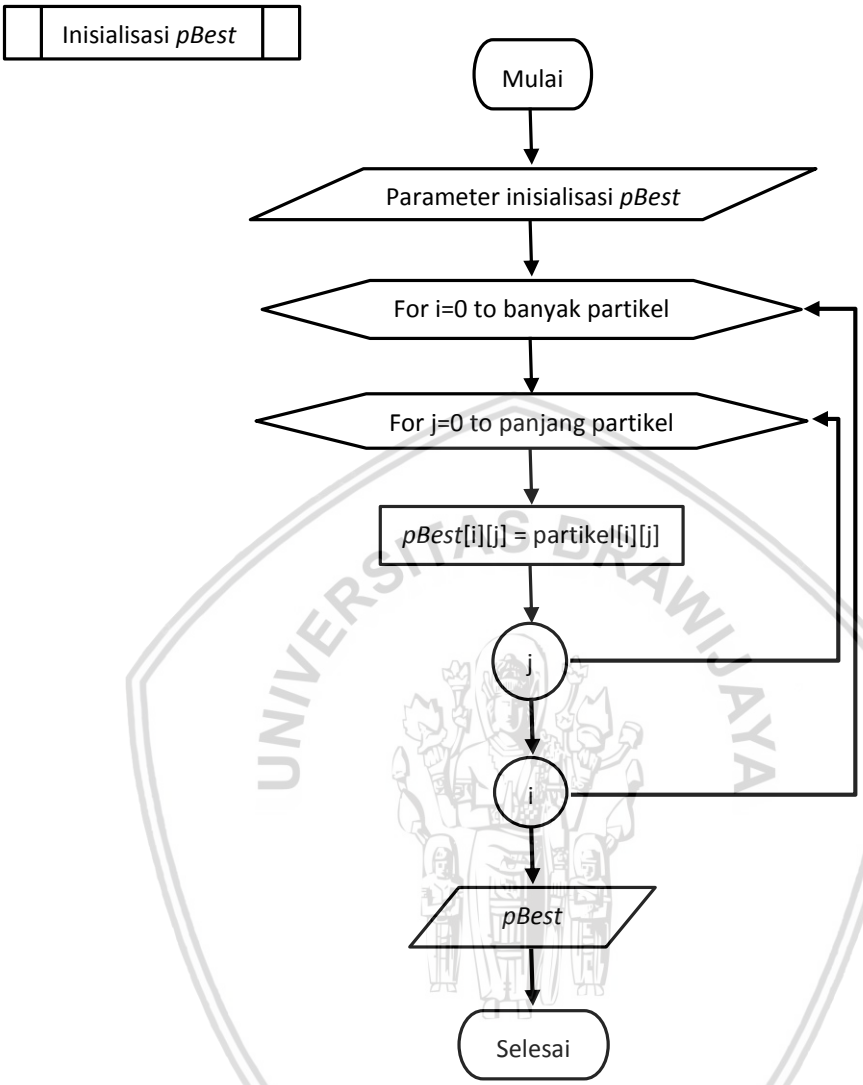


Gambar 4.11 Siklus inisialisasi kecepatan

Kecepatan awal seluruh partikel adalah 0, sehingga dilakukan perulangan untuk setiap dimensi, kemudian menginisialisasi setiap dimensi dengan nilai 0.

4.1.2.4 Siklus Inisialisasi *pBest*

Siklus inisialisasi *pBest* dapat dilihat pada Gambar 4.12.

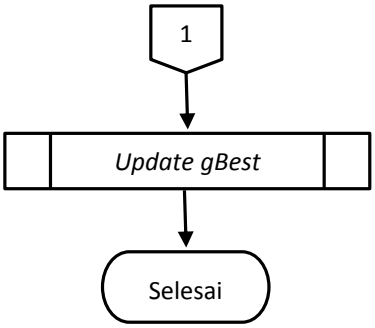


Gambar 4.12 Siklus inisialisasi *pBest*

Parameter yang digunakan adalah banyak partikel dan panjang partikel. Nilai *pBest* awal sama dengan nilai partikel. Untuk setiap *pBest*, diinisialisasikan nilai yang sama dengan partikel.

4.1.2.5 Siklus Inisialisasi *gBest*

Siklus inisialisasi *gBest* dapat dilihat pada Gambar 4.13.

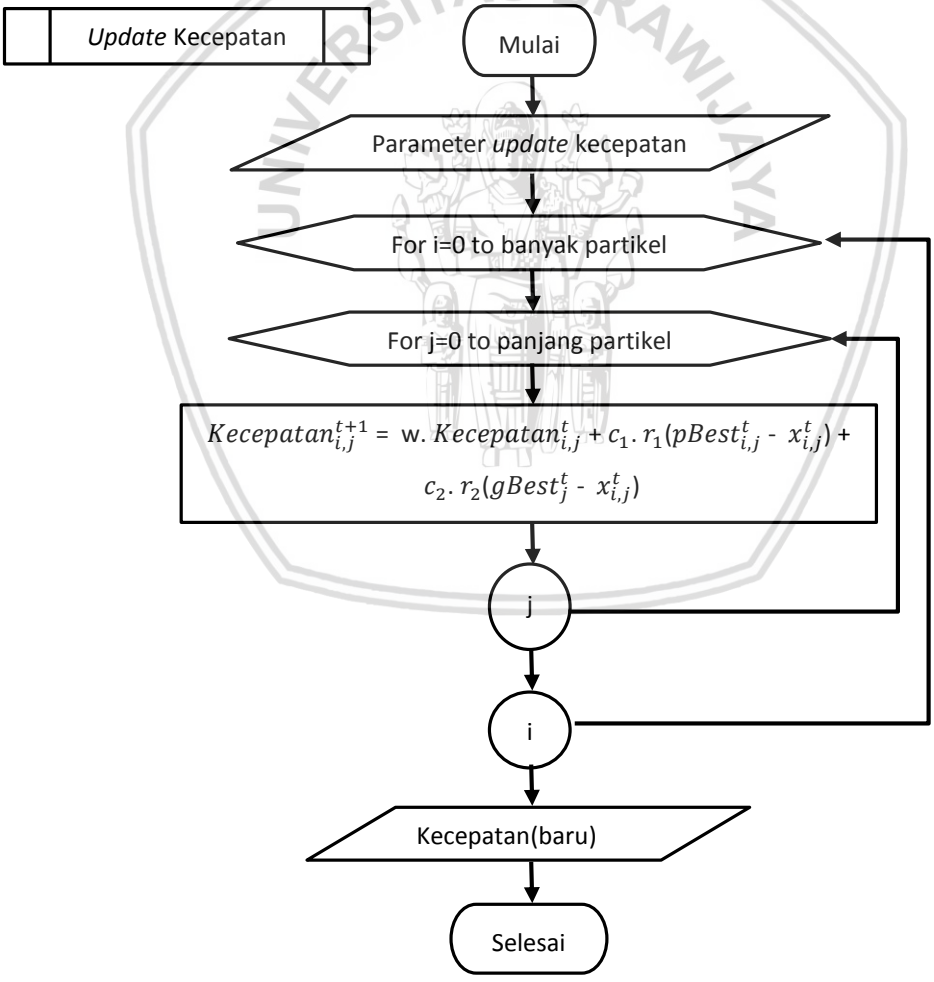


Gambar 4.13 Siklus inisialisasi *gBest*

Parameter yang digunakan adalah nilai *pBest*. Mencari nilai *gBest* awal sama dengan proses *update gBest*, sehingga proses inisialisasi *gBest* akan memanggil proses *update gBest*.

4.1.2.6 Siklus *Update* Kecepatan

Siklus *update* kecepatan dapat dilhat pada Gambar 4.14.

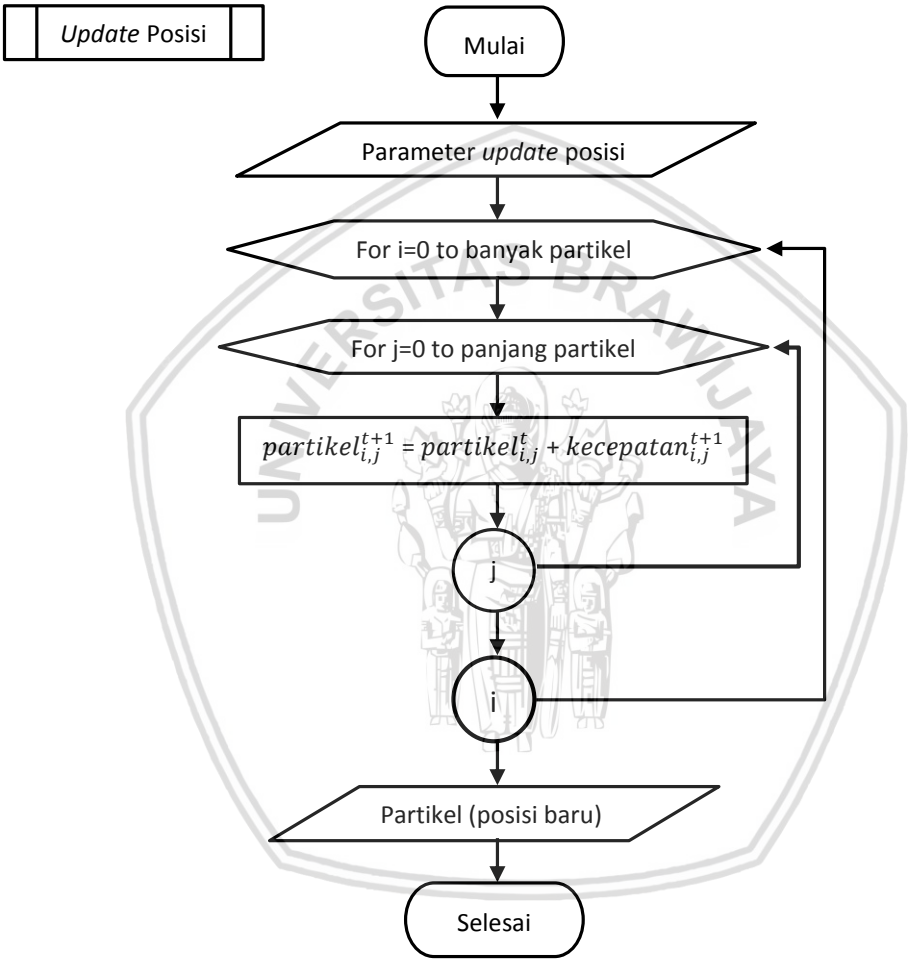


Gambar 4.14 Siklus *update* kecepatan

Parameter yang digunkan pada proses *update* kecepatan adalah *w, c1, c2, r1, r2, pBest, gBest,* partikel, banyak partikel dan panjang partikel. Parameter *i* digunakan untuk perulangan sebanyak partikel dan parameter *j* digunakan untuk perulangan sebanyak panjang partikel. Pada setiap kecepatan partikel lakukan *update* menggunakan Persamaan 2.7. *Output* dari proses ini adalah kecepatan partikel yang telah di-*update* (baru).

4.1.2.7 Siklus *Update* Posisi

Siklus *update* posisi dapat dilihat pada Gambar 4.15.

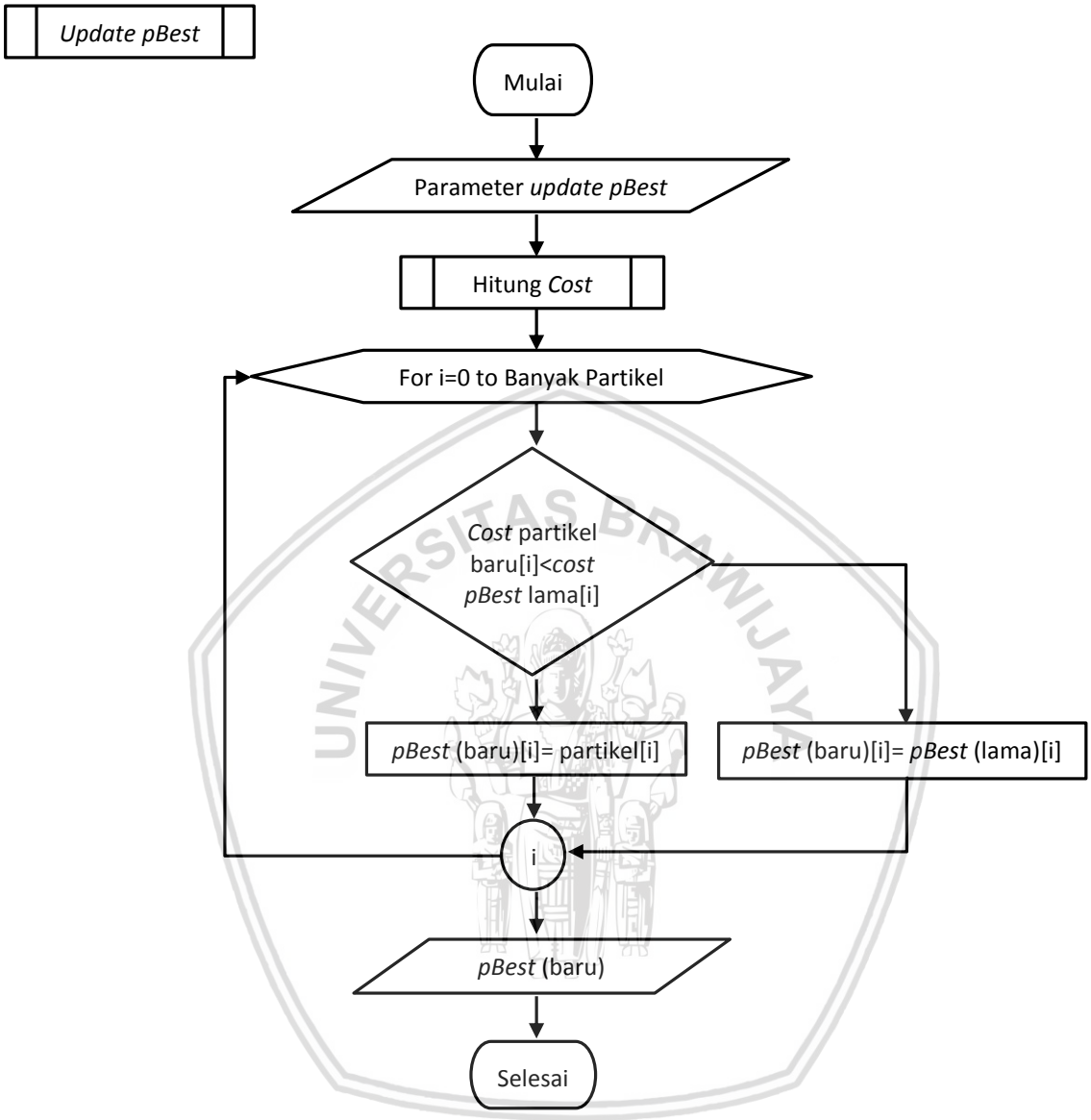


Gambar 4.15 Siklus *update* posisi

Parameter yang digunakan pada proses *update* posisi adalah partikel pada iterasi sebelumnya dan kecepatan partikel terbaru. Untuk setiap partikel (parameter perulangan *i*), lakukan *update* posisi partikel menggunakan Persamaan 2.8 pada setiap dimensi partikel (parameter perulangan *j*).

