TESIS – KI142502

PREDIKSI RELIABILITAS PERANGKAT LUNAK MENGGUNAKAN SUPPORT VECTOR REGRESSION DAN MODEL MINING

VIKA FITRATUNNANY INSANITTAQWA
5114201044

DOSEN PEMBIMBING
Dr. Ir. Siti Rochimah, MT.

PROGRAM MAGISTER
BIDANG KEAHLIAN REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

[Halaman ini sengaja dikosongkan]

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:

VIKA FITRATUNNANY INSANITTAQWA
Nrp. 5114201044

Dengan judul :
Prediksi Reliabilitas Perangkat Lunak Menggunakan Support Vector Regression dan Model Mining

Tanggal Ujian : 11-7-2017
Periode Wisuda : 2016 Genap

Disetujui oleh:

Dr. Ir. Siti Rochimah, MT. (Pembimbing 1)
NIP. 196810021994032001

Daniel Oranova Siahaan, S.Kom, MSc. PD.Eng. (Penguji 1)
NIP. 197411232006041001

Nurul Fajrin Ariyani, S.Kom, M.Sc (Penguji 2)
NIP. 198607222015042003

Adhatus Sholichah, S.Kom, M.Sc (Penguji 3)
NIP. 198508262015042002

Dekan Fakultas Teknologi Informasi ITS,

Dr. Agus Zainal Arifin, S. Kom., M. Kom.
NIP. 197208091995121001

[Halaman ini sengaja dikosongkan]

Prediksi Reliabilitas Perangkat Lunak Menggunakan Support Vector Regression dan Model Mining

Nama Mahasiswa : Vika Fitratunnany Insanittaqwa
NRP : 5114 201 044
Pembimbing : Dr. Ir. Siti Rochimah, MT.

ABSTRAK

Reliabilitas perangkat lunak didefinisikan sebagai probabilitas operasi perangkat lunak yang bebas dari kegagalan (*failure*) dalam sebuah periode waktu tertentu. Pemodelan reliabilitas perangkat lunak ini dapat dilakukan salah satunya dengan memanfaatkan data kegagalan perangkat lunak untuk melakukan prediksi kegagalan di masa datang. Salah satu arsitektur yang dipakai dalam pemodelan ini pada umumnya adalah dengan menggunakan beberapa data terakhir untuk melakukan prediksi. Padahal, kegagalan perangkat lunak dapat saja dipengaruhi oleh data yang terdahulu seperti yang telah dibuktikan pada satu penelitian, yang menggunakan teknik *model mining* untuk memilih data masukan terdahulu tersebut. Pada penelitian ini, diusulkan penggunaan Binary Particle Swarm Optimization (BPSO) sebagai metode *model mining* untuk melakukan prediksi reliabilitas perangkat lunak dengan menggunakan Support Vector Regression (SVR). Data yang dipakai atau tidak dipakai masing-masing disimbolkan dengan angka "1" atau "0" dan metode ini diujicobakan pada 6 data dari proyek perangkat lunak yang nyata, yaitu data FC1, FC2, FC3, TBF1, TBF2, dan TBF3. Keakuratan model yang diusulkan dibandingkan dengan prediksi yang tidak menggunakan *model mining* dengan mengukur nilai Mean Squared Error (MSE) dan Average Relative Prediction Error (AE). Metode SVR-BPSO yang diusulkan terbukti dapat menghasilkan prediksi yang lebih akurat, terutama untuk data FC1, FC2, dan FC3 yang bersifat stabil. Sifat data TBF yang berbeda dengan data FC menunjukkan bahwa data ini tidak cocok digunakan sebagai bahan uji coba metode yang diusulkan karena *time-between-failure* pada data tidak bergantung pada urutan kegagalan tertentu, seperti yang terlihat pada data TBF1, TBF2, dan TBF3. Pemilihan parameter SVR juga mempengaruhi keakuratan prediksi, dimana hal ini dapat diperbaiki pada penelitian selanjutnya. Secara umum, metode yang diusulkan telah dapat menghasilkan prediksi reliabilitas perangkat lunak dengan baik dan penggunaan *model mining* terbukti dapat memberikan manfaat yang nyata dalam bidang prediksi reliabilitas perangkat lunak.

Kata Kunci: *Binary Particle Swarm Optimization, Model Mining, Prediksi Reliabilitas Perangkat Lunak, Suppor Vector Regression.*

[Halaman ini sengaja dikosongkan]

Software Reliability Prediction based on Support Vector Regression and Model Mining

Student Name : Vika Fitratunnany Insanittaqwa
NRP : 5114 201 044
Supervisor : Dr. Ir. Siti Rochimah, MT.

ABSTRACT

Software reliability is defined as the pobability of failure-free software operation in certain period of time. The modelling of software reliability can be done in one way by using software failure data to predict the future failures. One architecture in this modelling is done generally by using the last few consecutive data to predict the future value, where actually the failure of a software can be dependent also to earlier data as showed in one research about the use of model mining to determine which data to use as prediction. In this research, we propose the use of Binary Particle Swarm Optimization (BPSO) as a model mining method to predict the reliability of software by using Support Vector Regression (SVR) as predictor. To determine which data to use in model mining, the data is symbolized with one "1" or "0" in the structure of BPSO particle. The proposed method is tested with 6 real data from real project, which are called FC1, FC2, FC3, TBF1, TBF2, and TBF3. The accuracy of the proposed model is compared with a predictor without model mining by computing the Mean Squared Error (MSE) and Average Relative Prediction Error (AE). The proposed SVR-BPSO method is proved to be able to predict more accurately, especially in FC1, FC2, and FC3 data which are more stable in nature. The use of TBF data sets proved to be inappropriate as it yields poor prediction results in TBF1, TBF2, and TBF3 data, which may have rooted from the differing nature with FC data. The method to choose SVR parameters can also affect the accuracy of prediction, which opens room for improvement in future research. In general, the proposed method is able to predict the reliability of a software and the use of model mining is important in effort to produce more accurate prediction in software failure data.

Keywords: *Binary Particle Swarm Optimization, Model Mining, Software Reliability Prediction, Suppor Vector Regression*.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT karena atas rahmat dan hidayah-Nya penulis dapat menyelesaikan menyelesaikan Tesis ini yang berjudul **"**Prediksi Reliabilitas Perangkat Lunak Menggunakan Support Vector Regression dan Model Mining**"**.

Tesis ini juga tidak akan terselesaikan tanpa bantuan dari berbagai pihak. Ucapan terima kasih dan penghormatan yang sebesar-besarnya penulis sampaikan kepada:

1. Allah SWT atas segala rahmat dan hidayah-Nya.
2. Kedua orang tua penulis atas doa, dukungan, dan kasih sayang yang tak terhingga.
3. Kedua adik kandung penulis yang selalu dapat memberikan tawa dan semangat.
4. Ibu Dr. Ir. Siti Rochimah, MT. selaku dosen pembimbing yang telah memberikan bimbingan, arahan, dan motivasi untuk membantu pengerjaan Tesis ini.
5. Bapak Daniel Oranova, S.Kom, MSc. PD.Eng., Ibu Nurul Fajrin Ariyani, S.Kom, M.Sc, dan Ibu Adhatus Sholichah, S.Kom, M.Sc selaku dosen penguji yang telah memberikan saran, arahan, dan koreksi dalam pengerjaan Tesis ini.
6. Bapak dan Ibu dosen Jurusan Teknik Informatika ITS yang telah banyak memberikan ilmu dan bimbingan yang tak ternilai harganya.
7. Seluruh staf dan karyawan Teknik Informatika ITS yang telah banyak memberikan kelancaran administrasi akademik.
8. Rekan-rekan Pascasarjana Teknik Informatika ITS terutama angkatan 2014 atas segala dorongan semangat dan kebersamaan semasa perkuliahan.
9. Pihak-pihak lain yang tidak sempat penulis sebutkan pada lembaran ini.

Penulis berharap Tesis ini dapat memberikan manfaat untuk kedepannya, namun penulis sadar bahwa Tesis ini masih jauh dari sempurna. Penulis mohon maaf sebesar-besarnya apabila terjadi kekurangan pada Tesis ini. Semoga kritik dan saran yang membangun dapat digunakan untuk perbaikan di masa yang akan datang.

Surabaya, Juli 2017

Vika Fitratunnany Insanittaqwa

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1. Latar Belakang

Komputer telah menjadi bagian penting pada aktifitas keseharian manusia. Salah satu komponen komputer adalah perangkat lunak, dimana perangkat lunak ini harus dapat menjalankan fungsinya tanpa terjadi kesalahan ataupun kegagalan. Untuk memastikan hal ini, proses pengujian perangkat lunak perlu dilakukan. Dalam proses ini, kesalahan (*fault*) ataupun kegagalan (*failure*) perangkat lunak harus ditemukan agar dapat dilakukan perbaikan. Hal ini bertujuan agar perangkat lunak yang diberikan pada pengguna adalah perangkat lunak yang memiliki tingkat reliabilitas yang tinggi.

Reliabilitas perangkat lunak didefinisikan sebagai probabilitas operasi perangkat lunak yang bebas dari kegagalan dalam sebuah periode waktu yang ditentukan pada lingkungan tertentu (American Institute of Aeronautics and Astronautics, 1993). Pada siklus hidup perangkat lunak, pengukuran reliabilitas perangkat lunak memiliki beberapa manfaat. Salah satu contohnya adalah pada tahap pengujian, dimana pengukuran reliabilitas perangkat lunak dapat digunakan sebagai acuan pengembang untuk memperkirakan seberapa besar sumber daya yang diperlukan untuk melakukan pengujian yang optimal. Dalam dunia penelitian, pemodelan reliabilitas perangkat lunak adalah topik yang aktif dan diminati karena pentingnya peranan pengukuran reliabilitas tersebut.

Terdapat dua macam pendekatan pemodelan reliabilitas perangkat lunak dalam literatur secara umum, yaitu Analytical Software Reliability Modeling (ASRM) atau pendekatan analitis dan Data-Driven Software Reliability Modeling (DDSRM) atau pendekatan data (Hu dkk., 2006). Dalam literatur, pendekatan ASRM banyak dikembangkan pada penelitian dengan pemodelan Software Reliability Growth Model (SRGM), dimana terdapat lebih dari 100 model yang telah diusulkan hingga saat ini semenjak pertama diperkenalkan pada tahun 1972 (Lyu, 2007). Namun, pendekatan ini dinilai memiliki beberapa kelemahan, salah

satunya adalah tidak adanya sebuah model universal yang dapat diterapkan pada semua situasi (Goel, 1985). Hal ini disebabkan karena pemodelan SRGM yang didasarkan pada beberapa asumsi yang bergantung pada jenis sistem tertentu.

Untuk mengatasi kelemahan ini, pendekatan DDSRM mulai banyak ditelusuri karena pendekatan ini hanya mengandalkan data kesalahan atau kegagalan perangkat lunak tanpa disertai asumsi yang terlalu ketat. Terdapat banyak metode yang diusulkan, misalnya Bayesian networks, model Autoregressive Integrated Moving Average (ARIMA) untuk *time series*, dan lain-lain. Diantara metode-metode tersebut, metode Machine Learning, seperti Artificial Neural Network (ANN) dan Support Vector Machine (SVM), merupakan metode yang populer dan banyak dikembangkan dalam penelitian. Penerapan ANN untuk melakukan prediksi reliabilitas perangkat lunak menuai banyak perhatian semenjak dilakukan pertama kali pada tahun 1991 (Karunanithi dkk., 1991) dan telah banyak pengembangan yang dilakukan untuk mendapatkan hasil prediksi yang lebih baik. Namun, metode ini dinilai memiliki beberapa kelemahan, diantaranya adalah sulitnya mendapatkan hasil yang stabil dan ancaman terjadinya *over-fitting*. Untuk mengatasi hal ini, SVM untuk regresi atau Support Vector Regression (SVR) mulai diterapkan dan mulai mendapatkan perhatian pada tahun 2005 sebagai metode yang dinilai lebih baik daripada ANN (Tian & Noore, 2005c). Beberapa teknik optimasi juga telah diusulkan untuk memperbaiki hasil prediksi SVR, diantaranya Simulated Annealing (SA) (Pai & Hong, 2006), Genetic Algorithm (GA) (Ho & Hsieh, 2009), metode hibrida GA-SA (Jin, 2011), Particle Swarm Optimization (PSO) (Qin, 2011), dan Improved Estimation of Distribution Algorithm (Jin & Jin, 2014).

