OPTIMASI PARAMETER *SUPPORT VECTOR MACHINE* (SVM) DENGAN *PARTICLE SWARM OPTIMIZATION* (PSO) UNTUK KLASIFIKASI PENDONOR DARAH DENGAN DATASET RFMTC

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
I Gusti Ngurah Ersania Susena
NIM: 115060807111151

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

PENGESAHAN

OPTIMASI PARAMETER *SUPPORT VECTOR MACHINE* (SVM) DENGAN *PARTICLE SWARM OPTIMIZATION* (PSO) UNTUK KLASIFIKASI PENDONOR DARAH DENGAN DATASET RFMTC

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
I Gusti Ngurah Ersania Susena
NIM: 115060807111151

Skripsi ini telah diuji dan dinyatakan lulus pada
3 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Muhammad Tanzil Furqon, S.Kom, M.CompSc	Randy Cahya Wihandika, S.ST, M.Kom
NIP: 19820930 200801 1 004	NIK: 201405 880206 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

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

PENGUJI

- Penguji I / Ketua Majelis

  Edy Santoso, S.Si, M.Kom

- Penguji II

  Ir. Sutrisno, M.T

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, 3 Agustus 2018

I Gusti Ngurah Ersania Susena

NIM: 115060807111151

DAFTAR RIWAYAT HIDUP

NAMA LENGKAP : I GUSTI NGURAH ERSANIA SUSENA
JENIS KELAMIN : LAKI-LAKI
TTL : MATARAM, 16 JUNI 1993
FAKULTAS : FAKULTAS ILMU KOMPUTER
UNIVERSITAS : UNIVERSITAS BRAWIJAYA
JURUSAN/PRODI : TEKNIK INFORMATIKA
NIM : 115060807111151
NO HP : 0812406798686
ALAMAT DI MALANG : JALAN SUNANKALIJAGA NO 16
AGAMA : HINDU
ANAK KE : 1 DARI 2 BERSAUDARA

RIWAYAT PENDIDIKAN

PENDIDIKAN	TEMPAT	TAHUN
TK YPRU	MATARAM	1998
SDN 2	MATARAM	2005
SMPN 2	MATARAM	2008
SMAN 1	MATARAM	2011

UCAPAN TERIMAKASIH

Penulis memanjatkan puji syukur kehadirat Tuhan Yang Maha Esa, karena dengan segala berkat dan pengetahuan-Nya yang datang dari segala penjuru, penulis mampu merampungkan skripsi yang berjudul "Optimasi Parameter *Support Vector Machine* (SVM) Dengan *Particle Swarm Optimization* (PSO) Untuk Klasifikasi Pendonor Darah Dengan Dataset RFMTC". Penulisan skripsi ini disusun dalam upaya penulis memenuhi syarat menjadi Sarjana Komputer.

Selama pengerjaan skripsi ini penulis mendapatkan banyak bantuan dari banyak pihak, baik dengan moril dan materiil. Dalam kesempatan ini penulis bermaksud ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Bapak Muhammad Tanzil Furqon, S.kom., M.CompSc., selaku pembimbing utama yang banyak memberikan waktu luang untuk memberikan ilmu dan membimbing penulis secara intensif dalam proses penyelesaian skripsi.
2. Bapak Randy Cahya Wihandika, S.ST., M.Kom., selaku pembimbing pendamping yang memberikan saran dan arahan untuk penyelesaian laporan skripsi.
3. Bapak Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D., selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya
4. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D., selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
5. Bapak Ir. Heru Nurwasito, M.Kom., selaku Wakil Ketua I Bidang Akademik Fakultas Ilmu Komputer Universitas Brawijaya.
6. Bapak Agus Wahyu Widodo, S.T., M.Cs., selaku Ketua Program Studi Teknik Informatika Universitas Brawijaya.
7. Bapak Muhammad Tanzil Furqon, S.Kom., M.CompSc, selaku sekretaris Program Studi Teknik Informatika Universitas Brawijaya.
8. Staff BK Fakultas Ilmu Komputer Bapak Prasetyo Iskandar, S.T., dan Ibu Wiwin Lukitohadi, S.H., S.Psi., CHRM., yang selalu memberikan dorongan semangat tanpa batas kepada penulis melalui bimbingan konseling secara intensif.
9. Segenap dosen Fakultas Ilmu Komputer Universitas Brawijaya atas ilmu pengetahuan yang diturunkan kepada penulis.

10. Segenap staff dan pegawai Fakultas Ilmu Komputer Universitas Brawijaya atas segala bantuan yang bersifat administratif.
11. Keluarga besar tercinta, Bapak I Gusti Gede Prajendra, S.H., Ibu Ir. Aryani dan adik I Gusti Arya Prawira Sanjaya yang selalu memberikan kepercayaan penuh dengan dukungan baik secara moril, materiil, bimbingan, kritik, saran dan doa yang tiada henti kepada penulis.
12. Seluruh keluarga besar Teknik Informatika angkatan 2011 Universitas Brawijaya.
13. Seluruh keluarga besar Unikahidha Universitas Brawijaya yang telah memberikan pengalaman berorganisasi bagi penulis.
14. Seluruh keluarga, saudara, kakak, dan adik yang penulis jumpai selama penulis menempuh studi di Fakultas Ilmu Komputer Universitas Brawijaya.
15. Seluruh saudara di malang yang telah membantu dukungan dan memberikan pengalaman yang sangat berharga bagi penulis selama kebersamaannya dalam penyelesaian skripsi penulis.
16. Serta semua pihak / saudara yang tidak dapat penulis sebutkan satu per satu baik yang terlibat secara langsung maupun tidak langsung demi rampungnya skripsi penulis.

Akhir kata atas segala doa, bimbingan, bantuan, kritik dan saran semoga semua pihak mendapat balasan yang sepadan dari Tuhan Yang Maha Esa. Dengan keterbatasan ilmu yang diraih, penulis sadar bahwa skripsi ini jauh dari tujuan dan manfaat yang ingin dicapai, maka dari itu atas segala kerendahan hati penulis mohon sekiranya pembaca dapat memberikan saran dan kritik yang membangun demi memperoleh tujuan dan manfaat yang ingin dicapai pada penelitian selanjutnya.

Malang, 3 Agustus 2018

Penulis

senaersania@gmail.com

ABSTRAK

Donor darah merupakan salah satu kegiatan kemanusiaan yang dilakukan secara sukarela. Darah merupakan salah satu unsur zat terpenting yang dimiliki manusia pada siklus kehidupan manusia. Dalam melaksanakan kegiatan donor darah, pemantauan ketersediaan stok kantung darah biasanya menjadi masalah utama. Untuk memenuhi stok kantung darah diperlukannya sistem yang dapat memprediksi perilaku pendonor darah. RFMTC (*Recency, Frequency, Monetary, Time, Churn Probability*) adalah metode RFM yang telah di modifikasi agar dapat melihat perilaku dari pendonor yang dapat mendonorkan darahnya kembali atau tidak mendonor. Maka dari itu dibutuhkan klasifikasi dari perilaku pendonor darah dengan menggunakan metode SVM-PSO. Dengan teknik SVM bekerja untuk mencari *hyperplane* yaitu garis pemisah antar kelas data. Kemudian teknik PSO bekerja untuk mencari nilai range parameter masukan yang dibutuhkan SVM agar mendapat nilai *hyperplane* yang optimal. Penelitian ini menggunakan 748 data dari UCI dataset dengan 4 fitur utama dan 2 kelas. Berdasarkan pengujian yang telah dilakukan didapatkan nilai akurasi sebesar 90% dengan nilai iterasi pelatihan SVM yang kecil dan nilai jumlah partikel PSO yang rendah.

Kata kunci: *RFMTC, Pendonor darah, SVM, PSO, SVM-PSO, Klasifikasi, Parameter, Optimasi.*

ABSTRACT

Blood donation is one of voluntary humanitarian activities. Blood is one of the most important substances that humans have in the human life cycle. In carrying out blood donation activities, monitoring the stock availability of blood bags is usually a major problem. To know ammount stock of blood bag we need a system that can predict the behavior of blood donors. RFMTC (Recency, Frequency, Monetary, Time, Churn Probability) is a modified RFM method in order to see the behavior of donors who can donate their blood or not to donate again. Therefore, SVM-PSO method needed to know classification of blood donors behavior. With SVM techniquesto find hyperplane that is the dividing line between data classes. Then the PSO technique to find the range of input parameters that SVM needed to get the optimal hyperplane value. This research uses 748 data from UCI dataset with 4 main features and 2 classes. Based on the test that has been done obtained the accuracy of 90% with the value of learning iteration SVM small and the value of the number of PSO particles are low.

***Keywords**: RFMTC, Blood Donors, SVM, PSO, SVM-PSO, Classification, Parameter, Optimization.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE PROGRAM

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Darah merupakan materi biologis yang hidup dan belum dapat diproduksi di luar tubuh manusia. Definisi dari donor darah adalah proses pemindahan darah dari pendonor yang sehat jasmani dan rohani kepada penerima donor darah (resipien). Proses pengambilan darah dilakukan secara sukarela dan disimpan di bank darah sebagai stok untuk dapat digunakan transfusi darah selanjutnya. Kegiatan donor darah di Indonesia seringkali dilakukan dan dikelola oleh PMI. Di luar itu ada juga komunitas donor darah yang berdiri secara sukarela. Pemenuhan kebutuhan darah sangat penting untuk meningkatkan pelayanan kesehatan dan membantu dalam upaya menyelamatkan nyawa seseorang (Kementerian Kesehatan Republik Indonesia, 2014).

Dalam kegiatan donor darah sangat dibutuhkan partisipasi dari masyarakat sendiri dengan sarana kesehatan yang ditunjang oleh ketersediaan fasilitas, sarana dan prasarana yang dapat menjamin ketersediaan darah dalam jumlah yang cukup, aman dan berkualitas (Kementerian Kesehatan Republik Indonesia, 2014). Minim kesadaran dari masyarakat sendiri untuk menjadi pendonor golongan sukarela yang menjadikan persediaan kantong darah di Unit Donor Darah (UDD) menjadi kurang tercukupi atau rendah. Data ideal untuk ketersediaan darah yang diperuntukkan untuk donor darah adalah 2,5% dari jumlah penduduk. Indonesia sendiri terdata pada tahun 2013 terdapat kekurangan kantong darah sebanyak 2.476.389 (Pusat Data dan Informasi Kementerian Kesehatan RI, 2014). Donor darah di Indonesia kebanyakan bersifat musiman atau dilakukan berkaitan dengan *event* tertentu, ini bertolak belakang dengan negara maju yang sangat rutin mendonor secara sukarela setiap tiga bulan (Pusat Data dan Informasi Kementerian Kesehatan RI, 2014). Menurut *American Red Cross*, darah hasil dari donor tidak dapat bertahan setelah 42 hari sejak didapatkan dari pendonor darah. Salah satu cara untuk memenuhi ketersediaan kantong darah yang mencukupi adalah dengan mempunyai data sumbangan rutin dari relawan yang sehat (Darwiche et al , 2010). Untuk memenuhi stok kantung darah diperlukannya sistem yang dapat memprediksi perilaku pendonor darah. RFMTC (*Recency, Frequency, Monetary, Time, Churn Probability*) adalah metode RFM yang telah di modifikasi agar dapat melihat perilaku dari pendonor yang dapat mendonorkan darahnya kembali atau tidak mendonor. Maka dari itu dibutuhkan klasifikasi dari perilaku pendonor darah dengan menggunakan metode SVM-PSO. Dengan teknik SVM bekerja untuk mencari *hyperplane* yaitu garis pemisah antar kelas data. Kemudian teknik PSO bekerja untuk mencari nilai range parameter masukan yang dibutuhkan SVM agar mendapat nilai *hyperplane* yang optimal. Berdasarkan masalah klasifikasi dan regresi dengan linier ataupun non-linier kernel *Support Vector Machine* (SVM) dapat digunakan yang dapat menjadi satu kemampuan algoritma pembelajaran untuk klasifikasi serta regresi. *Support Vector Machine*

(SVM) juga memiliki akurasi tinggi dengan kesalahan yang relatif kecil dan dapat digunakan untuk melakukan prediksi, selain memiliki kelebihan tentunya *Support Vector Machine* (SVM) juga memiliki kekurangan dalam menentukan nilai parameter yang optimal. Beberapa algoritma banyak direkomendasikan untuk mengoptimasi parameter pada *machine learning* seperti *Particle Swarm Optimization* (PSO). *Particle Swarm Optimization* (PSO) digunakan untuk menentukan parameter optimal dengan menemukan parameter yang memiliki akurasi terbaik saat mereka bergerak dalam *problem space* dari parameter *Support Vector Machine* (SVM).

Terdapat banyak penelitian sebelumnya yang menyatakan *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO) mampu mengoptimalkan nilai parameter dari klasifikasi tersebut. Seperti penelitian yang pernah dilakukan sebelumnya oleh Gur Emre (2014) tentang pertumbuhan tulang balita usia 0-6 tahun dengan menggunakan SVM dan PSO. Penelitian ini membandingkan akurasi SVM-PSO dengan metode lainnya dalam mengklasifikasikan pertumbuhan tulang balita dengan SVM, *Naïve Bayes*, C4.5 dan KNN. Dari hasil perbandingan didapatkan bahwa dengan metode SVM-PSO tingkat akurasi lebih tinggi yaitu 74,87% sedangkan SVM 73,82%, *Naive Bayes* 68,21%, C4.5 67,18% dan KNN 70,26. Ini membuktikan bahwa dengan penambahan optimasi nilai parameter sebelum melakukan iterasi pada pelatihan SVM, sistem mampu memberikan hasil klasifikasi yang akurasinya lebih tinggi.

Penelitian sebelumnya dilakukan oleh Nugroho (2018) tentang klasifikasi pendonor darah dengan menggunakan metode SVM pada dataset RFMTC. Penelitian ini dilakukan dengan mencoba rasio perbandingan kelas data, dimana perbandingan rasio optimal yang diperoleh adalah 50% : 50% dengan hasil keluaran akurasi 64,75%.

Penelitian lainnya juga dilakukan oleh Ayoub A., et al (2014) tentang mendeteksi risiko kanker payudara dengan menggunakan SVM dan PSO. Penelitian ini juga membandingkan hasil akurasi dari beberapa metode lainnya. Hasil akurasi yang didapatkan dari metode SVM-PSO adalah 97,66% sedangkan metode SVM 94,74%, NEF Class 97,20%, Fuzzy-GA1 97,36% dan Neuro-rule 2a 98,10%. Walaupun SVM-PSO nilai akurasi yang dihasilkan tidak sebesar metode Neuro-rule 2a namu SVM-PSO dapat memberikan nilai akurasi yang cukup besar. Untuk penelitian lebih lanjut lagi mungkin dibutuhkan modifikasi pada metode PSO agar nilai akurasi dapat lebih akurat lagi.

Penelitian juga dilakukan oleh Musyaffa et all., tentang memprediksi risiko penyakit liver dengan membandingkan metode SVM dan metode SVM-PSO. Hasil nilai akurasi yang diberikan oleh metode SVM sebesar 71,36% namun ketika diberikan optimasi pada parameter SVM dengan PSO akurasi meningkat menjadi 77,36%. Dapat disimpulkan bahwa nilai parameter untuk pelatihan SVM berpengaruh terhadap *sequential learning* yang ada pada metode SVM untuk mendapat nilai akurasi. Sehingga PSO disini sangat berperan penting dalam mencari partikel terbaik agar nilai akurasi keluaran dari SVM semakin meningkat.

Dengan demikian berdasarkan permasalahan diatas solusi untuk klasifikasi perilaku pendonor darah menggunakan metode *Support Vector Machine* (SVM) untuk mencari *hyperplane* garis pemisah antar kelas data dan dioptimasi dengan *Particle Swarm Optimization* (PSO) agar mampu memberikan interval parameter terbaik yang digunakan untuk proses klasifikasi. Maka pada skripsi ini akan dibangun suatu sistem sebagai alat optimasi untuk klasifikasi data perilaku pendonor darah dengan metode *Support Vector Machine* (SVM) dengan *Particle Swarm Optimization* (PSO) yang bertujuan untuk mendapatkan meminimalkan kesalahan generalisasi dan menentukan parameter terbaik dari *Support Vector Machine* (SVM). Maka judul yang diangkat pada skripsi ini adalah **Optimasi Akurasi *Support Vector Machine* (SVM) dengan *Particle Swarm Optimization* (PSO) untuk Klasifikasi Pendonor Darah Dengan RFMTC.**

1.2 Rumusan masalah

Rumusan masalah yang ada adalah sebagai berikut:

1. Bagaimana implementasi *Support Vector Machine* (SVM) dengan menggunakan *Particle Swarm Optimization* (PSO) untuk klasifikasi pendonor darah dengan RFMTC?
2. Bagaimana pengaruh optimasi parameter *Particle Swarm Optimization* (PSO) terhadap nilai akurasi *Support Vector Machine* (SVM) untuk klasifikasi pendonor darah dengan RFMTC?

1.3 Tujuan

Berikut ini adalah tujuan yang ingin dicapai dalam penelitian ini adalah

1. Menerapkan implementasi pengoptimalan parameter *Support Vector Machine* (SVM) dengan menggunakan *Particle Swarm Optimization* (PSO) untuk klasifikasi donor darah dengan RFMTC.
2. Menguji tingkat akurasi implementasi pengoptimalan parameter *Support Vector Machine* (SVM) dengan menggunakan *Particle Swarm Optimization* (PSO) untuk klasifikasi donor darah dengan RFMTC.

1.4 Manfaat

Manfaat yang diperoleh dari pengoptimalan *Support Vector Machine* (SVM) dengan menggunakan *Particle Swarm Optimization* (PSO) untuk klasifikasi donor darah dengan RFMTC adalah:

1. Dapat membantu klasifikasi pendonor darah dengan *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO).
2. Mengetahui parameter terbaik pada *Support Vector Machine* (SVM) dengan *Particle Swarm Optimization* (PSO) untuk klasifikasi pendonor darah dengan RFMTC.
3. Dapat mempermudah sistem informasi kelayakan pada dataset RFMTC dalam mengklasifikasikan pendonor darah.

1.5 Batasan masalah

Berdasarkan dengan latar belakang dan perumusan masalah yang luas telah diuraikan, agar pembahasan dalam penelitian ini tidak meluas, dibatasi hal-hal sebagai berikut:

1. Klasifikasi untuk pendonor darah sebatas 2 kelas dibuat berdasarkan *Support Vector Machine* (SVM) dengan *Particle Swarm Optimization* (PSO).
2. Data yang digunakan dalam penelitian ini adalah data *Blood Transfusion Service Center Data Set* UCI *repository, Hsin-Chu Taiwan,* sebanyak 748 data.
3. Parameter *Support Vector Machine* (SVM) yang dioptimasi adalah *lambda* (λ), *gamma* (γ), *Complexity(C)* dan *varian(σ)*.

1.6 Sistematika pembahasan

Sistematika dalam penulisan bertujuan memberikan gambaran penyusunan laporan yang meliputi beberapa bab. Sistematika penulisan yang diajukan untuk penyusunan laporan mencakup tujuh bab dengan masing-masing bab diuraikan secara garis besar penelitian adalah sebagai berikut :

BAB I Pendahuluan

Berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat penelitian, serta sistematika penulisan dari skripsi ini.

BAB II Landasan Kepustakaan

Menguraikan kajian pustaka dan dasar teori yang berhubungan dengan metode *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO).

BAB III Metodologi Penelitian

Membahas metodologi yang akan digunakan dalam penelitian dengan serangkaian langkah yang dilakukan peneliti untuk menyelesaikan permasalahan dalam pembuatan optimasi untuk klasifikasi pendonor darah dengan metode *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO).

BAB IV Perancangan

Perancangan berisi tentang perencanaan rancangan dari optimasi pemilihan pendonor darah dengan metode *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO). Mulai dari rancangan model, rancangan antar muka sistem, dan rancangan pengujian.

BAB V Implementasi

Implementasi membahas hasil dari penerapan bab perancangan yang telah disusun, bagaimana optimasi pemilihan pendonor darah dengan berdasarkan data yang ada, antar muka sistem, dan *source code* untuk mengembangkan aplikasi optimasi pemilihan pendonor darah dengan metode *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO).

BAB VI Pengujian dan pembahasan

Pengujian dan analisis ini menjelaskan tentang strategi pengujian dan teknik pengujian yang ada terhadap sistem yang direalisasikan.

BAB VII Penutup

Berisi kesimpulan dan pengembangan dan pengujian serta saran-saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini membahas landasan kepustakaan yang berhubungan dengan optimasi untuk klasifikasi pendonor darah yang bisa melakukan donor darah atau tidak bisa melakukan donor darah kembali dengan RFMTC. Pada pengklasifikasian untuk pendonor darah digunakan metode *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO)

2.1 Kajian Pustaka

Pada penulisan skripsi ini, dilakukan kajian terhadap penelitian-penelitian sebelumnya. Penelitian tersebut meliputi penyelesaian implementasi dari metode *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO)

Pada penelitian yang dilakukan Gur Emre (2014) tentang pertumbuhan tulang balita usia 0-6 tahun dengan menggunakan SVM dan PSO. Penelitian ini membandingkan akurasi SVM-PSO dengan metode lainnya dalam mengklasifikasikan pertumbuhan tulang balita dengan SVM, *Naïve Bayes*, C4.5 dan KNN. Dari hasil perbandingan didapatkan bahwa dengan metode SVM-PSO tingkat akurasi lebih tinggi yaitu 74,87% sedangkan SVM 73,82%, *Naive Bayes* 68,21%, C4.5 67,18% dan KNN 70,26. Ini membuktikan bahwa dengan penambahan optimasi nilai parameter sebelum melakukan iterasi pada pelatihan SVM, sistem mampu memberikan hasil klasifikasi yang akurasinya lebih tinggi.

Penelitian lainnya juga dilakukan Nugroho (2018) tentang klasifikasi pendonor darah dengan menggunakan metode SVM pada dataset RFMTC. Penelitian ini dilakukan dengan mencoba rasio perbandingan kelas data, dimana perbandingan rasio optimal yang diperoleh adalah 50% : 50% dengan hasil keluaran 64,75%.

Penelitian Ayoub A., et al (2014) tentang mendeteksi risiko kanker payudara dengan menggunakan SVM dan PSO. Penelitian ini juga membandingkan hasil akurasi dari beberapa metode lainnya. Hasil akurasi yang didapatkan dari metode SVM-PSO adalah 97,66% sedangkan metode SVM 94,74%, NEF Class 97,20%, Fuzzy-GA1 97,36% dan Neuro-rule 2a 98,10%. Walaupun SVM-PSO nilai akurasi yang dihasilkan tidak sebesar metode Neuro-rule 2a namu SVM-PSO dapat memberikan nilai akurasi yang cukup besar. Untuk penelitian lebih lanjut lagi mungkin dibutuhkan modifikasi pada metode PSO agar nilai akurasi dapat lebih akurat lagi.

Penelitian menggunakan metode *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO) untuk analisis kanker payudara. *Support Vector Machine* (SVM) sebagai model perhitungan komputasi cepat yang meminimalkan kesalahan generalisasi dan *Particle Swarm Optimization* (PSO) digunakan untuk teknik optimasi penentuan parameter terbaik dari *Support Vector Machine* (SVM) (Arafi et al., 2013)

2.2 Donor Darah

Donor darah adalah orang yang menyumbangkan darah mereka secara sukarela kepada orang lain yang memang memerlukan suplai darah dari luar tubuh dengan tujuan untuk membantu atau menyelamatkan nyawa mereka. Ini karena, sampai saat ini darah belum bisa di sintesis pada kondisi penyakit penyakit tertentu sehingga harus diambil dari orang lain dan transfusi pada penderita (Kementerian Kesehatan Republik Indonesia).

2.3 *Dataset* RFMTC

Variabel dari dataset RFMTC (*Recency, Frequency, Monetary value, Time since first purchase and Churn probability*) merupakan modifikasi dari teknik RFM, tiga variabel sederhana, yaitu *Recency of purchase, Frequency of purchase,* dan *Monetary value of purchase*, merupakan dasar dari teknik RFMTC. RFMTC merupakan hasil modifikasi dari Y, I-Cheng, Y, King-Jang, and T, Tao-Ming yang digunakan untuk meramalkan perilaku pendonor darah apakah pendonor tersebut termasuk klasifikasi yang potensial mendonorkan darahnya kembali atau tidak, Akurasi dari RFMTC ini menurut I-Cheng et all, lebih tinggi daripada RFM, Adapun penjelasan dari variabel RFMTC tersebut adalah :

a. *Recency* Jumlah bulan sejak terakhir menyumbangkan darah.
b. *Frequency* adalah jumlah berapakali donor.
c. *Monetary* adalah jumlah darah yang disumbangkan dalam c.c.
d. *Time* yaitu jumlah bulan sejak pertama menyumbangkan darah, *Churn Probability* yaitu *or* merupakan variabel yang merepresentasikan apakah pendonor mendonorkan darahnya kembali atau tidak, 1 menyatakan mendonorkan darah (*Donation*), 0 menyatakan tidak menyumbangkan darah (*non Donation*).