4.1.2.8 Siklus *Update pBest*

Siklus *update pBest* dapat dilihat pada Gambar 4.16.

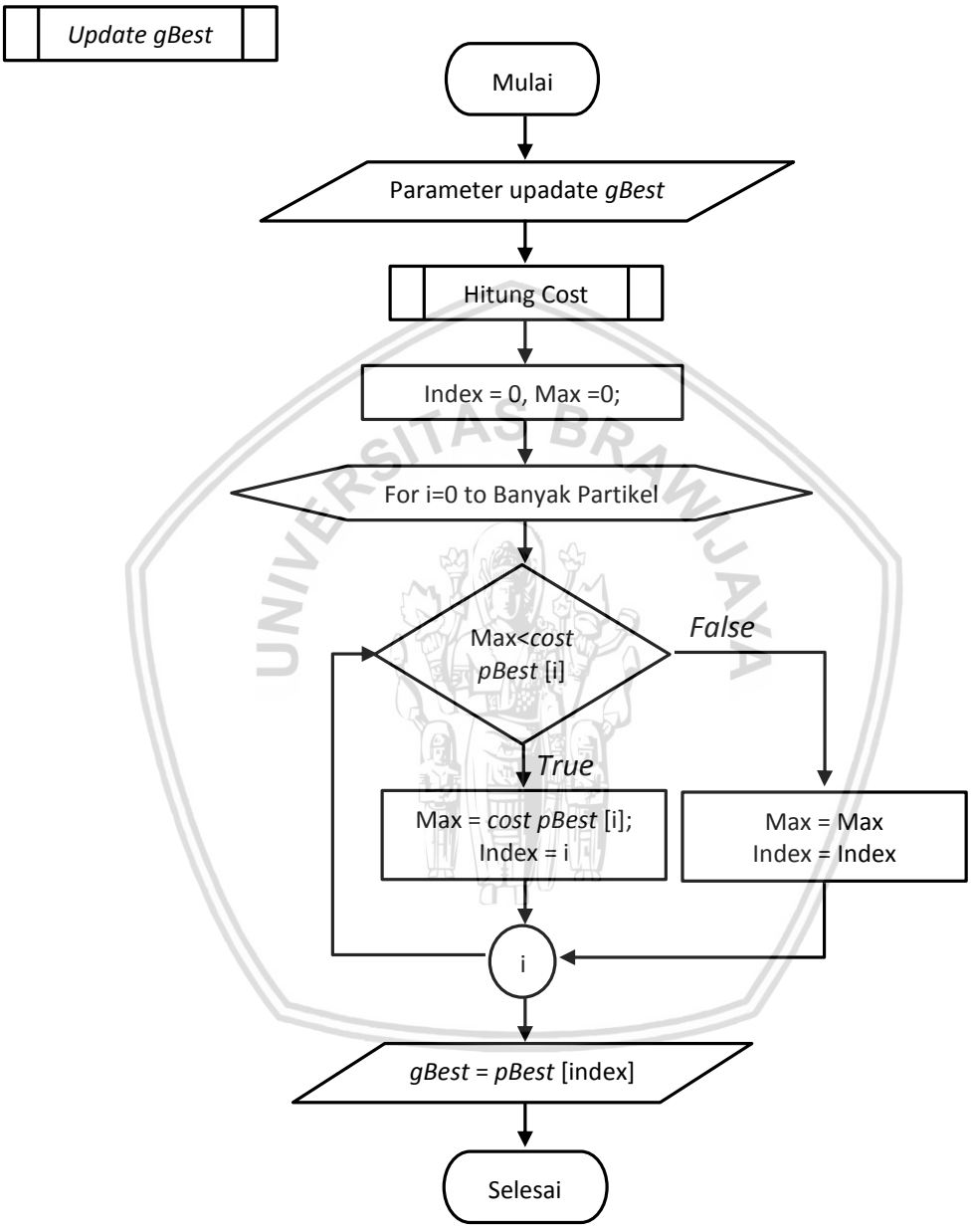


Gambar 4.16 Siklus *update pBest*

Proses *update pBest* adalah menentukan nilai *pBest* terbaru dengan cara membandingkan *cost* (memilih *cost* terkecil) *pBest* lama (iterasi sebelumnya) dan cost partikel terbaru (iterasi sekarang), parameter yang digunakan adalah banyak partikel, partikel terbaru dan *pBest* pada iterasi sebelumnya. Hitung Cost yang dilakukan pada proses *update pBest* adalah menghitung *cost* partikel terbaru menggunakan Persamaan 2.9 dan Persamaan 2.10. Apabila nilai *cost* partikel lebih kecil daripada nilai *cost pBest*, maka nilai *pBest* baru akan sama dengan nilai partikel. Bila sebaliknya, maka nilai *pBest* tidak akan berubah.

4.1.2.9 Siklus *Update gBest*

Proses *update gBest* digunakan untuk mencari nilai *gBest* terbaru (*pBest* dengan nilai *cost* terkecil). Siklus *update gBest* dapat dilihat pada Gambar 4.17.



Gambar 4.17 Siklus *update gBest*

Parameter yang digunakan adalah *pBest* dan banyak partikel. Variabel *Max* digunakan untuk menyimpan nilai *cost* yang akan dibandingkan, variabel *Index* digunakan untuk menyimpan index *pBest* dengan nilai *cost* terkecil.

4.2 Perhitungan Manual LVQ

Pada subbab ini akan dijelaskan perhitungan manualisasi pelatihan dan pengujian vektor bobot LVQ tanpa menggunakan algoritme PSO. Untuk meyederhanakan manualisasi, gejala yang digunakan hanya 5 gejala awal (vektor sebenarnya terdiri dari 45 gejala). Parameter yang akan digunakan adalah:

a. α = 0,1

b. *dec* α = 0,1

c. iterasi = 1

d. data latih = 5

e. data uji = 5

4.2.1 Vektor Bobot Awal LVQ

Vektor bobot awal yang akan digunakan adalah vektor bobot dari masing-masing kelas yang dipilih secara acak. Pada manualisasi ini menggunakan data D_061 untuk mewakili kelas target 1, D_059 untuk mewakili kelas target 2, D_084 untuk mewakili kelas target 3 dan D_043 untuk mewakili kelas target 4 seperti pada Tabel 4.1.

Tabel 4.1 Vektor bobot awal LVQ

Bobot	Vektor bobot				
W1	35	35	50	50	50
W2	35	35	50	15	35
W3	35	35	35	35	35
W4	15	15	15	15	15

4.2.2 Data Latih LVQ

Data latih digunakan sebagai pertimbangan untuk memperbaiki vektor bobot LVQ. Data latih dipilih secara acak. Data latih untuk manualisasi ini ada pada Tabel 4.2.

Tabel 4.2 Data latih LVQ

Data	Vektor					Target
D_045	50	50	50	50	50	1
D_097	35	50	50	35	35	2
D_080	15	35	15	35	15	3
D_046	15	15	15	15	15	4
D_056	35	35	35	35	35	2

4.2.3 Manualisasi Hitung Jarak *Euclidean* dan *Update* Vektor Bobot LVQ

Hitung jarak *euclidean* antara data latih pada Tabel 4.2 dan vektor bobot pada Tabel 4.1. Jarak data latih D_045 dengan vektor bobot dihitung sebagai berikut:

$D(w_{1,}\ L_1)= \sqrt{(35-50)^2+\ (35-50)^2+(50-50)^2+(50-50)^2+(50-50)^2}$

$D(w_{1,}\ \ L_1)$= 21.21

$D(w_{2,}\ L_1)= \sqrt{(15-50)^2+\ (15-50)^2+(15-35)^2+(15-35)^2+(15-50)^2}$

$D(w_{2,}\ L_1)$= 66.8

$D(w_{3,}\ L_1)= \sqrt{(35-50)^2+\ (35-50)^2+(35-35)^2+(15-35)^2+(35-50)^2}$

$D(w_{3,}\ L_1)$= 32.7

$D(w_{4,}\ L_1)= \sqrt{(35-50)^2+\ (35-50)^2+(35-35)^2+(15-35)^2+(50-50)^2}$

$D(w_{4,}\ L_1)$= 29.1

Keterangan:

$D(w,\ L)$ = jarak e*uclidean* antara w dan L.

w_i = vektor bobot ke-i

Li_i = data latih ke-i.

Jarak terkecil data latih pertama terhadap vektor bobot adalah 21.21, sehinngga data latih 1 terklasifikasi ke dalam kelas 1. Karena hasil perthitungan sesuai dengan kelas target, *update* vektor bobot dihitung menggunakan Persamaan 2.3 pada vektor bobot pertama. Hasilnya adalah seperti pada Tabel 4.3.

Tabel 4.3 Vektor bobot *update* pertama

Bobot	Vektor bobot				
W1	50	50	48.5	35	50
W2	15	15	15	15	15
W3	35	35	35	15	35
W4	35	35	35	15	50

Lakukan perulangan perhitungan jarak *euclidean* terhadap data latih kedua. Maka hasilnya data latih kedua terklasifikasi ke kelas 3. Karena hasil perhitungan tidak sesuai dengan kelas target, maka vektor bobot di-*update* dengan Persamaan 2.4 dengan hasil seperti pada Tabel 4.4.

Tabel 4.4 Vektor bobot *update* kedua

Bobot	Vektor bobot				
W1	50	50	48.5	35	50
W2	15	15	15	15	15
W3	33.5	37	35	13	37
W4	35	35	35	15	50

Ulangi proses hitung jarak *euclidean* dan *update* vektor bobot sampai semua data latih, maka hasil vektor bobot akhir adalah seperti pada Tabel 4.5.

Tabel 4.5 Vektor bobot akhir pelatihan LVQ

Bobot	Vektor bobot				
W1	36.5	36.5	50	50	50
W2	35	35	50	15	35
W3	35	33.5	33.5	35	35
W4	13	11.02	13	11.02	13

4.2.4 Manualisasi *Update* α LVQ

Setelah didapatkan vektor bobot akhir, maka *update* nilai α menggunakan Persamaan 2.5, maka nilai α adalah:

$\alpha(\text{baru}) = 0{,}1 \times 0{,}1$

$\alpha(\text{baru}) = 0{,}001$

4.2.5 Manualisasi Pengujian LVQ

Data uji yang akan digunakan pada pengujian ini adalah sepeti pada Tabel 4.6.

Tabel 4.6 Data uji manualisasi pengujian LVQ

Data uji	Bobot					Kelas target
D_071	35	35	50	35	35	1
D_041	50	15	35	35	15	2
D_087	35	35	15	35	15	3
D_100	15	15	15	15	15	4
D_012	50	50	50	35	50	1

Proses menghitung jarak *euclidean* data uji D_071 terhadap vektor bobot pada Tabel 4.5 sebagai berikut:

$D(w_{1,}\ U_1) = \sqrt{(36.5-35)^2 + (36.5-35)^2 + (50-50)^2 + (50-35)^2 + (50-35)^2}$

$D(w_{1,}\ U_1) = 21.31$

$D(w_{2,}\ U_1) = \sqrt{(35-35)^2 + (35-35)^2 + (50-50)^2 + (15-35)^2 + (35-35)^2}$

$D(w_{2,}\ U_1) = 20$

$D(w_{3,}\ U_1) = \sqrt{(35-35)^2 + (33.5-35)^2 + (33.5-50)^2 + (35-35)^2 + (35-35)^2}$

$D(w_{3,}\ U_1) = 16.56$

$D(w_{4,}\ U_1) =$
$\sqrt{(13-35)^2 + (11.02-35)^2 + (13-50)^2 + (11.02-35)^2 + (13-35)^2}.$

$D(w_{4,}\ U_1) = 59.051$

Keterangan:

$D(w, U_1)$ = jarak *Euclidean* antara w dan U.

w_i = vektor bobot ke-i

U_i = data uji ke-i.

Jarak terkecil antara data uji pertama terhadap vektor bobot adalah 16.56, maka data uji D_071 terklasifikasi ke dalam kelas 3.

Ulangi proses klasifikasi, maka hasil dapat dilihat seperti Tabel 4.7.

Tabel 4.7 Kesesuaian kelas data uji LVQ

Data uji	Kelas LVQ	Kelas target
D_071	3	1
D_041	3	2
D_087	3	3
D_100	4	4
D_012	1	1

4.2.6 Manualisasi Hitung Akurasi LVQ

Hitung akurasi menggunakan Persamaan 2.6, maka proses perhitungannya sebagai berikut:

Akurasi $= \frac{3}{5}$ x 100% = 60%

Dari hasil tersebut diketahui bahwa akurasi pada pengujian algoritme LVQ (tanpa PSO) adalah sebesar 60%.