Selain dari metode yang digunakan, pendekatan DDSRM juga dapat dibedakan dari arsitektur modelnya (Tian & Noore, 2005b) (Su & Huang, 2007). Dua macam arsitektur yang umumnya dipakai pada DDSRM adalah Single-Input Single-Output (SISO) dan Multiple-Delayed-Input Single-Output (MDISO). Karakteristik arsitektur SISO dapat dilihat dari pemodelan hubungan sebuah masukan x_t pada waktu t dengan sebuah pasangan keluaran y_t. Misalnya, hubungan waktu eksekusi kumulatif sebagai masukan dan jumlah akumulasi kegagalan perangkat lunak sebagai keluaran. MDISO, di lain pihak, mencoba

memodelkan hubungan diantara data kegagalan perangkat lunak, dimana data masukan yang digunakan umumnya berupa data *lagged time series* $(x_{t-1}, x_{t-2}, \dots, x_{t-z})$ dengan menggunakan z data sebagai masukan. Hal ini menyebabkan MDISO dianggap memiliki nilai yang lebih daripada model SISO biasa karena pada kenyataannya, reliabilitas perangkat lunak dipengaruhi oleh kondisi perangkat lunak di waktu yang lalu.

Jumlah masukan pada arsitektur MDISO dapat ditentukan dengan proses "trial-and-error" atau dari pengalaman sebelumnya. Pada awalnya, penelitian MDISO menggunakan asumsi bahwa kegagalan perangkat lunak hanya bergantung pada data kegagalan yang paling akhir saja. Namun, kemudian ditemukan bahwa kegagalan perangkat lunak dapat juga dipengaruhi oleh data yang terdahulu karena korelasi pada model *time series* dapat terjadi lebih rumit. Hal ini dibuktikan dengan penelitian dengan menggunakan metode *model mining* untuk menentukan data yang dipakai untuk melakukan prediksi (Yang dkk., 2010), dimana pada penelitian tersebut terbukti bahwa metode *model mining* dapat digunakan untuk mencari hubungan *time lags* yang terdahulu untuk mendapatkan hasil prediksi yang lebih akurat daripada data masukan yang mengambil data paling akhir saja.

Meskipun hal ini dinilai sebagai temuan yang baru, tidak banyak penelitian yang dilakukan untuk mengembangkan temuan ini lebih jauh. Padahal, terdapat kesempatan untuk menyelidiki apakah metode lain dapat diterapkan untuk melakukan *model mining* pada data kegagalan perangkat lunak. Binary Particle Swarm Optimization (BPSO) adalah algoritma pengembangan PSO yang dikhususkan untuk masalah diskrit. PSO itu sendiri adalah algoritma pencarian yang optimal dalam ruang solusi yang bersifat heuristik, dimana proses pencarian tersebut terinspirasi dari kumpulan partikel yang bergerak dengan arah masing-masing. Pada proposal ini, diusulkan sebuah penelitian mengenai prediksi reliabilitas perangkat lunak dengan menggunakan SVR dengan *model mining* yang dilakukan dengan algoritma BPSO. Metode BPSO dipilih karena kemampuannya dalam mencari solusi diskrit yang dapat diterapkan pada masalah *model mining*, dimana data yang dipakai atau tidak dipakai masing-masing dapat disimbolkan dengan, misalnya, angka "1" atau "0". Penggunaan metode ini

dimaksudkan untuk menemukan *time lags* terbaik yang dipakai untuk prediksi reliabilitas perangkat lunak.

Pada akhir penelitian ini, hasil yang ditemukan diharapkan dapat menunjukkan bahwa BPSO dapat digunakan sebagai metode *model mining* untuk melakukan prediksi reliabilitas perangkat lunak dengan menggunakan SVR. Penggunaan BPSO sebagai metode *model mining* juga diharapkan mampu menghasilkan hasil prediksi yang lebih akurat daripada prediksi yang dilakukan hanya dengan SVR saja.

1.2. Perumusan Masalah

Rumusan masalah pada penelitian ini adalah sebagai berikut:

1. Bagaimana BPSO dapat digunakan sebagai metode *model mining* pada data kegagalan perangkat lunak?
2. Berapa nilai keakuratan prediksi reliabilitas perangkat lunak dengan menggunakan SVR dan BPSO sebagai *model mining*?
3. Apakah penerapan BPSO sebagai *model mining* dapat menghasilkan nilai prediksi yang lebih akurat daripada prediksi yang dilakukan tanpa menggunakan *model mining*?

1.3. Tujuan

Penelitian ini bertujuan untuk menyelidiki apakah BPSO dapat digunakan dalam proses *model mining* dalam prediksi reliabilitas perangkat lunak dan bagaimana cara menerapkan metode tersebut. Selain itu, penelitian ini juga bertujuan untuk menyelidiki apakah *model mining* dengan menggunakan BPSO menghasilkan prediksi yang lebih akurat daripada prediksi tanpa menggunakan *model mining*.

1.4. Manfaat

Penelitian ini diharapkan dapat memberikan manfaat, terutama untuk para praktisi dan akademisi. Bagi para praktisi, khususnya pengembang perangkat lunak, penelitian ini dapat memberikan sumbangsih berupa acuan mengenai prediksi reliabilitas perangkat lunak yang tentunya dapat berguna pada tahap

pengujian perangkat lunak ataupun pada tahap operasi perangkat lunak. Bagi para akademisi, penelitian ini dapat memberikan manfaat berupa temuan baru dalam bidang DDSRM dengan arsitektur MDISO yang menggunakan *model mining* untuk mendapatkan data masukan.

1.5. Kontribusi Penelitian

Penelitian ini memberikan kontribusi dalam bidang prediksi reliabilitas perangkat lunak dengan pendekatan DDSRM dengan arsitektur MDISO dengan *model mining*. Lebih khusus lagi, penelitian ini memberikan kontribusi dengan penerapan algoritma BPSO sebagai metode baru dalam melakukan *model mining* dalam prediksi reliabilitas perangkat lunak dengan menggunakan SVR.

1.6. Batasan Masalah

Batasan masalah pada penelitian ini adalah sebagai berikut.

1. Arsitektur model reliabilitas perangkat lunak yang digunakan pada penelitian ini menggunakan MDISO dengan jumlah maksimal *sliding window* (jumlah data masukan) yang ditentukan sebelumnya. Penelitian mengenai pencarian jumlah *sliding window* yang optimal tidak akan diselidiki pada penelitian ini.
2. Penelitian ini menggunakan pustaka LIBSVM untuk menjalankan SVR dalam lingkungan pemrograman Matlab untuk mempermudah proses implementasi.

[Halaman ini sengaja dikosongkan]

BAB II
KAJIAN PUSTAKA DAN DASAR TEORI

2.1. Prediksi Reliabilitas Perangkat Lunak

Menurut American National Standard Institute (ANSI), definisi reliabilitas perangkat lunak adalah probabilitas operasi perangkat lunak yang bebas dari kegagalan (*failure*) dalam sebuah periode waktu yang ditentukan pada lingkungan tertentu (American Institute of Aeronautics and Astronautics, 1993). Karena dewasa ini perangkat lunak yang ada memiliki ukuran dan kompleksitas yang tinggi, muncul kebutuhan bagi pengembang perangkat lunak untuk dapat memprediksikan reliabilitas perangkat lunak. Manfaat dari prediksi ini antara lain adalah untuk menentukan apakah program telah siap dipasarkan pada konsumen dan berapa lama pengujian yang dibutuhkan. Prediksi ini juga berguna untuk menaksir jumlah kegagalan yang mungkin terjadi di lingkungan pengguna setelah program dijalankan. Selain itu, prediksi ini dapat membantu menentukan jumlah bantuan yang dibutuhkan untuk memperbaiki *defect* setelah perangkat lunak dipasarkan (Wood, 1996).

Dalam literatur, penelitian dalam bidang ini berfokus pada bagaimana mendesain sebuah model yang dapat secara akurat memprediksikan reliabiltias perangkat lunak dalam hal jumlah kegagalan, kecepatan kegagalan, atau intensitas kegagalan. Terdapat dua macam pendekatan pemodelan reliabilitas perangkat lunak secara umum, yaitu Analytical Software Reliability Modeling (ASRM) atau pendekatan analitis dan Data-Driven Software Reliability Modeling (DDSRM) atau pendekatan data (Hu dkk., 2006). Pendekatan ASRM menggunakan fungsi matematika dengan asumsi pada sistem tertentu untuk mendeskripsikan kegagalan perangkat lunak sehingga reliabilitas dapat diprediksi. Namun, model ini memiliki kelemahan karena tidak ada sebuah model yang dapat diterapkan secara efektif pada sistem yang berbeda. Pendekatan DDSRM kemudian diusulkan untuk memprediksikan reliabilitas perangkat lunak dari data kegagalan perangkat lunak yang telah lalu tanpa disertai asumsi pada sistem. Metode yang sering digunakan

pada pendekatan ini lebih banyak menggunakan teknik *soft computing* dengan regresi atau analisis *time series*.

2.2. ASRM dan DDSRM

Kegagalan dalam perangkat lunak dapat terjadi pada proses eksekusinya, baik itu dalam tahap pengujian maupun dalam tahap operasi. Pada tahap pengujian, terdapat waktu yang dibutuhkan untuk mendeteksi dan menghilangkan *defect* yang menyebabkan kegagalan dan reliabilitas perangkat lunak memiliki *trend* yang tumbuh sepanjang waktu (*growth*) (Cai dkk., 2001). Oleh karena itu, pemodelan reliabilitas secara analitis menghasilkan sebuah model yang bernama Software Reliability Growth Model (SRGM). Hal ini juga berkaitan dengan reliabilitas perangkat lunak yang tumbuh hingga waktu *release* (Karunanithi dkk., 1992c). Dalam literatur, banyak sekali ditemukan penelitian mengenai SRGM sejak diperkenalkan pertama kali pada 1972, dimana terdapat lebih dari 100 model yang telah diusulkan hingga saat ini (Lyu, 2007). Model ini dapat diklasifikasikan menjadi dua kelompok, yaitu model "time between failures" dan "failure count". Asumsi-asumsi yang dipakai dalam pemodelan bermacam-macam. Namun, asumsi yang paling penting adalah sebagai berikut (Amin dkk., 2013).

1. Waktu antara kegagalan yang independen.
2. Perbaikan yang seketika saat kesalahan terdeteksi.
3. Perbaikan kesalahan tanpa menghasilkan kesalahan yang baru.

Meskipun SRGM telah menyediakan cara untuk menilai, menaksir dan memprediksikan reliabilitas perangkat lunak, pendekatan ini memiliki beberapa kelemahan. Tidak ada sebuah model yang dapat diterapkan secara efektif pada sistem yang berbeda (Goel, 1985). Hal ini dapat diatasi dengan menggunakan beberapa model sekaligus untuk mendapatkan hasil prediksi yang lebih baik. Namun, proses ini memakan banyak waktu dan membutuhkan tenaga ahli untuk menentukan model yang tepat untuk lingkungan tertentu.

Pendekatan DDSRM diusulkan untuk mengatasi batasan pada ASRM. Prinsip utama pada pendekatan ini adalah dengan melakukan prediksi berdasarkan data kesalahan atau kegagalan perangkat lunak tanpa disertai asumsi yang terlalu ketat. Beberapa metode yang diusulkan untuk mengatasi batasan asumsi pada

ASRM adalah penggunaan *non-parametric statistics* (Robinson & Dietrich, 1987) (Barghout & Abdel-Ghaly, 1998) dan Bayesian Networks (Bai dkk., 2005) (Wiper dkk., 2012). Namun, metode-metode tersebut tidak dapat menjawab masalah tentang *applicability*. Terdapat pula prediksi dengan menggunakan analisa *time series*, terutama dengan menggunakan model ARIMA. Namun, metode ini masih berada pada tahap awal dan masih memiliki beberapa batasan (Amin dkk., 2013), salah satunya adalah ARIMA membutuhkan asumsi tersendiri pada data yang digunakan untuk membangun model yang benar.