2.4 *Support Vector Machine* (SVM)

Support Vector Machine (SVM) adalah sistem pembelajaran yang diperkenalkan untuk menyelesaikan masalah pengenalan pola oleh Vapnik. SVM adalah salah satu teknik klasifikasi data dengan proses pelatihan (*supervised learning*) untuk menemukan garis pemisah (*hyperplane*) yang terletak di tengah-tengah antara dua set objek dari dua kelas dengan fungsi persamaan 2.1 sehingga diperoleh ukuran *margin* dari dua kelas secara maksimal (Vapnik, 1997).

$$f(w,b) = x_i \cdot w + b \qquad (2.1)$$

Keterangan :

x_i : data ke-*i*
w : vektor yang tegak lurus terhadap *hyperplane*
b : nilai bias (*threshold*)

Persamaan bidang pembatas kelas positif dan kelas negatif (Vapnik, 1997) ditunjukkan pada persamaan 2.2 dan 2.3.

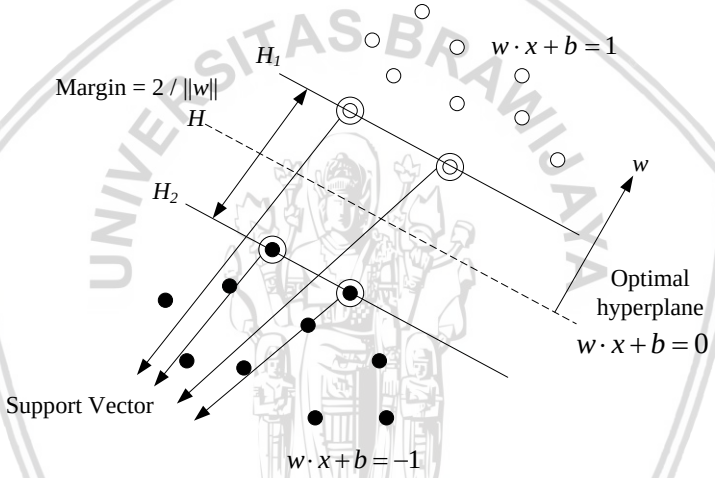
$$x_i \cdot w + b \geq 1 \text{ untuk } y_i = 1 \quad (2.2)$$

$$x_i \cdot w + b \leq 1 \text{ untuk } y_i = -1 \quad (2.3)$$

Keterangan :

y_i : kelas data ke-*i*

Titik yang terdekat dengan *hyperplane* disebut *support vector* ditunjukkan pada Gambar 2.1. *Margin* dapat ditentukan berupa jarak antara *support vector* positif dengan *support vector* negatif. SVM membutuhkan data *training* kelas positif dan kelas negatif. Pelatihan kelas positif dan kelas negatif ini dibutuhkan SVM untuk membuat keputusan terbaik dalam memisahkan data positif dengan data negatif di ruang n-dimensi.



Gambar 2.1 *Hyperplane*

Sumber : Vapnik (1997)

Jika kedua bidang pembatas direpresentasikan dalam pertidaksamaan dan *margin* dimaksimalkan maka ditemukan *hyperplane* terbaik dengan persamaan 2.4 dan 2.5 (Vapnik, 1997).

$$\textit{minimize } J_1[w] = \frac{1}{2}\|w\|^2 \quad (2.4)$$

$$y_i(x_i \cdot w + b) - 1 \geq 0 \text{ untuk } i = 1, \ldots, l \quad (2.5)$$

Keterangan :

J_1 : fungsi *minimize*

l : jumlah data

Sedangkan untuk mengklasifikasikan data yang tidak dapat dipisahkan secara linier formula SVM harus dimodifikasi karena tidak akan ada solusi yang ditemukan. Oleh karena itu, kedua bidang pembatas harus diubah sehingga lebih fleksibel (untuk kondisi tertentu) dengan penambahan variabel *slack* $\xi \geq 0$ dengan

parameter *C* untuk meminimalkan *misclassification* dan memperkecil nilai variabel *slack* (Cristianini, 2000) sesuai persamaan 2.6 dan 2.7.

$$\textit{minimize } J_1[w,\xi_i] = \left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i\right) \qquad (2.6)$$

$$y_i(x_i \cdot w + b) - 1 + \xi_i \geq 0 \text{ dimana } \xi_i \geq 0 \text{ untuk } i = 1, \ldots, l \qquad (2.7)$$

Keterangan :

C : parameter *user* bernilai bilangan positif (batasan *error*)

ξ : variabel *slack* (mengukur *error* dari data)

Bentuk persamaan 2.6 dengan batasan 2.7 juga dapat dibentuk menjadi *quadratic convex programming problem* tanpa batasan dalam bentuk *lagrangian multiplier* dengan persamaan 2.8.

$$\textit{minimize } L(w,b,\xi_i,\alpha,r) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i[y_i(x_i \cdot w + b) - 1 + \xi_i] - \sum_{i=1}^{l} r_i\xi_i \qquad (2.8)$$

$\alpha = (\alpha_1,\ldots,\alpha_l)$ dan $r = (r_1,\ldots,r_l)$ adalah *non-negative Lagrange multiplier*.

Persamaan 2.8 dapat dikonversi ke dalam *dualitas lagrange multiplier* yang lebih mudah pada proses perhitungan dengan persamaan 2.9, 2.10, dan 2.11.

$$\textit{maximize } L_D(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\alpha \cdot D\alpha \qquad (2.9)$$

$$\sum_{i=1}^{n} y_i\alpha_i = 0 \text{ dan } 0 \leq \alpha_i \leq 0 \text{ untuk } i = 1, \ldots, l \qquad (2.10)$$

$$D_{ij} = y_i y_j x \cdot x_i + y_i y_j \lambda^2 \qquad (2.11)$$

Keterangan :

λ : faktor penambah

Setelah optimal *multiplier* α ditemukan, maka klasifikasi dapat menggunakan fungsi yang ditunjukkan pada persamaan 2.12 (Cristianini, 2000).

$$f(x) = \text{sign}\left(\sum_{i=1}^{n}\alpha_i y_i x \cdot x_i + b\right) \qquad (2.12)$$

Nilai bias dapat dicari dengan persamaan 2.13 dengan nilai *w* pada persamaan 2.14.

$$b = -\frac{1}{2}\left(x_i^+ . w + x_i^- . w\right) \qquad (2.13)$$

$$w = \sum_{i=1}^{n}\alpha_i y_i x_i \qquad (2.14)$$

2.4.1 *Sequential Learning* SVM Untuk Klasifikasi

Dalam pembelajaran metode SVM untuk mencari *hyperplane* yang optimal digunakan *quadratic programming* (QP). Namun penyelesaian QP terkadang memiliki persamaan cukup kompleks, membutuhkan waktu eksekusi yang besar, dan rentan terhadap ketidakstabilan. Sehingga dikembangkan metode *sequential training* untuk klasifikasi pada SVM yang memberikan nilai optimal dari *langrage multiplier α* untuk masalah klasifikasi pada dimensi tinggi (Vijayakumar, 1999).

1. Inisialisasi $\alpha = 0$ dan beberapa parameter yaitu λ, γ, C, dan iterasi maksimum. Hitung matriks $[D]_{ij}$ dengan persamaan 2.15.

$$[D]_{ij} = y_i y_j \left(K(x_i, x_j) + \lambda^2\right) \text{ untuk } i, j = 1, \ldots, l \qquad (2.15)$$

   Keterangan :
   $K(x_i, x_j)$: fungsi kernel

2. Untuk i=1 hingga l, masing-masing hitung persamaan 2.16, 2.17, dan 2.18.

   a. $$E_i = \sum_{j=1}^{n} \alpha_j D_{ij} \qquad (2.16)$$

   b. $$\min(\max[\gamma(1 - E_i), -\alpha_i], C - \alpha_i) \qquad (2.17)$$

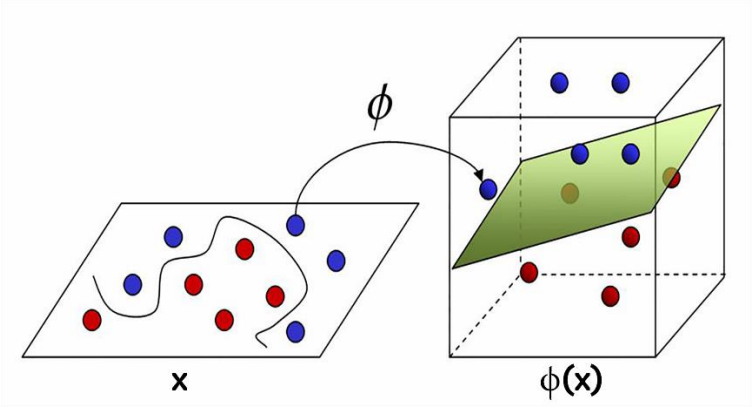
   c. $$\alpha_i = \alpha_i + \delta\alpha_i \qquad (2.18)$$

   Keterangan :
   E_i : *Error* data ke-i
   γ : parameter untuk mengontrol kecepatan proses *learning*
   $\delta\alpha$: fungsi pembatas *Lagrange multiplier*.

Langkah 2 dilakukan terus-menerus hingga kondisi telah mencapai konvergen atau iterasi maksimum tercapai.

2.4.2 Kernel

Banyak teknik pembelajaran yang dikembangkan dengan asumsi kelinieran. Sehingga algoritma yang dihasilkan terbatas untuk kasus-kasus yang linier. Karena itu, bila suatu kasus klasifikasi memperlihatkan ketidaklinieran, algoritma seperti *perceptron* tidak bisa mengatasinya. Secara umum, kasus-kasus di dunia nyata kebanyakan adalah kasus yang tidak linier.



Gambar 2.2 *Feature Space*

Sumber : Scholkopf (2002)

Gambar 2.3 bagian kiri, data tersebut sulit dipisahkan secara linier. Metoda kernel adalah salah satu cara untuk mengatasinya. Dengan metoda kernel suatu data x di *input space* dipetakan ke *feature space* dengan dimensi yang lebih tinggi sebagai $\phi : x \rightarrow \phi(x)$. Menghasilkan data x di *input space* menjadi $\phi(x)$ di *feature space* (Scholkopf, 2002).

Sering kali fungsi $\phi(x)$ tidak dapat dihitung, tetapi *dot product* dari dua vektor dapat dihitung baik di dalam *feature space*. Dengan kata lain, sementara $\phi(x)$ mungkin tidak diketahui, *dot product* $\phi(x).\phi(x')$ masih bisa dihitung di *feature space*. Sebagai konsekuensi, pembatas yang menjelaskan permasalahan dalam klasifikasi harus diformulasikan kembali sehingga menjadi bentuk *dot product*. Suatu fungsi kernel $k(x,x')$ bisa untuk menggantikan *dot product* $\phi(x).\phi(x')$. Kemudian di *feature space*, bisa dibuat suatu fungsi pemisah yang linier yang mewakili fungsi *non*-linier di *input space*. Dalam *input space*, data tidak bisa dipisahkan secara linier, tetapi bisa dipisahkan di *feature space*. Karena itu dengan memetakan data ke *feature space* menjadikan tugas klasifikasi menjadi lebih mudah (Santosa, 2005).

Fungsi kernel yang biasanya dipakai dalam literatur SVM (Haykin, 1999) :

1. Linier : $K(x,y) = x \cdot y$ (2.19)
2. *Polynomial of degree up to d* : $K(x,y) = (x \cdot y + c)^d$ (2.20)
3. Gaussian RBF : $K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$ (2.21)
4. Sigmoid : $K(x,y) = \tanh(\sigma(x \cdot y) + c)$ (2.22)

Pemilihan fungsi kernel yang tepat adalah hal yang sangat penting. Fungsi kernel akan menentukan *feature space* di mana fungsi klasifikasi akan dicari. Sepanjang fungsi kernelnya benar, SVM akan beroperasi secara benar meskipun tidak mengetahui pemetaan yang digunakan. Fungsi kernel yang benar diberikan oleh teori Mercer (Vapnik, 1997) dimana fungsi itu harus memenuhi syarat

kontinus dan *positive definite*. Pada penerapan metoda kernel, tidak perlu mengetahui pemetaan yang digunakan untuk satu per satu data, tetapi lebih penting mengetahui bahwa *dot produk* dua titik di *feature space* bisa digantikan oleh fungsi kernel. Fungsi kernel yang direkomendasikan untuk diuji pertama kali adalah fungsi kernel gaussian RBF karena memiliki performa yang mirip dengan kernel linier pada parameter tertentu (Hsu, 2004).

2.5 Particle Swarm Optimization (PSO)

Particle swarm optimization, disingkat sebagai PSO, didasarkan pada perilaku sebuah kawanan serangga, seperti semut, rayap, lebah atau burung. Algoritma PSO meniru perilaku sosial organisme ini. Perilaku sosial terdiri dari tindakan individu dan pengaruh dari individu-individu lain dalam suatu kelompok. Kata "partikel" menunjukkan individu, misalnya seekor burung dalam kawanan burung. Setiap individu atau partikel berperilaku saling terhubung dengan cara menggunakan kecerdasannya (*intelligence*) sendiri dan juga dipengaruhi perilaku kelompok kolektifnya. Dengan demikian, jika satu partikel atau seekor burung menemukan jalan yang tepat atau pendek menuju ke sumber makanan, sisa kelompok yang lain juga akan dapat segera mengikuti jalan tersebut meskipun lokasi mereka jauh di kelompok tersebut.

Metode optimasi yang didasarkan pada *swarm intelligence* ini disebut algoritma *behaviorally inspired* sebagai alternatif dari algoritma genetika, yang sering disebut *evolution-based procedures*. Algoritma PSO ini awalnya diusulkan oleh J. Kennedy dan R. C. Eberhart (Kennedy, 1995). Dalam konteks optimasi multivariabel, kawanan diasumsikan mempunyai ukuran tertentu atau tetap dengan setiap partikel posisi awalnya terletak di suatu lokasi yang acak dalam ruang multidimensi. Setiap partikel diasumsikan memiliki dua karakteristik posisi dan kecepatan. Memperbarui posisi ditunjukkan pada persamaan 2.23 dan besar kecepatan pada persamaan 2.24.

$$x_j(i+1) = x_j(i) + v_j(i+1) \tag{2.23}$$

$$v_j(i+1) = v_j(i) + c_1 r_1(p_{best} - x_j(i)) + c_1 r_1(g_{best} - x_j(i)) \tag{2.24}$$

Keterangan

x_j	: posisi partikel ke-*j*
i	: iterasi
v_j	: kecepatan partikel ke-*j*
c	: learning rate
r	: bilangan random
p_{best}	: posisi terbaik dari partikel
g_{best}	: posisi terbaik dari kelompok

- Setiap partikel bergerak dalam ruang / *space* tertentu dan mengingat posisi terbaik yang pernah dilalui atau ditemukan terhadap sumber makanan atau nilai fungsi objektif. Setiap partikel menyampaikan

informasi atau posisi bagusnya kepada partikel yang lain dan menyesuaikan posisi dan kecepatan masing-masing berdasarkan informasi yang diterima mengenai posisi yang bagus tersebut.

2.5.1 Modifikasi PSO

Dalam implementasinya, ditemukan bahwa kecepatan partikel dalam PSO diupdate terlalu cepat dan nilai minimum fungsi tujuan sering terlewati. Karena itu ada revisi atau perbaikan terhadap algoritma PSO standard. Perbaikan itu berupa penambahan suatu inersia ω untuk mengurangi kecepatan. Biasanya nilai ω dibuat sedemikian hingga semakin meningkat iterasi yang dilalui, semakin mengecil kecepatan partikel. Nilai ini bervariasi secara linier dalam rentang 0.9 hingga 0.4. Secara matematis perbaikan ini menghasilkan persamaan 2.25.

$$v_j(i+1) = \omega\, v_j(i) + c_1 r_1 (p_{best} - x_j(i)) + c_2 r_2 (g_{best} - x_j(i)) \qquad (2.25)$$

Keterangan

ω : bobot inersia

Bobot inersia ini diusulkan oleh Y. Shi dan R. C. Eberhart (Shi, 1998) untuk meredam kecepatan selama iterasi, yang memungkinkan kawanan burung menuju (konvergen) titik target secara lebih akurat dan efisien dibandingkan dengan algoritma aslinya. Formula ini adalah modifikasi dari formula kecepatan partikel. Nilai bobot inersia yang tinggi menambah porsi pencarian global (*global exploration*), sedangkan nilai yang rendah lebih menekankan pencarian lokal (*local search*). Untuk tidak terlalu menitikberatkan pada salah satu bagian dan tetap mencari area pencarian yang baru dalam ruang berdimensi tertentu, maka perlu dicari nilai bobot inersia (ω) yang secara imbang menjaga pencarian global dan lokal. Untuk mencapai itu dan mempercepat konvergensi, suatu bobot inersia yang mengecil nilainya dengan bertambahnya iterasi digunakan dengan persamaan 2.26 (Shi, 1998).

$$\omega(i) = \omega_{\max} - \left(\frac{\omega_{\max} - \omega_{\min}}{i_{\max}} \right) \cdot i \qquad (2.26)$$

dimana ω_{max} dan ω_{min} masing-masing adalah nilai awal dan nilai akhir bobot inersia, i_{max} adalah jumlah iterasi maksimum yang digunakan. Biasanya digunakan nilai ω_{max} = 0,9 dan ω_{min} = 0,4. Perubahan atau modifikasi formula untuk mengupdate kecepatan ini perlu memperhatikan bahwa jika nilai yang terlalu besar akan memungkinan suatu lokal optimum akan terlewati sehingga algoritma justru menemukan lokal optimal yang lain yang tidak lebih baik nilainya.

2.5.2 Bobot Inersia

Persamaan bobot inersia (ω) dapat berupa nilai konstan, penurunan, dan strategi *random* seperti yang ditunjukkan seperti berikut (Huang, 2006) :

1. Konstan : Bobot inersia diatur dengan nilai 1,0.
2. Penurunan : Dengan linier menurunkan bobot inersia dengan persamaan 2.27.

$$\omega(i) = \omega_{\max} - \left(\frac{\omega_{\max} - \omega_{\min}}{i_{\max}} \right) \cdot i \quad (2.27)$$

i adalah iterasi saat ini, i_{max} adalah iterasi maksimum yang telah ditetapkan dan $\omega_{\max}$, $\omega_{\min}$ adalah penepatan maksimum dan minimum dari bobot inersia

3. Random: Bobot inersia didefinisikan persamaan 2.28.

$$\omega = 0{,}5 - \left(\frac{rnd(\ )}{2{,}0} \right) \quad (2.28)$$

rnd() adalah nomor acak dalam rentang 0 hingga 1.

Kemiripan persamaan bobot inersia juga terdapat pada penelitian Abdulhamit Subasi dengan penambahan variabel untuk mengalikan kecepatan partikel, tetapi dengan penyebutan variabel yang berbeda yaitu faktor penyeimbang (*λ*). Faktor penyeimbang dicari menggunakan persamaan 2.29 dan persamaan 2.30 (Subasi, 2013).

$$\lambda = \frac{2}{\left| 2 - \varphi - \sqrt{\varphi^2 - 4\varphi} \right|} \quad (2.29)$$

dimana $\varphi = c_1 + c_2$

Dengan persamaan kecepatan partikel yang sedikit berbeda yaitu :

$$v_j(i+1) = \lambda \left[v_j(i) + c_1 r_1 \left(p_{best} - x_j(i) \right) + c_2 r_2 \left(g_{best} - x_j(i) \right) \right] \quad (2.30)$$

2.5.3 Fungsi *Fitness*

Akurasi klasifikasi yang didapat dari pelatihan SVM adalah kriteria yang digunakan untuk merancang sebuah fungsi fitness. Dengan demikian, untuk partikel dengan akurasi klasifikasi tinggi dapat ditentukan dari nilai *fitness* yang tinggi. Untuk memecahkan masalah beberapa parameter sekaligus, dapat dengan menciptakan fungsi *fitness* tujuan tunggal yang menggabungkan contohnya empat tujuan menjadi satu. Partikel dengan nilai *fitness* tinggi memiliki probabilitas tinggi untuk efek posisi partikel lain dari iterasi berikutnya, sehingga harus didefinisikan secara tepat (Huang, 2008).

Untuk persamaan *fitness*, akurasi klasifikasi (*acc*) atau *hit rate* yang menunjukkan persentase yang diklasifikasikan dengan benar dapat dievaluasi oleh persamaan 2.24 (Subasi, 2013).

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \quad (2.31)$$

Keterangan :
TP : *true positives*
TN : *true negatives*
FP : *false positives*
FN : *false negatives*

2.5.4 PSO-SVM untuk klasifikasi

Algoritma PSO-SVM menggabungkan dua metode *machine learning* dengan mengoptimalkan parameter SVM menggunakan PSO. PSO dimulai dengan partikel dipilih secara acak dan mencari partikel optimal dengan iterasi. Setiap partikel memiliki kecepatan atau pergerakan dan merupakan calon solusi parameter terbaik. Klasifikasi SVM dibangun untuk setiap kandidat solusi parameter sehingga dapat diketahui evaluasi kinerja tiap solusi parameter melalui akurasi yang didapat pada SVM. Algoritma PSO memandu pemilihan parameter potensial yang menyebabkan akurasi prediksi terbaik dapat ditemukan dengan efisien. Algoritma ini menggunakan partikel yang paling baik untuk dijadikan acuan pada kandidat partikel generasi berikutnya. Dengan demikian, rata-rata masing-masing kandidat partikel dalam populasi memiliki *fitness* lebih baik dari pendahulunya. Proses ini berlanjut sampai kinerja SVM konvergen dan gerakan partikel mendekati nol (Engelbrecht, 2007).

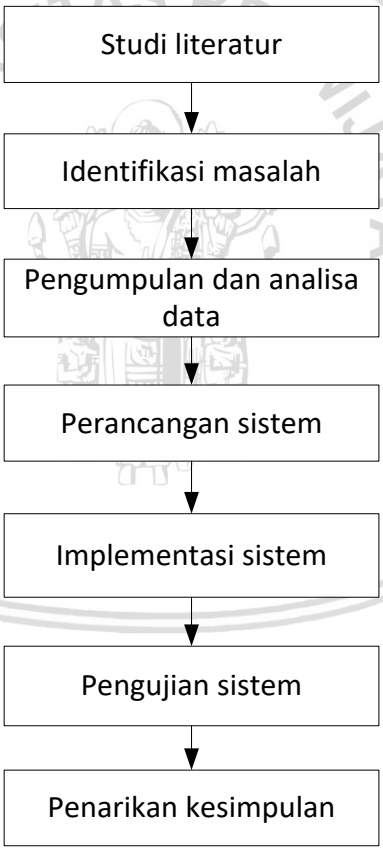
PSO digunakan untuk menemukan parameter optimal dengan menemukan parameter dengan akurasi terbaik saat mereka bergerak dalam *problem space* dari parameter SVM. PSO-SVM mengambil kelebihan dari resiko struktural minimal dari SVM dan kemampuan optimasi global secara cepat oleh PSO (Subasi, 2013).

BAB 3 METODOLOGI

Skripsi dengan judul klasifikasi akurasi *Support Vector Machine* (SVM) dengan *Particle Swarm Optimization* (PSO) untuk klasifikasi pendonor darah dengan RFMTC digolongkan sebagai penelitian yang bisa digunakan sebagai alat bantu dalam mengklasifikasikan pendonor darah yang bisa melakukan donor darah dan tidak bisa melakukan donor darah kembali.

3.1 Tahapan Penelitian

Bab metodologi penelitian membahas pentingnya metode penelitian yang berguna dalam mendukung peneliti agar menjelaskan tahapan penelitian secara umum. Metode yang dilakukan pada penelitian ini terdiri dari beberapa tahapan, yaitu, studi literature, identifikasi masalah, pengumpulan dan analisa data, perancangan sistem, implementasi sitem, pengujian sistem, dan penarikan kesimpulan. Tahapan dari penelitian tersebut diilustrasikan pada Gambar 3.1.



Gambar 3.1 Diagram Metodologi Penelitian

3.2 Studi Literatur

Studi literatur bertujuan untuk mempelajari konsep dasar teori yang diperoleh dari pustaka dan jurnal penelitian sebelumnya untuk menunjang penelitian yang berkaitan dengan akurasi *Support Vector Machine* (SVM) dengan

Particle Swarm Optimization (PSO) untuk klasifikasi pendonor darah dengan RFMTC. Penyusunan dasar teori dilakukan setelah mendapat beberapa referensi yang tepat untuk mendukung penelitian ini.