4.3 Perhitungan Manual LVQ Menggunakan PSO

Pada subbab ini akan dijelaskan perhitungan manualisasi pelatihan vektor bobot LVQ dengan menggunakan algoritme PSO. Algoritme PSO dijalankan dalam proses pelatihan LVQ, sehingga PSO merupakan proses yang berjalan terlebih dahulu.

4.3.1 Manualisasi PSO

Partikel yang digunakan dalam manualisasi PSO ini adalah partikel yang disederhanakan menjadi 20 dimensi, sedangkan panjang partikel yang digunakan sebenarnya adalah 180 dimensi. Parameter yang digunakan pada manualisasi PSO adalah:

a. Ukuran *swarm* = 3
b. Iterasi= 2
c. $C_1$1 =0,5 dan C_2=0,5
d. w =0,8
e. r_1=0,2 dan r_2=0,3

4.3.1.2 Representasi Partikel dan Perhitungan *Cost*

Panjang dimensi sebuah partikel didapat dari banyaknya parameter dikali banyak kelas target. Misalkan terdapat 5 parameter gejala ADHD dan 4 jenis kelas target (*inattention, hyperactivity, impulsivity*, dan tidak ADHD), maka panjang sebuah partikel adalah 20 dimensi. Dimensi 1 sampai 5 diisi bobot untuk mewakili kelas target 1 (*inattention*), dimensi 6 sampai 10 adalah bobot pada kelas target 2 (*hyperactivity*), dimensi 11 sampai 15 adalah bobot pada kelas target 3 (*impulsivity*) dan dimensi 16 sampai 20 adalah bobot pada kelas target 4 (tidak ADHD). Penentuan 4 data untuk mewakili masing- masing kelas pada sebuah partikel dilakukan secara acak. Misal terpilih data D_001, D_007, D_023 dan D_031 untuk menjadi 1 partikel, maka representasi partikel dapat dilihat pada Tabel 4.8.

Tabel 4.8 Representasi partikel PSO

Partikel	Dimensi Partikel																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
X1	50	50	35	35	50	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
	D_001					D_007					D_023					D_031				

Sebelum menghitung *cost*, terlebih dahulu menentukan data yang akan digunakan sebagai pembanding hasil klasifikasi dari partikel PSO sebagai vector bobot dan kelas target. Misalkan digunakan 5 data untuk menghitung *cost*. Data yang digunakan untuk menghitung *cost* dapat dilihat pada Tabel 4.9.

Tabel 4.9 Data latih PSO

No	Data	GEJALA					Target
		G_01	G_02	G_03	G_04	G_05	
1	D_045	50	50	50	50	50	1
2	D_097	35	50	50	35	35	2
3	D_080	15	35	15	35	15	3
4	D_046	15	15	15	15	15	4
5	D_056	35	35	35	35	35	2

Selanjutnya hitung jarak *euclidean* setiap data latih PSO dengan sebuah partikel. Setiap partikel merepresentasikan bobot untuk 4 kelas target, sehingga untuk mecari kelas suatu data latih PSO diperlukan perhitungan jarak *euclidean* secara berulang menggunakan 1 partikel yang sama namun dengan dimensi partikel yang berbeda.

Menghitung jarak *euclidean* antara data latih PSO pertama (D_045) dengan X1 pada Tabel 4.8:

$$D\left(X_{1,\sum_{j=1}^{5} j,}\, U_1\right) =$$

$$\sqrt{(50-50)^2 + (50-50)^2 + (35-50)^2 + (35-50)^2 + (50-50)^2}$$

$D\left(X_{1,\sum_{j=1}^{5} j,}\ U_1\right)$ = 21. 21

$D\left(X_{1,\sum_{j=5}^{10} j,}\ U_1\right)$ =

$$\sqrt{(15-50)^2 + (15-50)^2 + (15-50)^2 + (15-50)^2 + (15-50)^2}$$

$D\left(X_{1,\sum_{j=5}^{10} j,}\ U_1\right)$ = 78.26

$D\left(X_{1,\sum_{j=11}^{15} j,}\ U_1\right)$ =

$$\sqrt{(15-50)^2 + (15-50)^2 + (15-50)^2 + (15-50)^2 + (15-50)^2}$$

$D\left(X_{1,\sum_{j=11}^{15} j,}\ U_1\right)$= 78.26

$D\left(X_{1,\sum_{j=16}^{20} j,}\ U_1\right)$ =

$$\sqrt{(15-50)^2 + (15-50)^2 + (15-50)^2 + (15-50)^2 + (15-50)^2}$$

$D\left(X_{1,\sum_{j=16}^{20} j,}\ U_1\right)$ = 78.262

Keterangan:

$D\left(X_{,}\ U\right)$ = jarak *euclidean* antara X dan U.

$X_{i,\sum_{j=n}^{k} j,}$ = Partikel ke-i dimensi j=n sampai dimensi ke-k.

U_i =Data latih PSO ke-i.

Jarak *euclidean* terkecil adalah 21.21, maka kelas data latih PSO pertama terhadap partikel pertama adalah kelas 1.

Setelah perhitungan kelas dilakukan untuk semua data latih PSO terhadap partikel, hasil akhir perhitungan dapat dilihat pada Tabel 4.10.

Tabel 4.10 Kesesuaian data uji PSO

No	Data	Kelas	
		Target	PSO
1	D_045	1	1
2	D_097	2	1
3	D_080	3	3
4	D_046	4	2
5	D_056	2	1

Selanjutnya menghitung ketidaksesuaian antara target dan kelas hasil perhitungan partikel PSO. Dari Tabel 4.10 dapat diketahui ketidaksesuaian antara kelas target yang sesungguhya dengan kelas hasil perhitungan menggunakan partikel PSO sebagai vektor bobot. Total dari ketidaksesuaian tersebut yang akan digunakan untuk mendapatkan nilai *cost*. Apabila kelas PSO tidak sesuai dengan

kelas target, maka nilai ketidaksesuaian bernilai 1 dan apabila kelas PSO sesuai maka nilai ketidaksesuaian bernilai 0. Total ketidaksesuaian partikel X1 dapat dilihat pada Tabel 4.11.

Tabel 4.11 Hitung ketidaksesuaian partikel

No	Data	Kelas		Ketidaksesuaian
		Target	PSO	
1	D_045	1	1	0
2	D_097	2	1	1
3	D_080	3	3	0
4	D_046	4	2	1
5	D_056	2	1	1
Total				3

Jumlah ketidaksesuaian antara hasil perhitungan PSO dan kelas target sesungguhnya adalah 3, maka nilai *cost* dari partikel X1 adalah 3.

4.3.1.3 Inisialisasi Partikel Awal

Inisialisasi awal partikel pada iterasi ke-0 dibangkitkan secara acak seperti pada subbab 4.3.1.2 dengan banyak partikel (ukuran *swarm*) adalah 3, maka posisi awal dapat dilihat pada Tabel 4.12.

Tabel 4.12 Inisialisasi partikel PSO

Partikel	Posisi																			
X_1^0	50	50	50	35	35	50	15	35	35	15	35	35	35	35	35	15	35	15	35	35
X_2^0	35	50	50	35	50	15	15	15	15	15	35	35	15	35	15	15	35	15	15	35
X_3^0	50	50	50	50	50	35	50	50	15	35	15	35	15	50	15	15	15	15	15	15

Keterangan:

X_i^j =partikel ke-*i* iterasi ke-*j*

4.3.1.4 Inisialisasi Kecepatan Awal

Kecepatan pada iterasi ke-0 semua bernilai 0. Kecepatan awal partikel dapat dilihat pada Tabel 4.13.

Tabel 4.13 Inisialisasi kecepatan awal PSO

Kecepatan																				
V_1^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_2^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_3^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Keterangan:

V_i^j = kecepatan partikel ke-*i* iterasi ke-*j*

4.3.1.5 Inisialisasi *pBest* Awal

Nilai *pBest* pada iterasi ke-0 adalah sama dengan nilai posisi partikel dan dihitung setiap *cost pBest*. Nilai *pBest* awal dapat dilihat pada Tabel 4.10.

Tabel 4.14 Inisialisasi *pBest* awal PSO

$pBest$	Dimensi $pBest$									
	1	2	3	4	5	6	7	8	9	10
$pBest_1^0$	50	50	50	35	35	50	15	35	35	15
$pBest_2^0$	35	50	50	35	50	15	15	15	15	15
$pBest_3^0$	50	50	50	50	50	35	50	50	15	35

$pBest$	Dimensi $pBest$										Cost
	11	12	13	14	15	16	17	18	19	20	
$pBest_1^0$	35	35	35	35	35	15	35	15	35	35	3
$pBest_2^0$	35	35	15	35	15	15	35	15	15	35	3
$pBest_3^0$	15	35	15	50	15	15	15	15	15	15	1

4.3.1.6 Inisialisasi *gBest* Awal

Nilai *gBest* diambil dari *pBest* dengan *cost* terkecil. Inisialisasi *gBest* awal dapat dilihat pada Tabel 4.15.

Tabel 4.15 Inisialisasi *gBest* awal PSO

$gBest$	Dimensi $gBest$									
	1	2	3	4	5	6	7	8	9	10
$gBest^0$	50	50	50	50	50	35	50	50	15	35

$gBest$	Dimensi $gBest$										*Cost*
	11	12	13	14	15	16	17	18	19	20	
$gBest^0$	15	35	15	50	15	15	15	15	15	15	4

4.3.2 *Update* Kecepatan Partikel PSO

Perhitungan *update* kecepatan pada iterasi 1 menggunakan Persamaan 2.8. Hasil *update* kecepatan pada iterasi 1 dapat dilihat pada Tabel 4.16.

Tabel 4.16 Kecepatan partikel PSO iterasi 1

Kecepatan	Dimensi Kecepatan									
	1	2	3	4	5	6	7	8	9	10
V_1^1	0	0	0	7.5	7.5	-7.5	17.5	7.5	-10	10
V_2^1	7.5	0	0	7.5	0	10	17.5	17.5	0	10
V_3^1	0	0	0	0	0	0	0	0	0	0

Kecepatan	Dimensi Kecepatan									
	11	12	13	14	15	16	17	18	19	20
V_1^1	-10	0	-10	7.5	-10	0	-10	0	-10	-10
V_2^1	-10	0	0	7.5	0	0	-10	0	0	-10
V_3^1	0	0	0	0	0	0	0	0	0	0

4.3.3 *Update* Posisi

Hasil perhitungan *update* posisi pada iterasi 1 menggunakan persamaa 2.7 dapat dilihat pada tabel 4.17.

Tabel 4.17 Posisi partikel PSO iterasi 1

Partikel	Dimensi Partikel									
	1	2	3	4	5	6	7	8	9	10
X_1^1	50	50	50	42.5	42.5	42.5	32.5	42.5	25	25
X_2^1	42.5	50	50	42.5	50	25	32.5	32.5	15	25
X_3^1	50	50	50	50	50	35	50	50	15	35

Partikel	Dimensi Partikel										*Cost*
	11	12	13	14	15	16	17	18	19	20	
X_1^1	25	35	25	42.5	25	15	25	15	25	25	3
X_2^1	25	35	15	42.5	15	15	25	15	15	25	1
X_3^1	15	35	15	50	15	15	15	15	15	15	0

4.3.4 *Update pBest*

Dengan membandingkan nilai *cost* antara partikel terbaru (iterasi 1) dengan *pBest* pada iterasi sebelumnya (iterasi 0), maka didapatkan hasil berupa *pBest* terbaru seperti pada Tabel 4.18.

Tabel 4.18 *pBest* PSO iterasi 1

$pBest$	Dimensi $pBest$									
	1	2	3	4	5	6	7	8	9	10
$pBest_1^1$	50	50	50	35	35	50	15	35	35	15
$pBest_2^1$	42.5	50	50	42.5	50	25	32.5	32.5	15	25
$pBest_3^1$	50	50	50	50	50	35	50	50	15	35

$pBest$	Dimensi $pBest$										*Cost*
	11	12	13	14	15	16	17	18	19	20	
$pBest_1^1$	15	35	15	50	15	15	15	15	15	15	3
$pBest_2^1$	25	35	15	42.5	15	15	25	15	15	25	1
$pBest_3^1$	15	35	15	50	15	15	15	15	15	15	0

4.3.5 *Update gBest*

Hasil *update gBest* pada iterasi 1 dapat dilihat pada Tabel 4.19.

Tabel 4.19 *gBest* iterasi 1

$gBest$	Dimensi $gBest$									
	1	2	3	4	5	6	7	8	9	10
$gBest^1$	50	50	50	50	50	35	50	50	15	35

$gBest$	Dimensi $gBest$										*Cost*
	11	12	13	14	15	16	17	18	19	20	
$gBest^1$	15	35	15	50	15	15	15	15	15	15	4

4.3.6 Partikel Terbaik

Partikel terbaik merupakan representasi solusi yang dicari. Partikel terbaik dipilih pada iterasi terakhir. Sesuai pada Tabel 4.17, maka partikel terbaik, memiliki *cost* terendah, maka partikel terbaik adalah partikel ke-3. Partikel terbaik dapat dilihat pada Tabel 4.20.

Tabel 4.20 Partikel terbaik PSO

Partikel	Dimensi Partikel										
	1	2	3	4	5	6	7	8	9	10	
X_3^1	50	50	50	50	50	35	50	50	15	35	
Partikel	Dimensi Partikel										*Cost*
	11	12	13	14	15	16	17	18	19	20	
X_3^1	15	35	15	50	15	15	15	15	15	15	0

4.3.7 Vektor Bobot Awal LVQ -PSO

Partikel terbaik dari proses PSO akan digunakan sebagai vektor bobot pelatihan LVQ. Partikel akan dipecah untuk masing – masing bobot kelas, vektor bobot awal LVQ-PSO dapat dilihat pada Tabel 4.21.