Metode lainnya yang populer adalah dengan menggunakan teknik *machine learning*, seperti ANN dan SVM, untuk menjawab masalah *applicability* SRGM. Definisi permasalahan yang dipakai pada awalnya adalah sebagai berikut: jika terdapat data waktu eksekusi kumulatif $(x_1, x_2, \cdots, x_k) \in X_k(t)$, data *fault* yang diamati pada waktu tersebut $(y_1, y_2, \cdots, y_k) \in Y_k(t)$ hingga saat ini (waktu *t*), dan Δ adalah waktu eksekusi kumulatif dari x_{k+1} hingga x_{k+h}, maka prediksikan jumlah *fault* kumulatif $y_{k+h}(t + \Delta)$ pada waktu mendatang $k + h, x_{k+h}(t + \Delta)$. Definisi ini kemudian berkembang sepanjang perkembangan penelitian dengan penggunaan data masukan dan keluaran yang berbeda. Dalam hal jangkauan prediksi, batasan prediksi h harus ditentukan untuk menentukan seberapa jauh prediksi dilakukan. Untuk $h = 1$, prediksinya dinamakan "next-step-prediction" (atau "short-term prediction"). Untuk $h = n(\geq 2)$, prediksinya dinamakan "*n*-step-ahead prediction" (atau "long-term prediction") (Karunanithi dkk., 1992c). Dalam literatur lain, selain dua macam prediksi tersebut (yang disebut juga "variable-term prediction"), terdapat pula "end-point prediction" yang hanya memprediksikan kapan sistem menjadi stabil (Sitte, 1999).

Penerapan ANN untuk melakukan prediksi reliabilitas perangkat lunak menuai banyak perhatian semenjak dilakukan pertama kali pada tahun 1991 (Karunanithi dkk., 1991), dimana ditemukan bahwa Feed Forward ANN dapat melakukan prediksi lebih akurat dan memiliki tingkat *applicability* yang lebih baik daripada SRGM. Penelitian ini kemudian dilanjutkan dengan tiga penelitian lanjutan yang menunjukkan bahwa Jordan's NN dan Elman's NN juga dapat digunakan untuk melakukan prediksi yang lebih baik (Karunanithi & Whitley, 1992) (Karunanithi dkk., 1992b). Fuzzy NN kemudian diterapkan sebagai metode

untuk melakukan prediksi reliabilitas (Adnan & Yaacob, 1994). Metode NN juga telah dibandingkan dengan metode *parametric recalibration* dan ditemukan bahwa NN lebih unggul (Sitte, 1999). Non-parametric NN kemudian dibandingkan dengan metode regresi pada penelitian selanjutnya (Aljahdali dkk., 2001). Recurrent NN dibuktikan memiliki akurasi lebih baik daripada Feed Forward NN (Ho dkk., 2003). Beberapa teknik optimasi juga telah diusulkan untuk hasil prediksi yang lebih baik, yaitu Pseudoinverse Learning Algorithm (Guo & Lyu, 2004), GA (Tian & Noore, 2005) (Tian & Noore, 2005b), Fuzzy Min-Max Algorithm (Zemouri & Zerhouni, 2012), dan Imperialist Competitive Algorithm (Noekhah dkk., 2013). Beberapa model *ensemble* juga telah diusulkan dengan mengkombinasikan beberapa NN untuk mendapatkan hasil yang lebih akurat (Kiran & Ravi, 2008) (Zheng, 2009) (Li dkk., 2013). Meskipun ANN dinilai dapat melakukan pemodelan non-linear dengan mempelajari hubungan yang kompleks antara masukan dan keluaran tanpa harus memiliki pengetahuan tentang kondisi sistem, metode ini dinilai memiliki beberapa kelemahan. Misalnya, metode ini sulit mendapatkan hasil yang stabil dan memiliki ancaman terjadinya *over-fitting*.

Untuk mengatasi hal ini, SVM untuk regresi atau SVR mulai diterapkan dan mulai mendapatkan perhatian pada tahun 2005 (Tian & Noore, 2005c). Beberapa teknik optimasi juga telah diusulkan untuk memperbaiki hasil prediksi SVR, diantaranya Simulated Annealing (SA) (Pai & Hong, 2006), Genetic Algorithm (GA) (Ho & Hsieh, 2009), metode hibrida GA-SA (Jin, 2011), Particle Swarm Optimization (PSO) (Qin, 2011), dan Improved Estimation of Distribution Algorithm (Jin & Jin, 2014).

2.3. SISO dan MDISO

Selain dari metode yang digunakan, pendekatan DDSRM juga dapat dibedakan dari arsitektur modelnya (Tian & Noore, 2005b) (Su & Huang, 2007). Dua macam arsitektur yang umumnya dipakai pada DDSRM adalah Single-Input Single-Output (SISO) dan Multiple-Delayed-Input Single-Output (MDISO). Karakteristik arsitektur SISO dapat dilihat dari pemodelan hubungan sebuah masukan x_t pada waktu t dengan sebuah pasangan keluaran y_t. Misalnya,

hubungan waktu eksekusi kumulatif sebagai masukan dan jumlah akumulasi kegagalan perangkat lunak sebagai keluaran. MDISO, di lain pihak, mencoba memodelkan hubungan diantara data kegagalan perangkat lunak, dimana data masukan yang digunakan umumnya berupa data *lagged time series*. Hal ini menyebabkan MDISO dianggap memiliki nilai yang lebih daripada model SISO biasa karena pada kenyataannya, reliabilitas perangkat lunak dipengaruhi oleh kondisi perangkat lunak tersebut di waktu yang lalu. Sehingga, data masukan pengukuran reliabilitas perangkat lunak juga harus mempertimbangkan data kegagalan di waktu yang lalu.

Pada awalnya, penelitian dengan arsitektur MDISO mendefinisikan bahwa kegagalan x_i memiliki korelasi dengan beberapa kegagalan *z* yang paling akhir, yaitu $(x_{i-1}, x_{i-2}, \dots, x_{i-z})$, dimana *z* adalah dimensi vektor masukan (jumlah node masukan pada ANN atau SVM) yang terkadang disebut juga sebagai ukuran "sliding window", "fixed-length of moving window", atau "order of autoregressive terms". Penelitian terdahulu menggunakan beberapa nilai *z* dengan proses "trial-and-error" atau dari pengalaman sebelumnya yang berbeda-beda. Dua penelitian (Hu dkk., 2006) (Ho & Hsieh, 2009) menggunakan nilai z sebanyak 3. Tiga penelitian (Aljahdali dkk., 2001) (Benaddy dkk., 2011) (Lo, 2010) menggunakan nilai *z* sebanyak 4. Sitte (Sitte, 1999) menggunakan beberapa nilai *z*, yaitu 2, 4, 8, dan 16. Cai, dkk. (Cai dkk., 2001) menggunakan nilai 10, 20, 30, 40, dan 50. Li, dkk. (Li dkk., 2013) menggunakan nilai 2, 3, 4, dan 5. Yang dan Li (Yang & Li, 2007) menggunakan nilai 3, 4, dan 17. Jin dan Jin (Jin & Jin, 2014) menggunakan nilai 3 dan 4. Penelitian lain mencari jumlah z yang terbaik dengan algoritma yang berbeda-beda (Tian & Noore, 2005) (Tian & Noore, 2005b).

Penelitian-penelitian tersebut menggunakan masukan yang bersifat *delayed* dengan ukuran yang berbeda untuk mendapatkan hasil prediksi yang lebih akurat. Namun, jika dicermati lebih jauh, penelitian tersebut menggunakan asumsi bahwa kegagalan perangkat lunak hanya bergantung pada data kegagalan yang paling akhir saja. Padahal, korelasi pada model *time series* dapat terjadi lebih rumit. Sehingga, sebuah kegagalan perangkat lunak dimungkinkan dapat berkorelasi dengan beberapa kegagalan yang terdahulu.

Hal ini kemudian dibuktikan dengan penelitian lainnya bahwa kegagalan dapat memiliki korelasi dengan data yang terdahulu. Penelitian Yang dkk (Yang dkk., 2010) menunjukkan bahwa masukan pada MDISO belum tentu harus menggunakan *time lags* yang paling akhir. Data *time lags* yang digunakan pada model MDISO juga dapat diambil dari data terdahulu. Penentuan *time lags* terbaik yang dapat digunakan dalam proses prediksi dapat dilakukan dengan proses yang disebut *model mining.* Penemuan tersebut juga menunjukkan bahwa *model mining* dapat menghasilkan prediksi yang lebih akurat daripada data masukan paling akhir saja. Berdasarkan hasil penelitian ini, sebuah kegagalany_i, misalnya, dapat berkorelasi dengan x_{i-8}, x_{i-6}, dan x_{i-2}. Sehingga, pemodelan MDISO yang lebih umum dapat dituliskan seperti Persamaan (1).

$$y_i = F(x_{i-m_1}, x_{i-m_2}, \dots, x_{i-m_p}) \quad (1)$$

Pada Persamaan (1), y_i adalah sebuah observasi proses kegagalan perangkat lunak; $(x_{i-m_1}, x_{i-m_2}, \dots, x_{i-m_p})$ adalah *time lag* yang diambil pada *time series*; dan $F(\cdot)$ adalah model *time series*. Untuk melakukan sebuah prediksi, perlu ditentukan pula *time lag* yang akan digunakan pada $F(\cdot)$ untuk menentukan nilai *p* dan *m*. Dengan model ini, *time lag* yang digunakan pada prediksi belum tentu berasal dari data yang terakhir dan dapat berasal dari data yang terdahulu. Nilai *z* dan *time lags* yang terbaik dalam sebuah model dapat ditentukan dengan bantuan teknik kecerdasan komputasional.

2.4. Support Vector Regression

SVM pertama kali diusulkan berdasarkan teori pembelajaran statistika (Vapnik, 1995). Teknik ini memiliki prinsip untuk meminimalkan resiko structural untuk meminimalkan batas atas error pada generalisasi daripada meminimalkan *error* proses *training* seperti ANN. Dengan prinsip ini, SVM dapat mencapai struktur jaringan yang optimal dan memiliki generalisasi yang lebih baik daripada ANN. SVM pada awalnya dikembangkan untuk menyelesaikan masalah pengenalan pola. Namun, dengan adanya fungsi "ε-insensitive loss", SVM dapat diperluas untuk menyelesaikan masalah penaksiran regresi. Teknik ini juga dikenal dengan nama Support Vector Regression (SVR) dan telah

menunjukkan peforma yang sangat baikdalam memecahkan masalah regresi (Vapnik dkk., 1997).

SVR dapat menyelesaikan masalah prediksi linear atau non-linear dengan memetakan data masukan *x* ke high-dimensional feature space *F* dan menyelesaikannya dengan regresi linear pada feature space. Jika terdapat sebuah set data $G = \{(x_i, d_i)\}_i^n$, dimana x_i adalah vector masukan, d_i adalah vector keluaran (target), dan *n* adalah jumlah total pola data, SVR bertujuan untuk mengidentifikasi fungsi regresi untuk secara akurat memprediksi keluaran. Fungsi regresi linear pada feature space ditunjukkan pada Persamaan (2).

$$f(x) = \omega\phi(x) + b \qquad (2)$$

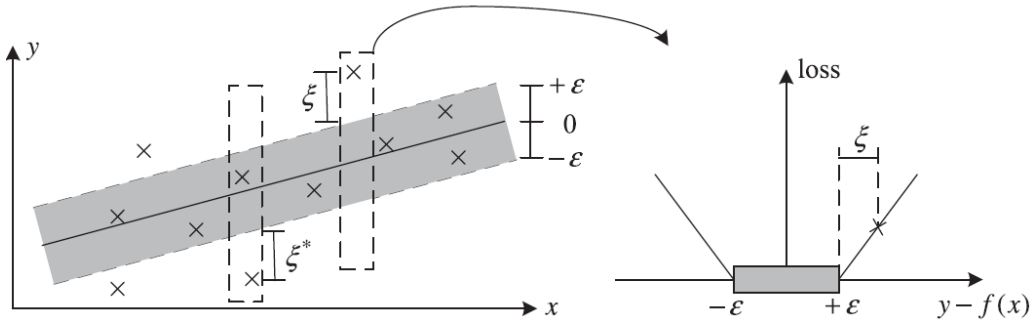
Pada Persamaan (2), ω dan b adalah koefisien, dan $\phi(x)$ adalah fungsi pemetaan feature space. Tujuan ummnya adalah untuk menemukan fungsi dengan tingkat "flatness" tertinggi. Nilai dari koefisien ω dan b dapat ditentukan dengan meinimalkan fungsi resiko regularisasi seperti pada Persamaan (3).

$$R = \frac{1}{2}\|\omega\|^2 + C\frac{1}{l}\sum_{i=1}^{l}|y_i - f(x_i)|_\varepsilon \qquad (3)$$

dimana

$$|y_i - f(x_i)|_\varepsilon = \begin{cases} 0 & \text{if } |y_i - f(x_i)| \le \varepsilon, \\ |y_i - f(x_i)| & \text{otherwise.} \end{cases}$$

Pada Persamaan (3), *C* adalah konstanta regularisasi yang menunjukkan trade-off antara kompleksitas struktur model $\frac{1}{2}\|\omega\|^2$ dan error empiris $\frac{1}{l}\sum_{i=1}^{l}|y_i - f(x_i)|_\varepsilon$. Menaikkan nilai *C* akan meningkatkan signifikansi resiko empiris relatif terhadap regularisasi. Hal ini hanya dapat diterima hanya jika error fitting lebih besar dari ε. Fungsi ε-insensitive loss digunakan untuk menstabilkan estimasi. Sehingga, ε dapat dilihat sebagai tabung yang memliki ukuran yang menggambarkan akurasi perkiraan seperti yang ditunjukkan pada Gambar 2.1.