3.3 Identifikasi Masalah

Identifikasi masalah merupakan tahap awal dalam membuat suatu pengembangan sistem. Tahap ini merupakan tahapan untuk membatasi masalah yang akan di implementasikan dalam aplikasi, mengumpulkan berbagai kebutuhan yang diperlukan sistem dan mengumpulkan data yang berkaitan dengan aplikasi yang dibuat. Identifikasi ini merupakan tahapan awal untuk menentukan keberhasilan dari tahap selanjutnya. Permasalahan yang menjadi latar belakang penelitian ini adalah menentukan tingkat akurasi optimasi dari metode *Support Vector Machine* (SVM) dengan *Particle Swarm Optimization* (PSO) untuk klasifikasi pendonor darah. Data yang digunakan pada penelitian ini diambil dari *Blood Transfusion Service Center Data Set* UCI *repository, Hsin-Chu Taiwan*.

3.4 Pengumpulan dan Analisis Data

Pengumpulan data donor darah berasal dari *Blood Transfusion Service Center Data Set* UCI *repository, Hsin-Chu Taiwan*. Data yang terkumpul adalah data real yang akan diolah sehingga data siap untuk diproses dengan menerapkan metode yang digunakan. Penelitian ini menggunakan sebanyak 748 data calon pendonor darah dan berdasarkan batasan masalah yang telah ditentukan.

3.5 Perancangan Sistem

Perancangan sistem bertujuan merancang urutan tahapan yang nantinya akan diterapkan pada tahap implementasi aplikasi. Pada tahap perancangan sistem, perancangan aplikasi klasifikasi akurasi *Support Vector Machine* (SVM) dengan *Particle Swarm Optimization* (PSO) untuk klasifikasi pendonor darah dengan RFMTC. Kebutuhan yang digunakan dalam perancangan sistem ini adalah sebagai berikut:

1. Spesifikasi Kebutuhan Hardware :
   - Laptop Toshiba Satellite E205 core i5
   - RAM 8 GB
   - DDR 3
   - VGA 2 GB
2. Spesifikasi Kebutuhan Software :
   - Microsoft Windows 7 Ultimate 64-bit sebagai sistem operasi
   - PHP merupakan bahasa pemrograman untuk menjalankan aplikasi sistem ini
   - MySql sebagai sistem manajemen database

- *Enterprise Architecture* sebagai aplikasi untuk membuat diagram

Kebutuhan data donor darah yang diperoleh dari *Blood Transfusion Service Center Data Set* UCI *repository, Hsin-Chu Taiwan*.

3.6 Implementasi Sistem

Implementasi merupakan tindakan lanjutan atau penerapan dari sebuah perencanaan yang disusun sebelumnya. Pada tahap implementasi diterapkan komponen-komponen yang berkaitan dengan implementasi *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* untuk klasifikasi pendonor darah.

3.7 Pengujian Sistem

Setelah tahap implementasi maka akan dilakukan tahap uji coba atau pengujian sistem. Data yang digunakan dalam pengujian merupakan data yang sudah tersedia. Tahap pengujian sistem dilakukan untuk mengetahui seberapa baik suatu sistem dan untuk memperoleh nilai akurasi yang baik.

3.8 Penarikan Kesimpulan

Penarikan kesimpulan dapat dilakukan setelah semua tahapan sebelumnya dilakukan, yang meliputi perancangan, implementasi, dan pengujian sistem. Kesimpulan diambil dari hasil pengujian beserta analisis terhadap sistem yang telah dibangun. Kemudian tahapan akhir yang dilakukan adalah penulisan saran. Saran dimaksudkan agar penelitian berikutnya mampu memperbaiki kekurangan yang terjadi, menyempurnakan penulisan, serta memberikan pertimbangan untuk pengembangan sistem selanjutnya.

BAB 4 PERANCANGAN

Pada bab membahas beberapa hal, yaitu perancangan antarmuka pengguna, perancangan algoritme, perhitungan manualisasi dan yang terakhir adalah perancangan uji coba dan evaluasi.

4.1 Identifikasi Permasalahan

Penelitian ini permasalahan yang harus dicari adalah memprediksi bisa atau tidaknya seseorang untuk dapat melakukan donor darah kembali sesuai dengan *Recency, Frequency, Monetary, Time* dan *Churn Probability* (*Class*) . Data masukan dari pasien nantinya akan diproses dalam bentuk *sequential learning* pada SVM dengan menggunakan parameter yang dioptimasi sebelumnya dengan menggunakan PSO. Data yang digunakan sebanyak 748 data pendonor darah sebagaimana ditunjukkan pada Tabel 4.1 (sampel data yang lengkap ditunjukkan pada lampiran)

Tabel 4.1 Sampel Data

No	Recency	Frequency	Monetary litre	Time	C
15	2	6	1.5	15	1
16	2	5	1.25	11	1
17	2	14	3.5	48	1
18	2	15	3.75	49	1
20	2	3	0.75	4	1
5	1	24	6	77	0
6	4	4	1	4	0
8	1	12	3	35	0
11	4	23	5.75	58	0
14	1	13	3.25	47	0

Prediksi pendonor darah ini menggunakan 5 fitur RFMTC yaitu *Recency, Frequency, Monetary, Time* dan *Churn Probability* (*Class*). Adapun penjelasan dari RFMTC sudah dijelaskan sebelumnya pada sub bab 2.4.

Parameter yang akan digunakan pada proses SVM dan PSO ini meliputi : *alpha* (α), *complexity* (C), *gamma* (γ), *lambda* (λ), perhitungan *kernel*, perhitungan matriks, nilai *error*, perhitungan *delta alpha*, perhitungan *alpha*, dan nilai bias. Semua perhitungan tersebut diperoleh sebelumnya dari proses pengoptimasian 4 parameter utama yaitu *lambda* (*λ*), *gamma* (*γ*), *complexity* (*C*) dan varian (*σ*) pada PSO dengan menguji nilai kecepatan partikel awal dengan kecepatan partikel yang sudah dihitung dengan PSO berdasarkan range nilai masing-masing parameter

tersebut. Pada fase menentukan parameter PSO kecepatan pertama diinisialisasi dengan kecepatan awalnya dalam keadaan tetap. Kemudian fase selanjutnya adalah mendapatkan nilai P_{best} yang didapatkan dari proses perhitungan SVM dan setelahnya ditentukan nilai $G_{best.}$ Nilai G_{best} sendiri ini adalah nilai terbaik yang didapatkan dari nilai-nilai P_{best} sebelumnya.

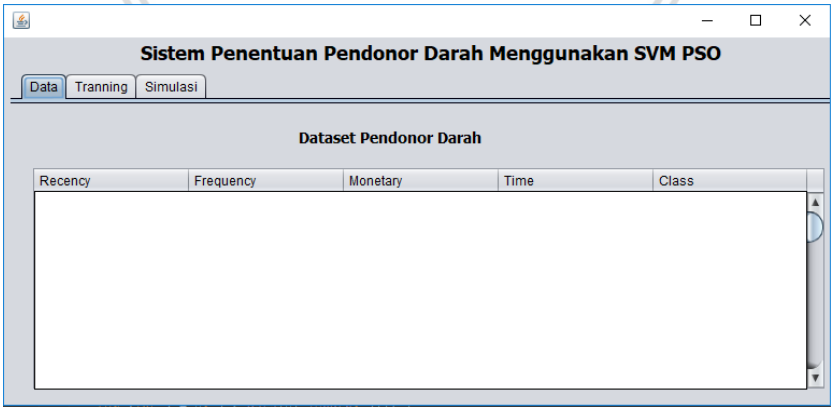
Fase selanjutnya adalah ketika melakukan tahapan *update* pada nilai parameter dibutuhkan nilai terbaik dari kumpulan nilai P_{best} atau bisa disebut dengan nilai G_{best}. Di fase *update* ini akan dilakukan penambahan nilai kecepatan partikel awal dengan nilai kecepatan partikel setelah dilakukannya perhitungan terhadap nilai P_{best} dan G_{best} untuk mendapatkan nilai posisi partikel selanjutnya. Nilai posisi partikel disini adalah nilai parameter yang sudah dioptimasi dan selanjutnya untuk dihitung pada perhitungan SVM. Perhitungan dilakukan sampai pada iterasi maksimumnya yaitu 4 kali iterasi untuk mendapatkan nilai posisi partikel yang terbaik.

4.2 Perancangan Antarmuka

Antarmuka sistem ini terdiri dari tiga halaman tab. Halaman tab pertama adalah halaman daftar data, data latih dan data uji yang digunakan dalam sistem. Halaman tab kedua adalah halaman masukan parameter algoritma *Particle Swarm Optimization* (PSO) dan parameter algoritma *Support Vector Machine* (SVM) serta hasil optimasi yang dihasilkan sistem. Halaman tab ketiga adalah hasil klasifikasi dari data uji dan *form* untuk memeriksa kondisi pasien berupa klasifikasi status resiko menurut sistem sesuai parameter pada halaman tab kedua.

4.2.1 Rancangan Halaman Data

Rancangan halaman data adalah halaman yang berisi keseluruhan data maupun pembagian data latih dan data uji yang digunakan pada sistem ini. Pada halaman ini menampilkan table dataset yang telah dimuat untuk proses SVM dan PSO. Rancangan halaman data ditampilkan pada Gambar 4.1.



Gambar 4.1 Halaman data

4.2.2 Rancangan Halaman Parameter dan Hasil Akurasi

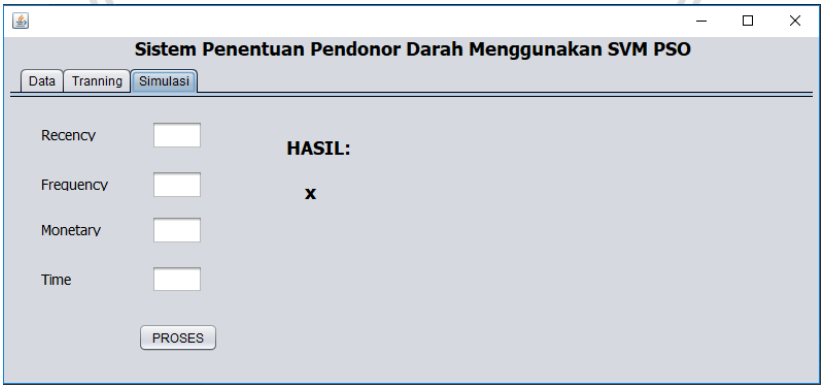
Rancangan halaman parameter merupakan halaman untuk memasukkan parameter range SVM dan parameter PSO. Pada halaman batasan range SVM ini akan menampilkan bagan untuk memasukkan nilai *lamda, gamma,* konstanta dan *Varian* berdasarkan range yang telah ditentukan sebelumnya pada proses PSO.

Gambar 4.2 Halaman Parameter dan Hasil Akurasi

Rancangan halaman parameter dan hasil akurasi adalah halaman yang berisi form untuk memasukkan nilai-nilai parameter algoritma dan menampilkan hasil akurasi dan hasil optimasi pada tiap iterasi algoritma.

4.2.3 Rancangan Halaman Hasil Klasifikasi dan *Form* Klasifikasi

Rancangan halaman hasil klasifikasi dan *form* klasifikasi adalah halaman yang berisi tabel yang menampilkan hasil klasifikasi dari data uji menurut sistem dan sesuai data laboratorium pada sisi kiri. Sedangkan sisi kanan adalah *form* untuk seseorang dapat memeriksa status pendonor darah dengan memasukkan data yang diperlukan. Rancangan halaman hasil klasifikasi dan *form* klasifikasi ditampilkan pada Gambar 4.3.



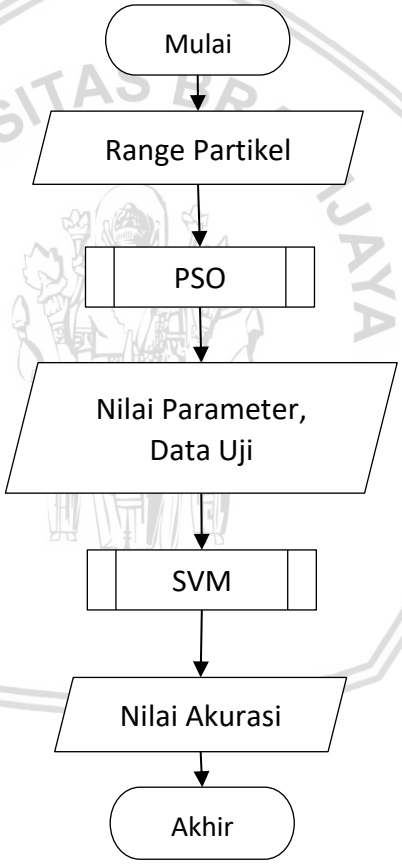
Gambar 4.3 Halaman Tabel Proses SVM-PSO

4.3 Perancangan Algoritme

Diagram alir (*flowchart*) pada algoritme untuk perhitungan menggunakan *Support Vector Machine* (SVM) dan *Particle Swarm Optimization* (PSO) merupakan penyelesaian masalah secara bertahap melalui fase-fase yang dapat memprediksi hasil perhitungan pada penelitian ini.

4.3.1 Fase *Particle Swarm Optimization* (PSO) dan *Support Vector Machine* (SVM)

Dalam studi kasus ini perhitungan dilakukan dengan melakukan pengoptimalan parameter SVM agar dalam perhitungannya nanti diperoleh nilai yang akurasinya tinggi. Optimasi nilai parameter dilakukan dengan metode PSO. Berikut adalah langkah-langkah dari metode PSO dan SVM ditunjukkan pada Gambar 4.4.



Gambar 4.4 PSO-SVM

Proses diagram alir PSO – SVM :

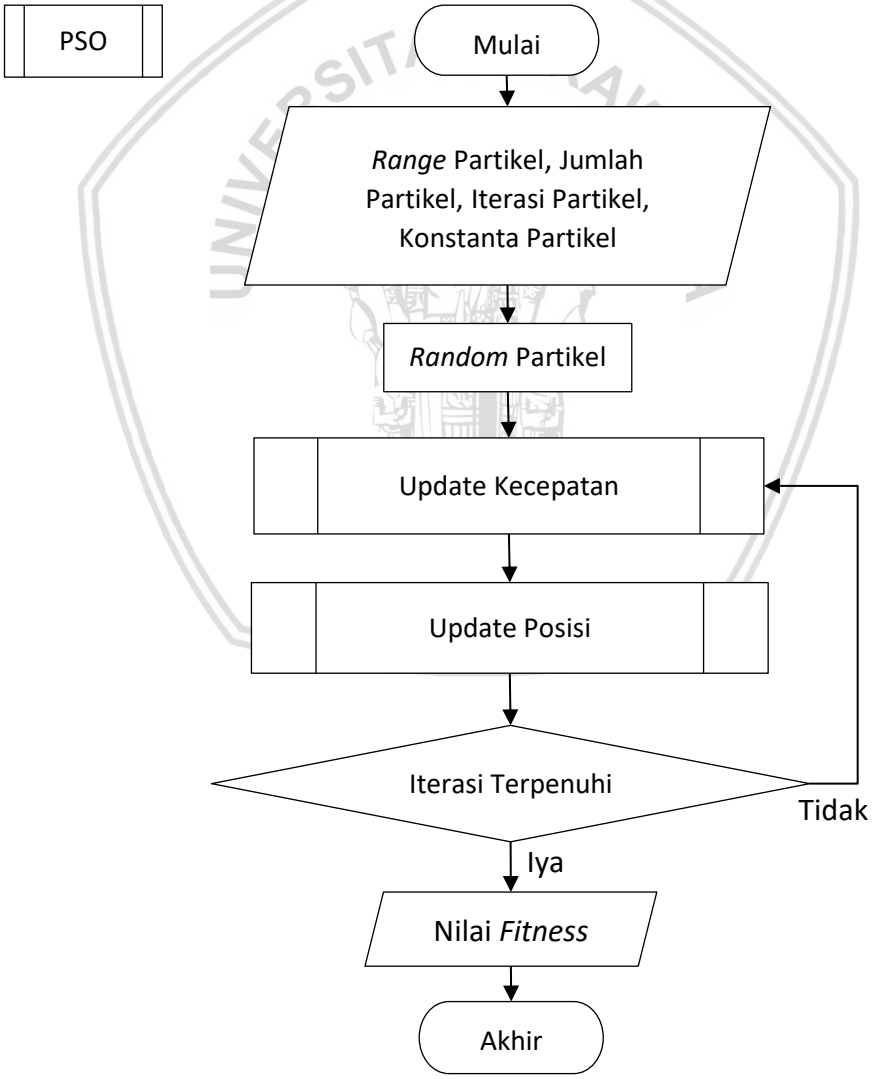
1. Sistem membutuhkan nilai range pada partikel untuk melakukan perhitungan PSO.
2. Melakukan proses perhitungan PSO untuk mendapatkan nilai parameter pada perhitungan selanjutnya di SVM.

3. Proses penentuan nilai parameter dari perhitungan PSO dapat digunakan sebagai inputan data *training* untuk perhitungan SVM.
4. Sistem menghasilkan nilai akurasi dari perhitungan SVM yang optimal berdasarkan parameter yang optimal setelah dihitung sebelumnya dengan PSO.

4.3.2 Fase *Particle Swarm Optimization* (PSO)

Pada fase PSO disini adalah mengoptimalkan nilai parameter agar mendapatkan nilai akurasi yang baik. Optimasi yang dilakukan dengan metode PSO ini agar dapat memperbaiki nilai parameter berulang-ulang hingga iterasi yang diinginkan. Tujuan utama agar dari studi kasus ini agar mendapat nilai akurasi yang terbaik.

Berikut adalah langkah-langkah dari pengoptimasian parameter dengan metode PSO ditunjukkan pada Gambar 4.5.



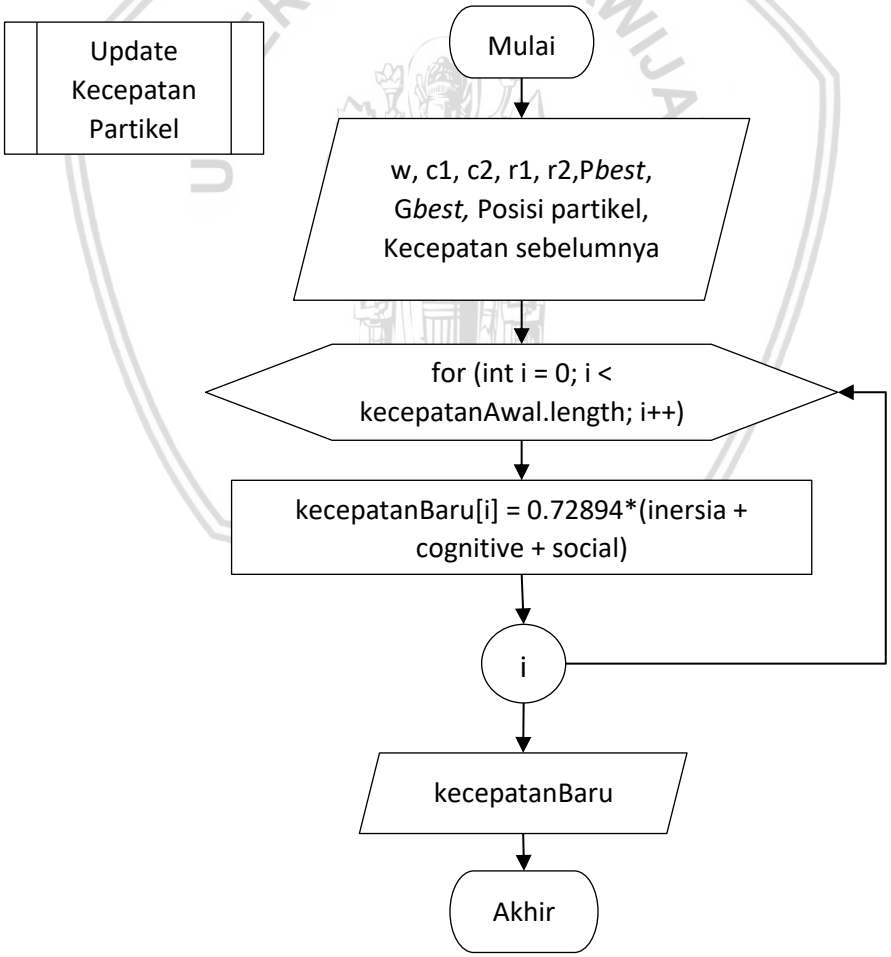
Gambar 4.5 Diagram Alir *(Flowchart)* PSO

Proses diagram alir dari PSO adalah sebagai berikut :

1. Sistem membutuhkan nilai pada partikel yang dibutuhkan PSO yakni, *range* partikel, jumlah partikel, iterasi dan konstanta pada partikel.
2. Melakukan proses inisialisasi nilai pada sebanyak *range* nilai parameter secara *random.*
3. Menghitung nilai *fitness* dengan menggunakan rumus nilai *fitness* pada persamaan 2.31.
4. Melakukan *update* kecepatan partikel, *update* posisi partikel untuk mendapatkan nilai parameter terbaru selama iterasi belum terpenuhi.
5. Sistem menghasilkan nilai parameter yang optimal untuk klasifikasi perhitungan SVM klasifikasi pendonor darah.

4.3.2.2 *Update* Kecepatan Partikel

Update kecepatan partikel dilakukan untuk mendapatkan hasil yang optimal dalam melakukan perhitungan *update* posisi partikel selanjutnya. Berikut diagram alir *update* kecepatan partikel ditunjukkan pada Gambar 4.6.



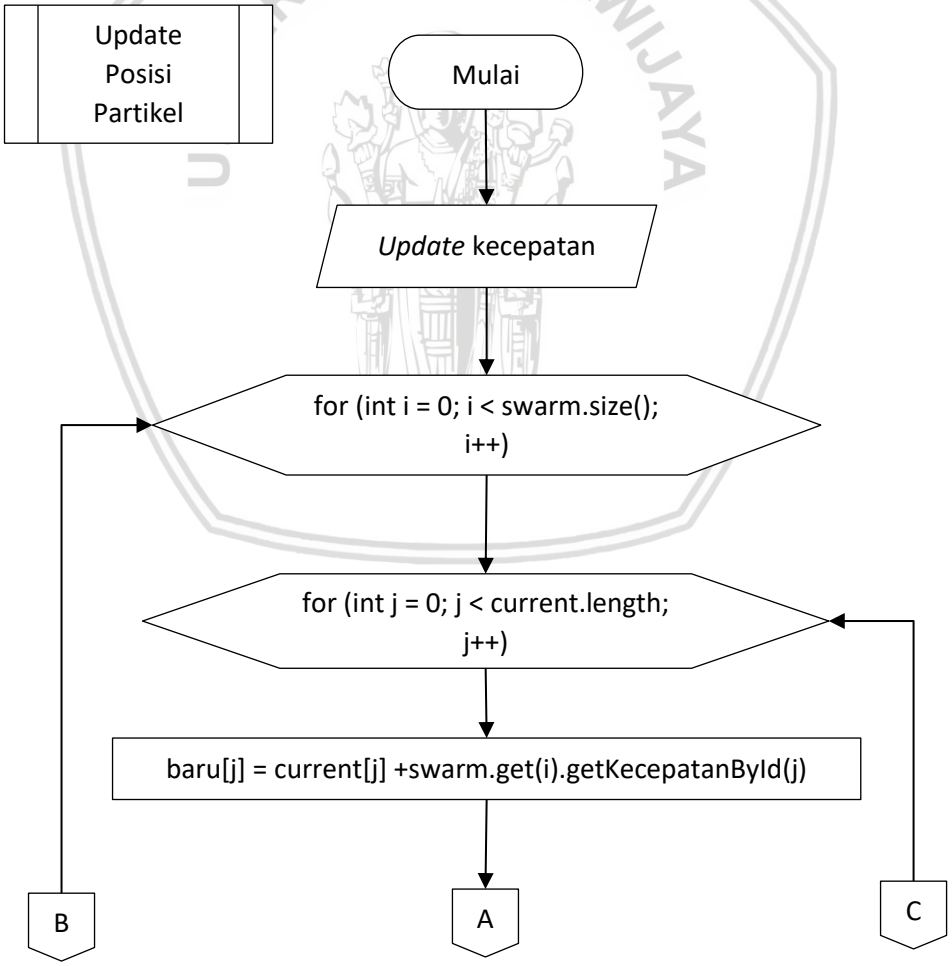
Gambar 4.6 Diagram Alir *(Flowchart) Update* Kecepatan Partikel

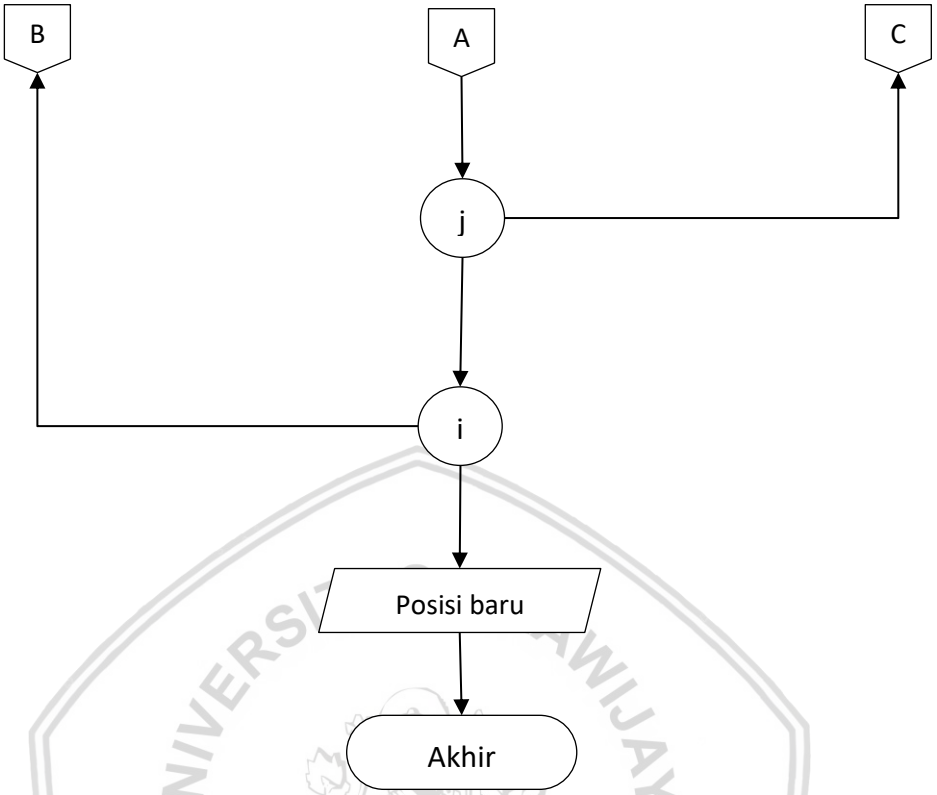
Proses diagram alir dari *update* kecepatan partikel adalah sebagai berikut :

1. Sistem membutuhkan nilai bobot inersia, konstanta, bilangan acak, kecepatan pada iterasi sebelumnya, posisi P_{best} dan G_{best} partikel.
2. Sistem melakukan kondisi perulangan sebanyak jumlah panjang data pada kecepatan awal untuk dibutuhkan dalam menghitung kecepatan baru.
3. Proses mengubah kecepatan partikel dengan menggunakan perhitungan kecepatan partikel yang ditunjukkan pada persamaan 2.24.
4. Mendapatkan nilai kecepatan partikel baru yang siap dipindahkan pada proses perhitungan posisi partikel baru.