Tabel 4.21 Vektor bobot awal LVQ-PSO

Bobot	Vektor bobot				
W1	50	50	50	50	50
W2	35	50	50	15	35
W3	15	35	15	50	15
W4	15	15	15	15	15

4.3.8 Manualisasi Pelatihan Vektor Bobot LVQ-PSO

Data latih yang digunakan pada proses ini adalah data latih PSO seperti pada Tabel 4.9 dan vektor bobot yang digunkan adalah seperti pada Tabel 4.21. Untuk proses selanjutnya, lakukan perhitunan yang sama pada subbab 4.2. Hasil vektor bobot pada pelatihan LVQ-PSO adalah seperti pada Tabel 4.22.

Tabel 4.22 Vektor bobot akir pelatihan LVQ-PSO

Bobot	Vektor bobot				
W1	50	50	50	50	50
W2	35	51.5	51.5	18.8	35
W3	15	35	15	51.5	15
W4	15	15	15	15	15

4.3.9 Manualisasi Pengujian LVQ-PSO

Pada pengujian ini, data uji yang digunakan adalah sama seperti pada Tabel 4.6. Dengan melakukan proses perhitungan yang sama (sub-subbab 4.2.5) antara data uji dan vektor bobot pada Tabel 4.22, maka hasil kesesuaian terdapat pada Tabel 4.23.

Tabel 4.23 Kesesuaian data uji LVQ-PSO

Data uji	Kelas LVQ	Kelas target
D_071	1	1
D_041	4	2
D_087	3	3
D_100	4	4
D_012	1	1

4.3.10 Manualisasi Hitung Akurasi LVQ-PSO

Hitung akurasi menggunakan Persamaan 2.12, maka proses perhitungannya:

Akurasi $= \frac{4}{5}$ x 100% = 80%

Dari hasil tersebut diketahui bahwa akurasi pada pengujian algoritme LVQ-PSO adalah sebesar 80%.

4.4 Perancangan Pengujian

Pengujian dibagi menjadi 3, yaitu pengujian PSO, pengujian LVQ dan pengujian LVQ-PSO.

4.4.1 Pengujian PSO

Pengujian PSO dilakukan untuk mengetahui nilai variabel yang akan memberikan hasil *fitness* (*cost* terkecil) partikel terbaik.

4.4.1.1 Pengujian Bobot Inersia

Nilai bobot inersia terbesar (*Wmax*) dan bobot inersia terkecil (*Wmin*) akan mempengaruhi kecepatan eksplorasi partikel untuk menemukan solusi terbaik. Parameter yang akan digunakan adalah pasangan *Wmax* dan *Wmin* antara 0,4 sampai 0,9. Perancangan pengujian bobot inersia dapat dilihat pada Tabel 4.24.

Tabel 4.24 Perancangan pengujian bobot inersia

No	Bobot inersia *Wmin*;*Wmax*	Percobaan ke-					Rata-rata *fitness*
		1	2	3	4	5	
1	0,4; 0,5						
2	0,4; 0,6						
3	0,4; 0,7						
4	0,4; 0,8						
5	0,4; 0,9						
6	0,5; 0,6						
7	0,5; 0,7						
8	0,5; 0,8						
9	0,5; 0,9						
10	0,6; 0,7						

11	0,6; 0,8						
12	0,6; 0,9						
13	0,7; 0,8						
14	0,7; 0,9						
15	0,8; 0,9						

4.4.1.2 Pengujian Ukuran *Swarm*

Pengujian ukuran *swarm* dilakukan untuk mengetahui ukuran *swarm* (banyak partikel) yang menghasilkan solusi terbaik. Setiap pengujian dilakukan sebanyak 5 kali dengan ukuran *swarm* yang akan digunakan adalah 5, 10, 25, 50 dan 100. Perancangan pengujian ukuran swarm dapat dilihat pada Tabel 4.25.

Tabel 4.25 Perancangan pengujian ukuran *swarm*

No	Banyak partikel	Percobaan ke-					Rata – rata *fitness*	Rata-rata waktu komputasi (detik)
		1	2	3	4	5		
1	5							
2	10							
3	25							
4	50							
5	100							

4.4.1.3 Pengujian Maksimal Iterasi PSO

Pengujian maksimal iterasi dilakukan untuk mengetahui iterasi maksimal yang menghasilkan solusi terbaik. Setiap pengujian dilakukan sebanyak 5 kali denganbanyak iterasi yang akan digunakan adalah 5, 10, 25, 50 dan 100. Perancangan pengujian maksimal iterasi dapat dilihat pada Tabel 4.26.

Tabel 4.26 Perancangan pengujian maksimal iterasi

No	Maksimal Iterasi	Percobaan ke-					Rata – rata *Fitness*	Rata-rata waktu komputasi (detik)
		1	2	3	4	5		
1	5							
2	10							
3	25							
4	50							
5	100							

4.4.2 Pengujian LVQ

Pengujian LVQ Pengujian untuk mengetahui nilai variabel yang akan memberikan hasil akurasi terbaik. Pengujian LVQ dilakukan dengan menggunakan partikel terbaik PSO sebagai vektor bobot awal pelatihan LVQ.

4.4.2.1 Pengujian *Learning rate* LVQ

Nilai *learning rate* atau α akan mempengaruhi perubahan vektor bobot, maka dari itu pengujian α dilakukan untuk mengetahui nilai α yang akan menghasilkan akurasi yang paling baik. Nilai α yang akan digunakan adalah 0,1, 0,2, 0,3, 0,4, 0.5, 0,6, 0,7, 0,8 dan 0,9. Perancangan pengujian *learning rate* pelatihan LVQ dapat dilihat pada Tabel 4.27.

Tabel 4.27 Perancangan pengujian *learning rate*

No	*Learning rate* (α)	Percobaan ke-					Rata – rata akurasi (%)
		1	2	3	4	5	
1	0,1						
2	0,2						
3	0,3						
4	0,4						
5	0.5						
6	0,6						
7	0,7						
8	0,8						
9	0,9						

4.4.2.2 Pengujian Pengurang *Learning rate*

Nilai pengurang *learning rate* atau *dec* α akan mempengaruhi perubahan nilai α , maka dari itu pengujian nilai *dec α* dilakukan untuk mengetahui nilai *dec* α yang akan menghasilkan akurasi yang paling baik. Nilai *dec α* yang akan digunakan adalah 0,1, 0,2, 0,3, 0,4, 0.5, 0,6, 0,7, 0,8 dan 0,9. Perancangan pengujian *dec α* dapat dilihat pada Tabel 4.28.

Tabel 4.28 Perancangan pengujian pengurang *learning rate*

No	Pengurang *learning rate* (dec α)	Percobaan ke					Rata – rata akurasi (%)
		1	2	3	4	5	
1	0,1						
2	0,2						
3	0,3						
4	0,4						
5	0.5						
6	0,6						
7	0,7						
8	0,8						
9	0,9						

4.4.3 Pengujian LVQ-PSO

Pengujian LVQ-PSO dilakukan untuk mengetahui tingkat akurasi algortima LVQ yang telah dioptimasi dengan PSO.

4.4.3.1 Pengujian Menggunakan Parameter Terbaik

Pengujian ini menggunakan nilai variabel terbaik yang telah didapatkan dari pengujian-pengujian sebelumnya. Pengujian ini menggunakan parameter jumlah data uji LVQ yang berbeda, yaitu 10, 20, 30, 40 dan 50. Perancangan pengujian LVQ-PSO dengan parameter terbaik dapat dilihat pada Tabel 4.29.

Tabel 4.29 Perancangen pengujian LVQ-PSO parameter terbaik

No	Banyak data latih; Banyak data uji	Percobaan ke-					Rata – rata akurasi (%)	Rata-rata Waktu komputasi (detik)
		1	2	3	4	5		
1	90;10							
2	80;20							
3	70;30							
4	60;40							
5	50;50							

4.4.3.2 Pegujian Perbandingan LVQ – PSO dan LVQ

Pengujian ini dilakukan pada kedua algoritme (LVQ-PSO dan LVQ) untuk mengetahui apakah algoritme yang dioptimasi menghasilkan akurasi yang lebih baik. Parameter yang akan diujikan adalah banyak data uji yang digunakan, yaitu 10, 20, 30, 40 dan 50. Setiap algoritme akan diuji sebanyak 5 kali pada setiap parameter. Perancangan pengujian perbandingan antara LVQ-PSO dapat dilihat pada Tabel 4.30 dan perancangan pengujian LVQ dapat dilihat pada Tabel 4.31.

Tabel 4.30 Perancangan pengujian akurasi LVQ-PSO

No	Banyak data latih; Banyak data uji	Percobaan ke-					Rata – rata akurasi (%)	Rata-rata Waktu komputasi (detik)
		1	2	3	4	5		
1	90;10							
2	80;20							
3	70;30							
4	60;40							
5	50;50							

Tabel 4.31 Perancangan pengujian akurasi LVQ

No	Banyak data latih; Banyak data uji	Percobaan ke-					Rata – rata akurasi (%)	Rata-rata Waktu komputasi (detik)
		1	2	3	4	5		
1	90;10							
2	80;20							
3	70;30							
4	60;40							
5	50;50							

BAB 5 IMPLEMENTASI

Pada bab ini akan berisi implementasi algoritme. Bab ini berisi 3 subbab sesuai dengan kelas implementasi, yaitu kelas PSO, kelas Pelatihan LVQ dan kelas Pengujian LVQ.

5.1 Implementasi Algoritme PSO

Algoritme PSO digunakan untuk mencari vektor bobot terbaik yang menghasilkan akurasi tertinggi. Variabel yang digunakan dalam implementasi algoritme PSO adalah:

a. *PanjangPartikel* = 180.

b. BanyakPartikel ditentukan sesuai *input*an ketika program dijalankan.

c. BanyakKelas =4, sesuai dengan banyaknya kelas klasifikasi.

5.1.1 Implementasi Inisialisasi Partikel

Inisialisasi partikel merupakan proses untuk menentukan nilai posisi awal partikel. Jumlah partikel yang digunakan sesuai dengan jumlah yang ditentukan (ketika program dijalankan). Setiap partikel merepresentasikan vektor dari 4 kelas klasifikasi LVQ. Kode implementasi dapat dilihat pada Gambar 5.1.

```
1  public void inisialisasi_Partikel() {
2   File File_Input = new File("F:\\ADHD.xls");
3   try {
4    Workbook workboook = Workbook.getWorkbook(File_Input);
5    Sheet sheet = workboook.getSheet(0);
6    partikel = new double[banyakPartikel][panjangPartikel];
7    for (int i = 0; i < banyakPartikel; i++) {
8     for (int j = 0; j < panjangPartikel; j++) {
9      for (int k = 1; k <= banyakKelas; k++) {
10      int baris = random.nextInt(102 - 3) + 3;
11      if (Integer.parseInt(sheet.getCell(47,
            baris).getContents()) == k) {
12       for (int l= 0; l + 2 < sheet.getColumns() - 1; l++){
          partikel[i][j]=Integer.parseInt(sheet.getCell(l +2,
                        baris).getContents());
13        j++;
14       }
15      } else {
16        k--;}}}}
17  } catch (IOException | IndexOutOfBoundsException
      BiffException e){}}
```

Gambar 5.1 Implementasi inisialisasi partikel

Keterangan:

1. Baris 2, deklarasi file data *input* (database) yang digunakan adalah "ADHD.xls".
2. Baris 6, inisialisasi ukuran array partikel yang akan digunakan.
3. Baris 7, perulangan sebanyak partikel yang akan digunakan.
4. Baris 8, perulangan sebanyak panjang partikel (180 dimensi).
5. Baris 9, perulangan sebanyak kelas klasifikasi. Batas nilai k adalah 1 sampai 4, sesuai kelas klasifikasi.
6. Baris 10, menetukan baris dari file *input* secara acak (terdapat 100 data). Pada file “ADHD.xls”, baris data dimulai dari baris ke 3 sampai 102.
7. Baris 11, melakukan pengecekan, apabila kelas target pada data *input* sesuai dengan nilai k saat ini.
8. Baris 12-13, bila syarat baris 11 terpenuhi, maka masukkan selruh vektor (45 dimensi) dari data *input* ke dalam array partikel.
9. Baris 15-16, bila syarat pada baris 11 tidak terpenuhi, lakukan penguragan nilai pada variabel k dan lakukan perulangan untuk memilih baris secara acak.
10. Baris 9-16, agar setiap partikel dipastikan merepresentasikan vektor setiap kelas klasifikasi.

5.1.2 Implementasi Inisialisasi Kecepatan

Kecepatan awal partikel adalah 0. Kode implementasi inisialisasi kecepatan dapat dilihat pada Gambar 5.2.

```
1  public void inisialisasi_Kecepatan() {
2   kecepatan = new double[banyakPartikel][panjangPartikel];
3   for (int i = 0; i < banyakPartikel; i++) {
4    for (int j = 0; j < panjangPartikel; j++) {
5     kecepatan[i][j] = 0;}}}
```

Gambar 5.2 Implementasi inisialisasi kecepatan

Keterangan:

1. Baris 2, inisialisasi ukuran array kecepatan yang akan digunakan.
2. Baris 3-5, menginisialisasi setiap dimensi array dengan nilai 0.

5.1.3 Implementasi Inisialisasi *pBest*

Inisialisasi *pBest* berfugsi untuk menentukan nilai awal *pBest*. Nilai *pBest* awal adalah sama dengan partikel awal. Kode implementasi inisialisasi *pBest* dapat dilihat pada Gambar 5.3.

```
1  public void inisialisasi_pBest() {
2   pBest = new double[banyakPartikel][panjangPartikel];
3   for (int i = 0; i < banyakPartikel; i++) {
4    for (int j = 0; j < panjangPartikel; j++) {
5     pBest[i][j] = partikel[i][j];}}}
```

Gambar 5.3 Implementasi inisialisasi *pBest*

Keterangan:

1. Baris 2, menginisialisasi ukuran array *pBest*.
2. Baris 3-4, inisialisasi setiap dimensi *pBest* dengan nilai yang sama dengan partikel.