Gambar 2.1. Penggambaran ε pada SVR linear (Schölkopf & Smola, 2002)

Pada Gambar 2.1, terdapat dua variabel "slack" ξ dan ξ^* yang menggambarkan deviasi positif dan negatif. Variabel ini memiliki posisi non-zero di luar area [-ε,ε]. Trade-off antara kompleksitas model (flatness) dan titik-titik yang berada di luar tabung (variabel slack) ditentukan dengan meminimalkan Persamaan (4).

$$|y_i - f(x_i)| - \varepsilon = \begin{cases} \xi & \text{if } y_i - f(x_i) > \varepsilon, \\ \xi^* & \text{if } f(x_i) - y_i > \varepsilon. \end{cases} \tag{4}$$

Dengan demikian, fungsi resiko regularisasi dapat ditulis sebagai Persamaan (5).

$$R = \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{l}(\xi + \xi^*) \tag{5}$$

dimana

$$\begin{cases} y_i - \omega\phi(x_i) - b \le \varepsilon + \xi_{i,} & i = 1,2,\ldots,l, \\ \omega\phi(x_i) + b - y_i \le \varepsilon + \xi_{i,} & i = 1,2,\ldots,l, \\ \xi + \xi_i \ge 0, & i = 1,2,\ldots,l. \end{cases}$$

Menurut kondisi Karush-Kuhn-Tucker, masalah optimasi dapat ditransformasikan dengan memaksimalkan masalah dual Lagrangian seperti pada Persamaan (6).

$$L(\alpha_i - \alpha_i^*) = \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) - \varepsilon\sum_{i=1}^{l}(\alpha_i - \alpha_i^*) - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(x_i, x_j), \tag{6}$$

dengan batasan

$$\sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0,$$

$$0 \leq \alpha_i, \alpha_i^* \leq C, \quad i = 1,2,\ldots,l.$$

Pada Persamaan (6), $K(x_i, x_j)$ adalah fungsi kernel. Fungsi kernel yang sering dipakai pada prakteknya adalah polynomial dan Gaussian. Selain itu, α_i dan α_i^* adalah multiplier Langrange yang memenuhi $\alpha_i \cdot \alpha_i^* = 0$ untuk $i = 1,2,\cdots,l$. Setelah α_i dan α_i^* didapatkan, fungsi regresi dapat ditulis kembali sebagai Persamaan (7).

$$f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\, K(x_i, x_j) + b \tag{7}$$

Persamaan Kernel Gaussian dapat ditulisan seperti Persamaan (8).

$$K(x_i, x_j) = \exp\left[\frac{-(x_i - x_j)^2}{2\sigma^2}\right] \tag{8}$$

Untuk membangun model SVR yang efisien, beberapa parameter perlu dispesifikasikan. Beberapa parameter tersebut adalah sebagai berikut.

1) Fungsi kernel
2) Parameter regularisasi C
3) Bandwidth fungsi kernel (σ^2)
4) Ukuran tabung fungsi ε-insensitive loss

Pemilihan nilai parameter yang tidak tepat akan berujung pada "over-fitting" atau "under-fitting". Umumnya, peneliti masih menggunakan prosedur standar dengan trial-and-error, membangun dengan parameter yang berbeda, lalu mengujinya dengan validasi untuk mendapatkan parameter yang optimal. Salah satu cara yang mudah namun efektif untuk menentukan nilai parameter SVR adalah dengan menggunakan "crossvalidation" (Smola & Schölkopf, 2004). Crossvalidation didasarkan pada sebuah ide bahwa kesalahan prediksi pada sebuah subset pada sampel *training* yang tidak digunakan pada proses *training* adalah identik pada kesalahan yang diharapkan itu sendiri. Terdapat beberapa strategi cross-validation, diantaranya 10-fold crossvalidation, leave-one out error (*l*-fold crossvalidation), bootstrap dan algoritma turunan yang mengestimasi kesalahan crossvalidation itu sendiri.

Dalam penggunaan teknik SVR, proses "*scaling*" dan normalisasi juga dapat dilakukan pada data. Proses ini dinilai dapat mengurangi waktu konvergensi *training* (Liu dkk., 1995) dan mempermudah perbandingan banyak data dengan jangkauan nilai yang berbeda-beda (Sitte, 1999). Dalam literatur, terdapat beberapa cara untuk melakukan *scaling* atau normalisasi pada data. Salah satu cara yang paling banyak digunakan adalah dengan melakukan *scaling* secara absolut, yaitu dengan merubah nilai pada data dengan jangkauan [0.0,1.0] dengan persamaan tertentu. Selain cara absolut, terdapat pula *scaling* yang merubah nilai pada data menjadi nilai baru dengan jangkauan [0.1, 0.9] dengan Persamaan (9) (Tian & Noore, 2005c).

$$i' = \frac{0.8}{i_{max} - i_{min}} i + \left(0.9 - 0.8 \times \frac{i_{max}}{i_{max} - i_{min}}\right) \qquad (9)$$

Pada Persamaan (12), i' adalah hasil proses *scaling*, i adalah nilai sebenarnya, i_{max} adalah nilai paling besar pada data, dan i_{min} adalah nilai paling kecil pada data. *Scaling* dengan jangkauan ini dinilai dapat menghasilkan prediksi lebih baik daripada hasil yang didapatkan dari *scaling* secara absolut (Karunanithi dkk., 1992c).

2.5. Binary Particle Swarm Optimization

BPSO adalah pengembangan dari algoritma Particle Swarm Optimization (PSO). PSO pertama kali diperkenalkan oleh Kennedy dan Eberhart sebagai sebuah algoritma optimasi yang terinspirasi dari perilaku kumpulan organisme atau "swarm" (Kennedy & Eberhart, 1995). Pada algoritma PSO, "swarm" sering digambarkan sebagai kumpulan partikel yang masing-masing memiliki atribut berupa posisi dan kecepatan yang digunakan untuk mencari solusi yang paling optimal secara iteratif. Posisi partikel merepresentasikan solusi dan kecepatan partikel menggambarkan seberapa cepat partikel bergerak pada ruang solusi. Pada akhir algoritma, partikel dengan posisi terbaik akan diambil sebagai solusi permasalahan.

Pertama-tama, sebuah populasi partikel diinisialisasi dengan posisi acak pada ruang solusi. Untuk mencari solusi yang terbaik, partikel harus bergerak dengan sebuah kecepatan menuju ke posisi lain pada ruang solusi. Posisi tersebut

kemudian akan dievaluasi dengan fungsi fitness tertentu, dimana posisi yang terbaik akan direkam dan disajikan sebagai solusi akhir.

Ruang solusi algoritma PSO dapat diasumsikan sebagai ruang dengan dimensional *d*. Partikel ke-*i* dapat direpresentasikan dengan vektor posisi $P_i = (p_{i1}, p_{i2}, \dots, p_{id})$ dan kecepatannya dengan $V_i = (v_{i1}, v_{i2}, \dots, v_{id})$. Tiap kali partikel bergerak, posisi terbaik yang pernah dikunjungi oleh partikel tersebut disimpan pada vektor $P_{ibest} = (p_{i1}, p_{i2}, \dots, p_{id})$ yang disebut juga "personal best". Posisi terbaik dari semua partikel juga disimpan pada sebuah vektor $P_{gbest} = (p_{g1}, p_{g2}, \dots, p_{gd})$ yang disebut juga "global best". Kecepatan dan posisi partikel akan selalu diperbarui hingga kondisi berhenti tercapai.

BPSO adalah sebuah algoritma yang dikembangkan dari PSO untuk mengatasi masalah diskrit (Kennedy & Eberhart, 1997). Partikel dapat menentukan masalah "ya" atau "tidak", "benar" atau "salah", "1" atau "0", dan sebagainya. Perbedaan BPSO dengan PSO terletak pada definisi kecepatannya, dimana kecepatan partikel pada BPSO didefinisikan sebagai probabilitas berubahnya nilai "1" atau "0". Dengan demikian, kecepatan harus dibatasi dengan jangkauan [0,1] dan pemetaan harus dilakukan pada nilai kontinu agar muat pada jangkauan tersebut. Normalisasi dapat diimplementasikan dengan fungsi sigmoid pada Persamaan (10).

$$v_{ij}^{'}(t) = sig\big(v_{ij}(t)\big) = \frac{1}{1 + e^{-v_{ij}(t)}} \qquad (10)$$

Kecepatan dapat diperbaruhi dengan Persamaan (11).

$$v_i(t+1) = wv_i(t) + c_1\varphi_1(p_i - x_i(t)) + c_2\varphi_2(p_g - x_i(t)) \qquad (11)$$

Pada Persamaan (11), c_1 dan c_2 adalah konstanta positif, φ_1 dan φ_2 adalah dua variabel acak dengan distribusi uniform antara 0 dan 1, w adalah inersia weight yang menunjukkan efek vektor kecepatan yang lama pada vektor yang baru. Terdapat batas atas V_{max} untuk kecepatan partikel untuk mencegah partikel bergerak terlalu cepat. Nilai ini biasanya diinisialisasi sebagai fungsi jangkauan masalah. Misalnya, jika jangkauan untuk semua x_{ij} adalah [-50,50], maka V_{max} adalah proporsional dengan 50.

Posisi partikel kemudian diperbarui dengan Persamaan (12).

$$x_{ij}(t+1) = \begin{cases} 1, & if\ r_{ij} < sig(v_{ij}(t+1)) \\ 0, & otherwise \end{cases} \quad (12)$$

Pada Persamaan (12), r_{ij} adalah angka acak uniform dengan jangkauan [0,1].

2.6. LIBSVM

LIBSVM adalah pustaka untuk implementasi SVM yang telah dikembangkan sejak tahun 2000 (Chang & Lin, 2011). Tujuan pengembangan LIBSVM adalah untuk membantu praktisi untuk mengaplikasikan SVM secara mudah. Penggunaan LIBSVM pada umumnya membutuhkan dua langkah. Pertama, "*training*" pada data dapat dilakukan untuk menghasilkan sebuah model yang dapat melakukan klasifikasi atau regresi. Kedua, menggunakan model tersebut untuk memprediksikan data uji dalam proses "*testing*". LIBSVM memiliki beberapa modul yang dapat digunakan untuk berbagai aplikasi SVR, diantaranya adalah C-Support Vector Classification (C-SVC), v-SVC, estimasi distribusi (one-class SVM), ε-SVR, dan v-SVR.

C-SVC merupakan implementasi dari SVM untuk masalah klasifikasi. Hal ini berbeda dengan v-SVC yang memiliki parameter *v*, yaitu sebuah parameter baru yang diperkenalkan pada tahun 2000 (Schölkopf dkk., 2000). One-class SVM adalah implementasi dari algoritma yang diusulkan tentang distribusi dalam masalah high-dimension (Schölkopf dkk., 2001). ε-SVR dan v-SVR digunakan untuk masalah regresi, dimana v-SVR menggunakan parameter *v* untuk mengontrol jumlah support vector.

2.7. Penelitian Terkait

Penelitian tentang prediksi reliabilitas perangkat lunak dengan menggunakan pendekatan DDSRM dengan teknik *machine learning* mulai mendapatkan perhatian semenjak diperkenalkan pada tahun 1991. Rangkuman penelitian-penelitian tersebut dapat dilihat pada Tabel 2.1.