4.3.2.3 *Update* Posisi Partikel

Diagram alir (*flowchart*) untuk proses perhitungan *update* posisi partikel ditunjukkan pada Gambar 4.7.





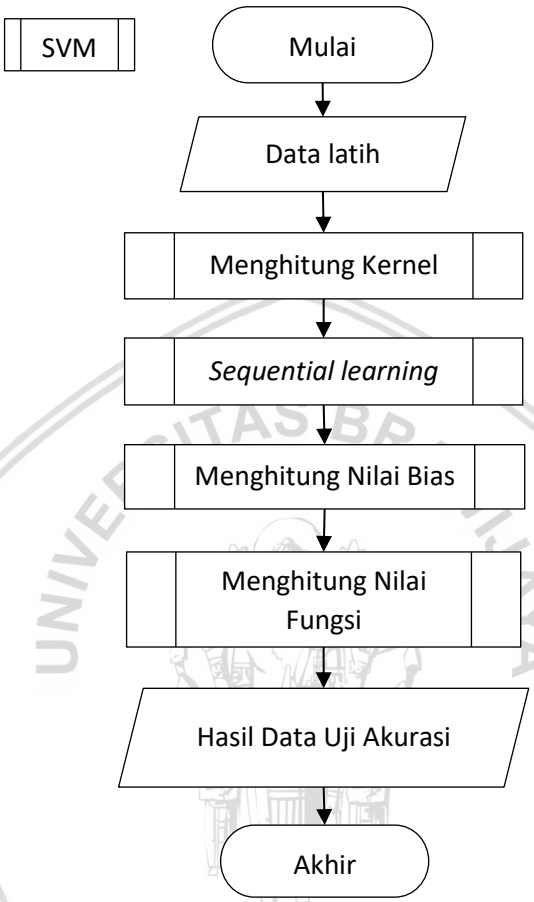
Gambar 4.7 Diagram Alir *(Flowchart) Update* Posisi Partikel

Proses diagram alir dari *update* posisi partikel adalah sebagai berikut :

1. Sistem membutuhkan hasil nilai dari *update* kecepatan sebelumnya yang sudah dilakukan pada proses perhitungan *update* kecepatan partikel.
2. Sistem kemudian melakukan kondisi perulangan sebanyak panjang partikel untuk menjadi masukan pada proses perhitungan *update* posisi partikel.
3. Sistem melakukan kondisi perulangan lagi sebanyak panjang *update* posisi partikel untuk menjadi masukan agar dapat memperbarui setiap proses perhitungan *update* posisi partikel.
4. Kemudian sistem melakukan proses perhitungan *update* posisi partikel yang ditunjukkan pada persamaan 2.23.
5. Sistem memperoleh keluaran berupa posisi baru tiap-tiap partikel.

4.3.3 Fase *Support Vector Machine* (SVM)

Diagram alir (*flowchart*) untuk perhitungan menggunakan metode klasifikasi SVM berupa perhitungan kernel, sequential learning, menghitung bias, dan menghitung f(x) ditunjukkan pada Gambar 4.8.



Gambar 4.8 Diagram Alir *(Flowchart)* SVM

Proses klasifikasi status pendonor darah dengan SVM adalah sebagai berikut :

1. Sistem membutuhkan nilai data latih untuk melakukan perhitungan pada kernel SVM.
2. Nilai data latih yang dibutuhkan digunakan untuk menghitung nilai kernel berbentuk perhitungan matriks.
3. Melakukan proses perhitungan nilai *sequential learning,* dalam melakukan proses *sequential learning* terdapat beberapa langkah proses lagi di dalamnya.

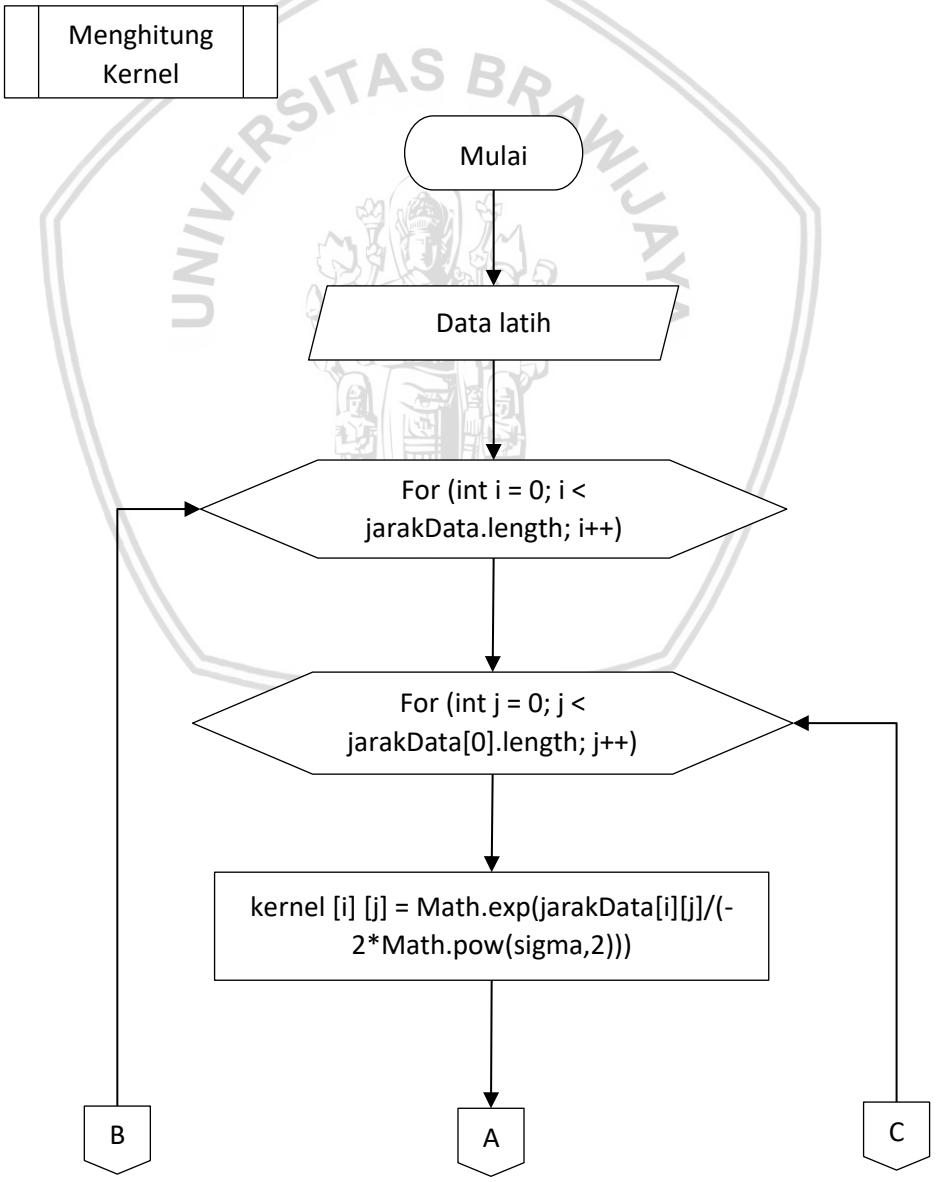
   Berikut proses yang dilakukan dalam perhitungan *sequential learning* :

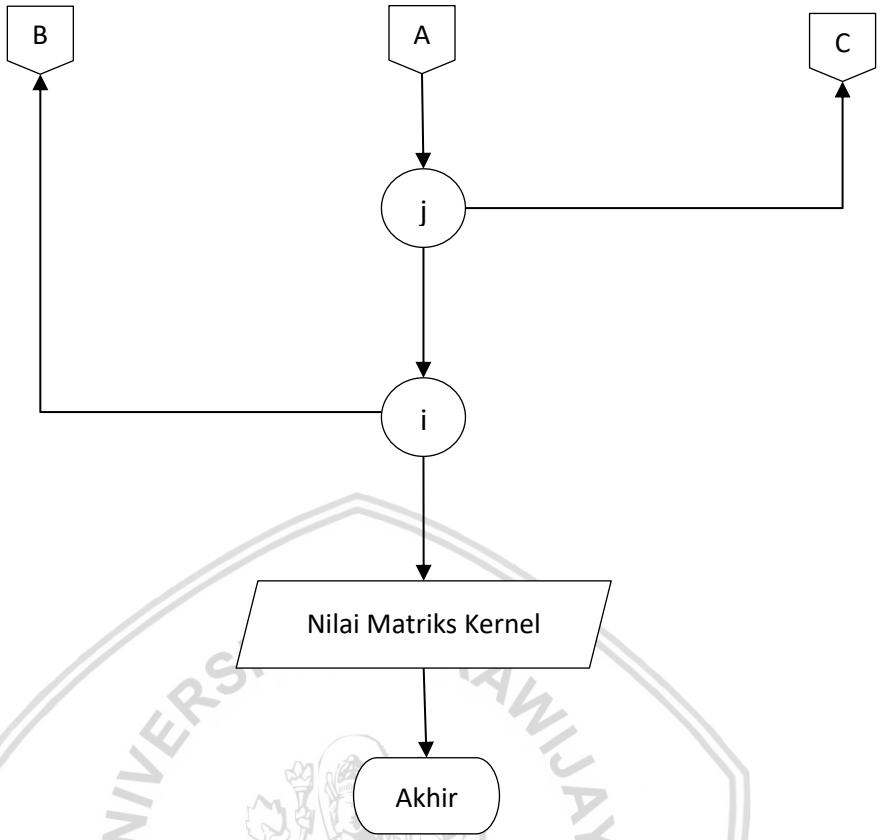
   a. Menghitung nilai matriks *hessian*
   b. Menghitung nilai *error*
   c. Menghitung nilai *delta alpa*
   d. Menghitung nilai *alpa*

4. Sistem kemudian melakukan perhitungan nilai bias yang digunakan untuk menghitung nilai dari fungsi SVM setelah mendapat nilai α terbaru dari tiap data pada proses sebelumnya.
5. Melakukan proses menghitung nilai fungsi *f(x)* setelah mendapatkan nilai bias dari langkah sebelumnya. Proses dilakukan dengan menghitung data *testing* dengan menjumlahkan nilai bobot data *testing* dengan nilai bias yang dihasilkan dari hasil perhitungan data *training* sebelumnya.
6. Mendapatkan hasil data uji akurasi untuk digunakan sebagai nilai *fitness* pada proses PSO.

4.3.3.2 Menghitung Kernel

Diagram alir *(flowchart)* untuk menghitung nilai kernel dengan perhitungan matriks ditunjukkan pada Gambar 4.9.





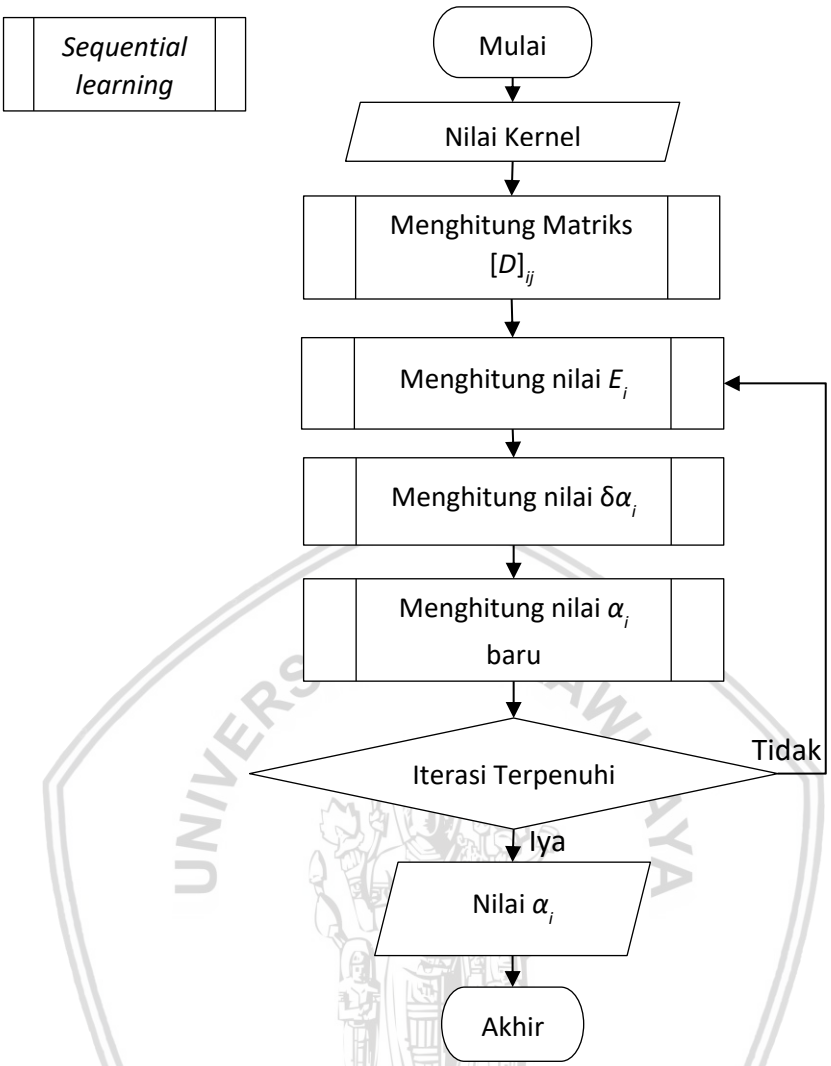
Gambar 4.9 Diagram Alir *(Flowchart)* Menghitung Kernel

Proses menghitung nilai kernel pada SVM ditunjukkan sebagai berikut :

1. Sistem membutuhkan data latih sebagai perhitungan matriks kernel sesuai dengan banyaknya data *training* yang akan digunakan yaitu sebanyak 10 data *training*.
2. Menginputkan pengulangan sebanyak panjang data *training* yang yang digunakan untuk melakukan perhitungan kernel *gaussian* RBF.
3. Memulai proses perhitungan kernel dengan menggunakan rumus perhitungan kernel *gaussian RBF* dengan persamaan 2.21.
4. Proses perhitungan dilakukan sampai dengan banyaknya data latih yang digunakan dalam melakukan perhitungan kernel *gaussian* RBF.
5. Sistem mendapatkan hasil nilai perhitungan matriks kernel *gaussian* RBF.

4.3.3.3 Fase *Sequential learning*

Pada fase algoritme *sequential* sesuai Vijayakumar adalah proses perhitungan iterasi pelatihan pada SVM. Berikut adalah langkah-langkah dari *sequential learning* ditunjukkan pada Gambar 4.10.



Gambar 4.10 Diagram Alir *(Flowchart) Sequential Learning*

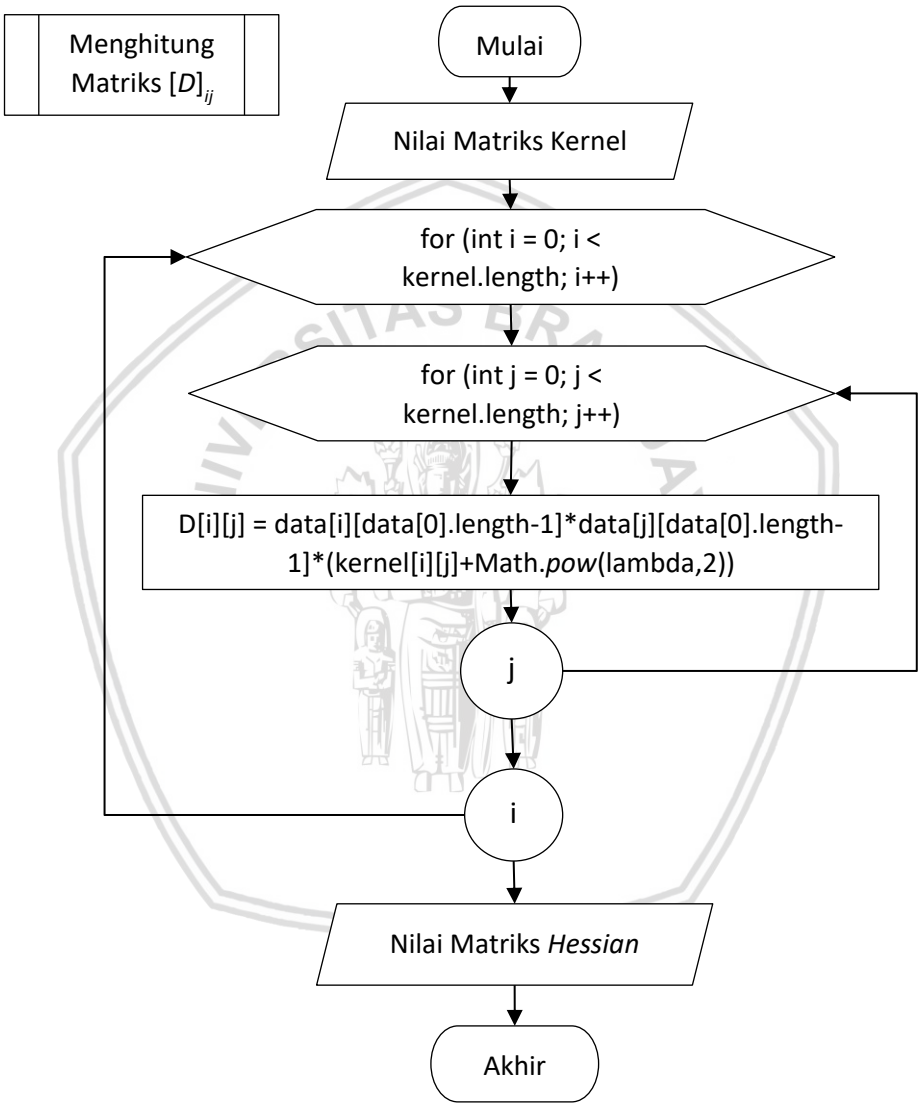
Proses diagram alir dari *sequential learning* adalah sebagai berikut :

1. Sistem membutuhkan nilai kernel pada perhitungan sebelumnya untuk dapat melakukan perhitungan *sequential learning.*
2. Sistem kemudian melakukan perhitungan matriks *hessian* $[D]_{ij}$ dengan menggunakan nilai *lambda* (λ) dan nilai kernel sebelumnya untuk dapat melakukan perhitungan nilai *error.*
3. Melakukan proses perhitungan nilai *error* dengan menggunakan nilai kernel dan nilai *alpha* 0 sebagai parameter utama dengan melakukan sebanyak 3 kali iterasi perhitungan. Iterasi dilakukan agar mendapatkan nilai titik yang paling optimal untuk dapat memaksimalkan hasil nilai akurasi.
4. Setelah mendapat nilai *error* sistem kemudian melakukan perhitungan *delta alpa* dengan menggunakan nilai *error, gamma* (γ) dan nilai *complexity* (C) untuk dapat membatasi nilai *alpa baru* (α) agar selalu bernilai ≥ 0, dan membatasi nilai α baru agar selalu bernilai $\leq C$.

5. Jika iterasi tidak terpenuhi maka sistem akan kembali melakukan iterasi sampai pada titik yang optimal dan kembali ke tahap perhitungan nilai *error, delta alpha,* dan *alpha* baru ketika belum mencapai titik nilai *alpha* baru yang optimal.

Menghitung Matriks *Hessian* $[D]_{ij}$

Diagram alir (*flowchart*) untuk menghitung nilai matriks *hessian* $[D]_{ij}$ ditunjukkan pada Gambar 4.11.



Gambar 4.11 Diagram Alir *(Flowchart) Matriks Hessian*

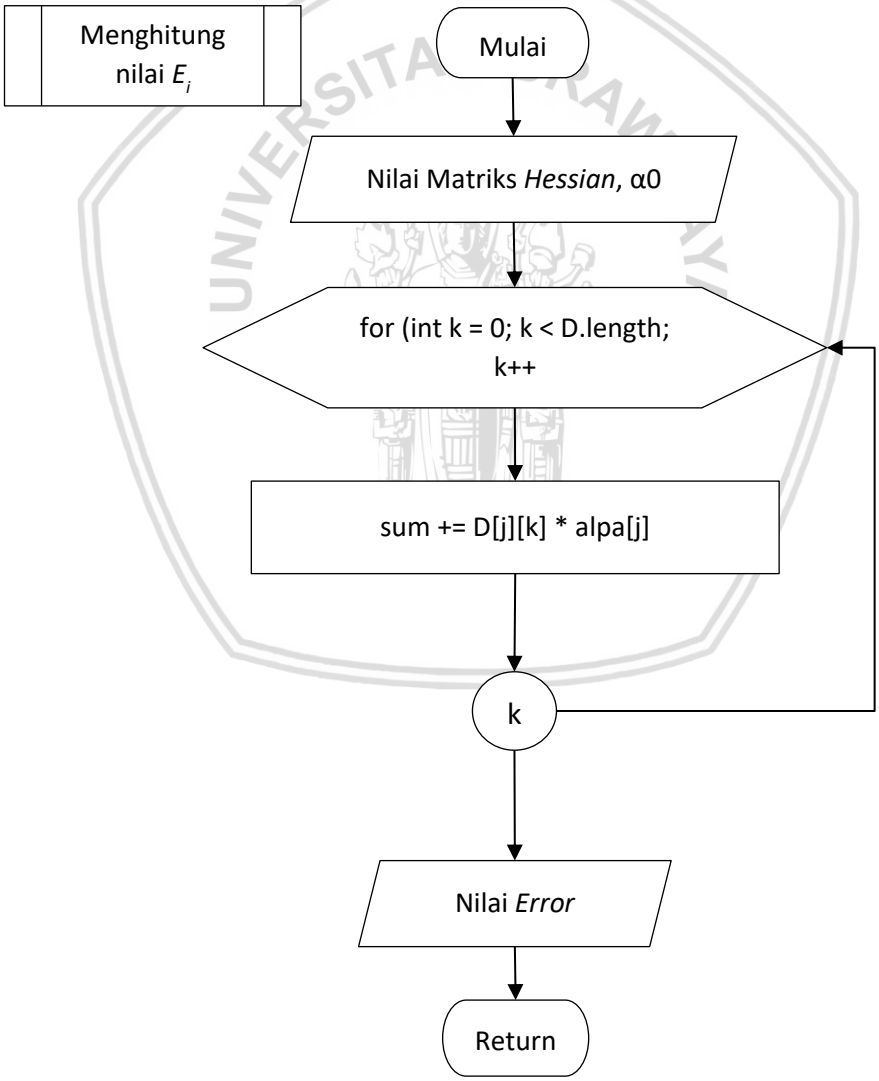
Proses diagram alir (*flowchart*) menghitung matriks *Hessian* adalah sebagai berikut :

1. Sistem membutuhkan nilai matriks kernel *gaussian* RBF dan nilai *lambda* (λ) untuk dapat melakukan perhitungan matriks *hessian* $[D]_{ij}$.