5.1.4 Implementasi Hitung *Cost* Partikel

Hitung *cost* partikel berfungsi untuk menghitung *cost* dan *fitness* dari setiap partikel. Variabel yang digunakan pada proses ini adalah:

a. Array *euclidean*, digunakan untuk menyimpan jarak *euclidean* data latih PSO/ *input* terhadap partikel. Array *euclidean* berdimensi sebanyak data latih PSO x 6. Dimensi 0 digunakan untuk menyimpan jarak antara data latih PSO dengan partikel dimesi 0-44. Dimensi 1 digunakan untuk menyimpan jarak antara data latih PSO dengan partikel dimensi 44-89. Dimensi 2 digunakan untuk menyimpan jarak antara data latih PSO dengan partikel dimesi 90-134. Dimensi 3 digunakan untuk menyimpan jarak antara data latih PSO dengan partikel dimesi 135-179. Dimensi 4 digunakan untuk menyimpan kelas klasifikasi data latih PSO (jarak *euclidean* terkecil dari dimensi 1 sampai 4). Dimensi 5 digunakan untuk menyimpan nilai *cost*/ ketidaksesuaian antara kelas data latih PSO hasil perhitungan (dimensi 5) dengan kelas data latih PSO seseungguhnya (kelas target). Isi array e*uclidean* akan ter-*reset* setiap perhitungan *cost* 1 partikel.

b. Array *fitness*, menyimpan *cost* seluruh partikel.

Kode implementasi hitung *cost* dapat dilihat pada Gambar 5.4.

```
1  public double[][] hitungCost(double partikel[][],
   double[][] fitness) {
2   int x, y, z;
3    for (int i = 0; i < banyakPartikel; i++) {
4     euclidean = new double[banyakDataLatihPSO][6];
5     double cost = 0;
6      for (int j = 0; j < banyakDataLatihPSO; j++) {
7       for (z = 0, x = 0; z < banyakKelas; z++) {
8        double sum = 0;
9        for (y = 0; y < 45; y++, x++) {
10        double b = Math.pow((partikel[i][x] -
                    dataLatihPSO[j][y]), 2);
11        sum += b;
```

```
12          }
13          euclidean[j][z] = Math.sqrt(sum);
14         }
15        euclidean[j][z] = Mencariminimum(euclidean[j]);
16        euclidean[j][z + 1] = kesesuaian(euclidean[j][z], j);
17        cost += euclidean[j][z + 1];
18        }
19       fitness[i][0] = i + 1;
20       fitness[i][1] = cost;
21       fitness[i][2] = ((banyakDataLatihPSO - cost) /
                        banyakDataLatihPSO);
22     }
23   return fitness;}
```

Gambar 5.4 Implementasi hitung *cost* partikel

Keterangan:

1. Baris 3, perulangan sebanyak partikel.
2. Baris 4, inisialisasi ukuran array *Euclidean*.
3. Baris 5, inisialisasi variabel *cost*. Variabel ini, digunakan untuk menyimpan total ketidasesuaian untuk 1 partikel.
4. Baris 6, perulangan untuk setiap data latih PSO.
5. Baris 7, perulangan sebanyak kelas klasifikasi.
6. Baris 8, inisialisasi variabel *sum*, variabel ini digunakan untuk menyimpan jumlah kuadrat jarak *euclidean* antara data latih PSO dengan partikel (dimensi tertentu).
7. Baris 9-12, menghitung jarak *euclidean* kuadrat data latih PSO terhadap partikel dimensi ke-j x ke-y.
8. Baris 13, menyimpan jarak *euclidean* ke dalam array (dimensi 1-4).
9. Baris 15, memanggil fungsi *Mencariminimum* untuk menetukan kelas klasifikasi data latih PSO kemudian menyimpan kelas ke dalam array *euclidean* (dimensi ke 5).
10. Baris 16, memanggil fungsi *kesesuaian* untuk menghitung *cost*.
11. Baris 17, mendapatkan total ketidaksesuaian untuk 1 partikel terhadap seluruh data latih PSO.
12. Baris 19-20, menyimpan total *cost* dan *fitness* untuk semua partikel.
13. Baris 23, mengembalikan array *fitness*.

5.1.5 Implementasi Menentukan Kelas

Menentukan kelas berfungsi untuk mengklasifikasikan sebuah data latih PSO/ *input* dengan cara mencari jarak terdekat data latih PSO terhadap suatu kelas. Kode implementasi menetukan kelas dapat dilihat pada Gambar 5.5.

```
1  public int Menentukan_kelas(double euclidean[]) {
2   int i, kelas = 0;
3   double min = 1000.0;
4   for (i = 0; i < 4; i++) {
5    if (min > euclidean[i]) {
6     min = euclidean[i];
7     kelas = i;
8    }
9   }
10 return kelas + 1;}
```

Gambar 5.5 Implementasi menentukan kelas

Keterangan:

1. Baris 2-3, inisialisasi variabel $kelas$ dan min.
2. Baris 4-9, mencari nilai terkecil pada setiap dimensi array. Nilai terkecil akan dijadikan nilai dari *min* dan indeksnya (parameter *i*) akan dijadikan *kelas*.
3. Baris 10, mengembalikan kelas klasifikasi. Kelas klasifikasi target memiliki range 1-4 sedangkan array *euclidean* berdimensi 0-3, maka nilai kembalian adalah *kelas+1.*

5.1.6 Implementasi Hitung Kesesuaian

Hitung kesesuaian digunakan untuk mengetahui apakah kelas suatu data latih PSO sama dengan kelas sesungguhnya (kelas target). Kode implementas hitung kesesuaian dpat dilihat pada Gambar 5.6.

```
1  public double Hitung_kesesuaian(double kelas, int indeks) {
2   if (dataLatihPSO[indeks][45] == kelas) {
3    return 0;
4   } else {
5  return 1;}}
```

Gambar 5.6 Implementasi hitung kesesuaian

Keterangan:

1. Baris 2-4, syarat apabila kelas data latih PSO sesungguhnya (kelas target) sama dengan nilai variabel *kelas* (hasil penghitungan jarak *euclidean* terkecil) sama, maka kembalikan nilai 0.
2. Baris 5, bila syarat tidak terpenuhi maka kembalikan nilai 1.

5.1.7 Impelemetasi *Update gBest*

Proses *update gBest* digunakan untuk mencari nilai *gBest* baru. Nilai *gBest* didapatkan dari *pBest* yang memiliki nilai *fitness* terbesar. *Update gBest* digunakan juga saat inisialisasi awal *gBest*. Kode implementasi *update gBest* dapat dilihat pada Gambar 5.7.

```
1   public void update_gBest() {
2    hitungCost(pBest, fitness1);
3    gBest = new double[panjangPartikel];
4    int indexgBest = 0;
5    int i, j;
6    double max = 0;
7    for (i = 0; i < banyakPartikel; i++) {
8     if (max < fitness1[i][2]) {
9      max = fitness1[i][2];
10     indexgBest = i;}}
11   for (j = 0; j < pBest[0].length; j++) {
12    gBest[j] = pBest[indexgBest][j];}
13  }
```

Gambar 5.7 Implementasi *update gBest*

Keterangan:

1. Baris 2, memanggil fungsi *hitungCost* untuk mendapatkan nilia *fitness* dan *cost* dari setiap *pBest*.
2. Baris 4-6, inisialisasi variabel *indexgBest* dan *max*.
3. Baris 7-10, lakukan pengecekan pada setiap *pBest*, apabila nilai *fitness pBest* ke-*i* lebih besar daripada nilai *max*, nilai *fitness* tersebut akan menjadi nilai *max* baru dan variabel *i* akan dipilih menjadi *indexgBest*.
4. Baris 11-13, perulangan untuk panjang partikel dan memasukkan nilai *pBest* index ke-*indexgBest* ke dalam *gBest* baru.

5.1.8 Implementasi *Update* Kecepatan

Proses *update* kecepatan merupakan perhitungan untuk mendapatkan kecepatan partikel terbaru. Kode implementasi *update* kecepatan dapat dilihat pada Gambar 5.8.

```
1   public void update_kecepatan() {
2    double vel;
3    for (int i = 0; i < banyakPartikel; i++) {
4     for (int j = 0; j < panjangPartikel; j++) {
       vel = (w * kecepatan[i][j]) + (c1 * r1 * (pBest[i][j] -
5            partikel[i][j])) + (c2 * r2 * (gBest[j] -
             partikel[i][j]));
6     if (vel > vmax) {
```

```
7        kecepatan[i][j] = vmax;
8       } else if (vel < vmin) {
9        kecepatan[i][j] = vmin;
10      } else {
11       kecepatan[i][j] = vel;}}}}
```

Gambar 5.8 Implementasi *update* kecepatan

Keterangan:

1. Baris 2, inisialisasi variabel *vel*, digunakan untuk menyimpan nilai perhitungan kecepatan sebelum dimasukkan ke dalam array *kecepatan*.
2. Baris 3, perulangan sebanyak partikel.
3. Baris 4, perulangan sebanyak panjang partikel.
4. Baris 5, hitung kecepatan baru menggunakan Persamaan 2.7.
5. Baris 6-11, melakukan *velocity clamping* pada variabel *vel* kemudian memasukkan nilai *vel* ke dalam array *kecepatan.*

5.1.9 Implementasi *Update pBest*

Proses *update pBest* digunakan untuk mencari nilai *pBest* terbaru. Nilai *pBest* baru didapatkan dengan membandingkan nilai *cost* antara partikel terbaru (iterasi saat ini) dan *pBest* lama (iterasi saat ini-1). Kode implementasi *update pBest* dapat dilihat pada Gambar 5.9.

```
1  public void update_pBest() {
2          hitungCost(partikel, fitness2);
3          for (int i = 0; i < banyakPartikel; i++) {
4              if (fitness2[i][1] < fitness1[i][1]) {
5                  pBest[i] = partikel[i];}
6          }
7      }
```

Gambar 5.9 Implementasi *update pBest*

Keterangan:

1. Baris 2, memanggil fungsi *hitungCost* untuk menghitung *fitness* partikel terbaru.
2. Baris 3, perulangan sebanyak partikel.
3. Baris 4, apabila nilai *cost* partikel terbaru ke-*i* lebih kecil dari *pBest* lama ke-*i*, maka nilai *pBest* baru ke-*i* sama dengan partikel ke-*i*.

5.1.10 Implementasi *Update* Posisi

Update posisi merupakan proses untuk mencari nilai partikel terbaru. Kode implmentasi *update* posisi dapat dilihat pada Gambar 5.10.

```
1  public void update_posisi() {
2   for (int i = 0; i < banyakPartikel; i++) {
3    for (int j = 0; j < panjangPartikel; j++) {
4     partikel[i][j] = kecepatan[i][j] + partikel[i][j];}}}
```

Gambar 5.10 Implementasi *update* posisi

Keterangan:

1. Baris 2-4, *update* posisi partikel dengan menjumlahkan kecepatan terbaru(iterasi saat ini) dan posisi partikel lama(iterasi-1).

5.2 Implementasi Pelatihan LVQ

Pelatihan LVQ berjalan setelah didapatkan partikel terbaik dari proses PSO.

5.2.1 Implementasi Vektor Bobot Awal

Proses vektor bobot awal bertujuan untuk merubah partikel terbailk PSO (panjang partikel PSO 1 x 180 dimensi) menjadi vektor bobot LVQ menjadi berukuran 4 x 45, ukuran tersebut berarti terdapat 4 kelas klasifikasi dengan 45 vektor gejala ADHD. Kode implementasi vektor bobot awal dapat dilihat pada Gambar 5.11.

```
1  Public void vektorBobotAwal() {
2   vektorBobot = new double[4][45];
3   for (int i = 0; i < kelas; i++) {
4    for (int k = 0, j = i * 45; j < (i * 45 + 45); j++, k++){
5      vektorBobot[i][k] = pso.individuTerbaik[j];}}}
```

Gambar 5.11 Implementasi vektor bobot awal

Keterangan:

1. Bari 2, inisialisasi ukuran array *vektorBobot.*
2. Baris 3, perulangan sebanyak kelas klasifikasi, yaitu 4.
3. Baris 4, perulangan sebanyak panjang vektor bobot. Variabel *k* adalah sesuai panjang vektor bobot pelatihan LVQ, yaitu 45 dimensi, sedangkan variabel *j* adalah dimensi partikel PSO, yaitu degan batas 180 diimensi. Pada baris ini, partikel PSO akan dipotong setiap 45 dimensi.
4. Baris 5, memasukkan nilai individu terbaik PSO ke dalam array *vektorBobot.*

5.2.2 Implementasi Hitung *Euclidean* Kelas

Hitung *euclidean* kelas merupakan proses untuk mengklasifikasikan suatu data latih PSO ke dalam salah satu kelas target. Kode implementasi hitung *euclidean* dapat dilihat pada Gambar 5.12.

```
1   public int hitung EuclideanKelas(double[] dataLatihke) {
2    int kelasHitung;
3    double sum, akar;
4    double[][]euc = new double[kelas];
5    for (int i = 0; i < kelas; i++) {
6     sum = 0;
7     for (int j = 0; j < panjangVektor; j++) {
8      sum += Math.pow((dataLatihke[j]-vektorBobot[i][j]), 2);}
9     akar = Math.sqrt(sum);
10    euc[i] = akar;}
11   kelasHitung = pso.Menentukan_kelas(euc);
12  return kelasHitung;}
```

Gambar 5.12 Implementasi hitung *euclidean* kelas

Keterangan:

1. Baris 2-3, inisialisasi variabel.
2. Baris 4, inisialisasi array *euc*.
3. Baris 5-6, perulangan sebanyak kelas (4) dan inisialisasi variabel *sum*, variabel *sum* akan kembali ke 0.
4. Baris 7-8, perulangan sebanyak panjang vektor(45). Variabel *sum* menyimpan jarak *euclidean* kuadrat antara vektor bobot dan data latih.
5. Baris 9-10, variabel *akar* meyimpan jarak *euclidean* dan disimpan dalam array *euc*.
6. Baris 11, mencari kelas data latih dengan memanggil fungsi dari kelas PSO, *Menentukan_kelas.*
7. Mengembalikan kelas klasifikasi data latih.