Tabel 2.1. Rangkuman Penelitian Terdahulu dengan ANN

No.	Judul	Rujukan	Metode yang Diusulkan	Temuan
1	Prediction of software reliability using neural networks	(Karunanithi dkk., 1991)	Penggunaan Feed Forward NN (FFNN) untuk prediksi reliabilitas perangkat lunak.	Penelitian menunjukan bahwa NN dapat melakukan prediksi tanpa membuat asumsi tertentu pada sistem dan hanya bergantung pada data.
2	Prediction of Software Reliability Using Feedforward and Recurrent Neural Nets	(Karunanithi & Whitley, 1992)	Penggunaan FFNN dan Jordan's NN untuk melakukan prediksi.	Penelitian ini menunjukkan bahwa Semi-Recurrent NN dapat juga digunakan untuk melakukan prediksi.
3	Prediction of software reliability using connectionist models	(Karunanithi dkk., 1992b)	Penggunaan FFNN, Jordan's NN, dan Elman's NN untuk melakukan prediksi.	Penelitian ini menunjukkan bahwa Recurrent NN dapat juga digunakan untuk melakukan prediksi.
4	Using Neural Networks in Reliability Prediction	(Karunanithi dkk., 1992c)	Pemaparan penggunaan NN untuk prediksi reliabilitas secara umum.	
5	An Integrated Neural-Fuzzy System of Software Reliability Prediction	(Adnan & Yaacob, 1994)	Penggunaan metode fuzzy NN untuk melakukan prediksi reliabilitas.	Penggunaan metode fuzzy dapat memperbaiki hasil prediksi.
6	Comparison of software reliability growth predictions: NN vs parametric recalibration	(Sitte, 1999)	Perbandingan antara NN dan *parametric recalibration* sebagai metode alternatif SRGM.	Penelitian menunjukkan bahwa NN lebih baik daripada *parametric recalibration*.

No.	Judul	Rujukan	Metode yang Diusulkan	Temuan
7	Prediction of Software Reliability: A Comparison between Regression and NN Non-parametric models	(Aljahdali dkk., 2001)	Perbandingan antara Non-Parametric NN dan teknik regresi.	Penelitian menunjukkan bahwa Non-Parametric NN lebih baik daripada *parametric recalibration*.
8	On the neural network approach in software reliability modeling	(Cai dkk., 2001)	Pemaparan penggunaan FFNN untuk prediksi reliabilitas secara umum.	
9	A study of connectionist models for software reliability prediction	(Ho dkk., 2003)	Penggunaan Elman's NN yang dimodifikasi untuk melakukan prediksi.	
10	A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data	(Guo & Lyu, 2004)	Penggunaan PIL untuk mengoptimalisasi FFNN dan generalisasi penggabungan beberapa NN.	
11	Evolutionary neural network modeling for software cumulative failure time prediction	(Tian & Noore, 2005)	Penggunaan GA untuk mengoptimalisasi jumlah neuron masukan dan jumlah layer tersembunyi pada NN.	
12	On-line prediction of software reliability using an evolutionary connectionist model	(Tian & Noore, 2005b)	Penggunaan GA untuk mengoptimalisasi jumlah neuron masukan dan jumlah layer tersembunyi pada NN. Prediksi dilakukan secara *online*.	Prediksi dapat dilakukan secara *online*, dimana model dapat beradaptasi jika terdapat data baru yang masuk.

No.	Judul	Rujukan	Metode yang Diusulkan	Temuan
13	Early Software Reliability Prediction with Extended ANN Model	(Hu dkk., 2006)	Penggunaan data *failure* perangkat lunak pada proyek terdahulu untuk memprediksikan terjadinya *failure* pada awal waktu pengembangan.	Data terdahulu dapat dimanfaatkan untuk melakukan prediksi proyek baru yang belum memiliki banyak data *failure* yang cukup.
14	Neural Network-Based Approaches for Software Reliability Estimation using Dynamic Weighted Combinational Models	(Su & Huang, 2007)	Analisa NN dari sudut pandang matematis dan penurunan ekspresi matematis dari SRGM untuk diterapkan pada NN.	Penelitian menggunakan analisa matematis untuk menurunkan model yang dapat secara akurat memprediksikan reliabilitas.
15	Software Reliability Prediction by Soft Computing Techniques	(Kiran & Ravi, 2008)	Penggunaan model *ensemble* (penggabungan beberapa model NN) dengan Back Propragation NN.	Penggunaan model *ensemble* dapat menghasilkan prediksi yang lebih akurat daripada satu model NN saja.
16	Software Reliability Prediction Based on Discrete Wavelet Transform and Neural Network	(Jin dkk., 2009)	Penggunaan Wavelet Transform dan NN untuk melakukan prediksi reliabilitas.	
17	Predicting software reliability with neural network ensembles	(Zheng, 2009)	Penggunaan model *ensemble* (penggabungan beberapa model NN) untuk melakukan prediksi reliabilitas.	
18	Prediction of Software Reliability Using Feed Forward Neural Networks	(Singh & Kumar, 2010)	Penggunaan FFNN untuk prediksi reliabilitas perangkat lunak.	

No.	Judul	Rujukan	Metode yang Diusulkan	Temuan
19	Study of Software Reliability Prediction Based on GR Neural Network	(Wu & Yang, 2011)	Penggunaan General Regression NN untuk prediksi reliabilitas perangkat lunak.	Penelitian ini menunjukkan bahwa General Regression NN dapat juga digunakan untuk melakukan prediksi.
20	Evolutionary prediction for cumulative failure modeling: A comparative study	(Benaddy dkk., 2011)	Penggunaan Real-coded GA untuk mengestimasi model NN dan auto-regresi yang terbaik.	
21	Software Reliability Prediction using Neural Network with Encoded Input	(Bisi & Goyal, 2012)	Penggunaan skema encoding yang berbeda, yaitu NN with Exponential Encoding (NNE) dan NN with Logarithmic Encoding (NNL) untuk melakukan prediksi.	
22	Autonomous and adaptive procedure for cumulative failure prediction	(Zemouri & Zerhouni, 2012)	Penggunaan algoritma Fuzzy Min-Max untuk mengoptimalkan model NN secara *online.*	Prediksi dapat dilakukan secara *online*, dimana model dapat beradaptasi jika terdapat data baru yang masuk.
23	Neural Network Ensemble Based on K-means Clustering Individual Selection and Application for Software Reliability Prediction	(Li dkk., 2013)	Penggunaan K-Means untuk memilih beberapa model NN yang digunakan secara *ensemble* untuk prediksi realiabilitas perangkat lunak.	Penelitian memperkenalkan cara untuk memilih *ensemble* dengan metode klasterisasi.
24	Software Reliability Prediction Model Based On Ica Algorithm and Mlp Neural Network	(Noekhah dkk., 2013)	Penggunaan ICA sebagai algoritma optimasi untuk Multilayer Perceptron NN untuk melakukan prediksi.	

No.	Judul	Rujukan	Metode yang Diusulkan	Temuan
25	Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction	(Roy dkk., 2014)	Penggabungan beberapa SRGM tradisional dari pembelajaran FFNN dan pemanfaatan Real-coded GA untuk optimasi model NN.	

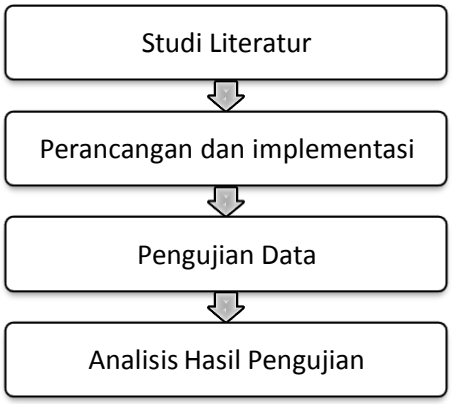
Tabel 2.2. Rangkuman Penelitian Terdahulu dengan SVR

No.	Judul	Rujukan	Metode yang Digunakan	Temuan
1	Dynamic Software Reliability Prediction: An Approach Based On Support Vector Machines	(Tian & Noore, 2005c)	Penggunaan SVR untuk melakukan prediksi reliabilitas secara *online.*	Prediksi dapat dilakukan secara *online*, dimana model dapat beradaptasi jika terdapat data baru yang masuk.
2	Prediction of Software Reliability by Support Vector Regression	(Wang & Chen, 2005)	Penggunaan SVR untuk melakukan prediksi reliabilitas.	
3	Software reliability forecasting by support vector machines with simulated annealing algorithms	(Pai & Hong, 2006)	Penggunaan SA untuk melakukan optimasi parameter SVR.	
4	A Study on Software Reliability Prediction Based on Support Vector Machines	(Yang & Li, 2007)	Penelitian ini menggunakan SVR untuk menyelidiki data yang lebih baik digunakan untuk prediksi realibilitas.	Penelitian ini menemukan bahwa data *failure* kumulatif lebih baik digunakan daripada data inter-failure time.

No.	Judul	Rujukan	Metode yang Digunakan	Temuan
5	Prediction of software reliability by support vector regression with genetic algorithms	(Ho & Hsieh, 2009)	Real-value GA digunakan sebagai algoritma optimasi parameter SVR.	
6	Early Software Reliability Prediction Based on Support Vector Machines with Genetic Algorithms	(Lo, 2010)	GA digunakan sebagai algoritma optimasi parameter SVR untuk prediksi reliabilitas di awal masa pengembangan.	
7	A generic data-driven software reliability model with model mining technique	(Yang dkk., 2010)	GA digunakan sebagai algoritma *model mining*, dimana *time lags* yang digunakan untuk melakukan prediksi belum tentu data yang terakhir. Selain itu, GA digunakan untuk optimasi parameter SVR.	Penelitian ini menunjukkan bahwa *model mining* dapat digunakan untuk mengetahui hubungan internal data dan memperbaiki hasil prediksi.
8	Software reliability prediction based on support vector regression using a hybrid genetic algorithm and simulated annealing algorithm	(Jin, 2011)	GA dan SA diintegrasi untuk optimasi parameter e-SVR.	
9	Software reliability prediction model based on support vector regression with improved estimation of distribution algorithms	(Jin & Jin, 2014)	Estimation of Distribution Algorithm (EDA) digunakan untuk memperbaiki populasi dan mengoptimasi parameter SVR.	

BAB III
METODOLOGI PENELITIAN

Bab ini memaparkan tentang metodologi penelitian yang digunakan pada penelitian ini, yang terdiri dari (1) studi literatur, (2) perancangan metode yang diusulkan dan implementasi kode, (3) pengujian data, dan (4) analisis hasil pengujian. Tahapan metodologi penelitian ini dapat dilihat pada Gambar 3.1.



Gambar 3.1. Tahapan metodologi penelitian

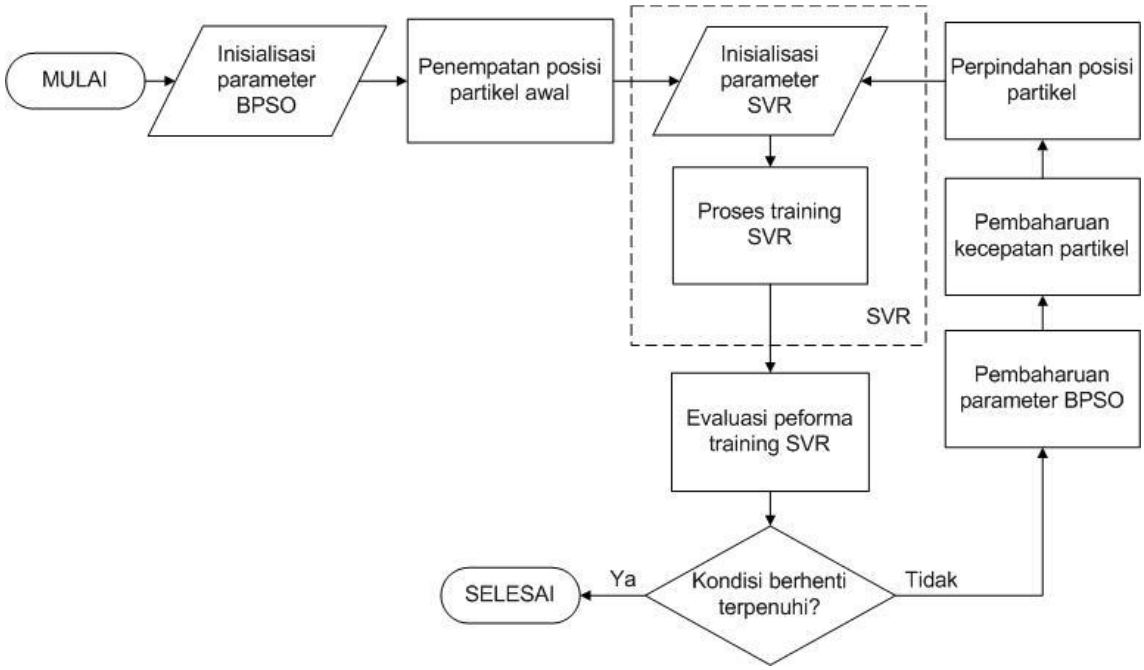
Penjelasan dari tahapan metode penelitian pada Gambar 3.1 akan dijelaskan secara rinci sebagai berikut.