2. Melakukan proses perhitungan matriks *hessian* dengan menginputkan nilai kernel *gaussian* RBF sebelumnya dan nilai *lambda* yang diperoleh sebelumnya dari proses perhitungan range parameter untuk partikel PSO.
3. Memulai proses perhitungan dengan rumus perhitungan matriks *hessian* $D_{[i][j]}$ pada persamaan 2.15.
4. Sistem menghasilkan nilai matriks *hessian* $D_{[i][j]}$ yang dapat digunakan selanjutnya dalam melakukan perhitungan nilai *error.*

Menghitung Nilai *Error*

Diagram alir (*flowchart*) untuk menghitung nilai *error* ditunjukkan pada Gambar 4.12.



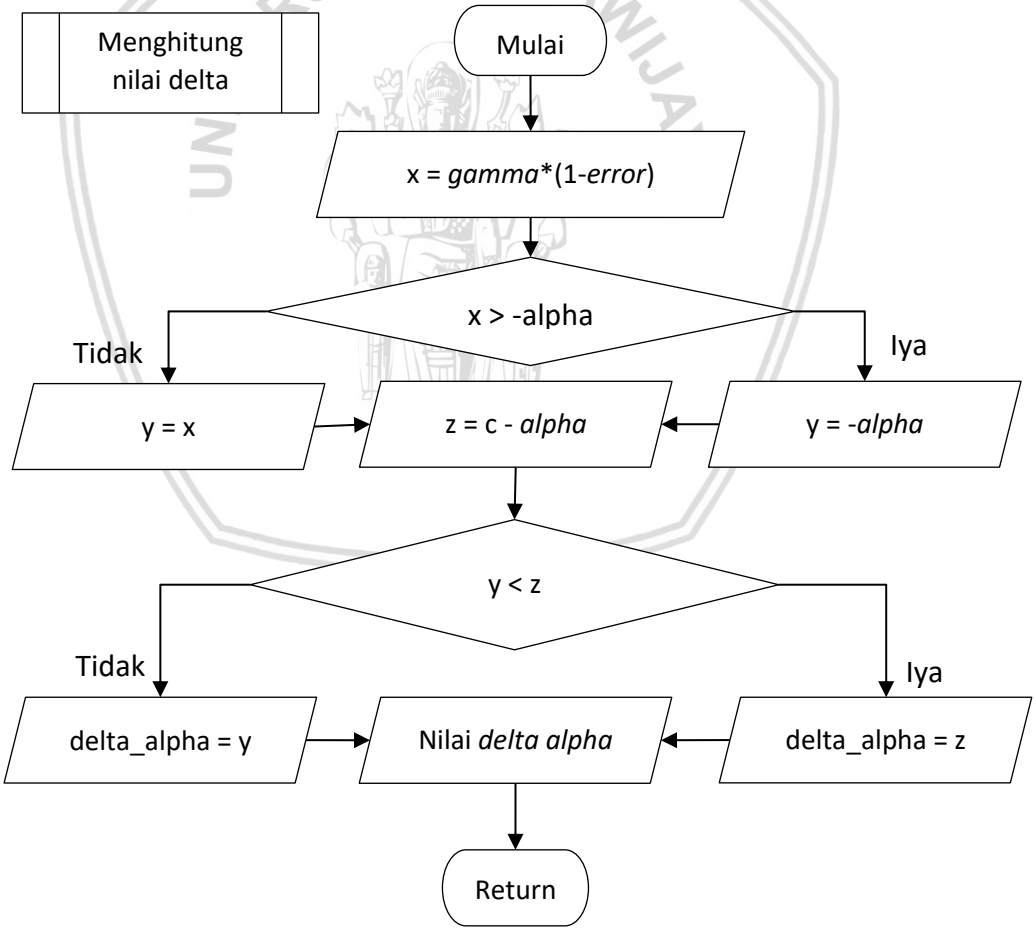
Gambar 4.12 Diagram Alir *(Flowchart) Nilai Error*

Proses diagram alir (*flowchart*) menghitung nilai *error* adalah sebagai berikut :

1. Sistem membutuhkan nilai data matriks *hessian* $D_{[i][j]}$ untuk dapat melakukan perhitungan nilai *error.*
2. Melakukan inisialiasi sepanjang jumlah data matriks *hessian* $D_{[i][j]}$ dan perulangan sesuai banyaknya panjang jumlah data matriks *hessian*.
3. Melakukan proses perhitungan nilai *error* dengan rumus perhitungan nilai *error* yang ditunjukkan pada persamaan 2.16.
4. Sistem menghasilkan nilai *error* yang digunakan selanjutnya untuk melakukan proses perhitungan nilai *delta alpha.*

Menghitung Nilai *Delta Alpha*

Diagram alir (*flowchart*) untuk menghitung nilai *delta alpha* ditunjukkan pada Gambar 4.13.



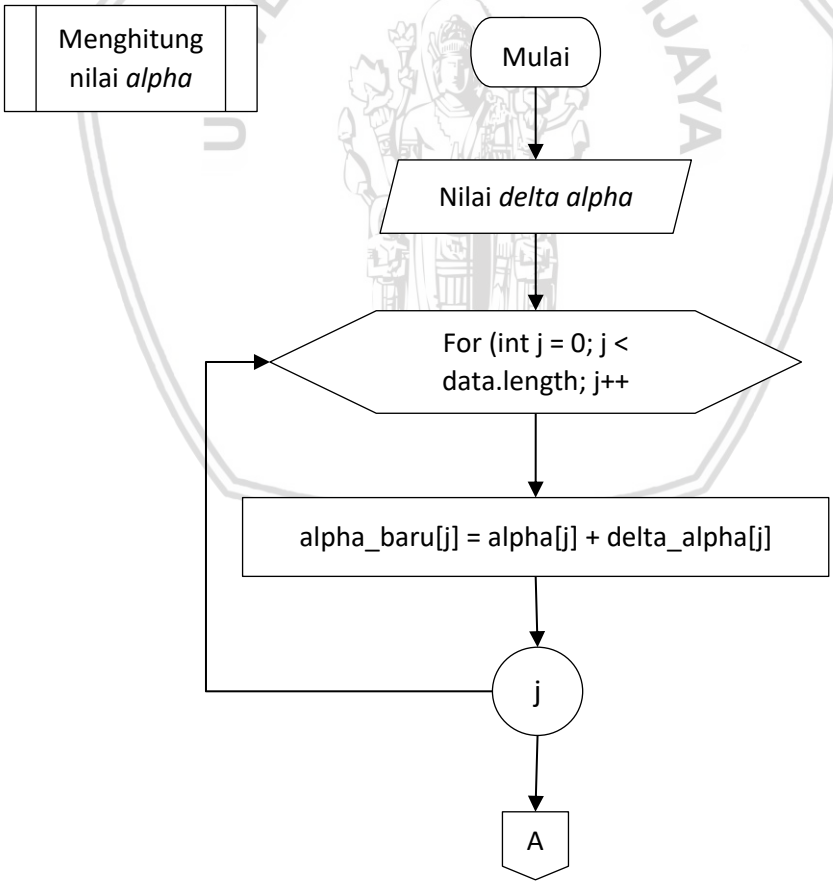
Gambar 4.13 Diagram Alir *(Flowchart) Delta Alpha*

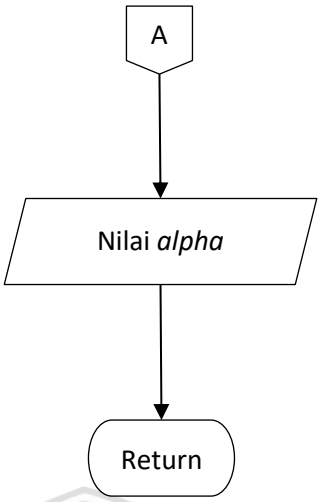
Proses diagram alir (*flowchart*) untuk menghitung *delta alpha* adalah sebagai berikut :

1. Sistem membutuhkan nilai *error* pada proses sebelumnya untuk dapat melakukan proses perhitungan *delta alpha.*
2. Melakukan inisialisasi sebanyak jumlah data yang akan dihitung.
3. Melakukan proses perhitungan nilai *delta alpha* dengan persamaan 2.17.
4. Melakukan proses pengulangan sampai pada titik nilai yang diperlukan untuk mendapatkan *delta alpha*.
5. Sistem menghasilkan nilai *delta alpha* yang digunakan selanjutnya untuk menghitung nilai *alpha* baru.

Menghitung Nilai *Alpha* Baru

Diagram alir (*flowchart*) untuk menghitung nilai *alpha* baru ditunjukkan pada Gambar 4.14.





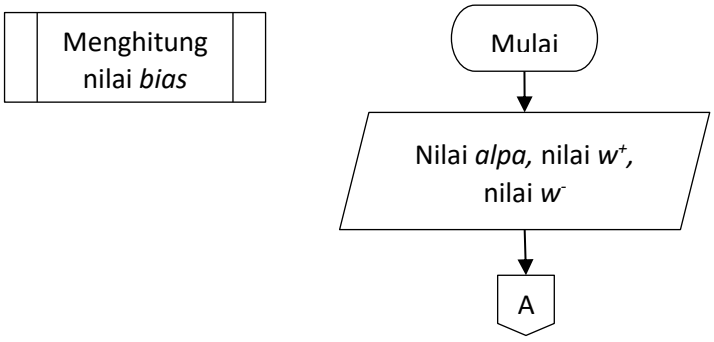
Gambar 4.14 Diagram Alir *(Flowchart) Alpha* Baru

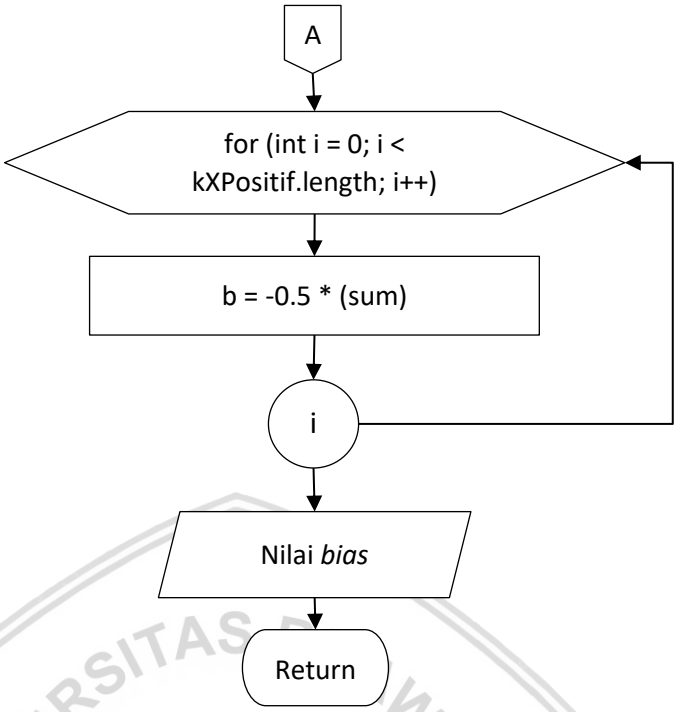
Proses diagram alir (*flowchart*) untuk menghitung nilai *alpha* baru adalah sebagai berikut :

1. Sistem mebutuhkan nilai *delta alpha* untuk dapat melakukan perhitungan mencari nilai *alpha* baru.
2. Sistem melakukan inisialisasi sebanyak jumlah data yang akan dihitung.
3. Melakukan proses perhitungan nilai *alpha* baru dengan rumus persamaan 2.18.
4. Melakukan proses pengulangan sampai pada nilai *alpha* yang optimal.
5. Mendapatkan nilai *alpha* yang optimal yang selanjutnya dapat digunakan untuk mencari nilai bias.

4.3.3.4 Menghitung Nilai *Bias*

Diagram alir (*flowchart*) untuk menghitung nilai *bias* ditunjukkan pada Gambar 4.15.





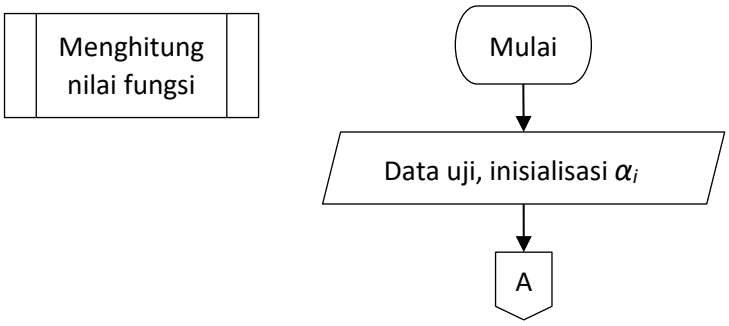
Gambar 4.15 Diagram Alir *(Flowchart)* Nilai *Bias*

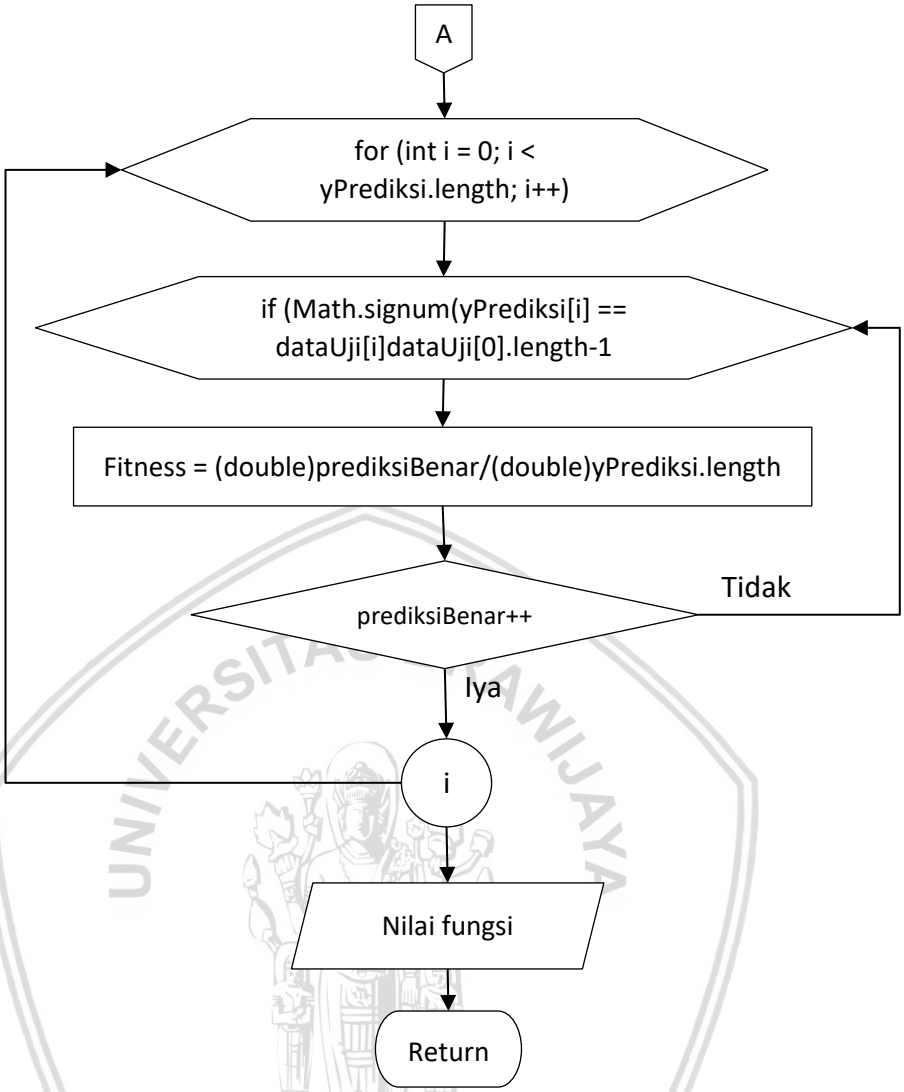
Proses diagram alir (*flowchart*) untuk menghitung nilai *bias* adalah sebagai berikut :

1. Sistem membutuhkan nilai *alpha* baru nilai w^+ dan w^- untuk dapat melakukan perhitungan nilai *bias.*
2. Sistem selanjutnya melakukan inisialisasi banyaknya jumlah data yang dihitung.
3. Melakukan proses perhitungan nilai *bias* dengan rumus persamaan 2.13.
4. Mendapatkan nilai *bias* yang selanjutnya dapat digunakan dalam mencari perhitungan nilai fungsi.

4.3.3.5 Menghitung Nilai Fungsi

Diagram alir (*flowchart*) untuk menghitung nilai fungsi ditunjukkan pada Gambar 4.16.





Gambar 4.16 Diagram Alir *(Flowchart)* Nilai Fungsi

Proses diagram alir (*flowchart*) untuk menghitung nilai fungsi adalah sebagai berikut :

1. Sistem membutuhkan data uji terpilih dan hasil insialisasi dari nilai α_i beserta data hasil uji kernel.
2. Melakukan inisialisasi data sebanyak data yang akan diuji.
3. Melakukan proses perhitungan nilai fungsi dengan rumus fungsi dengan persamaan 2.12.
4. Sistem mendapatkan nilai fungsi yang optimal.

4.4 Perhitungan Manualisasi

Pada tahap perhitungan manualisasi yang akan dilakukan dalam studi kasus pendonor darah ini bertujuan untuk mengoptimalkan nilai parameter yang dipakai dalam mendapatkan nilai akurasi yang baik untuk pengklasifikasian pendonor darah. Perhitungan manulaisasi akan mendapatkan hasil nilai parameter terbaik untuk klasifikasi pendonor darah dengan dataset (*Recency, Frequency, Monetary, Time, Churn probability* (Class) RFMTC.

4.4.1 Normalisasi Data

Pada tahap normalisasi data ini akan dilakukan suatu proses normalisasi terlebih dahulu dengan menggunakan rumus normalisasi *Min-Max*. Normalisasi data dibutuhkan agar *range* data atau rentang data pada suatu fitur yang ada pada studi kasus ini tidak terlalu jauh. Pada tahap manualisasi ini akan menormalisasi data *training* berjumlah 10 data. berikut ini adalah proses perhitungan normalisasi data dengan menggunakan rumus normalisasi *Min-Max* :

Xbaru = (((Xlama – Minlama)/(Maxlama – Minlama)) * (Maxbaru – Minbaru)) + Minbaru

Contoh perhitungan normalisasi untuk fitur-fitur dapat diterangkan dalam menghitung fitur *Recency, Frequency, Monetary, Time* sebagai berikut :

Recency = (2 - 1) / (4 - 1) * (1 - 0) + 0 = 0,333333

Frequency = (6 – 3) / (24 – 3) * (1 – 0) + 0 = 0,142857

Monetary = (1,5 – 0,75) / (5,75 – 0,75) * (1 – 0) + 0 = 0,15

Time = (15 – 4) / (77 – 4) * (1 – 0) + 0 = 0,150685

Diketahui bahwa Xbaru adalah nilai yang dinormalisasi, Maxlama dan Minlama adalah nilai maksimum dan minimum pada fitur *Recency* (R) yaitu nilai Max 4 dan nilai Min 1. Untuk Maxbaru dan Minbaru adalah range dari nilai normalisasi yaitu 1 dan 0. Tabel data uji yang telah dinormalisasi ditunjukkan pada Tabel 4.2.

Tabel 4.2 Sampel Data Normalisasi

Data	Recency	Frequency	Monetary	Time	C
1	0.333333	0.142857	0.15	0.150685	1
2	0.333333	0.095238	0.1	0.09589	1
3	0.333333	0.52381	0.55	0.60274	1
4	0.333333	0.571429	0.6	0.616438	1
5	0.333333	0	0	0	1
6	0	1	1.05	1	-1
7	1	0.047619	0.05	0	-1
8	0	0.428571	0.45	0.424658	-1
9	1	0.952381	1	0.739726	-1
10	0	0.47619	0.5	0.589041	-1

4.4.2 Tahap Perhitungan

Tahap perhitungan dalam penelitian ini dibagikan dalam beberapa proses mulai dari menentukan nilai parameter pada formulasi PSO sampai pada menguji akurasi perhitungan pada SVM. Tahap perhitungan dimulai dari menentukan nilai parameter yang optimal dengan menggunakan metode PSO kemudian setelah mendapatkan nilai parameter yang optimal dilakukan perhitungan nilai akurasi dengan metode SVM. Berikut adalah tahapan – tahapan dalam proses perhitungan mulai dari tahap menentukan nilai parameter yang akan dioptimasi sampai mendapatkan nilai akurasi perhitungan :

1. Menentukan nilai parameter.
2. Mendapatkan nilai acak parameter sebagai nilai partikel untuk dioptimasi.
3. Menginputkan nilai parameter yang digunakan untuk proses perhitungan SVM.
4. Memasukkan nilai data latih dan parameter yang dibutuhkan untuk melakukan proses menghitung nilai kernel.
5. Menghitung nilai *kernel*.
6. Menghitung nilai *error*.
7. Menghitung nilai *delta alpha*.
8. Menghitung nilai *alpha* baru.
9. Menghitung nilai *bias*.
10. Menghitung nilai *f(x)*.
11. Mendapatkan nilai *fitness* yang digunakan untuk memproses perhitungan optimasi kecepatan partikel terbaru.
12. Mendapatkan kecepatan partikel terbaru.
13. Menentukan posisi partikel terbaru dari perhitungan kecepatan awal dengan kecepatan partikel terbaru.
14. Melakukan pengulangan kembali dari proses 2 – 13 sesuai dengan iterasi maksimal yang dibutuhkan.

4.4.2.1 Menentukan Nilai Parameter

Dalam menentukan nilai parameter pada studi kasus ini menggunakan 4 parameter yang akan dioptimasi, yaitu nilai *lambda* (λ), *gamma* (γ), *complexity* (C), dan varian (σ). Dengan memberikan nilai batas bawah dan batas atas setiap parameter pada Tabel 4.3.

Tabel 4.3 *Range* Parameter

Batas	λ	γ	C	σ
Bawah	0,01	0,0001	0,01	1
Atas	5	10	10000	1000

Proses memberikan parameter PSO dan nilai *random* untuk partikel awal adalah dengan menentukan jumlah partikel = 4, iterasi maksimum = 5, dan nilai *random* untuk partikel awal dari parameter yang akan dioptimasi sesuai range batas atas dan batas bawah tiap parameter. Dengan memberikan 4 nilai partikel dari 4 parameter (*λ, γ, C* dan *σ*) yang dioptimasi ditunjukkan pada Tabel 4.4.

Tabel 4.4 *Random* Partikel

Partikel	λ	γ	C	σ
$x_1(0)$	0,1	0,2	1000	75
$x_2(0)$	0,2	0,4	2000	150
$x_3(0)$	0,3	0,6	3000	225
$x_4(0)$	0,4	0,8	4000	300

4.4.2.2 Proses Menginput Nilai Parameter

Nilai parameter yang akan ditentukan disini berdasarkan pada 4 nilai parameter yaitu *lambda* (*λ*), *gamma* (*γ*), *Complexity* (*C*) dan varian (*σ*). *Dataset* parameter yang digunakan ditunjukkan pada Tabel 4.5.

Tabel 4.5 *Dataset* Parameter

λ	γ	C	σ
0,1	0,2	1000	75

4.4.2.3 Kernel

Pada proses perhitungan kernel berdasarkan pada data uji yang sudah dinormalisasi sebelumnya ditunjukkan pada tabel 4.2. Salah satu kernel yang direkomendasikan untuk digunakan adalah kernel *gaussian* RBF. Berikut adalah proses penggunaan kernel *gaussian* RBF ditunjukkan pada persamaan 2.21.