5.2.3 Implementasi *Update* Bobot

Update bobot merupakan proses memperbarui vektor bobot pelatihan LVQ. Kode implementasi *update* bobot dapat dilihat pada Gambar 5.13.

```
1   public void UpdateBobot(int kelas,int index,double alpha){
2    if (kelas == dataLatih[index][45]) {
3     for (int i = 0; i < 45; i++) {
       vektorBobot[kelas - 1][i] = vektorBobot[kelas - 1][i] +
4                                (alpha * (dataLatih[index][i]
                                 - vektorBobot[kelas - 1][i]));
5     }
6    } else {
7     for (int i = 0; i < 45; i++) {
       vektorBobot[kelas - 1][i] = vektorBobot[kelas - 1][i] -
8                                 (alpha * (dataLatih[index][i]
                                 - vektorBobot[kelas - 1][i]));
9   }}}
```

Gambar 5.13 Implementasi *update* bobot

Keterangan:

1. Baris 2-5, bila dari kelas pehitungan data latih PSO sesuai dengan kelas target, maka *update* vektor bobot menggunakan Persamaan 2.3.
2. Baris 2-5, bila dari kelas pehitungan data latih PSO tidak sama dengan kelas target, maka *update* vektor bobot menggunakan Persamaan 2.4.

5.2.4 Implementasi *Update Learning rate* (α)

Kode implementasi untuk menghitung nilai α terbaru dapat dilihat pada Gambar 5.14.

```
1  public float UpdateAlpha(double alpha, double pengaliA) {
2   double a = alpha * pengaliA;
3  return (float) a;
4  }
```

Gambar 5.14 Implementasi *update* α

Keterangan:

1. Baris 2, hitung nilai *α* baru menggunakan Persamaan 2.5.
2. Baris 3, kembalikan nilai *α* terbaru.

5.3 Implementasi Pengujian LVQ

Pengujian LVQ merupakan proses untuk mengklasifikasikan data latih PSO ke dalam satu kelas target dengan menggunakan bobot akhir dari proses pelatihan LVQ sebagai vektor acuan. Pengujian LVQ juga dapat digunakan untuk mendapatkan tingkat akurasi implementasi algoritme LVQ-PSO.

5.3.1 Implementasi Menghitung Akurasi

Mengitung akurasi merupakan proses untuk mengetahui akurasi implementasi algortima. Kode implementasi menghitung akurasi dapat dilihat pada Gambar 5.15.

```
1  public void akurasi() {
2   kesesuaian = new double[banyakDataUji][2];
3   int pinalty = 0;
4    for (int i = 0; i < banyakDataUji; i++) {
5     kesesuaian[i][0]=hitungEuclideanKelas(dataPengujian[i]);
6     System.out.print(kesesuaian[i][0]);
7     if (kesesuaian[i][0] == dataPengujian[i][45]) {
8      kesesuaian[i][1] = 0;
9      pinalty += kesesuaian[i][1];
10    }else{
11     kesesuaian[i][1] = 1;
12     pinalty += kesesuaian[i][1];
13   }}
```

```
14    double fitness = banyakDataUji - pinalty;
15    double akurasi = (fitness / banyakDataUji) * 100;
16    System.out.println("\nAKURASI: " + akurasi + "%");
17  }
```

Gambar 5.15 Implementasi menghitung akurasi

Keterangan:

1. Baris 2, inisialisasi array *kesesuaian*. Variabel ini digunakan untuk menyimpan kelas hasil klasifikasi data latih PSO menggunakan algoritme LVQ-PSO.
2. Baris 3, Inisialisasi variabel *penalty*, untuk menyimpan total ketidaksesuaian antara kelas target data latih PSO dan kelas klasifikasi hasi perhitungan LVQ-PSO.
3. Baris 4, lakukan perulangan sebanyak data latih PSO.
4. Baris 5, Dapatkan kelas klasifikasi data latih PSO ke-*i* dan masukkan hasilnya ke dalam array *kesesuaian* indeks ke[*i*][0].
5. Baris 7-13, lakukan pengecekan, apabila kelas target sesuai dengan kelas perhitungan maka nilai *penalty* adalah +0, bila tidak sesuai maka nilai *penalty* +1.
6. Baris 14-16, hitung akurasi dengan Persamaan 2.12.

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab ini akan berisi hasil pengujian dan hasil analisis sesuai dengan rancangan pengujian. Pengujian dibagi menjadi tiga subbab, yaitu pengujian PSO, pengujian LVQ, pengujian LVQ-PSO.

6.1 Pengujian PSO

Pengujian PSO dilakukan untuk mengetahui nilai parameter yang akan memberikan hasil *fitness* partikel terbaik.

6.1.1 Pengujian Bobot Inersia

Nilai bobot inersia terbesar (*Wmax*) dan bobot inersia terkecil (*Wmin*) akan mempengaruhi kecepatan eksplorasi partikel untuk menemukan solusi terbaik. Parameter yang akan digunakan adalah pasangan *Wmax* dan *Wmin* antara 0,4 sampai 0,9. Pengujian bobot inersia dilakukan dengan nilai variabel lain sebagai berikut:

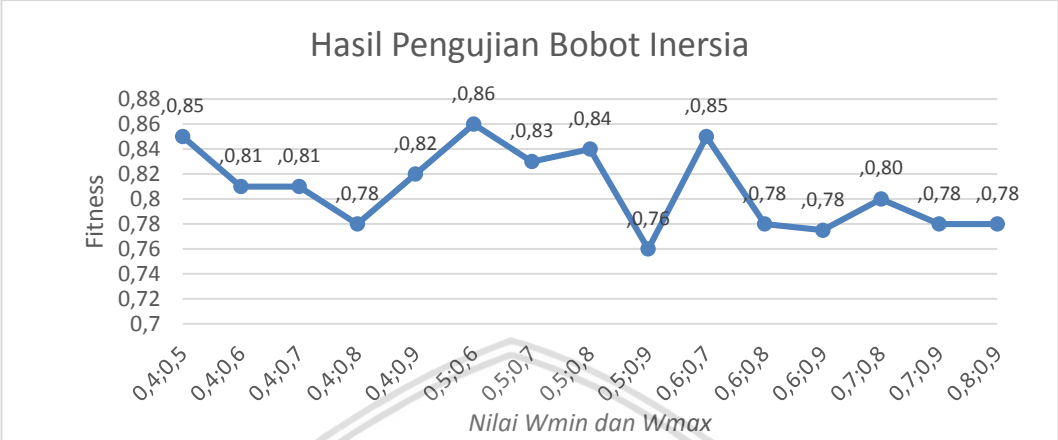
a. Ukuran *swarm* = 10
b. Data latih PSO = 20
c. Maksimal iterasi PSO = 10
d. r_1 = 0,5; r_2 = 0,5
e. c_1 = 1,5; c_2 = 1,5

Hasil pengujian bobot inersia dapat dilihat pada Tabel 6.1.

Tabel 6.1 Hasil pengujian bobot inersia

No	Bobot inersia *Wmin*;*Wmax*	Percobaan ke-					Rata-rata *fitness*
		1	2	3	4	5	
1	0,4;0,5	0,85	0,9	0,95	0,95	0,6	0,85
2	0,4;0,6	0,85	0,9	0,75	0,8	0,75	0,81
3	0,4;0,7	0,9	0,8	0,75	0,8	0,8	0,81
4	0,4;0,8	0,7	0,9	0,8	0,75	0,75	0,78
5	0,4;0,9	0,85	0,75	0,85	0,95	0,7	0,82
6	0,5;0,6	0,9	0,85	0,7	0,95	0,9	0,86
7	0,5;0,7	0,85	0,8	0,9	0,85	0,75	0,83
8	0,5;0,8	0,9	0,9	0,8	0,8	0,8	0,84
9	0,5;0;9	0,6	0,8	0,9	0,75	0,75	0,76
10	0,6;0,7	0,85	0,9	0,85	0,95	0,7	0,85
11	0,6;0,8	0,65	0,7	0,8	0,9	0,85	0,78
12	0,6;0,9	0,8	0,8	0,9	0,8	0,7	0,775
13	0,7;0,8	0,85	0,75	0,9	0,7	0,8	0,8
14	0,7;0,9	0,8	0,95	0,7	0,75	0,7	0,78
15	0,8;0,9	0,85	0,85	0,75	0,75	0,7	0,78

Untuk memudahkan membaca hasil pengujian pasangan bobot inersia, maka dibuat grafik seperti pada Gambar 6.1.



Gambar 6.1 Hasil pengujian bobot inersia

Berdasarkan hasil pengujian pada Gambar 6.1, diketahui bahwa rata-rata *fitness* terbaik adalah 0,86, yang didapatkan ketika menggunakan *Wmax* = 0,6 dan *Wmin* = 0,5. Hasil ini merepresentasikan bahwa semakin besar jarak antara *Wmin* dan *Wmax* menghasilkan *fitness* partikel yang semakin menurun karena partikel berpindah terlalu cepat sehingga semakin besar kemungkinan melewati posisi optimal, sedangkan nilai *Wmax* terlalu kecil menyebabkan partikel berpindah semakin lama sehingga lebih lama menuju posisi optimal.

6.1.2 Pengujian Ukuran *Swarm*

Pengujian ukuran *swarm* dilakukan untuk mengetahui ukuran *swarm* (banyak partikel) yang menghasilkan solusi terbaik. Setiap pengujian dilakukan sebanyak 5 kali dengan ukuran *swarm* yang akan digunakan adalah 5, 10, 25, 50 dan 100. Parameter pengujian lain adalah:

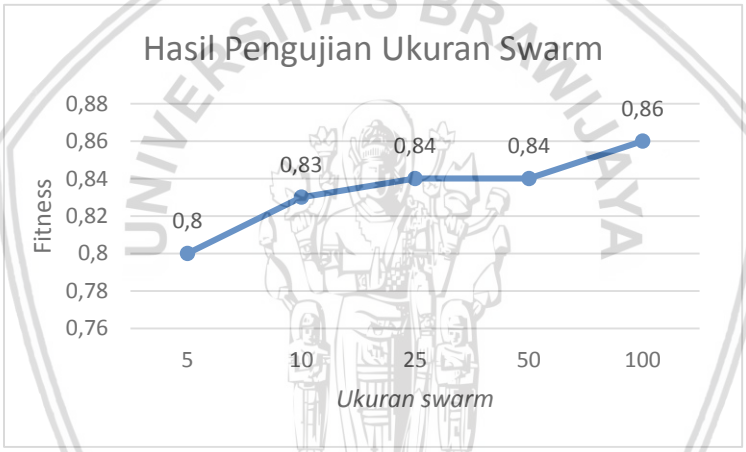
a. *Wmax* = 0,6; Wmin =0,5
b. Data latih PSO = 20
c. Maksimal iterasi PSO = 10
d. r_1 = 0,5; r_2 = 0,5
e. c_1 = 1,5; c_2 = 1,5

Hasil pengujian ukuran *swarm* dapat dilihat pada Tabel 6.2.

Tabel 6.2 Hasil pengujian ukuran *swarm*

No	Banyak partikel	Percobaan ke-					Rata – rata *fitness*	Rata-rata Waktu komputasi (detik)
		1	2	3	4	5		
1	5	0,8	0,9	0,8	0,7	0,8	0,8	3,2
2	10	0,9	0,9	0,8	0,8	0,75	0,83	3,5
3	25	0,8	0,8	0,8	0,9	0,9	0,84	4
4	50	0,85	0,7	0,9	0,95	0,8	0,84	5,3
5	100	0,95	0,9	0,8	0,95	0,7	0,86	6,2

Untuk memudahkan membaca hasil pengujian ukuran *swarm*, maka dibuatlah grafik seperti pada Gambar 6.2.



Gambar 6.2 Hasil pengujian ukuran *swarm*

Berdasarkan hasil pada Tabel 6.2, diketahui bahwa ukuran *swarm* yang menghasilkan *fitness* tertinggi adalah 100 partikel dengan nilai rata-rata *fitness* terbaik sebesar 0,86. Sedangkan ukuran *swarm* dengan nilai rata-rata *fitness* terkecil adalah 5 partikel dengan nilai *fitness* 0,8. Hal ini menunjukkan bahwa semakin besar ukuran *swarm*, maka semakin besar pula ruang pencarian solusi (partikel terbaik). Semakin besar ukuran *swarm*, maka semakin lama waktu komputasi.

6.1.3 Pengujian Maksimal Iterasi PSO

Pengujian maksimal iterasi dilakukan untuk mengetahui jumlah iterasi yang menghasilkan solusi terbaik. Setiap pengujian dilakukan sebanyak 5 kali denganbanyak iterasi yang akan digunakan adalah 5, 10, 25, 50 dan 100. Berdasarkan pengujian ukuran *swarm*, maka pada pengujian maksimal iterasi kali ini variabel yang akan digunakan adalah:

a. *Wmax*=0,6; *Wmin*=0,5

b. Data latih PSO =20

c. Ukuran *swarm*= 100

d. r_1 = 0,5; r_2 = 0,5

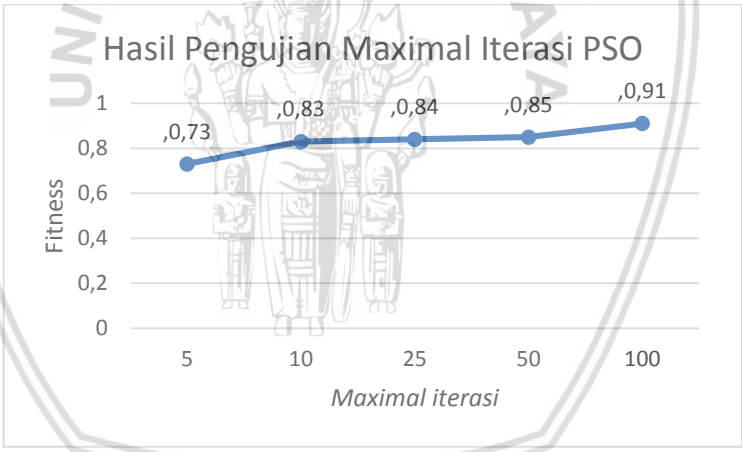
e. c_1 = 1,5; c_2 = 1,5

Hasil pengujian maksimal iterasi dapat dilihat pada Tabel 6.3.