3.1. Studi Literatur

Studi literatur adalah proses pengkajian artikel penelitian terdahulu yang berhubungan dengan topik penelitian ini. Referensi yang digunakan adalah referensi yang berkaitan dengan prediksi reliabilitas perangkat lunak, khususnya bagi pendekatan yang menggunakan teknik DDSRM dengan arsitektur MDISO. Selain itu, literatur yang berkaitan dengan metode yang diusulkan juga dipelajari, antara lain metode SVR dan metode BPSO. Studi literatur tambahan juga akan dilakukan untuk membantu mendesain langkah kerja penelitian, terutama mengenai data kesalahan atau kegagalan perangkat lunak dan kakas bantu yang dapat digunakan pada penelitian.

3.2. Perancangan dan Implementasi

Dalam penelitian ini, akan diimplementasikan sebuah sistem yang dapat melakukan prediksi reliabilitas perangkat lunak dengan menggunakan SVR dan *model mining* dengan BPSO (metode SVR-BPSO). Implementasi metode ini akan dilakukan pada lingkungan pemrograman Matlab 7.0 dengan memanfaatkan *tools* LibSVM untuk menjalankan beberapa perintah yang berkaitan dengan SVR. Sistem yang akan dibangun memerlukan proses *training* dan *testing* untuk menguji keakuratan metode yang diusulkan. Proses *training* pada penelitian ini memiliki tahapan yang dapat digambarkan pada diagram alir pada Gambar 3.2.



Gambar 3.2. Diagram alir proses *training* SVR-BPSO pada penelitian

Dibutuhkan data kegagalan perangkat lunak untuk menjalankan tahapan yang digambarkan pada Gambar 3.2. Penjelasan secara lebih rinci tentang data dan metode ini akan dijelaskan pada subbab berikut.

3.2.1. Data Kegagalan Perangkat Lunak

Terdapat beberapa data yang dipakai sebagai subjek pada penelitian ini yang didapatkan dari perangkat lunak sesungguhnya. Data tersebut dapat dibagi menjadi dua macam, yaitu data Time Between Failure (TBF) dan Failure Count

(FC). Pada penelitian ini, data-data tersebut disebut juga sebagai TBF1, TBF2, TBF3, FC1, FC2, dan FC3. TBF1 dan TBF2 adalah data kegagalan perangkat lunak yang dipublikasikan oleh Data and Analysis Center of Software (Musa, 1979) yang berisi data kegagalan dengan catatan waktu antar kegagalan dalam satuan milidetik dan detik. TBF3 adalah data yang didapatkan dari sebuah *workstation* pada Centre for Software Reliability (Lyu, 1996) yang berisi data kegagalan dengan catatan waktu antar kegagalan dalam satuan menit. FC1 adalah data dari sistem *tracking* bug pada laman web Xfce (Tamura & Yamada, 2005). FC2 adalah data dari sebuah produk jaringan *wireless* (Jeske dkk., 2005). FC3 adalah data dari sistem TROPICO R-1500, sebuah sistem Electronic Switching dari Brazil (Kanoun dkk., 1991). Masing-masing data dapat dilihat pada Lampiran.

Selain menggunakan dua macam data yang berbeda, data yang dipakai pada penelitian ini memiliki catatan kegagalan yang bervariasi. TBF1, TBF2, dan TBF3 memiliki jumlah data time-between-failure untuk 86, 207, dan 397 kegagalan. Sedangkan FC1, FC2, dan FC3 memiliki data jumlah total kegagalan dalam 21, 51, dan 81 minggu waktu eksekusi kumulatif perangkat lunak. Dengan adanya dua kelompok data dan jumlah kegagalan yang bervariasi, diharapkan metode yang diusulkan pada penelitian ini dapat diuji dengan baik.

Dalam penelitian ini, data akan digunakan untuk proses *training* dan *testing* model yang diusulkan. Sebanyak 80% data secara terurut akan digunakan sebagai data *training*, sedangkan 20% data terakhir akan digunakan sebagai data *testing*. Dalam proses *training*, data akan diolah terlebih dahulu dengan proses *scaling* dengan jangkauan nilai [0.1, 0.9] untuk meminimalkan dampak *scaling* secara absolut. Persamaan yang digunakan untuk proses ini ditunjukkan pada Persamaan (9). Tabel 3.1 menunjukkan data asli FC1 dan data FC1 setelah proses *scaling*.

Pada Tabel 3.1, terlihat data asli FC1 pada bagian kiri dan data FC1 setelah *scaling* pada bagian kanan. Kolom "Week" menunjukkan waktu eksekusi perangkat lunak dalam satuan minggu dan kolom "CNF" menunjukkan jumlah Cumulative Number of Failure (jumlah kegagalan yang terdeteksi) dalam jangkauan waktu tersebut. Proses *training* akan dilakukan untuk data minggu

pertama hingga minggu ke-17 karena memenuhi 80% porsi data untuk data setelah *scaling*.

Tabel 3.1. Data asli FC1 dan data setelah *scaling*

Data Asli		**Data setelah *scaling***	
Week	**CNF**	**Week**	**CNF**
1	16	0.1	0.1
2	24	0.15	0.155172414
3	26	0.2	0.168965517
4	38	0.25	0.251724138
5	45	0.3	0.3
6	51	0.35	0.34137931
7	60	0.4	0.403448276
8	67	0.45	0.451724138
9	74	0.5	0.5
10	89	0.55	0.603448276
11	98	0.6	0.665517241
12	108	0.65	0.734482759
13	114	0.7	0.775862069
14	117	0.75	0.796551724
15	124	0.8	0.844827586
16	127	0.85	0.865517241
17	132	0.9	0.9
18	135	0.95	0.920689655
19	150	1	1.024137931
20	164	1.05	1.120689655
21	167	1.1	1.14137931

Karena proses *training* dilakukan pada data minggu pertama hingga minggu ke-17, terlihat pada Tabel 3.1 bahwa data setelah *scaling* menghasilkan nilai pada jangkauan [0.1, 0.9] untuk data *training* tersebut. Dengan menggunakan

Persamaan (9), dapat diketahui bahwa nilai i_{max} dan i_{min} untuk kolom Week pada data *training* masing-masing adalah bernilai 1 dan 17, sedangkan nilai i_{max} dan i_{min} untuk kolom CNF masing-masing adalah 16 dan 132. Nilai i_{max} dan i_{min} ini kemudian juga dipakai untuk menghitung proses *scaling* data minggu ke-18 hingga ke-21. Dengan kata lain, nilai i_{max} dan i_{min} yang digunakan adalah nilai pada data *training*, sehingga hasil *scaling* data *testing* pada Tabel 3.1 memiliki nilai yang lebih besar dari jangkauan [0.1, 0.9].

3.2.2. Inisialisasi Parameter BPSO

Terdapat beberapa parameter algoritma BPSO perlu diinisialisasi, yaitu c_1, c_2, Vmax, dan w. Parameter c_1 dan c_2 diinisialisasi dengan nilai 2.0 berdasarkan aplikasi PSO yang dipakai pada banyak aplikasi (Eberhart & Shi, 2001). Vmax diinisialisasi pada ±4.0 berdasarkan ukuran yang direkomendasikan untuk masalah BPSO (Kennedy dkk., 2001). Parameter *w* diinisialisasi sebesar 0.8 berdasarkan anjuran dari penelitian lain untuk nilai Vmax $\geq$ 3 (Shi & Eberhart, 1998). Kondisi berhenti algoritma BPSO pada penelitian ini adalah ketika kondisi konvergen tercapai dengan selisih *threshold* sebesar 5×10^{-6}.

Jumlah partikel merupakan salah satu parameter lainnya yang perlu diinisialisasi. Dalam algoritma PSO secara umum, tidak ada aturan khusus untuk menentukan jumlah tersebut. Dalam penelitian ini, dilakukan beberapa percobaan awal dengan menjalankan metode yang diusulkan pada data FC1 dengan jumlah partikel yang berbeda untuk mencari jumlah partikel yang ideal dengan batasan waktu penelitian. Hasil waktu eksekusi dan jumlah iterasi yang diperlukan untuk mencapai konvergensi yang didapatkan dari percobaan awal tersebut dapat terlihat pada Tabel 3.2.

Pada Tabel 3.2, terlihat bahwa telah dilakukan percobaan metode yang diusulkan dengan jumlah partikel 2, 3, 4, 5, dan 6. Secara teori, jumlah partikel yang semakin banyak akan mempermudah ditemukannya solusi permasalahan yang paling optimal. Namun, dapat diamati bahwa jumlah partikel yang semakin banyak membuat waktu eksekusi metode yang semakin lama. Padahal, penelitian ini dibatasi pula oleh jangkauan waktu yang terbatas. Oleh karena itu, perlu dilakukan pembatasan jumlah partikel agar penelitian dapat diselesaikan dalam

waktu yang tersedia. Pada penelitian ini, jumlah partikel ditentukan sebanyak 5 karena percobaan awal pada Tabel 3.2 mengindikasikan bahwa iterasi yang diperlukan untuk mencapai konvergen lebih sedikit daripada jumlah partikel yang lainnya.

Tabel 3.2. Hasil percobaan data FC1 dengan jumlah partikel yang berbeda

No.	Jumlah Partikel	Waktu Eksekusi	Jumlah Iterasi Konvergen
1	2	2 jam 1 menit	21
2	3	2 jam 40 menit	15
3	4	3 jam 33 menit	15
4	5	4 jam 30 menit	14
5	6	6 jam 23 menit	22

Selain itu, setiap partikel memiliki komponen sliding window maksimal sejumlah *z*. Pada penelitian ini, *z* ditetapkan dengan nilai 8 yang menandakan bahwa terjadinya kegagalan perangkat lunak y_i dipengaruhi oleh 8 kegagalan perangkat lunak yang terjadi sebelumnya. Oleh karena itu, struktur sebuah partikel pada penelitian ini dapat digambarkan pada Gambar 3.3.

p_{i1}	p_{i2}	p_{i3}	p_{i4}	p_{i5}	p_{i6}	p_{i7}	p_{i8}

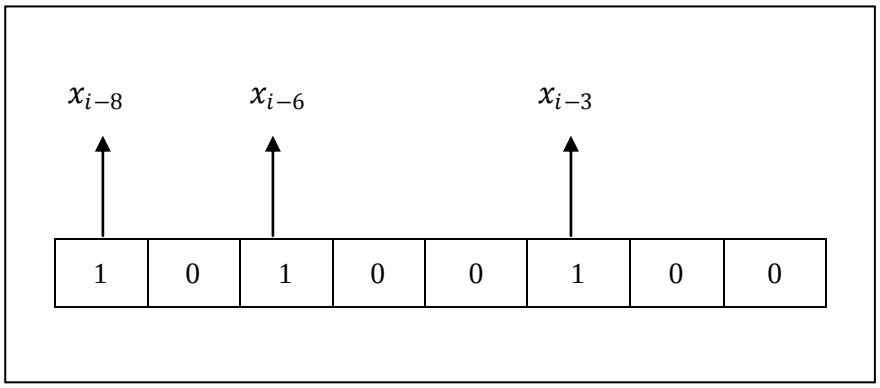
Gambar 3.3. Struktur sebuah partikel pada BPSO

Pada Gambar 3.3., terlihat bahwa sebuah partikel *P* terdiri dari 8 sub-partikel p_i yang merepresentasikan batas ukuran *sliding window* yang digunakan. Sub-partikel p_i dapat bernilai nol (“0”) atau satu (“1”).

3.2.3. Penempatan Posisi Partikel Awal

Posisi pada BPSO direpresentasikan dengan string biner yang bernilai nol (“0”) atau satu (“1”) pada sub-partikelnya, dimana nilai satu mewakili *lags* yang akan digunakan sebagai masukan SVR. Kombinasi biner ini menentukan model *time series* pada proses prediksi reliabilitas perangkat lunak. Misalnya, partikel

dengan nilai biner "10100100" menunjukkan bahwa data x_{i-3}, x_{i-6} dan x_{i-8} akan digunakan sebagai masukan seperti yang ditunjukkan pada Gambar 3.4.