Nilai varian (*σ*) menggunakan nilai yang telah ditentukan sebelumnya pada langkah pertama yaitu *σ* = 75. Data pengujian menggunakan 10 data uji maka kernel *gaussian* RBF dapat dihasilkan dalam bentuk matrik 10 x 10. Berikut ini adalah contoh perhitungan pada kernel matrik 10 x 10.

$$K(x_1, y_2) = exp\left\{\begin{bmatrix}(0{,}3333 - 0{,}3333)^2 + \\ (0{,}1428 - 0{,}0952)^2 + \\ (0{,}15 - 0{,}1)^2 + \\ (0{,}1506 - 0{,}0958)^2\end{bmatrix}/[(-2) \times 75^2]\right\} = 0{,}9999$$

Contoh perhitungan diatas adalah contoh perhitungan data matriks pada kernel *gaussian* RBF (tabel data perhitungan kernel *gaussian* RBF yang lengkap akan ditampilkan pada lampiran)

4.4.2.4 Perhitungan Matriks *Hessian*

Pada proses perhitungan matriks $[D]_{ij}$ menggunakan nilai dari *lambda* (*λ*) yang akan ditampilkan pada persamaan 2.15.

Nilai *lambda* (λ) menggunakan nilai sebelumnya pada langkah pertama yaitu λ = 0,2 dan juga menggunakan nilai sebelumnya dari perhitungan kernel *gaussian* RBF. Pada data pengujian menggunakan 10 data *training* dan menghasilkan bentuk matrik 10 x 10. Berikut ini adalah contoh perhitungan pada perhitungan matriks $[D]_{ij.}$

$[D]_{12}$ = 1 x 1 x (0,9999 + $0{,}1^2$) = 1,0099

Contoh perhitungan diatas adalah contoh perhitungan data pada perhitungan matriks $[D]_{12}$.

4.4.2.5 Menghitung Nilai Error

Pada proses perhitungan nilai *error* adalah langkah selanjutnya yang kita cari setelah mendapatkan nilai proses perhitungan matrik *hessian*. Tujuan dari penghitungan nilai *error* ini adalah untuk mendapatkan nilai alpha terbaru dari setiap data *training*. Iterasi pada studi kasus ini menggunakan 3 kali proses pembaruan. Perhitungan nilai *error* akan ditampilkan dalam persamaan 2.16.

Berikut adalah contoh perhitungan nilai *error* pada iterasi 2 dengan nilai α yaitu 0,2 yang didapatkan dari hasil perhitungan iterasi 1 pada nilai *error*.

$$E_1 = 0{,}2 \times 1{,}01 + 0{,}2 \times 1{,}00999 + \cdots + 0{,}2 \times (-1{,}00996) = 8{,}4282$$

Berikut adalah tabel penghitungan iterasi pada nilai *error* ditunjukkan pada Tabel 4.6.

Tabel 4.6 Iterasi nilai *error*

No	Iterasi 1	Iterasi 2	Iterasi 3
1	0	8.4282E-05	9.08565E-05
2	0	9.23829E-05	0.000107058
3	0	1.88407E-05	-4.002E-05
4	0	1.26199E-05	-5.246E-05
5	0	0.000107957	0.000138205
6	0	5.13305E-05	0.000180345
7	0	-9.4459E-05	-0.00011121
8	0	-4.21295E-05	-6.5564E-06
9	0	4.56698E-05	0.000169027
10	0	-2.90102E-05	1.96802E-05

4.4.2.6 Menghitung Nilai Delta Alpha

Proses perhitungan nilai *delta alpha* dapat dihitung setelah mendapatkan hasil perhitungan dari nilai *error* dengan nilai *γ* dan *C* yang ditunjukkan pada persamaan 2.17.

Berikut adalah contoh perhitungan untuk mencari nilai $\delta\alpha_1$ iterasi 2.

$$\delta\alpha_1 = min\{max[0{,}2 \times (1 - 8{,}4282); 0{,}2]; 1000 - 0{,}2\}$$
$$= min\{0{,}11722; 999{,}8\}$$

$= 0{,}19998$

Fungsi *max* pada persamaan 4.8 agar bisa membatasi nilai α baru agar selalu bernilai ≥ 0, fungsi *min* untuk membatasi nilai α baru agar selalu bernilai $\leq C$. Berikut adalah tabel penghitungan iterasi pada nilai *delta alpha* ditunjukkan pada Tabel 4.7.

Tabel 4.7 Iterasi Nilai *Delta Alpha*

No	Iterasi 1	Iterasi 2	Iterasi 3
1	0.2	0.199983144	0.199981829
2	0.2	0.199981523	0.199978588
3	0.2	0.199996232	0.200008004
4	0.2	0.199997476	0.200010492
5	0.2	0.199978409	0.199972359
6	0.2	0.199989734	0.199963931
7	0.2	0.200018892	0.200022242
8	0.2	0.200008426	0.200001311
9	0.2	0.199990866	0.199966195
10	0.2	0.200005802	0.199996064

4.4.2.7 Menghitung Nilai Alpha

Proses perhitungan nilai *alpha* dapat dihitung setelah mendapatkan hasil perhitungan dari nilai *delta alpha* dengan nilai λ = 0,1 untuk perhitungan matriks $[D]_{ij}$, serta nilai γ = 0,2 dan nilai C = 1000 untuk setiap proses dalam pembaruan nilai α. Berikut ditunkukan perhitungan pembaruan nilai α pada persamaan 2.18.

Berikut adalah contoh untuk perhitungan dalam mencari nilai α_1 iterasi 2.

$$\alpha_1 = 0{,}2 + 0{,}19998 = 0{,}39998$$

Berikut adalah tabel penghitungan iterasi 3 pada nilai *alpha* baru ditunjukkan pada Tabel 4.8.

Tabel 4.8 Iterasi Nilai *Alpha* Baru

No	Iterasi 1	Iterasi 2	Iterasi 3
1	0.2	0.399983144	0.599964972
2	0.2	0.399981523	0.599960112
3	0.2	0.399996232	0.600004236
4	0.2	0.399997476	0.600007968
5	0.2	0.399978409	0.599950768
6	0.2	0.399989734	0.599953665
7	0.2	0.400018892	0.600041134
8	0.2	0.400008426	0.600009737
9	0.2	0.399990866	0.599957061
10	0.2	0.400005802	0.600001866

4.4.2.8 Menghitung Nilai Bias

Proses perhitungan nilai bias adalah untuk menghitung nilai dari fungsi SVM. Setelah mendapatkan nilai α terbaru dari tiap data pada proses sebelumnya, dicari juga α terbesar dari kelas positif dan kelas negatif untuk digunakan dalam penentuan x^+ dan x^- yang digunakan untuk persamaan bias. Perhitungan nilai bias ditunjukkan dengan bentuk persamaan 2.13.

Berikut contoh perhitungan untuk mencari nilai bias.

$$b = -0{,}5 \times \begin{bmatrix} 0{,}59999 \times 1 \times (0{,}99996 + 0{,}99997) + \\ 0{,}59999 \times 1 \times (0{,}99995 + 0{,}99997) + \\ \cdots + \\ 0{,}6000 \times (-1) \times (0{,}99999 + 0{,}99999) \end{bmatrix} = -8{,}52057$$

4.4.2.9 Menghitung *f(x)*

Proses menghitung nilai *f(x)* dilakukan setelah mendapatkan nilai bias dari langkah sebelumnya. Proses perhitungan ini dilakukan terhadap data testing dengan menjumlahkan nilai dari total bobot data testing dengan nilai bias yang dihasilkan dari perhitungan sebelumnnya. Perhitungan nilai *f(x)* disini untuk mendapatkan nilai *hyperplane.* Perhitungan *hyperplane* diketahui dengan bentuk persamaan 2.12.

Berdasarkan contoh perhitungan *hyperplane* pada persamaan 4.7 didapatkan fungsi untuk menentukan status dari data uji.

$$f(x_1) = sign\left\{ \begin{bmatrix} 0{,}32933 \times 1 \times 0{,}94362 + \\ 0{,}43527 \times 1 \times 0{,}8343 + \\ \cdots + \\ 0{,}32021 \times (-1) \times 0{,}13861 \end{bmatrix} + 0{,}27873 \right\} = 1$$

Nilai *f(x)* sama dengan 1 merupakan nilai menunjukkan status benar dan bisa mendonorkan darah sedangkan *f(x)* sama dengan -1 nilai menunjukkan status salah dan tidak bisa mendonorkan darah. Berikut ditunjukkan akurasi hasil klasifikasi pada tabel 4.9.

Tabel 4.9 Akurasi Hasil Klasifikasi

Data Uji	*f(x)*	Status Sistem
20	1	BENAR
21	1	BENAR
22	1	BENAR
23	1	BENAR
24	1	BENAR
25	1	BENAR
26	1	BENAR
27	1	BENAR

28	-1	SALAH
29	-1	SALAH

Dengan diketahui hasil dari status sistem dari data uji yang telah diproses sebelumnya, dapat ditemukan akurasi sistem dengan melakukan perhitungan *f(x)* sistem dengan persamaan 2.31.

$$acc = \frac{8}{10}\ x\ 100\% = 80\%$$

Sehingga dari menghitung persamaan diatas didapatkan akurasi sebesar 80% untuk digunakan sebagai nilai *fitness* pada PSO.

4.4.2.10 Hasil Akurasi

Hasil akurasi tertinggi yang di dapatkan dari beberapa partikel *x* pada perhitungan SVM. Hasil tertinggi digunakan sebagai alat ukur seberapa besar akurasi dari sistem ini. Dengan perhitungan partikel *x* pada perhitungan SVM masing – masing perhitungan partikel *x* memperoleh akurasi 80%. Untuk hasil akurasi dengan akurasi tertinggi akan ditunjukkan pada Tabel 4.10.

Tabel 4.10 Hasil Akurasi

Partikel	λ	γ	C	σ	*Fitness*
$X_1(0)$	0.1	0.2	1000	75	80%
$X_2(0)$	0.2	0.4	2000	150	80%
$X_3(0)$	0.3	0.6	3000	225	80%
$X_4(0)$	0.4	0.8	4000	300	80%

4.4.2.11 Menentukan *Fitness*

Proses menentukan nilai *fitness* diperoleh dari mengambil nilai akurasi dari metode SVM yang menggunakan nilai parameter berupa nilai partikel (*x*) pada tabel 4.10. Dengan metode perhitungan SVM didapatkan *fitness* berupa akurasi dari tiap partikel ditunjukkan pada Tabel 4.11.

Tabel 4.11 *Fitness* Partikel

Fitness	Akurasi
$f_1(0)$ =	80%
$f_2(0)$ =	80%
$f_3(0)$ =	80%
$f_4(0)$ =	80%

Perhitungan kecepatan partikel dilakukan setelah diketahui nilai *fitness* dari tiap partikel melalui perhitungan akurasi SVM. Tujuan dari menghitung kecepatan partikel agar dapat mengetahui P_{best} dari tiap partikel dan G_{best} dari semua partikel. P_{best} adalah nilai terbaik yang didapatkan dari semua partikel sedangkan G_{best} adalah nilai terbaik yang didapatkan dari semua $P_{best.}$ Berdasarkan tabel 4.11

dengan membandingkan nilai partikel *fitness* masing-masing didapatkan nilai P_{best} dan G_{best} ditunjukkan pada Tabel 4.12.

Tabel 4.12 Nilai Partikel Terbaik

	λ	γ	C	σ
$P_{best,1}$ =	0.1	0.2	1000	75
$P_{best,2}$ =	0.2	0.4	2000	150
$P_{best,3}$ =	0.3	0.6	3000	225
$P_{best,4}$ =	0.4	0.8	4000	300
G_{best} =	0.1	0.2	1000	75

4.4.2.12 Menghitung Kecepatan Partikel

Proses perhitungan kecepatan partikel dengan ditentukan terlebih dahulu nilai (c) secara random misalkan c_1 = 2,05; c_2 = 2,05 dan r_1 = 0,2 tujuannya agar partikel mendekati titik optimal. Kecepatan partikel dapat ditunjukkan dengan persamaan 2.24.

Dalam perhitungan kecepatan partikel perlu menentukan nilai variabel $\lambda_{PSO.}$ Nilai variabel PSO ditunjukkan dengan persamaan 2.29.

Sedangkan dalam proses penentuan nilai λ_{PSO} dibutuhkan nilai variabel φ. Nilai variabel φ ditunjukkan dengan persamaan 2.29.

Berikut adalah perhitungan λ_{PSO} dengan nilai c_1 = 2,05 dan c_2 = 2,05.

$$\varphi = 2{,}05 + 2{,}05 = 4{,}1$$

$$\lambda_{PSO} = \frac{2}{\left|2 - 4{,}1 - \sqrt{4{,}1^2 - 4 \times 4{,}1}\right|} = 0{,}729$$

Variabel λ_{PSO} pada persamaan 4.8 berbeda dengan parameter λ pada SVM yang dioptimasi. Berikut ini adalah contoh perhitungan dari mencari kecepatan partikel kedua untuk parameter λ pada iterasi pertama, sedangkan untuk hasil perhitungan kecepatan seluruh partikel saat iterasi pertama ditunjukkan pada Tabel 4.13.

$$v_2(1) = 0{,}729 \times [0 + 2{,}05 \times 0{,}2 \times (0{,}2 - 0{,}2) + 2{,}05 \times 0{,}2 \times (0{,}1 - 0{,}2)]$$
$$= -0{,}02992$$

Tabel 4.13 Kecepatan Partikel

Kecepatan	λ	γ	C	σ
$v_1(1)$ =	0	0	0	0
$v_2(1)$ =	-0.02992	-0.05985	-299.236	-22.4427
$v_3(1)$ =	-0.05985	-0.11969	-598.472	-44.8854
$v_4(1)$ =	-0.08977	-0.17954	-897.708	-67.3281

4.4.2.13 Menentukan Posisi Baru Partikel

Pergerakan dari tiap partikel menuju titik yang optimal menghasilkan posisi baru yang nantinya dapat menemukan titik optimal baru. Posisi baru dari tiap partikel ditemukam dengan menambahkan posisi awal dengan kecepatan partikel yang ditunjukkan pada persamaan 2.23.

Berikut contoh perhitungan untuk mencari posisi baru partikel kedua untuk parameter λ pada iterasi pertama.

$$x_2(1) = 0{,}2 + (-0{,}02992)$$
$$= 0{,}17008$$

Untuk hasil perhitungan seluruh posisi baru dari tiap partikel saat iterasi pertama ditunjukkan pada Tabel 4.14.

Tabel 4.14 Posisi Baru Partikel

Partikel	λ	γ	C	σ
$x_1(1)$ =	0.1	0.2	1000	75
$x_2(1)$ =	0.170076	0.340153	1700.764	127.5573
$x_3(1)$ =	0.240153	0.480306	2401.528	180.1146
$x_4(1)$ =	0.310229	0.620458	3102.292	232.6719

4.4.2.14 Hasil Optimasi

Hasil optimasi diperoleh setelah melakukan penghitungan sistem secara sampai iterasi maksimal yang dibutuhkan. Setelah setiap partikel telah melewati *range* partikel secara keseluruhan maka setiap pembaruan posisi partikel pada perhitungan manualisasi ini mendapatkan nilai akurasi *fitness* tertinggi 80%. Berikut perhitungan hasil optimasi ditunjukkan pada Tabel 4.15.

Tabel 4.15 Hasil Optimasi

Iterasi G_{best}	λ	γ	C	σ	*Fitness*
0	0.1	0.2	1000	75	80%
1	0.1	0.2	1000	75	80%
2	0.1	0.2	1000	75	80%
3	0.1	0.2	1000	75	80%

4.5 Perancangan Uji Coba dan Evaluasi

Dalam menentukan nilai parameter terbaik dari SVM-PSO harus dilakukan pengujian program dengan uji coba agar mendapatkan dan mengetahui parameter yang optimal. Uji coba yang dilakukan tersebut antara lain adalah :

1. Uji coba menentukan nilai batas bawah dan batas atas yang membatasi ruang pencarian dari tiap dimensi partikel.
2. Uji coba menentukan jumlah iterasi α yang optimal.
3. Uji coba mencari kombinasi nilai konstanta akselerasi yang optimal.
4. Uji coba menentukan jumlah partikel yang optimal.
5. Uji coba menentukan jumlah iterasi partikel yang optimal.

Uji coba ini dilakukan dengan pengujian tingkat akurasi dengan membandingkan data aktual dengan data prediksi dari algoritma SVM dan PSO. Pada penelitian ini nilai akurasi didapatkan dari melakukan 10 percobaan untuk masing-masing proses pengujian.

4.5.2 Uji Coba Batas Ruang Pencarian Dimensi

Uji coba batas ruang pengujian dimensi yaitu uji coba yang dilakukan untuk mengetahui *range* ruang pencarian dimensi agar dapat menghasilkan kombinasi parameter SVM yang optimal. Nilai *range* tersebut akan mempengaruhi nilai parameter SVM. *Range* yang diuji coba *range* dengan variasi berbeda dari tiap parameter.

a. Dimensi parameter λ

Rancangan uji coba *range* ruang pencarian tiap dimensi parameter λ dapat dilihat pada Tabel 4.16.

Tabel 4.16 Rancangan Uji Coba *Range* Dimensi λ

λ	Akurasi Pengujian Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,0001											
0,001											
0,01											
0,1											
1											
10											
20											
30											
50											
100											

b. Dimensi parameter *γ*

Rancangan uji coba *range* ruang pencarian tiap dimensi parameter *γ* dapat ditunjukkan pada Tabel 4.17.

Tabel 4.17 Rancangan Uji Coba *Range* Dimensi *γ*

γ	Akurasi Pengujian Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,00000001											
0,0000001											
0,000001											
0,00001											
0,0001											
0,001											
0,01											
0,1											
1											

c. Dimensi parameter *C*

Rancangan uji coba Panjang *range* ruang pencarian tiap dimensi parameter *C* ditunjukkan pada Tabel 4.18.

Tabel 4.18 Rancangan Uji Coba *Range* Dimensi *C*

C	Akurasi Pengujian Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,001											
0,01											
0,1											
1											
10											
100											
1000											
10000											

d. Dimensi parameter σ

Rancangan ui coba *range* ruang pencarian tiap dimensi parameter σ dapat ditunjukkan pada Tabel 4.19.

Tabel 4.19 Rancangan Uji Coba *Range* Dimensi σ

σ	Akurasi Pengujian Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,001											
0,01											
0,1											
1											
10											
100											
1000											
10000											

4.5.3 Uji Coba Jumlah Iterasi Pelatihan α

Uji coba jumlah iterasi α adalah uji coba sistem untuk mengetahuin iterasi pelatihan α agar mendapatkan nilai α yang optimal. Variasi jumlah iterasi yang digunakan adalah 10. Rancangan uji coba jumlah iterasi pelatihan α ditunjukkan pada Tabel 4.20.

Tabel 4.20 Rancangan Uji Coba *Range* Dimensi α

Jumlah Iterasi	Akurasi Percobaan Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
10											
20											
50											
100											
200											
300											
400											
600											
800											
1000											

4.5.4 Uji Coba Kombinasi Nilai Konstanta Akselerasi

Uji coba kombinasi nilai konstanta akselerasi adalah uji coba pada sistem untuk mengetahui kombinasi nilai konstanta akselerasi pertama (c_1) dan kedua (c_2) terbaik agar dapat menghasilkan parameter *Support Vector Machine* paling optimal. Uji coba pada nilai konstanta akselerasi pertama (c_1) dan kedua (c_2) menggunakan nilai berbeda. Rancangan uji coba ditunjukkan pada Tabel 4.21.

Tabel 4.21 Rancangan Uji Coba Kombinasi Nilai Konstanta Akselerasi

Nilai c_1	Nilai c_2	Akurasi Percobaan Ke-*i*										Rata-rata
		1	2	3	4	5	6	7	8	9	10	
0	0											
0,5	0,5											
1	1											
1,5	1,5											
2	2											
2,5	2,5											
3	3											
3,5	3,5											
5	5											

4.5.5 Uji Coba Jumlah Partikel

Uji coba banyaknya jumlah partikel adalah uji coba pada sistem untuk mengetahui jumlah partikel dari algoritma *Particle Swarm Optimization* yang digunakan agar dapat menghasilkan parameter *Support Vector Machine* paling optimal. Variasi jumlah partikel yang digunakan adalah 10. Rancangan uji coba partikel ditunjukkan pada Tabel 4.22.

Tabel 4.22 Rancangan Uji Coba Jumlah Partikel

Jumlah Partikel	Akurasi Percobaan Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
10											
20											
30											
40											
50											
60											
70											
80											
90											
100											

4.5.6 Uji Coba Jumlah Iterasi Partikel

Uji coba banyaknya jumlah iterasi partikel adalah uji coba uji coba sistem untuk mengetahui jumlah iterasi partikel dari algoritma *Particle Swarm Optimization* yang digunakan agar menghasilkan parameter *Support Vector Machine* paling optimal. Variasi jumlah maksimum iterasi yang digunakan adalah 10. Rancangan uji coba jumlah partikel ditunjukkan pada Tabel 4.23.

Tabel 4.23 Rancangan Uji Coba Jumlah Iterasi Partikel

Jumlah Iterasi	Akurasi Percobaan Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
10											
20											
30											
40											
50											
60											
70											
80											

BAB 5 IMPLEMENTASI

Bab ini membahas tentang implementasi dari perangkat lunak berdasarkan hasil analisis kebutuhan dan perancangan yang telah dibuat. Bagian – bagian yang akan dibahas adalah tentang implementasi algoritma dan implementasi antarmuka.

5.1 Implementasi Algoritme

Berdasarkan perancangan sistem yang telah dijabarkan pada Bab 4, selanjutnya akan dibahas implementasi algoritme berdasarkan dengan perancangan sistem yang dibuat sebelumnya. Implementasi ini menggunakan Bahasa pemrograman *java*.

5.1.1 Implementasi Kernel

Proses perhitungan kernel berikut ini merupakan proses perhitungan dengan menggunakan metode SVM yang memakai *dot product* dari 2 data *training* yang berada pada ruang vektor berdimensi tinggi. Dengan menggunakan fungsi kernel RBF dengan keluaran yang dihasilkan berupa matriks kernel berindeks n x n. Berikut implementasi perhitungan matriks kernel ditunjukkan pada Tabel 5.1.

Kode Program 5.1 Implementasi Kernel

No	Kode Program
1	`private double[][] hitungKernelRBF(double[][] jarakData) {`
2	`double[][]kernel = new double[jarakData.length][jarakData.length];`
3	`for (int i = 0; i < jarakData.length; i++) {`
4	`for (int j = 0; j < jarakData[0].length; j++) {`
5	`kernel[i][j]=Math.exp(jarakData[i][j]/(2*Math.pow(sigma,2)));`
6	`}`
7	`}`
8	`return kernel.clone();`
9	`}`

1. Untuk perhitungan nilai kernel ditunjukkan baris ke-5.

5.1.2 Implementasi Matriks Hessian

Proses perhitungan *Matriks Hessian* ini merupakan proses pertama pada algoritma *sequential learning* SVM. Perhitungan *Matriks Hessian* ini diproses menggunakan data input berupa nilai dari perhitungan kernel dan menggunakan nilai dari data konstanta *lambda* (λ). Hasil keluaran nantinya akan berupa nilai matriks berindeks n x n, dimana n itu adalah banyak jumlah data *training*. Berikut implementasi perhitungan *Matriks Hessian* ditunjukkan pada Tabel 5.2.

Kode Program 5.2 Implementasi Matriks Hessian

No	Kode Program
1	`private void hitungD(double[][] kernel) {`
2	`for (int I = 0; I < kernel.length; i++) {`
3	`for (int j = 0; j < kernel.length; j++) {`
4	

```
5                 D[i][j]=data[i][data[0].length-1]*data[j][data[0].length-
6   1]*(kernel[i][j]+Math.pow(lambda,2));
7               }
8           }
        }
```

1. Untuk perhitungan nilai *matriks hessian* pada tiap data ditunjukkan pada baris ke-4 dan ke-5

5.1.3 Implementasi Pelatihan *α*

Proses pelatihan ini diulang sampai pada banyaknya jumlah iterasi maksimum yang ditentukan. Proses yang pertama kali dilakukan adalah menghitung nilai *error (E)* untuk semua data training. Proses yang kedua dilakukan dengan menghitung nilai *delta alpha* ($\delta\alpha_i$) dan selanjutnya untuk proses terakhir adalah mencari nilai *alpha* baru *(α)*. Berikut merupakan implementasi dari perhitungan pelatihan *alpha* baru *(α)* ditunjukkan pada Tabel 5.3.