Tabel 6.3 Hasil pengujian maksimal iterasi

No	Maksimal Iterasi	Percobaan ke-					Rata – rata *Fitness*	Rata-rata Waktu komputasi (detik)
		1	2	3	4	5		
1	5	0,8	0,5	0,7	0,8	0,85	0,73	2,8
2	10	0,9	0,7	0,8	0,95	0,8	0,83	5
3	25	0,85	0,95	0,7	0,8	0,9	0,84	15,7
4	50	0,95	0,85	0,9	0,7	0,85	0,85	28,4
5	100	1	0,95	0,8	0,95	0,9	0,91	62,5

Untuk memudahkan membaca hasil pengujian maksimal iterasi, maka dibuatlah grafik seperti pada Gambar 6.3.



Gambar 6.3 Hasil pengujian maksimal iterasi

Dari hasil pengujian maksimal iterasi, rata-rata *fitness* terbaik terdapat pada maksimal itrasi 100 dengan nilai *fitness* 0,91 dan rata-rata *fitness* terendah terdapat pada maksimal iterasi 5 dengan nilai 0,73. Hal ini menunjukkan bahwa semakin tinggi jumlah maksimal iterasi, maka semakin baik nilai *fitness* karena semakin banyak dilakukan perbaikan pada posisi partikel mendekati nilai optimal. Namun berdasarkan Tabel 6.3, diketahui bahwa semakin tinggi maksimal iterasi, maka semakin lama waktu komputasi yang dibutuhkan.

6.2 Pengujian LVQ

Pengujian LVQ Pengujian untuk mengetahui nilai variabel yang akan memberikan hasil akurasi terbaik. Pengujian LVQ dilakukan dengan menggunakan partikel terbaik PSO sebagai vektor bobot awal pelatihan LVQ.

6.2.1 Pengujian *Learning Rate* LVQ

Nilai *learning rate* atau *α* akan mempengaruhi perubahan vektor bobot, maka dari itu pengujian *α* dilakukan untuk mengetahui nilai *α* yang akan menghasilkan akurasi yang paling baik. Nilai *α* yang akan digunakan adalah 0,1 sampai 0,9 dengan interval 0,1. Variabel yang digunakan pada pengujian ini adalah:

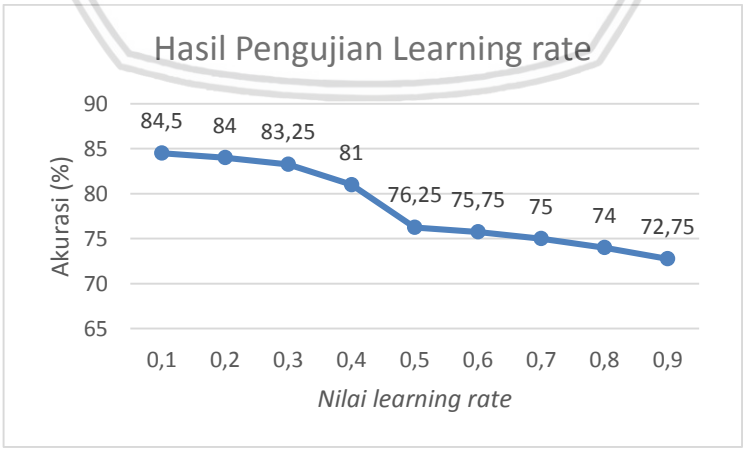
a. α minimum= 0.0000000000000001
b. Pengali α= 0,1
c. Iterasi maksimal= 100
d. Jumlah data uji = 80
e. Jumlah data latih = 20

Hasil pengujian *learning rate* dapat dilihat pada Tabel 6.4

Tabel 6.4 Hasil pengujian *learning rate*

No	*Learning rate* (α)	Percobaan ke-					Rata – rata akurasi (%)
		1	2	3	4	5	
1	0,1	82,5	82,5	92,5	75	90	84,5
2	0,2	88,75	82,5	86,25	85	77,5	84
3	0,3	70	77,5	85	86,25	97,5	83,25
4	0,4	81,25	81,25	87,5	78,75	76,25	81
5	0,5	75	81,25	63,75	90	71,25	76,25
6	0,6	80	82,5	76,25	65	75	75,75
7	0,7	81,25	75	61,25	71,25	86,25	75
8	0,8	78,75	75	71,25	87,5	57,5	74
9	0,9	71,25	77,5	72,5	82,5	60	72,75

Untuk memudahkan membaca hasil pengujian *learning rate* makan dibuat grafik hasil pengujian *learning rate* pada Gambar 6.4.



Gambar 6.4 Grafik hasil pengujian *learning rate*

Berdasarkan hasil pengujian *learning rate* pada Gambar 6.4, diketahui bahwa rata-rata akurasi terbaik terdapat pada nilai *learning rate* 0,1 dan nilai rata-rata

akurasi terendah terdapat pada nilai *learning rate* 0,9. Hasil ini menunjukkan bahwa semakin besar nilai *learning rate*, maka semakin kecil tingkat akurasi. Hal ini dikarenakan semakin besar nilai α semakin cepat perubahan nilai pada vektor bobot setiap kali dilakukan pembaruan vektor bobot pada proses pelatihan LVQ.

6.2.2 Pengujian Pengurang *Learning Rate*

Nilai pengurang *learning rate* atau *dec* α akan mempengaruhi perubahan nilai α , maka dari itu pengujian nilai *dec α* dilakukan untuk mengetahui nilai *dec* α yang akan menghasilkan akurasi yang paling baik. Nilai dec α yang akan digunakan adalah 0,1, 0,2, 0,3, 0,4, 0.5, 0,6, 0,7, 0,8 dan 0,9. Variabel yang digunakan pada pengujian ini adalah:

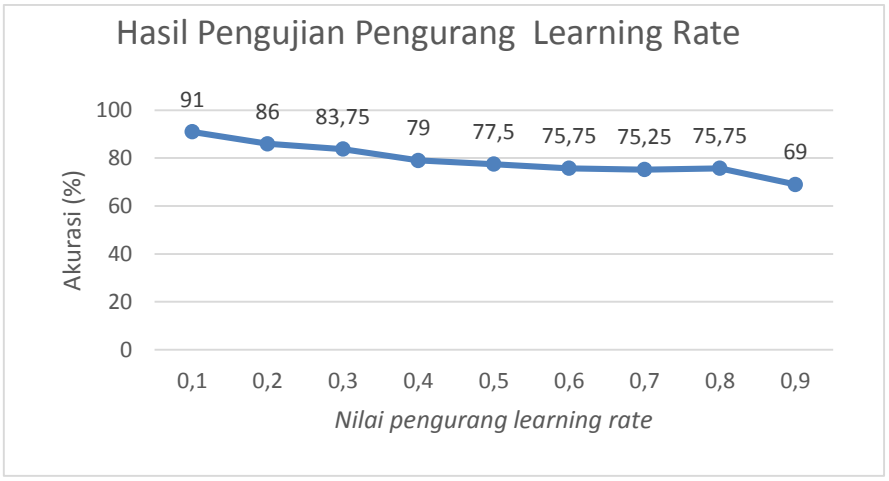
a. α minimum= 0.0000000000000001
b. *Learning rate*= 0,1
c. Iterasi maksimal= 100
d. Jumlah data uji =80
e. Jumlah data latih = 20

Hasil pengujian *learning rate* dapat dilihat pada Tabel 6.5.

Tabel 6.5 Hasil pengujian pengurang *learning rate*

No	Pengurang *learning rate* (dec α)	Percobaan ke					Rata – rata akurasi (%)
		1	2	3	4	5	
1	0,1	93,75	82,5	92,5	90	96,25	91
2	0,2	87,5	88,75	76,25	91,25	86,25	86
3	0,3	85	68,75	91,25	76,25	97,5	83,75
4	0,4	75	81,25	95	67,5	76,25	79
5	0,5	63,75	81,25	95	76,25	71,25	77,5
6	0,6	76,25	80	91,25	60	71,25	75,75
7	0,7	82,5	75	61,25	71,25	86,25	75,25
8	0,8	80	78,75	63,75	67,5	88,75	75,75
9	0,9	71,25	58,75	60	82,5	72,5	69

Untuk memudahkan membaca hasil pengujian nilai dec α maka dibuatlah grafik seperti Gambar 6.5.



Gambar 6.5 Hasil pengujian nilai pengurang *learning rate*

Berdasarkan hasil pengujian pada Gambar 6.5 dikethui bahwa hasil akurasi paling tinggi adalah 91% pada penggunaan nilai dec a=0,1, sedagkan nilai akurasi terendah adalah 69% ketika penggunaan nilai dec α=0,9. Hal ini menujukkan bahwa semakin besar nilai dec a, semakin kecil tingkat akurasi.

6.3 Pengujian LVQ-PSO

Pengujian LVQ-PSO dilakukan untuk mengetahui tingkat akurasi algortima LVQ yang telah dioptimasi dengan PSO.

6.3.1 Pengujian menggunakan parameter terbaik

Pengujian ini menggunakan nilai variabel yang diharapkan akan memberikan nilai akurasi paling optimal. Panegujian ini menggunakan parameter jumlah data uji LVQ yang berbeda, yaitu 10, 20, 30, 40 dan 50. Sebanyak 100 data yang tersedia, dibagi menjadi 2, yaitu data latih dan data uji. Nilai variabel yang digunakan sesuai hasil pengujian-pengujian sebelumnya, yaitu:

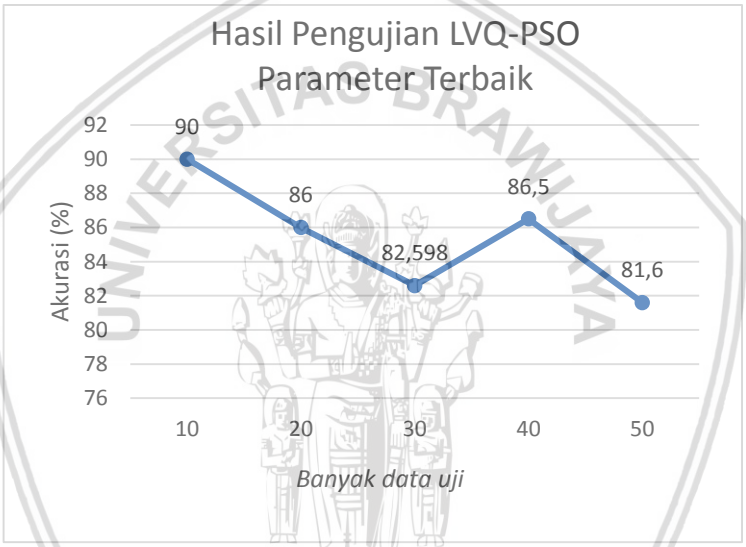
a. *Wmax*=0,6 ; *Wmin*=0,5
b. Ukuran *swarm*= 100
c. Maksimal iterasi PSO= 100
d. R1=0.5; r2= 0.5
e. C1= 1.5; c2=1.5
f. α minimum= 0.0000000000000001
g. *Learning rate*= 0,1
h. Pengurang α=0,1
i. Iterasi maksimal LVQ= 100

Hasil pengujian LVQ-PSO dengan parameter terbaik dapat dilihat pada Tabel 6.6.

Tabel 6.6 Hasil pengujian LVQ-PSO parameter terbaik

No	Banyak data latih; Banyak data uji	Percobaan ke-					Rata – rata akurasi (%)	Rata-rata Waktu komputasi (detik)
		1	2	3	4	5		
1	90;10	80	100	90	90	90	90	84,2
2	80;20	80	90	80	100	80	86	83,8
3	70;30	73,33	80	83,33	83,33	93	82,598	74,2
4	60;40	85	77,5	92,5	87,5	90	86,5	73
5	50;50	84	80	80	78	86	81,6	72,6

Untuk memudahkan membaca hasil pengujian LVQ-PSO menggunakan parameter terbaik, dibuatlah Gambar 6.6.



Gambar 6.6 Hasil pengujian LVQ-PSO parameter terbaik

Berdasarkan hasil pengujian LVQ-PSO pada Gambar 6.6 diketahui rata-rata tingkat akurasi dari setiap percobaan yang telah dilakukan. Tingkat akurasi terbaik adalah 90% dan tingkat akurasi terendah adalah 81,6%. Semakin banyak data uji yang digunakan (sedikit data latih), maka semakin cepat waktu komputasi.

6.3.2 Pegujian Perbandingan LVQ – PSO dan LVQ

Pengujian ini dilakukan pada kedua algoritme (LVQ-PSO dan LVQ) untuk mengetahui apakah algoritme yang dioptimasi menghasilkan akurasi yang lebih baik. Parameter yang akan diujikan adalah banyak data uji yang digunakan, yaitu 10, 20, 30, 40 dan 50. Setiap algoritme akan diuji sebanyak 5 kali pada setiap parameter, kemudian dihitung rata- rata waktu dan akurasi dan dibandingkan hasilnya. Pada setiap percobaan, algoritme LVQ-PSO dan LVQ menggunakan data uji yang sama, hasil acak dari totoal 100 data. Variabel yang digunakan pada variabel ini adalah:

a. *Wmax*=0,6 ; *Wmin*=0,5
b. Ukuran *swarm*= 100
c. Maksimal iterasi PSO= 100
d. R1=0.5; r2= 0.5
e. C1= 1.5; c2=1.5
f. α minimum= 0.0000000000000001
g. *Learning rate*= 0,1
h. Pengurang α=0,1
i. Iterasi maksimal LVQ= 100

Hasil pengujian algoritme LVQ-PSO dapat dilihat pada Tabel 6.7.