Gambar 3.4. Interpretasi posisi pada sebuah partikel BPSO

Untuk memulai metode SVR-BPSO, partikel perlu memiliki posisi awal dan kecepatan awal. Posisi dari masing-masing sub-partikel akan diinisialisasi secara acak, sehingga masing-masing partikel akan memiliki kombinasi nilai nol ("0") atau satu ("1") yang berbeda-beda pada tiap kali metode SVR-BPSO dijalankan. Kecepatan juga perlu diinisialisasi secara acak untuk tiap subpartikel, dimana nilai kecepatan berada pada jangkauan desimal antara -4.0 hingga 4.0.

3.2.4. Inisialisasi Parameter SVR dan Proses *Training* SVR

Data yang dipakai pada proses *training* dipilih berdasarkan posisi partikel. Misalnya, partikel pertama pada percobaan awal dengan posisi "10100100" menunjukkan bahwa data x_{i-3}, x_{i-6} dan x_{i-8} digunakan sebagai masukan untuk memprediksikan nilai y_i, sedangkan data x_{i-1}, x_{i-2}, x_{i-4}, x_{i-5}, dan x_{i-7} tidak digunakan sebagai masukan. Sehingga, pemilihan data perlu dilakukan berdasarkan posisi partikel tertentu.

Setelah dilakukan pemilihan masukan, proses *training* dapat dilakukan dengan aturan sebagai berikut: 1) kernel yang akan digunakan adalah kernel Gaussian karena fungsi kernel ini dinilai mampu menghasilkan prediksi yang lebih baik daripada fungsi kernel lainnya (Jin & Jin, 2014); 2) parameter *C*, σ, dan ε ditentukan dengan metode 5-fold cross-validation seperti yang telah dilakukan pada penelitian terdahulu (Ho & Hsieh, 2009). Pemilihan ketiga parameter ini dapat diimplementasikan dengan fungsi Matlab seperti pada Gambar 3.5. Dengan

menggunakan fungsi ini, parameter yang terbaik dapat didapatkan setelah beberapa iterasi.

```
bestcv = 10000; bestc=0; bestg=0; beste=0; cVal = 200;
while(cVal < 5000)
  gVal = 0.1;
  while(gVal < 1)
      eVal = 0.001;
      while(eVal < 0.01)
        cmd = ['-s 3 -t 2 -v 5 -c ', num2str(cVal), ' -g ',
num2str(gVal), ' -p ', num2str(eVal)];
        cv = svmtrain(labeltrain, datatrain, cmd);
        if (cv <= bestcv)
          bestcv = cv; bestc = cVal; bestg = gVal; beste = eVal;
        end
        eVal = eVal + 0.001;
      end
    gVal = gVal + 0.1;
  end
  cVal = cVal + 200;
end
```

Gambar 3.5. Fungsi pemilihan parameter SVR pada Matlab

3.2.5. Evaluasi Peforma

Evaluasi peforma yang dimaksud adalah perhitungan nilai keakuratan hasil *training*. Perhitungan yang diambil adalah nilai Mean Squared Error (MSE) yang dapat dilihat pada Persamaan (13).

$$MSE_{training} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \qquad (13)$$

Pada Persamaan (13), $\hat{y}_i$ adalah keluaran yang didapatkan dari model (observasi perkiraan atau prediksi) dan y_i adalah data kegagalan yang pada data (observasi sebenarnya). MSE digunakan untuk mengukur besarnya rata-rata nilai kuadrat kesalahan prediksi. Sehingga, nilai MSE yang kecil menunjukkan prediksi yang semakin akurat.

3.2.6. Kondisi Berhenti

Algoritma pada penelitian ini akan berhenti setelah memenuhi kondisi pemberhentian. Kondisi yang dipakai pada penelitian ini adalah hingga hasil MSE mencapai kondisi konvergen. Algoritma akan berhenti dan memberikan keluaran berupa model prediksi yang paling optimal dan kombinasi masukan yang akan

digunakan pada proses *training*. Jika kondisi konvergen belum tercapai, maka algoritma akan dilanjutkan ke langkah berikutnya.

3.2.7. Pembaruan Parameter BPSO

Terdapat beberapa parameter BPSO akan diperbarui pada tahap ini, yaitu P_{pbest} dan P_{gbest}. P_{pbest} dan P_{gbest} berisi salinan posisi partikel terbaik secara personal dan global. Dengan demikian, P_{pbest} dan P_{gbest} memiliki struktur yang sama dengan partikel BPSO.

Tiap partikel memiliki P_{pbest} masing-masing yang akan diperbarui jika peforma partikel yang terbaru lebih baik daripada peforma P_{pbest} yang disimpan. Hal ini berbeda dengan P_{gbest} yang mencerminkan peforma partikel yang terbaik diantara partikel lainnya. Nilai P_{gbest} akan diperbarui jika ada catatan P_{pbest} yang lebih baik daripada P_{gbest}.

3.2.8. Pembaruan Kecepatan dan Posisi Partikel

Kecepatan dan posisi partikel kemudian juga akan diperbarui dengan beberapa langkah berikut. Kecepatan baru partikel dihitung dengan Persamaan (11). Variabel φ_1 dan φ_2 pada persamaan tersebut dihasilkan dengan membuat bilangan acak. Nilai sigmoid dari kecepatan baru tersebut lalu dihitung dengan Persamaan (10). Setelah itu, dihasilkan bilangan acak r_{ij} untuk semua partikel. Posisi baru partikel kemudian dapat ditentukan dengan Persamaan (12). Kecepatan baru dan posisi baru yang didapatkan digunakan untuk iterasi berikutnya.

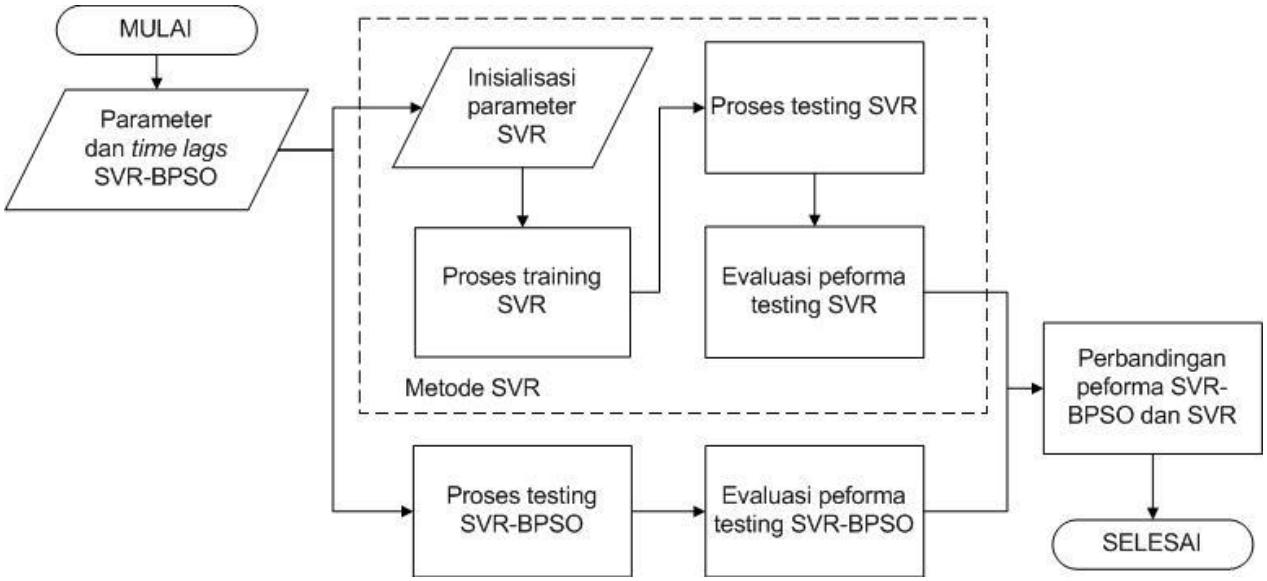
Pada iterasi berikutnya, proses *training* berulang lagi dengan kombinasi masukan yang berbeda. Setelah kondisi berhenti tercapai, partikel yang tercatat pada P_{gbest} akan diambil sebagai solusi terbaik.

3.3. Analisis Pengujian

Pengujian dilakukan untuk mengetahui bahwa metode yang diusulkan dapat menghasilkan prediksi reliabilitas perangkat lunak yang akurat. Untuk mengetahui hal ini, skenario pengujian dirancang dan dilakukan setelah proses *training*.

3.3.1. Inisialisasi Parameter BPSO

Pengujian dilakukan dengan membandingkan hasil yang didapatkan pada metode yang diusulkan (metode SVR-BPSO) dengan metode tanpa *model mining* (Metode SVR). Metode SVR memiliki alur yang sedikit berbeda dengan metode SVR-BPSO, dimana masukan yang digunakan diambil dengan jumlah tertentu tanpa dilakukan pemilihan dengan algoritma BPSO. Nilai peforma keduanya akan dibandingkan untuk mengetahui keakuratan metode yang diusulkan. Diagram alir skenario pengujian dapat dilihat pada Gambar 3.6.



Gambar 3.6. Diagram alir skenario pengujian

Pada Gambar 3.6, inisialisasi parameter SVR dilakukan secara acak dan proses *training* dilakukan dengan memasukkan data dengan ukuran *sliding window* yang disesuaikan dengan ukuran yang dipakai pada metode SVR-BPSO. Misalnya, jika metode SVR-BPSO menghailkan P_{gbest} dengan nilai "01001010", maka ukuran *sliding window* yang dipakai untuk metode SVR adalah 7. Nilai ini didapatkan dari data terjauh yang dipakai sebagai masukan pada kombinasi biner tersebut. Hal ini dilakukan untuk dapat menilai keefektifan penggunaan teknik *model mining*. Data yang akan digunakan untuk proses *testing* pada kedua metode adalah 20% rekaman terakhir dari keseluruhan data.

3.3.2. Parameter Pengujian

Nilai peforma yang digunakan untuk menilai keakuratan hasil *testing* kedua metode adalah nilai MSE yang dapat dihitung dengan Persamaan (14) dan Average Relative Prediction Error (AE) yang dapat dihitung dengan Persamaan (15).

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{14}$$

Pada Persamaan (14), $\hat{y}_i$ adalah prediksi atau keluaran yang didapatkan dari model dan y_i adalah data sebenarnya. Nilai MSE yang kecil menandakan kesalahan prediksi pada data *testing* yang lebih kecil pula.

$$AE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{Y}_i - Y_i}{Y_i}\right| \times 100\% \tag{15}$$

$$i = \left(\frac{i^{'} - 0.9}{0.8} \times (i_{max} - i_{min})\right) + i_{max} \tag{16}$$

Pada Persamaan (15), $\hat{Y}_i$ dan Y_i adalah data prediksi dan data sebenarnya setelah dilakukan *scaling* ulang dengan Persamaan (16) (Tian & Noore, 2005c). Nilai AE menunjukkan persentase kesalahan prediksi pada data tanpa *scaling*, dimana hal ini dapat digunakan untuk mengukur tingkat kesalahan prediksi dalam jangkauan data sebenarnya.

[Halaman ini sengaja dikosongkan]

BAB IV
UJI COBA DAN EVALUASI

Bab ini membahas evaluasi hasil pengujian metode yang diusulkan. Pembahasan pertama adalah tentang hasil prediksi yang didapat dari keenam data uji beserta penjelasan perbandingan nilai MSE dan AE. Pembahasan kedua adalah tentang analisa hasil pengujian.