Kode Program 5.3 Implementasi Pelatihan *α*

No	Kode Program
1	`private double hitungDeltaAlpha(double error, double alpha) {`
2	`        double delta_alpha;`
3	`        double x = gamma*(1 - error);`
4	`        double y;`
5	
6	`        if (x > -alpha) {`
7	`            y = x;`
8	`        } else {`
9	`            y = -alpha;`
10	`        }`
11	`        double z = c - alpha;`
12	`        if (y < z) {`
13	`            delta_alpha = y;`
14	`        } else {`
15	`            delta_alpha = z;`
16	`        }`
17	
18	`        return delta_alpha;`
19	`    }`

1. Untuk menginisiasi nilai *alpha* ditunjukkan pada baris ke-6 dan sampai baris ke-10.
2. Untuk melakukan perhitungan nilai *alpha* ditunjukkan pada baris ke-11.
3. Untuk mendapatkan nilai *alpha* baru ditunjukkan pada baris ke-12 sampai dengan baris ke-19.

5.1.4 Implementasi Perhitungan Bias

Proses perhitungan mencari nilai bias diperlukan nilai dari jumlah bobot dari kelas positif dan jumlah bobot dari kelas negatif. Nilai bobot disini melambangkan hasil perhitungan dari *alpha,* nilai *kernel* daripada data yang merupakan kelas positif dan kelas negatif, dan juga kelas dari data positif atau negatif seperti yang

sudah dijelaskan pada sebelumnya. Berikut merupakan implementasi dari perhitungan bias ditunjukkan pada Tabel 5.4.

Kode Program 5.4 Implementasi Perhitungan Bias

No	Kode Program
1	`private double hitungBias(double[] kXPositif, double[] kXNegatif) {`
2	`      double bias;`
3	`      double sum = 0;`
4	`      for (int i = 0; i < data.length; i++) {`
5	`      sum+=alpha[i]*data[i][data[0].length1]*(kXPositif[i]+kXNegatif[i]);`
6	`        }`
7	`        bias = -0.5*sum;`
8	`        return bias;`
9	`    }`

1. Untuk menginisiasi sebanyak panjang data yang dibutuhkan ditunjukkan pada baris ke-4.
2. Untuk melakukan perhitungan nilai penjumlahan dari hasil perkalian *alpha*, *lambda* dan kernel dari tiap data ditunjukkan pada baris ke-5.
3. Untuk melakukan perhitungan nilai bias ditunjukkan pada baris ke-8.

5.1.5 Implementasi Perhitungan *f(x)*

Proses perhitungan ini dilakukan pada data *testing* dengan menggunakan penjumlahan dari nilai total bobot data *testing* terhadap nilai bias yang didapatkan dari perhitungan sebelumnya. Hasil yang didapatkan adalah nilai dari fungsi linier *f(x)*, yang dimana jika hasil yang dihitung bernilai positif maka masuk pada kelas 1, sedangkan jika hasil yang dihitung bernilai negative maka masuk pada kelas -1. Berikut merupakan implementasi dari perhitungan *f(x)* ditunjukkan pada Tabel 5.5.

Kode Program 5.5 Implementasi Perhitungan Nilai Prediksi atau Nilai *F(x)*

No	Kode Program
1	`for private double[] hitungYPrediksi(double[][] kernelDataUji) {`
2	`        double[] yPrediksi = new double[dataUji.length];`
3	`        for (int i = 0; i < dataUji.length; i++) {`
4	`            yPrediksi[i] = 0;`
5	`            for (int j = 0; j < kernelDataUji.length; j++) {`
6	`                yPrediksi[i]+=alpha[j]*data[j][data[0].length-`
7	`1]*kernelDataUji[i][j];`
8	`            }`
9	`            yPrediksi[i] += bias;`
10	`        }`
11	`        return yPrediksi;`
12	`    }`
13	

1. Untuk menginisiasi banyak panjang data yang diuji ditunjukkan pada baris ke-3 samapai dengan baris ke-5.
2. Untuk perhitungan nilai *alpha, lambda* dan kernel data yang diuji ditunjukkan pada baris ke-6 dan baris ke-7.
3. Perhitungan nilai *f(x)* ditunjukkan pada baris ke-9.

5.1.6 Implementasi Perhitungan Kecepatan Partikel

Proses perhitungan demi memilih nilai besar kecepatan partikel memerlukan nilai pada faktor penyeimbang, kecepatan pada partikel sebelumnya, konstanta kecepatan, bilangan *random*, P_{best} dan G_{best}. Hal ini ditunjukkan pada iterasi ke-0, kecepatan awal tetap memiliki nilai 0, sedangkan pada iterasi berikutnya dapat menggunakan persamaan pembaruan dengan perhitungan kecepatan partikel. Berikut merupakan implementasi dari perhitungan kecepatan partikel ditunjukkan pada Tabel 5.6.

Kode Program 5.6 Implementasi Perhitungan Kecepatan Partikel

No	Kode Program
1	`private double[] hitungKecepatan(int partikelIdx, Partikel partikel) {`
2	`        double w = 1; // w -> range 0,8...1,2`
3	`        double[] kecepatanAwal = partikel.getKecepatan();`
4	`        double[] kecepatanBaru = new double[kecepatanAwal.length];`
5	`        double r1 = generateRandomDouble(0,1);`
6	`        double r2 = generateRandomDouble(0,1);`
7	
8	`        for (int I = 0; I < kecepatanAwal.length; i++) {`
9	`            double inersia = w*kecepatanAwal[i];`
10	
11	`doublecognitive=(c1*r1)*(pBest.get(partikelIdx).getParameter()[i]partikel`
12	`.getParameter()[i]);`
13	`doublesocial=(c2*r2)*(gBest.getParameter()[i]-`
14	`partikel.getParameter()[i]);`
15	`            kecepatanBaru[i] = 0.72894*(inersia + cognitive + social);`
16	`        }`
17	`        return kecepatanBaru;`
18	`    }`

1. Untuk menginisiasi sebanyak panjang data tiap partikel ditunjukkan pada baris ke-8 dan ke-9
2. Untuk melakukan perhitungan kecepatan *lambda, gamma, C* dan *varian* ditunjukkan pada baris ke-11 sampai dengan baris ke-14.
3. Untuk melakukan perhitungan nilai kecepatan partikel ditunjukkan pada baris ke-15.

5.1.7 Implementasi Perhitungan Posisi Baru Partikel

Proses perhitungan menemukan nilai pada posisi baru partikel adalah dengan menjumlahkan posisi lama dengan kecepatan partikel. Ditunjukkan pada iterasi ke-0, posisi awal partikel diperoleh dari nilai *random*, sedangkan iterasi berikutnya dapat menggunakan persamaan pembaruan dengan perhitungan posisi baru partikel ditunjukkan pada Tabel 5.7.

Kode Program 5.7 Implementasi Perhitungan Posisi Baru Partikel

No	Kode Program
1	`private void updatePartikel() {`
2	`        for (int I = 0; I < swarm.size(); i++) {`
3	`            double[] current = swarm.get(i).getParameter();`
4	`            double[] baru = new double[current.length];`
5	`            for (int j = 0; j < current.length; j++) {`

```
6                   baru[j] = current[j] + swarm.get(i).getKecepatanById(j);
7               }
8               swarm.get(i).setParameter(baru);
9           }
10      }
```

1. Untuk menginisiasi posisi baru partikel sepanjang banyak partikel ditunjukkan pada baris ke-5.
2. Untuk melakukan perhitungan posisi baru nilai partikel ditunjukkan pada baris ke-6.

5.1.8 Implementasi Perhitungan nilai P_{best}

Setelah mengetahui nilai *fitness* pada setiap partikel yang ada, proses perhitungan untuk menemukan nilai terbaik dari suatu partikel dilambangkan sebagai P_{best}, dengan bobot nilai akurasi terbesar atau nilai *fitness*. Menemukan nilai P_{best} tidak menggunakan persamaan dalam mencarinya, melainkan dengan membandingkan *fitness* pada posisi partikel yang pernah ditempati pada suatu partikel tersebut. Nilai P_{best} digunakan selanjutnya dalam menemukan nilai G_{best}. Berikut implementasi dari perhitungan nilai P_{best} ditunjukkan pada Tabel 5.8.

Kode Program 5.8 Implementasi Perhitungan Nilai P_{best}

No	Kode Program
1	`private void updatePBest(ArrayList<Partikel> swarm) {`
2	`for (int i = 0; i < swarm.size(); i++) {`
3	`//  debugArray(swarm.get(i).getParameter());`
4	`System.out.print(pBest.get(i).getFitness()+""+swarm.get(i).getFitness()`
5	`+ " ");`
6	`if (pBest.get(i).getFitness() < swarm.get(i).getFitness()) {`
7	`Partikel p = new Partikel();`
8	`p.setParameter(swarm.get(i).getParameter());`
9	`p.setFitness(swarm.get(i).getFitness());`
10	`pBest.set(i, p);`
11	`}`
12	`System.out.println();`
13	`}`
14	`}`

1. Untuk melakukan pencarian partikel sebanyak panjang dari partikel P_{best} ditunjukkan pada baris ke-2.
2. Untuk memberikan nilai pada tiap P_{best} untuk tiap partikel ditunjukkan pada baris ke-6 sampai dengan baris ke-10.

5.1.9 Implementasi Perhitungan nilai G_{best}

Proses perhitungan untuk menemukan nilai G_{best} memerlukan nilai *fitness* dari semua partikel terhadap iterasi secara keseluruhan. Nilai G_{best} tidak menggunakan persamaan dalam mencarinya, tetapi nilai G_{best} diperoleh dari partikel yang mempunyai nilai akurasi terbaik. Berikut merupakan implementasi dari perhitungan nilai G_{best} ditunjukkan pada Tabel 5.9.

Kode Program 5.9 Implementasi Perhitungan Nilai G_{best}

```
No  Kode Program
1   private void updateGBest(ArrayList<Partikel> pBest) {
2           if (gBest == null) {
3               gBest = new Partikel();
4               gBest.setParameter(pBest.get(0).getParameter());
5               gBest.setFitness(pBest.get(0).getFitness());
6           }
7
8           for (int I = 0; I < pBest.size(); i++) {
9               if (gBest.getFitness() < pBest.get(i).getFitness()) {
10                  gBest.setParameter(pBest.get(i).getParameter());
11                  gBest.setFitness(pBest.get(i).getFitness());
12              }
13          }
14      }
```

1. Untuk menyatakan kondisi mendapatkan nilai G_{best} diambil dari partikel yang mempunyai nilai P_{best} dinyatakan pada baris ke-2 sampai dengan baris ke-5.
2. Untuk menginisiasi sepanjang banyaknya partikel nilai P_{best} ditunjukkan pada baris ke-8.
3. Untuk menyatakan kondisi memperbarui perhitungan G_{best} ditunjukkan pada baris ke-9 sampai dengan baris ke-11.

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini membahas tentang proses pada pengujian optimasi SVM. Proses ini dilakukan dengan menggunakan pengujian akurasi. Pengujian akurasi ini digunakan untuk dapat mengukur akurasi dari sistem dengan membandingkan hasil keluaran sistem dan hasil dataset laboratorium.

6.1 Pengujian Batas Ruang Pencarian Dimensi

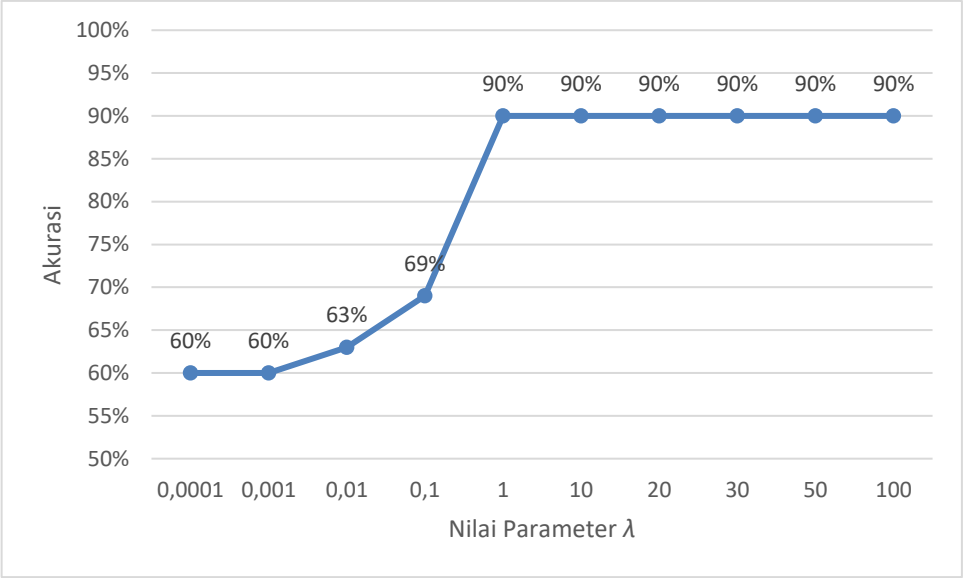
Untuk menentukan batas ruang pencarian dimensi terbaik dalam kasus ini dilakukan pengujian pada batas ruang dimensinya. Dilakukan pengujian *range* ruang pencarian dimensi pada *lambda* (λ), batas batas yang akan diuji dimulai dari 0,0001, 0,001, 0,01, 0,1, 1, 10, 20, 30, 50, dan 100. Berikut *range* dimensi parameter yang digunakan :

a. Range pencarian ruang pada dimensi γ : 0,0001-10000.
b. Range pencarian ruang pada dimensi C : 0,1-100.
c. Range pencarian ruang pada dimensi σ : 0,01-0,1.
d. Jumlah partikel : 10.
e. Jumlah iterasi maksimum : 10.
f. Konstanta akselerasi pertama dan kedua : 2 dan 2.
g. Jumlah iterasi α : 10.

Pengujian range ruang pada λ akan dilakukan sebanyak 10 kali. Hasil pengujian ditunjukkan pada Tabel 6.1.

Tabel 6.1 Pengujian Pada *Range* Ruang Pencarian Dimensi λ

λ	Akurasi Pengujian Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,0001	60%	60%	60%	60%	60%	60%	60%	60%	60%	60%	60%
0,001	60%	60%	60%	60%	60%	60%	60%	60%	60%	60%	60%
0,01	60%	60%	60%	60%	90%	60%	60%	60%	60%	60%	63%
0,1	70%	90%	90%	80%	60%	60%	60%	60%	60%	60%	69%
1	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
10	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
20	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
30	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
50	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
100	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%



Gambar 6.1 Grafik Pengujian Untuk Pencarian Dimensi *λ*

Berdasarkan pada grafik pengujian pencarian dimensi *λ* yang ditunjukkan pada Gambar 6.1, pada percobaan pengujian nilai *lambda* 0,0001, 0,001, 0,01, 0,1, 1, 10, 20, 30, 50 dan 100 mendapatkan akurasi tertinggi 90%. Nilai akurasi tertinggi didapatkan dari nilai *lambda* 1, 10, 20 dan selanjutnya mencapai titik konvergen sampai nilai parameter 100. Dari hasil pengujian nilai *lambda* didapatkan bahwa semakin *lambda* menjauhi nilai 0 maka akan mendapatkan nilai akurasi yang semakin membaik, tapi akan mengakibatkan lamanya proses pembelajaran dan komputasi pada perhitungan matriks *hessian* karena nilai *lambda* yang digunakan semakin besar (Vijayakumar). Pada penelitian ini nilai akurasi optimal didapat pada rentang parameter 0,1-100.

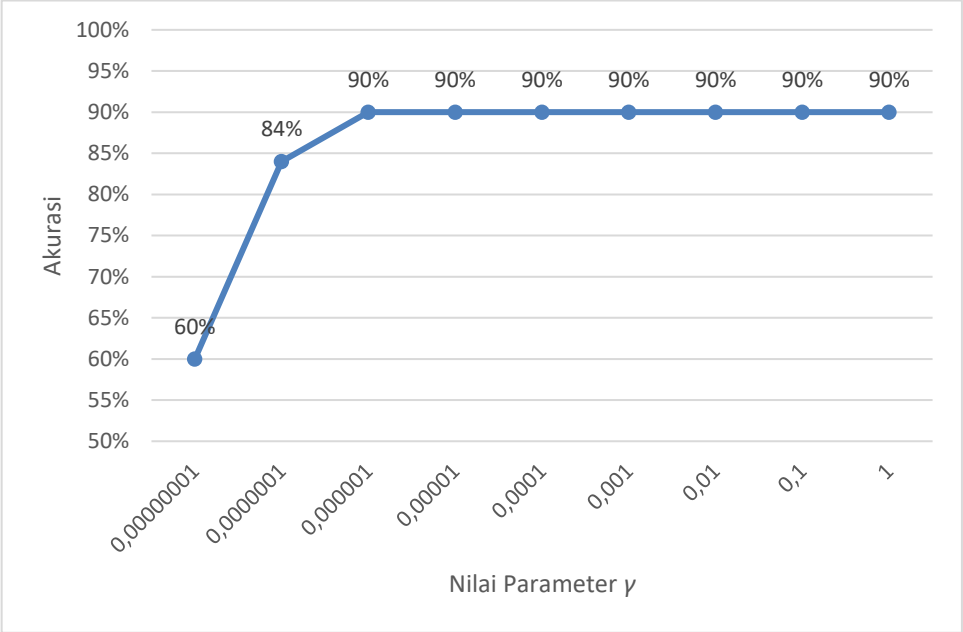
Pada pengujian *range* ruang dimensi pada *gamma* (*γ*), batas yang akan digunakan adalah 0,0001, 0,001, 0,01, 0,1, 1, 10, 100, 1000, dan 10000. Berikut *range* dimensi parameter yang digunakan :

a. Range pencarian ruang pada dimensi *λ* : 0,1-100.
b. Range pencarian ruang pada dimensi *C* : 0,1-100.
c. Range pencarian ruang pada dimensi *σ* : 0,01-0,1.
d. Jumlah partikel : 10.
e. Jumlah iterasi maksimum : 10.
f. Konstanta akselerasi pertama dan kedua : 2 dan 2.
g. Jumlah iterasi *α* : 10.

Pengujian range ruang pada *γ* akan dilakukan sebanyak 10 kali. Hasil pengujian ditunjukkan pada Tabel 6.2.

Tabel 6.2 Pengujian Pada *Range* Ruang Pencarian Dimensi *γ*

γ	Akurasi Pengujian Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,00000001	60%	60%	60%	60%	60%	60%	60%	60%	60%	60%	60%
0,0000001	60%	90%	90%	90%	90%	90%	60%	90%	90%	90%	84%
0,000001	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
0,00001	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
0,0001	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
0,001	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
0,01	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
0,1	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
1	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%



Gambar 6.2 Grafik Pengujian Untuk Pencarian Dimensi *γ*

Berdasarkan pada grafik pengujian pencarian dimensi *γ* yang ditunjukkan pada Gambar 6.2, pada percobaan pengujian nilai *gamma* 0,00000001, 0,0000001 0,000001, 0,00001, 0,0001, 0,001, 0,01, 0,1, dan 1 mendapatkan akurasi sebesar 90%. Dari hasil pengujian nilai *gamma(γ)* dapat disimpulkan bahwa semakin rendah nilai *gamma(γ)* maka tingkat laju pembelajaran akan berjalan lebih lama dan semakin tinggi nilai *gamma(γ)* maka tingkat laju pembelajaran akan berjalan lebih cepat. Pada penelitian ini nilai akurasi yang optimal didapat pada rentang 0,000001-1.

Pengujian dilakukan lagi untuk mencari ruang pencarian dimensi *complexity (C*), batas yang akan diuji yaitu 0,001, 0,01, 0,1, 1, 10, 100, 1000 dan 10000. Berikut *range* dimensi parameter yang digunakan :

a. Range pencarian ruang pada dimensi λ : 0,1-100.

b. Range pencarian ruang pada dimensi γ : 0,000001-1.

c. Range pencarian ruang pada dimensi σ : 0,01-0,1.

d. Jumlah partikel : 10.

e. Jumlah iterasi maksimum : 10.

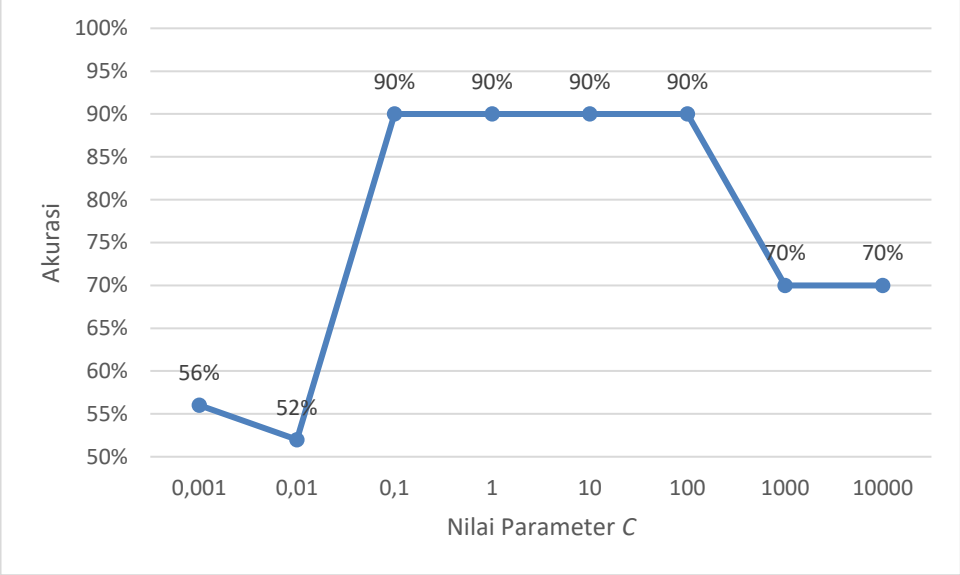
f. Konstanta akselerasi pertama dan kedua : 2 dan 2.

g. Jumlah iterasi α : 10.

Pengujian *range* ruang pada *C* akan dilakukan sebanyak 10 kali. Hasil pengujian ditunjukkan pada Tabel 6.3.

Tabel 6.3 Pengujian Pada *Range* Ruang Pencarian Dimensi *C*

C	Akurasi Pengujian Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,001	60%	40%	60%	60%	40%	60%	60%	60%	60%	60%	56%
0,01	60%	60%	60%	60%	60%	60%	40%	40%	40%	40%	52%
0,1	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
1	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
10	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
100	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
1000	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%
10000	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%



Gambar 6.3 Grafik Pengujian Untuk Pencarian Dimensi *C*

Berdasarkan pada grafik pengujian pencarian dimensi *C* yang ditunjukkan pada Gambar 6.3, pada percobaan pengujian nilai *C* 0,001, 0,01, 0,1, 1, 10, 100, 1000, dan 10000 mendapatkan akurasi sebesar 90%. Dapat disimpulkan bahwa nilai *C* yang semakin kecil akan memberikan pengaruh yang sangat kecil terhadap penalti

untuk meminimalkan nilai error dan ketika nilai *C* semakin besar maka nilai error dapat diminimalkan dan akurasi yang didapatkan semakin membaik dari hasil terburuk sebelumnya. Pada penelitian ini nilai akurasi yang optimal didapat pada rentang 0,1-100.

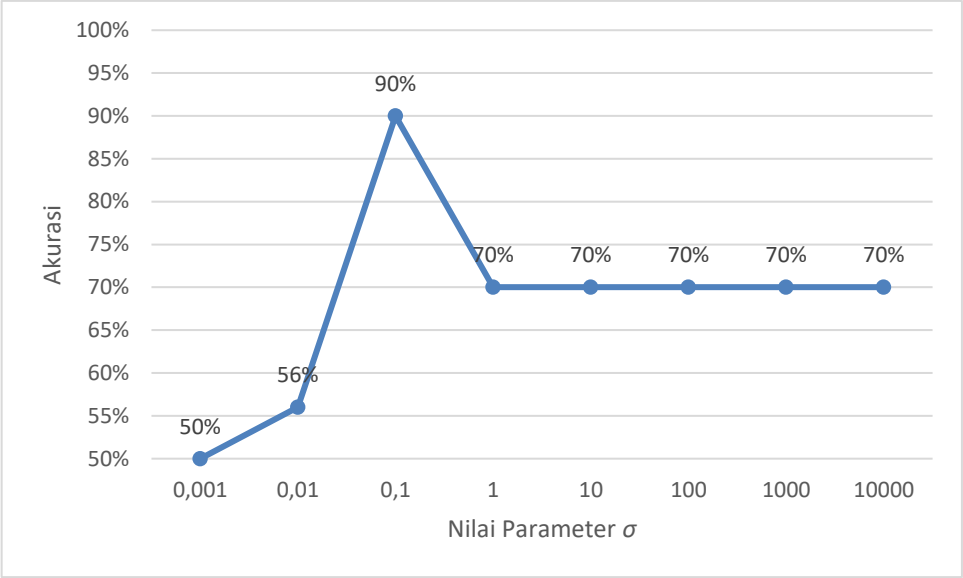
Selanjutnya melakukan pengujian pencarian dimensi varian *(σ),* dengan batas yang akan diuji adalah 0,001, 0,01, 0,1, 1, 10, 100, 1000 dan 10000. Berikut *range* dimensi parameter yang digunakan :

a. Range pencarian ruang pada dimensi λ : 0,1-100.
b. Range pencarian ruang pada dimensi γ : 0,000001-1.
c. Range pencarian ruang pada dimensi *C* : 0,1-100.
d. Jumlah partikel : 10.
e. Jumlah iterasi maksimum : 10.
f. Konstanta akselerasi pertama dan kedua : 2 dan 2.
g. Jumlah iterasi α : 10.