Tabel 6.7 Hasil pengujian akurasi LVQ-PSO

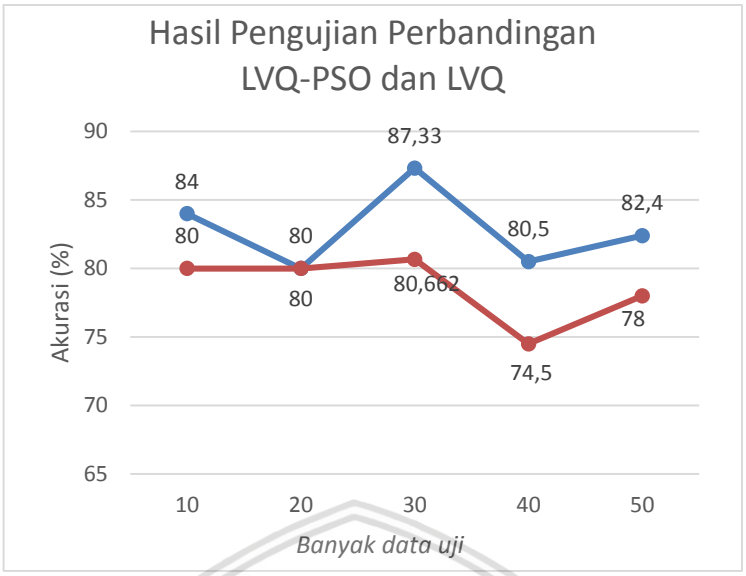
No	Banyak data latih; Banyak data uji	Percobaan ke-					Rata – rata akurasi (%)	Rata-rata Waktu komputasi (detik)
		1	2	3	4	5		
1	90;10	90	90	90	80	70	84	84,6
2	80;20	85	70	95	70	80	80	84,2
3	70;30	86,6	90	83,3	83,33	93,33	87,33	70,8
4	60;40	80	82,5	77,5	87,5	75	80,5	68,8
5	50;50	82	82	86	78	84	82,4	68,4

Hasil pengujian algoritme LVQ tanpa PSO dapat dilihat pada Tabel 6.8.

Tabel 6.8 Hasil pengujian akurasi LVQ tanpa PSO

No	Banyak data latih; Banyak data uji	Percobaan ke-					Rata – rata akurasi (%)	Rata-rata Waktu komputasi (detik)
		1	2	3	4	5		
1	90;10	70	80	90	80	80	80	6,4
2	80;20	80	90	80	80	70	80	5,4
3	70;30	83,3	86,6	80	76,6	76,6	80,6	4,8
4	60;40	77,5	67,5	80	75	72,5	74,5	3,6
5	50;50	88	80	80	64	78	78	3,2

Untuk memudahkan melihat perbedaan akurasi algoritme LVQ-PSO dan LVQ, maka dibuatlah grafik pada Gambar 6.7.



Gambar 6.7 Grafik hasil pengujian perbandingan akurasi

Berdasarkan hasil pengujian pada Gambar 6.7, diketahui bahwa akurasi algoritme LVQ-PSO paling tinggi adalah 87,3% dan akurasi terendah adalah 80%, sedangkan akurasi tertinggi LVQ adalah 80,6% dan akurasi terkecil adalah 74,5%. Secara rata-rata algoritme LVQ-PSO lebih baik dari algoritme LVQ, yaitu rata-rata akurasi algoritma LVQ adalah 82,84% dan algoritma LVQ adalah78,63%. Hal ini terjadi karena vektor yang digunakan pada pelatihan masing-masing algoritme berbeda. Pada algoritme LVQ-PSO vektor bobot awal sudah lebih optimal (lebih baik daripada vektor bobot pada algoritme LVQ). Meskipun begitu, bukan bararti algoritme LVQ akan selalu mendapatkan hasil akurasi yang lebih buruk dari LVQ-PSO, hal ini terjadi pada rata-rata pengujian dengan data uji sebanyak 20. Berdasarkan Tabel 6.7 dan Tabel 6.8 diketahui komputasi algoritme LVQ-PSO membutuhkan waktu yang lebih lama dari pada algoritme LVQ.

BAB 7 PENUTUP

Pada bab ini akan berisi kesimpulan berdasarkan hasil pengujian dan analisis yang telah dilakukan dan berisi saran untuk penelitian berikutnya.

7.1 Kesimpulan

Berdasarkan pengujian yang telah dilakukan, maka didapatkan kesimpulan yang akan mejawab masalah penelitian, yaitu:

1. Algoritme PSO digunakan untuk mengoptimasi vektor bobot awal proses pelatihan LVQ, sehingga urutan algoritme yang berjalan adalah PSO, pelatihan LVQ kemudian pengujian LVQ. Siklus algoritme PSO adalah: pertama tentukan ukuran *swarm*, inisialisasi setiap partikel yang merepresentasikan vektor bobot setiap kelas klasifikasi LVQ, inisialisasi kecepatan partikel, inisialisasi *pBest* dan inisialisasi *gBest.* Untuk setiap iterasi PSO, lakukan *update* kecepatan, *update* posisi, *update pBest* dan *update gBest*. Pada iterasi terakhir, pilih partikel dengan *fitness* terbesar sebagai partikel terbaik. Partikel terbaik PSO kemudian dijadikan vektor bobot awal yang akan digunakan pada proses pelatihan LVQ dan pengujian LVQ.

2. Pengujian dilakukan pada algoritme LVQ-PSO untuk mengetahui tingkat akurasi. Pengujian dilakukan dengan jumlah data uji yang berbeda-beda sebanyak lima kali. Parameter-parameter terbaik dari pengujian sebelumnya adalah *Wmax* 0,6, *Wmin* 0,5, ukuran *swarm* 100, maksimal iterasi PSO 100, α 0,1, dan *dec α 0,1*. Dengan mengunakan parameter-parameter tersebut, hasil akurasi tertinggi adalah 90% dan hasil akurasi terendah adalah 81,6%.

3. Tingkat akurasi LVQ-PSO dan LVQ didapatkan setelah melakukan pengujian pada setiap algoritme menggunakan data uji yang sama. Terdapat lima macam pengujian menggunakan jumlah data uji yang berbeda. Setiap pengujian diulang sebanyak lima kali. Hasil pengujian menunjukkan rata-rata akurasi tertinggi algortima LVQ-PSO adalah 87,3% dengan waktu komputasi 84,6 detik dan akurasi terendah adalah 80% dengan waktu komputasi 84,2 detik. Algoritma LVQ memiliki nilai rata-rata tertinggi adalah 80,6% dengan waktu komputasi 4,8 detik dan akurasi terkecil adalah 74,5% dengan waktu komputasi 3,6 detik. Dari hasil tersebut, maka dapat disimpulkan bahwa rata-rata akurasi algoritme LVQ-PSO lebih baik daripada LVQ untuk klasifikasi jenis ADHD pada anak usia dini meskipun algoritme LVQ-PSO membutuhkan waktu komputasi yang lebih lama.

7.2 Saran

Penelitian saat ini masih jauh dari sempurna dan masih dapat dikembangkan agar dapat mengurangi waktu komputasi dan meningkatkan tingkat akurasi. Saran yang diberikan untuk penelitian berikutnya adalah:

1. Menggunakan algoritme lain yang dapat mengoptimasi selain PSO, seperti algoritme genetika (GA) untuk mengurangi waktu komputasi dan meningkatkan tingkat akurasi.
2. Melakukan pengujian untuk mencari nilai semua variabel terbaik yang akan memberikan nilai akurasi yang tinggi.

DAFTAR PUSTAKA

American Psychiatric Association, 1994. *Diagnostic and Statistical Manual of Mental Disorders Fourth Edition*. Washington, DC.

Arifien, Z., Indriati, I., &Fauzi, M.A., 2016. P*endeteksi Jenis Attention Deficit Hyperactivity Disorder (ADHD) pada Anak Usia Dini Menggunakan Algoritme Learning Vector Quantization (LVQ)*. Repositori Jurnal Mahasiswa PTIIK UB, Doro Jurnal Volume 8 – Number 17. Tersedia di: < http://filkom.ub.ac.id/doro/archives/detail/DR00162201612> [Diakses 25 Februari 2017]

Asriningtias, S. R., Dachlan, H. S., & Yudaningtyas, E., 2015. *Optimasi Training Neural Network Menggunakan Hybrid Adaptive Mutation PSO-BP.* Jurnal EECCIS Vol. 9, No. 1, Juni 2015.

Budianita, E. & Prijodiprodjo, W., 2013. *Penerapan Learning Vector Quantization (LVQ) untuk Klasifikasi Status Gizi Anak.* IJCCS, Vol.7, No.2, July 2013, pp. 155~166 ISSN: 1978-1520

Cholissodin, I., 2016. *Dasar-Dasar Algoritme PSO*. Universitas Brawijaya.

Cholissodin, I., 2016. *Konsep Swarm Intelligence*. Universitas Brawijaya.

Fadila, P. N., Indriati, I. & Ratnawati, D. E., 2016. *Identifikasi Jenis Attention Deficit Hyperactivity Disorder (ADHD) pada Anak Usia Dini Menggunakan Algoritme Neighbor Weighted K-Nearest Neighbor (NWKNN)*. Repositori Jurnal Mahasiswa PTIIK UB, Doro Jurnal Volume 8 - Number 10. Tersedia di < http://filkom.ub.ac.id/doro/archives/detail/DR00094201612> [Diakses 25 Februari 2017]

Kompas, 2012. *Kecenderungan Bunuh Diri Penderita ADHD* [*online*]. Tersedia di: <http://health.kompas.com/read/2012/08/17/10123980/kecenderungan.bunuh.diri.penderita.adhd> [Diakses 19 Februari 2017]

Laxsmana, R. H., Indriati, I., Ratnawati D.E, 2016. *Pendeteksi Jenis Attention Deficit Hyperactivity Disorder (ADHD) pada Anak Usia Dini Menggunakan Algoritme Fuzzy Knearest Neighbor*. Repositori Jurnal Mahasiswa PTIIK UB, Doro Jurnal Volume 8 - Number 10. Tersedia di: < http://filkom.ub.ac.id/doro/archives/detail/DR00095201612> [Diakses pada 25 Februari 2017]

Mahmudy, W.F., 2015, Dasar-Dasar Algoritma Evolusi, Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya, Malang.

Novita, N. dan Siswati, S. 2010. *Pengaruh Social Stories Terhadap Keterampilan Sosial Anak Dengan Attention-Deficit Hyperactivity Disorder (ADHD) Studi Eksperimental Desain Kasus Tunggal Di Sekolah Alam Ar-Ridho Semarang*. Jurnal Psikologi Undip, 8(2), 102-116. Tersedia di: <http://www.ejournal.undip.ac.id/index.php/psikologi/article/viewFile/2955/2641> [Diakses 26 Februari 2017]

Nurmalahudin, 2013. *Perancangan Algoritme Belajar Jaringan Saraf Tiruan Menggunakan Particle Swarm Optimization (PSO)*. JurnalPOROS TEKNIK, Volume 5, No.1,Juni 2013: 18 - 23. Tersedia di <http://ejurnal.poliban.ac.id/index.php/porosteknik/article/view/16/8> [Diakses 6 Maret 2017]

Nurmawati, M., Indriati, I., Ratnawati, D. E., 2016. *Pendeteksi Jenis Attention Deficit Hyperactivity Disorder (ADHD) pada Anak Usia Dini Menggunakan Algoritme Linear Discriminant Analysis*. Repositori Jurnal Mahasiswa PTIIK UB, Doro Jurnal Volume 8 - Number 15. Tersedia di: < http://filkom.ub.ac.id/doro/ archives/detail/DR00142201612> [Diakses 25 Februari 2017]

Ramanda, Kresna, 2015. *Penerapan Particle Swarm Optimization sebagai Seleksi Fitur Prediksi Kelahiran Prematur pada Algoritme Neural Network*. Jurnal Teknik Komputer AMIK BSI 1 (2), 178-183. Tersedia di: < http://ejournal.bsi.ac.id/assets/files/JTK-178-183-Kresna.pdf> [Diakses 18 Mei 2017]

Rifqi, M., Cholissodin, I. &Santoso, E., 2016. *Optimasi Vektor Bobot Pada Learning Vector Quantization Dengan Algoritme Genetika Untuk Klasifikasi Tingkat Resiko Penyakit Stroke.* Repositori Jurnal Mahasiswa PTIIK UB, Volume 8 - Number 14. Tersedia di: < http://filkom.ub.ac.id/doro/archives/detail /DR00134201612# [Diakses 25 Februari 2017]

Rohman, F. F. dan Fauzijah, A., 2008. *Rancang Bangun Aplikasi Sistem Pakar untuk Menentukan Jenis Gangguan Perkembangan Pada Anak*. Media Informatika, Vol. 6, No. 1, Juni 2008, 1-23 ISSN: 0854-4743. Tersedia di: <http://jurnal.uii.ac.id/index.php/media-informatika/article/viewFile/106/66> [Diakses 26 Februari 2017]

Sutojo, T., Mulyanto, E & Suhartono, V, 2011. *Kecerdasan Buatan*. Yogyakarta. Penerbit ANDI.

Tanoyo, Diana Purnamasari, 2013. *Attention-Deficit/Hyperactivity Disorder Diagnosis And Treatment*.E-Jurnal Medika Udayana 2.7 (2013): 1179-1197. Tersedia di:< http://ojs.unud.ac.id/index.php/eum/article/view/5811/4374> [Diakses 8 Maret 2017]