Pengujian *range* ruang pada *σ* akan dilakukan sebanyak 10 kali. Hasil pengujian ditunjukkan pada Tabel 6.4.

Tabel 6.4 Pengujian Pada *Range* Ruang Pencarian Dimensi *σ*

σ	Akurasi Pengujian Ke-*i*										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,001	40%	60%	60%	40%	40%	40%	60%	40%	60%	60%	50%
0,01	40%	60%	60%	60%	60%	60%	60%	60%	40%	60%	56%
0,1	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
1	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%
10	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%
100	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%
1000	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%
10000	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%



Gambar 6.4 Grafik Pengujian Untuk Pencarian Dimensi *σ*

Berdasarkan pada grafik pengujian pencarian dimensi *σ* yang ditunjukkan pada Gambar 6.4, pada percobaan pengujian nilai *sigma (σ)* 0,001, 0,01, 0,1, 1, 10, 100, 1000, dan 10000 mendapatkan nilai akurasi sebesar 90%. Dapat dilihat pada grafik bahwa dengan nilai parameter *sigma* 0,001 dan 0,01 akurasi yang didapatkan sangat rendah namun kemudian membaik ketika melewati 0,01 dan 0,1. Dan ketika melewati 0,1 nilai akurasi menurun lalu konvergen sampai pada nilai 10000. Ini membuktikan bahwa jika nilai *sigma* yang diberikan terlalu kecil akan terjadinya ovefit pada data latih dan jika bernilai besar akan membuat komputasi berjalan lebih lama untuk fungsi yang kompleks. Pada penelitian ini nilai akurasi yang optimal didapat pada rentang 0,01-0,1.

6.2 Pengujian Jumlah Iterasi Pelatihan *α*

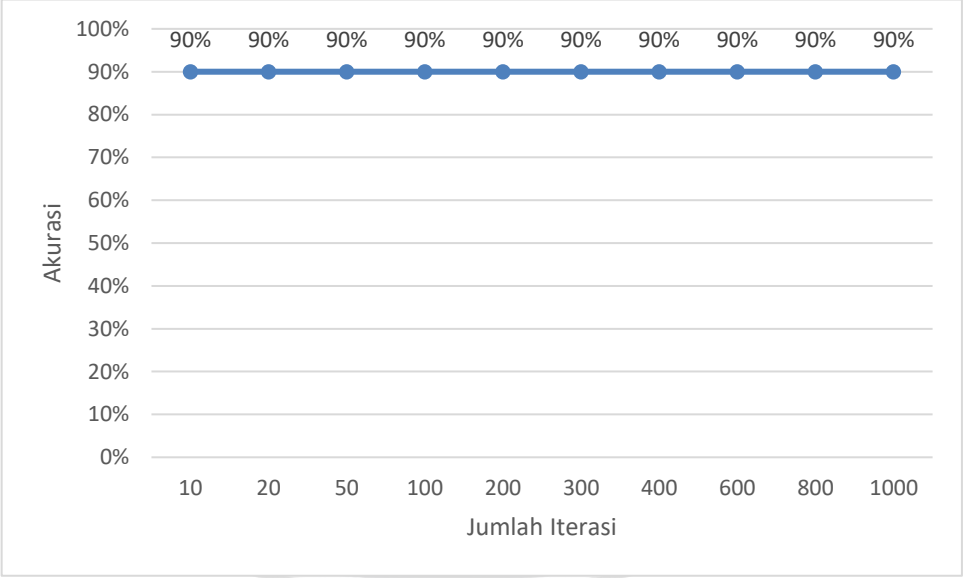
Pengujian ini dilakukan untuk dapat menentukan jumlah iterasi pelatihan *α* yang paling terbaik agar mendapatkan solusi terbaik dari tingkat akurasi yang dihasilkan. Jumlah iterasi yang digunakan yaitu 10, 20, 50, 100, 200, 300, 400, 600, 800, dan 1000. Berikut *range* parameter yang digunakan untuk menguji iterasi pelatihan *α* adalah sebagai berikut :

a. Range pencarian ruang pada dimensi *λ* : 0,1-100.
b. Range pencarian ruang pada dimensi γ : 0,000001-1.
c. Range pencarian ruang pada dimensi *C* : 0,1-100.
d. Range pencarian ruang pada dimensi *σ* : 0,01-0,1.
e. Jumlah partikel : 10.
f. Jumlah iterasi maksimum : 10.
g. Konstanta akselerasi pertama dan kedua : 2 dan 2.

Pengujian pada iterasi pelatihan α dilakukan sebanyak 10 kali. Hasil pengujian ditunjukkan pada Tabel 6.5.

Tabel 6.5 Hasil Pengujian Jumlah Iterasi Pelatihan α

Jumlah Iterasi	Akurasi Percobaan Ke-i										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
10	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
20	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
50	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
100	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
200	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
300	90%	90%	90%	90%	80%	90%	90%	90%	90%	90%	90%
400	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
600	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
800	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
1000	90%	90%	80%	90%	90%	90%	90%	90%	90%	90%	90%



Gambar 6.5 Grafik Pengujian Untuk Pencarian Dimensi α

Berdasarkan grafik pengujian untuk pencarian iterasi pelatihan α yang ditunjukkan pada Gambar 6.5, dengan menunjukkan akurasi terbaik didapatkan pada jumlah iterasi pelatihan α sebesar 10. Dengan iterasi pelatihan α yang semakin besar menunjukkan bahwa nilai akurasi tetap sama. Dapat dilihat dari perkembangan grafik yang memungkinkan semakin banyaknya iterasi pelatihan maka nilai akurasi yang dihasilkan sama.

6.3 Pengujian Kombinasi Nilai Konstanta Akselerasi

Pengujian pada nilai kombinasi nilai konstanta akselerasi ini digunakan untuk dapat memilih nilai kombinasi pada konstanta akselerasi yang terbaik sehingga menghasilkan solusi terbaik yang didasarkan pada tingkat akurasi yang dihasilkan.

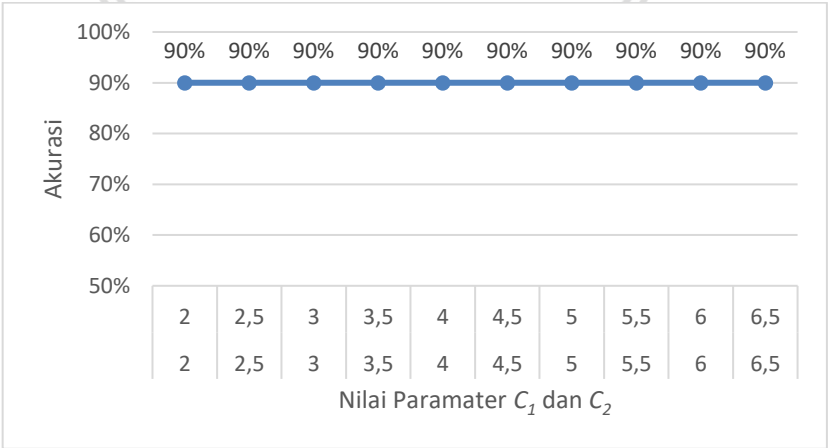
Nilai konstanta yang akan diuji yaitu 0, 0,5, 1, 1,5, 2, 2,5, 3, 3,5, 4, 4,5, dan 5 dari setiap konstanta akselerasi. Dan berikut adalah parameter yang digunakan pada uji coba kombinasi nilai konstanta akselerasi :

a. Range pencarian ruang pada dimensi λ : 0,1-100.
b. Range pencarian ruang pada dimensi γ : 0,000001-1.
c. Range pencarian ruang pada dimensi C : 0,1-100.
d. Range pencarian ruang pada dimensi σ : 0,01-0,1.
e. Jumlah partikel : 10.
f. Jumlah iterasi maksimum : 10.
g. Jumlah iterasi α : 10.

Pengujian pada kombinasi nilai konstanta akselerasi ini dilakukan sebanyak 10 kali. Hasil pengujian ditunjukkan pada Tabel 6.6.

Tabel 6.6 Hasil Pengujian Pada Kombinasi Nilai Konstanta Akselerasi

Nilai c_1	Nilai c_2	Akurasi Pengujian Ke-*i*										Rata-rata
		1	2	3	4	5	6	7	8	9	10	
2	2	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
2,5	2,5	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
3	3	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
3,5	3,5	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
4	4	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
4,5	4,5	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
5	5	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
5,5	5,5	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
6	6	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
6,5	6,5	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%



Gambar 6.6 Grafik Pengujian Untuk Pencarian Kombinasi Konstanta Akselerasi

Berdasarkan grafik pengujian untuk pencarian nilai kombinasi konstanta akselerasi yang ditunjukkan pada Gambar 6.6, akurasi yang terbaik didapatkan

pada c_1 = 2 dan c_2 = 2. Pada grafik menunjukkan bahwa semakin besar jumlah partikel maka nilai akurasi yang dihasilkan tetap sama. Semakin besar nilai partikel yang diinputkan nilai akurasi cenderung konvergen.

6.4 Pengujian Jumlah Partikel

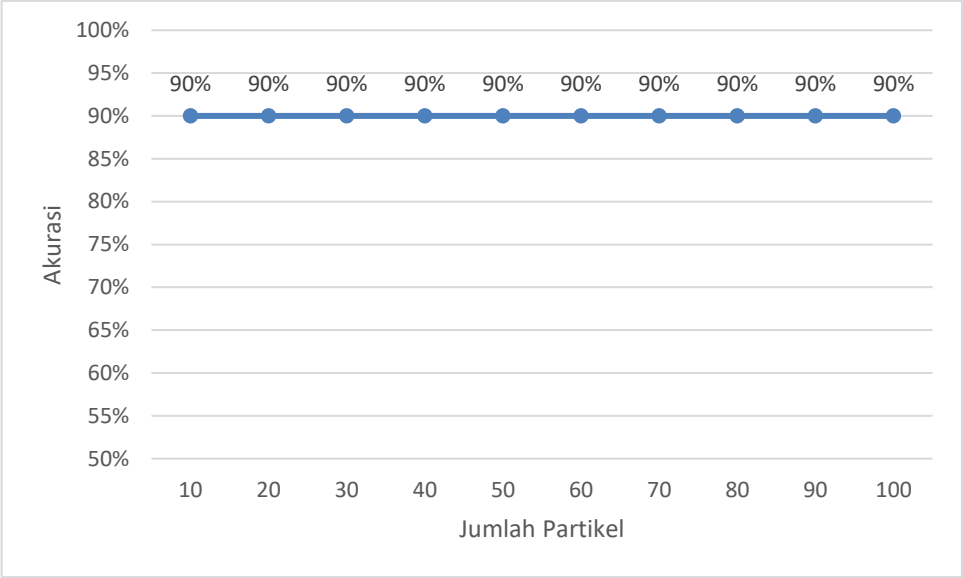
Pengujian pada jumlah partikel digunakan untuk dapat menentukan jumlah partikel yang terbaik berdasrakan tingkat akurasi yang dihasilkan. Jumlah partikel yang akan di uji yaitu 10, 20, 30, 40, 50, 60, 70, 80, 90, dan 100. Berikut *range* parameter yang digunakan untuk menguji coba jumlah partikel yaitu :

a. Range pencarian ruang pada dimensi λ : 0,1-100.
b. Range pencarian ruang pada dimensi γ : 0,000001-1.
c. Range pencarian ruang pada dimensi C : 0,1-100.
d. Range pencarian ruang pada dimensi σ : 0,01-0,1.
e. Jumlah iterasi maksimum : 10.
f. Konstanta akselerasi pertama dan kedua : 2 dan 2.
g. Jumlah iterasi α : 10.

Pengujian pada jumlah partikel ini akan dilakukan sebanyak 10 kali. Hasil pengujian ini ditunjukkan pada Tabel 6.7.

Tabel 6.7 Hasil Pengujian Jumlah Partikel

Jumlah Partikel	Akurasi Percobaan Ke-i										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
10	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
20	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
30	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
40	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
50	90%	80%	90%	90%	90%	90%	90%	90%	90%	90%	90%
60	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
70	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
80	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
90	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
100	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%



Gambar 6.7 Grafik Pengujian Untuk Pencarian Nilai Jumlah Partikel

Berdasarkan grafik pengujian untuk pencarian nilai jumlah partikel pada Gambar 6.7, akurasi terbaik didapatkan dari jumlah partikel sebanyak 10. Pada grafik menunjukkan bahwa semakin besar jumlah partikel maka nilai akurasi yang dihasilkan tetap sama, namun dapat membuat eksekusi program semakin melambat karena banyaknya jumlah partikel yang diinputkan. Semakin besar nilai partikel yang diinputkan nilai akurasi cenderung konvergen.

6.5 Pengujian Jumlah Iterasi Partikel

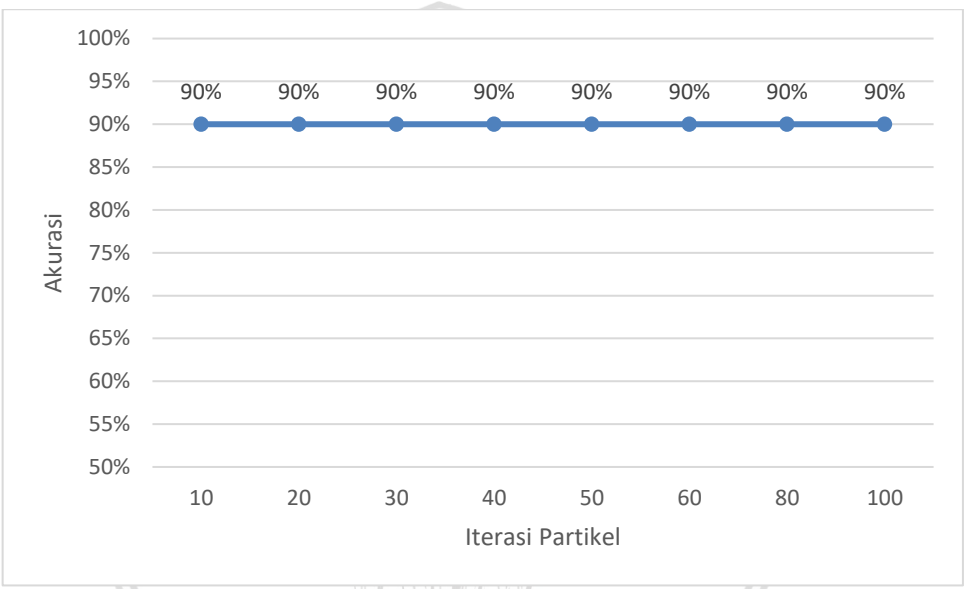
Pengujian pada jumlah iterasi partikel ini digunakan untuk dapat menentukan jumlah iterasi partikel terbaik agar dapat menghasilkan solusi terbaik berdasarkan pada tingkat akurasi yang dihasilkan. Jumlah dari iterasi yang akan diuji yaitu 10, 20, 30, 40, 50, 60, 80, dan 100. Berikut *range* parameter yang akan digunakan untuk melakukan uji coba dari jumlah iterasi partikel yaitu :

a. Range pencarian ruang pada dimensi λ : 0,1-100.
b. Range pencarian ruang pada dimensi γ : 0,000001-1.
c. Range pencarian ruang pada dimensi C : 0,1-100.
d. Range pencarian ruang pada dimensi σ : 0,01-0,1.
e. Jumlah partikel : 10.
f. Konstanta akselerasi pertama dan kedua : 2 dan 2.
g. Jumlah iterasi α : 10.

Pengujian pada jumlah iterasi partikel ini akan dilakukan sebanyak 10 kali. Hasil pengujian ini ditunjukkan pada Tabel 6.8.

Tabel 6.8 Hasil Pengujian Jumlah Iterasi Partikel

Jumlah Iterasi	Akurasi Percobaan Ke-i										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
10	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
20	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
30	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
40	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
50	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
60	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
80	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
100	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%



Gambar 6.8 Grafik Pengujian Untuk Pencarian Nilai Jumlah Iterasi Partikel

Berdasarkan grafik pengujian untuk pencarian nilai jumlah iterasi partikel pada Gambar 6.8, akurasi terbaik yang didapatkan dari jumlah iterasi partikel sebesar 10. Semakin besar nilai jumlah iterasi partikel maka semakin besar juga akurasi yang didapatkan. Namun dapat memperlambat program dalam menjalankan sistem karena banyaknya jumlah iterasi partikel yang dilakukan. Dilihat pada grafik semakin besar nilai jumlah iterasi partikel maka akurasi yang dihasilkan cenderung konvergen.

BAB 7 PENUTUP

Pada bab ini membahas mengenai proses pengujian dan kesimpulan dari pengujian optimasi pada SVM. Prsoses pengujian ini dilakukan dengan menguji nilai akurasi. Pengujian dari nilai akurasi ini digunakan untuk dapat mengukur tingkat akurasi dari sistem dengan membandingkan hasil keluaran dari sistem dan nilai hasil dataset Laboratorium.

7.1 Kesimpulan

Kesimpulan yang didapatkan berdasarkan hasil perancangan, implementasi, dan pengujian maka didapatkan kesimpulan sebagai berikut :

1. Untuk mengimplementasikan metode *Support Vector Machine* dan *Particle Swarm Optimization* (SVM-PSO) untuk klasifikasi pendonor darah dengan dataset (RFMTC) dengan melakukan beberapa tahap yakni, inisialisasi nilai partikel, normalisasi data, menghitung kernel, menghitung *sequential learning,* menghitung nilai bias, menghitung nilai akurasi klasifikasi dan menentukan nilai P_{best} dan nilai G_{best}. Tahap ini diawali dengan menentukan terlebih dahulu nilai partikel pada PSO untuk tiap parameter yang dibutuhkan dalam melakukan perhitungan SVM. Nilai partikel yang didapatkan dari proses PSO dijadikan sebagai nilai masukan untuk tiap parameter yang dibutuhkan SVM untuk mencari nilai *fitness* dan digunakan lagi pada proses PSO untuk mencari partikel terbaik. Dalam proses SVM dilakukan terlebih dahulu menormalisasi dataset dikarenakan nilai *range* antar data yang cukup jauh lalu menghitung nilai kernel dan melakukan proses perhitungan *sequential learning* SVM. Kemudian memperoleh nilai α baru dan nilai bias yang digunakan selanjutnya dalam mengolah data *testing.* Hasil akhir merupakan akurasi yang didapatkan dengan membandingkan nilai *class actual* dan *predicted class* pada SVM. Akurasi yang terbentuk dari perhitungan SVM dijadikan sebagai nilai *fitness* tiap partikel PSO. Dari setiap nilai *fitness* partikel PSO yang terbentuk kemudian ditentukan nilai P_{best} dan G_{best} yang digunakan sebagai keluaran terbaik dan hasil optimasi terbaik SVM dan PSO.
2. Berdasarkan pengujian yang dilakukan pada setiap parameter SVM dan PSO ini dapat disimpulkan bahwa akurasi terbaik sebesar 90% pada parameter dengan jumlah partikel PSO yang rendah dan dengan iterasi pelatihan SVM yang kecil, detail parameter yang diperoleh yaitu range pencarian dimensi *lambda (λ)* = 0,1-100, range pencarian dimensi *gamma (γ)* = 0,000001-1, range pencarian dimensi *complexity(C)* = 0,1-100, range pencarian dimensi varian *(σ)* = 0,01-0,1, jumlah partikel = 10, jumlah iterasi maksimum = 10, konstanta akselerasi pertama (c_1) = 2 dan kedua (c_2) = 2, dan jumlah iterasi pelatihan α = 10.

7.2 Saran

Berdasarkan hasil penelitian yang saya lakukan, adanya saran untuk melakukan penelitian lebih lanjut lagi yaitu :

1. Dengan membandingkan metode SVM-PSO ini dengan menggunakan *multiclass* dan dengan menambahkan metode yang lain sehingga dapat menambah referensi nilai akurasi terbaik dalam membandingkannya pada metode yang lain.
2. Sebaiknya untuk pengimplementasian dataset yang digunakan dalam penelitian selanjutnya menggunakan dataset rumah sakit setempat ataupun Palang Merah Indonesia agar dapat memprediksi dengan jelas perilaku pendonor darah sehingga dapat mencegah stok kekurangan kantung darah.

DAFTAR PUSTAKA

American Cancer Society. *Blood Transfusion and Donation.* Tersedia di: http://www.cancer.org [Diakses 3 Maret 2017]

Arafi, A., Fajr, R. & Bouroumi, A., 2014. *Breast Cancer Data Analysis Using Support Vector Machines and Particle Swarm Optimization. IEEE.* Morocco.

Darwiche, M., Feuilloy, M., Bousaleh, G., & Schang, D., 2010. *Prediction of Blood Transfusion Donation. IEEE,* 978.

Engelbrecht, A.P., 2007. *Computational Intelligence, An Introduction, Second edition*, JohnWiley & Sons Ltd., England.

Evelyn, C.A., 2006. *Anatomi dan Fisiologi untuk Paramedis.* Jakarta: Gramedia Pustaka Utama.

Gur Emre., et al., 2014. *Support Vector Machine Classification Based On Particle Swarm Optimization for Bone Age Determination. ScienceDirect. Engineering Faculty, Computer Engineering Dept., Konya, Turkey.*

Hassan, M., et al., 2013. *Face Recognition based on Opposition Particle Swarm Optimization and Support Vector Machine. IEEE International Conference on signal and image processing applications* (ICSIPA). Malaysia.

Haykin, S., 1999. *Neural Network: A Comprehensive Foundation*. Prentice Hall, New Jersey.

Hsu, Chih-Wei et al. 2004. *A Practical Guide to Support Vector Classification. Department of Computer Science and Information Engineering*, National Taiwan University.

Huang, Cheng, L., & Dun, J.F., 2008. *A distributed PSO–SVM hybrid system with feature selection and parameter optimization*. Huafan University, Taipei, Taiwan.

Kementerian Kesehatan Republik Indonesia. *Ketersediaan Darah Ditentukan Partisipasi Masyarakat Menjadi Donor*. Tersedia di: <www.depkes.go.id/article/print/16060300001/Ketersediaan-darah-ditentukan-partisipasi-masyarakat-menjadi-donor> [Diakses 1 Maret 2017]

Kennedy, j., & Eberhart R.C., 1995. *Particle Swarm Optimization. Proc. IEEE international conference on neural networks, vol. 4, pp. 1942-1948.*

Musyaffa N., & Rifai B., 2018. *Model Support Vector Machine Berbasis Particle Swarm Optimization Untuk Prediksi Penyakit Liver.* Sistem Informasi STMIK Nusa Mandiri. Jakarta.

Pusat Data dan Informasi Kementerian Kesehatan RI., 2014. *Situasi Donor Darah di Indonesia.*

Santosa, B., 2011. Metode Metaheuristik Konsep dan Implementasi. Guna Widya. Surabaya.

Scholkopf, B., & Smola A.J., 2002. *Learning with Kernels*. The MIT Press, Cambridge, Massachusetts.

Subasi, A., 2013. *Classification of EMG signals using PSO optimized SVM for diagnosis of neuromuscular disorders*. International Burch University. Sarajevo, Bosnia and Herzegovina.

Tomar, D., Prasad, B.R. & Agarwal, S., 2014. *An Efficient Parkinson Disease Diagnosis System Based on Least Squares Twin Support Vector Machine and Particle Swarm Optimization*. India.

Xu, X., et al. 2014. *Enhanced Support Vector Machine Using Parallel Particle Swarm Optimization*. *International Conferenceon Natural Computation.* China.

I-Cheng, Y., et al., 2008. *Knowledge discovery on RFM model using Bernoulli sequence. Expert Systems With Application.* China.

Shi, Y., & Eberhart, R.C., 1998. *Parameter selection in particle swarm optimization*. In V. W. Porto, N. Saravanan, D. Waagen, and A. Eibe, editors, *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, *page* 591-600. Springer-Verlag.

Vapnik, V.N., 1999. *An overview of statistical learning theory. Neural networks, IEEE transactions on vol. 10 no. 5, pp.* 1817-1824.

Wu, Q., Yan, H.S. & Yang, H.B., 2008. *A Forecasting Model Based Support Vector Machine and Particle Swarm Optimization. Workshop on Power Electronics and Intelligent Transportation System.* China.