4.1. Evaluasi Hasil Pengujian

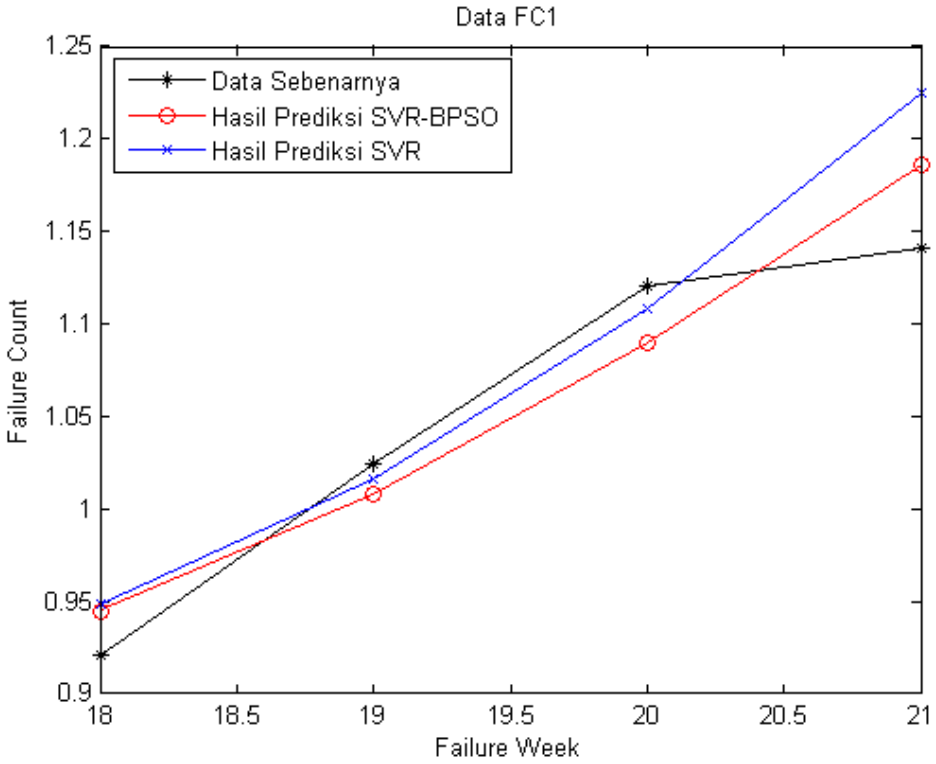
Subbab ini membahas tentang hasil pengujian dan evaluasi dari hasil prediksi yang didapatkan dari metode SVR-BPSO dan metode SVR pada enam data uji. Perbandingan nilai MSE dan AE dilakukan untuk mengukur tingkat akurasi metode yang diusulkan.

4.1.1. Evaluasi Hasil Pengujian pada data FC1

Pada penelitian ini, metode SVR-BPSO diujicobakan sebanyak 5 kali untuk tiap data. Hal ini dilakukan karena metode SVR-BPSO dapat menghasilkan keluaran yang berbeda pada tiap uji coba yang disebabkan oleh penempatan posisi dan kecepatan partikel yang bersifat acak. Selain itu, penentuan jumlah percobaan ini juga didasarkan pada jangka waktu penelitian yang terbatas. Pada tiap percobaan, nilai MSE dan AE metode SVR-BPSO akan dibandingkan dengan metode SVR (tanpa menggunakan *model mining*) untuk dapat mengetahui nilai keakuratan prediksi metode yang diusulkan. Hasil masing-masing percobaan secara detail dapat dilihat pada bagian Lampiran, sedangkan hasil rata-rata prediksi untuk data FC1 dapat dilihat pada Gambar 4.1.

Dari Gambar 4.1, terlihat bahwa proses *testing* dilakukan pada 4 data, yaitu kegagalan pada minggu ke-18, 19, 20, dan 21, seperti yang terlihat pada sumbu X (Failure Week). Pada tiap minggu tersebut, ditemukan kegagalan perangkat lunak dalam jumlah tertentu yang dapat dilihat pada sumbu Y (Failure Count). Dapat diamati bahwa metode SVR-BPSO dan metode SVR memiliki keakuratan yang

dapat dibandingkan, dimana masing-masing metode memiliki keakuratan prediksi yang lebih baik pada minggu-minggu tertentu. Misalnya, pada minggu ke-18 dan ke-21, metode SVR-BPSO dapat memprediksikan jumlah kegagalan dengan lebih baik daripada metode SVR. Sedangkan, pada minggu ke-19 dan ke-20, metode SVR-BPSO menghasilkan prediksi yang kurang akurat daripada metode SVR.



Gambar 4.1. Hasil prediksi SVR-BPSO dan SVR untuk Data FC1

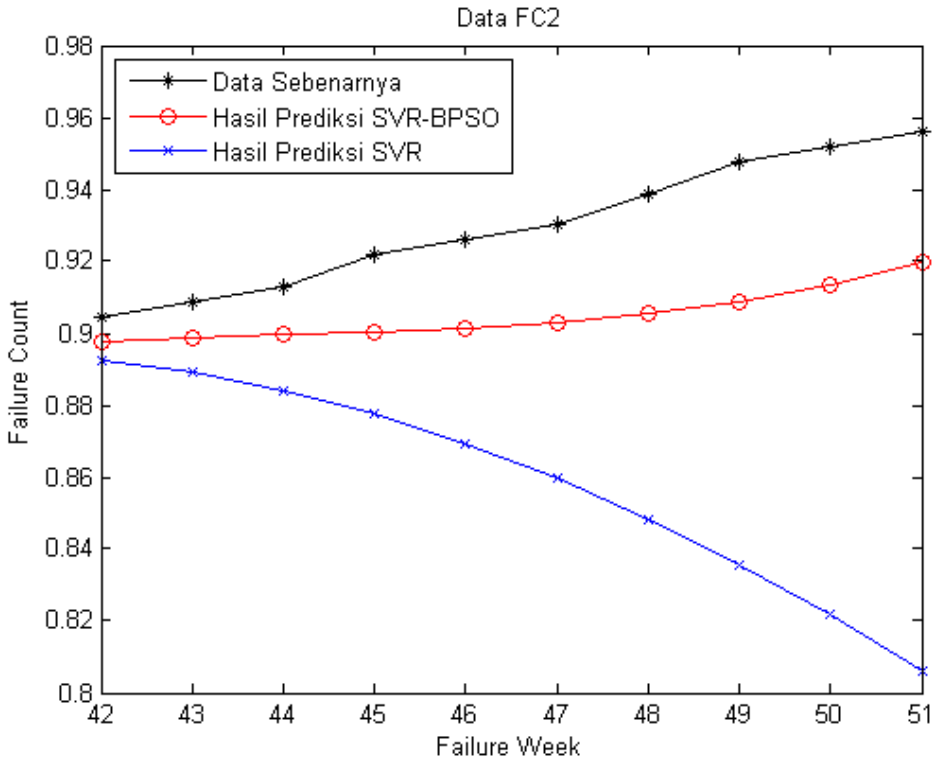
Keakuratan prediksi ini dapat dihitung dengan lebih rinci dengan mencari nilai MSE dan AE kedua metode. Dari rata-rata hasil percobaan, nilai MSE rata-rata yang didapatkan untuk metode SVR-BPSO adalah 0.00095, sedangkan nilai MSE rata-rata untuk metode SVR adalah 0.002. Dari nilai MSE ini, metode SVR-BPSO memiliki keakuratan yang lebih baik daripada metode SVR.

Hasil prediksi yang didapatkan ini kemudian dinormalisasi kembali dengan skala yang sesuai dengan data asli FC1. Dari nilai normalisasi ini, dapat dihitung keakuratan AE. Metode SVR-BPSO memiliki nilai AE sebesar 2.68% dan metode SVR memiliki nilai AE sebesar 3.03%. Dari kedua nilai AE ini, terlihat bahwa

metode SVR-BPSO memiliki keakuratan yang lebih baik setelah hasil prediksi dinormalisasi seperti skala aslinya.

4.1.2. Evaluasi Hasil Pengujian pada data FC2

Setelah dilakukan percobaan sebanyak 5 kali, didapatkan hasil prediksi untuk metode SVR-BPSO dan SVR untuk data FC2. Hasil rata-rata prediksi untuk data FC2 dapat dilihat pada Gambar 4.2.



Gambar 4.2. Hasil prediksi SVR-BPSO dan SVR untuk Data FC2

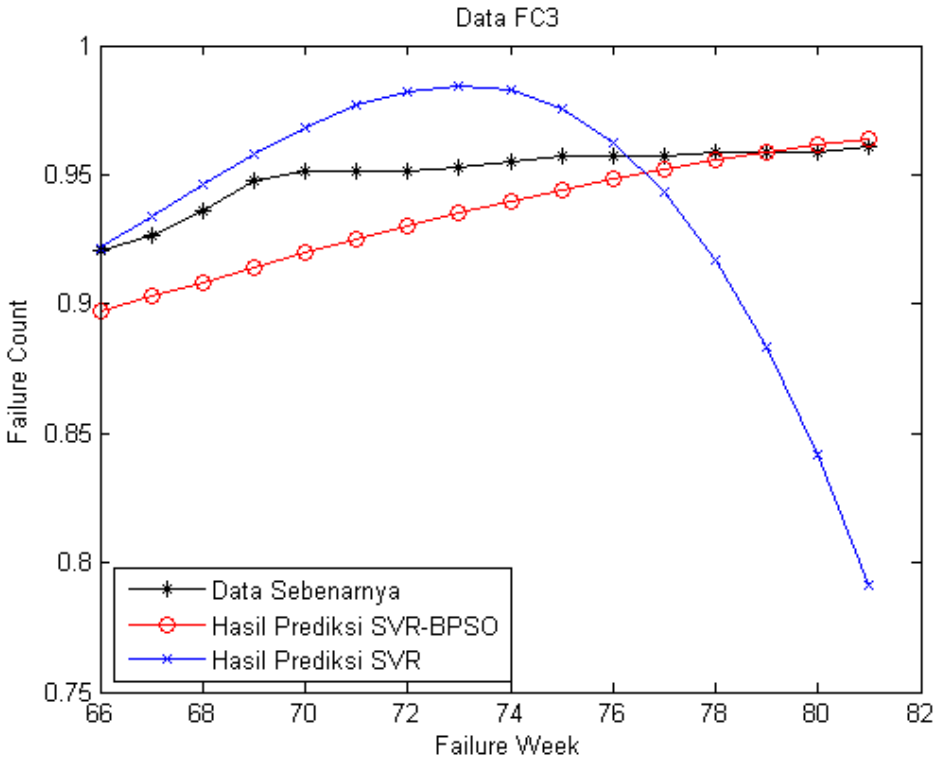
Dari Gambar 4.2, terlihat bahwa proses *testing* dilakukan pada 10 data, yaitu kegagalan pada minggu ke-42 hingga minggu ke-51. Dapat diamati bahwa metode SVR-BPSO dapat menghasilkan prediksi yang lebih baik daripada metode SVR pada semua minggu, dimana nilai prediksi yang dihasilkan tersebut cenderung mendekati data asli FC2. Dengan kata lain, metode SVR-BPSO cenderung dapat mengikuti *trend* data yang naik, sedangkan metode SVR tidak dapat mengikuti *trend* tersebut.

Setelah dilakukan perhitungan, nilai MSE rata-rata yang didapatkan untuk metode SVR-BPSO adalah 0.00077, sedangkan nilai MSE rata-rata untuk metode SVR adalah 0.0072. Dari nilai MSE ini, metode SVR-BPSO memiliki keakuratan yang lebih baik daripada metode SVR.

Hasil prediksi yang didapatkan ini kemudian dinormalisasi kembali dengan skala yang sesuai dengan data asli FC2. Dari nilai normalisasi ini, dapat dihitung keakuratan AE. Metode SVR-BPSO memiliki nilai AE sebesar 2.92% dan metode SVR memiliki nilai AE sebesar 8.30%. Dari kedua nilai AE ini, terlihat bahwa metode SVR-BPSO memiliki keakuratan yang lebih baik setelah hasil prediksi dinormalisasi seperti skala aslinya.

4.1.3. Evaluasi Hasil Pengujian pada data FC3

Setelah dilakukan percobaan sebanyak 5 kali, didapatkan hasil prediksi untuk metode SVR-BPSO dan SVR untuk data FC3. Hasil rata-rata prediksi untuk data FC3 dapat dilihat pada Gambar 4.3.



Gambar 4.3. Hasil prediksi SVR-BPSO dan SVR untuk Data FC3

Dari Gambar 4.3, terlihat bahwa proses *testing* dilakukan pada 16 data berdasarkan titik-titik pemetaan pada grafik, yaitu kegagalan pada minggu ke-66 hingga minggu ke-81. Selain itu, terlihat bahwa metode SVR-BPSO dan SVR memiliki kemampuan prediksi yang berbeda untuk data FC3. Metode SVR-BPSO cenderung dapat mengikuti *trend* data yang naik, sedangkan metode SVR tidak dapat mengikuti *trend* tersebut. Nilai MSE rata-rata yang didapatkan untuk metode SVR-BPSO adalah 0.00037, sedangkan nilai MSE rata-rata untuk metode SVR adalah 0.0034. Dari nilai MSE ini, metode SVR-BPSO memiliki keakuratan yang lebih baik daripada metode SVR.

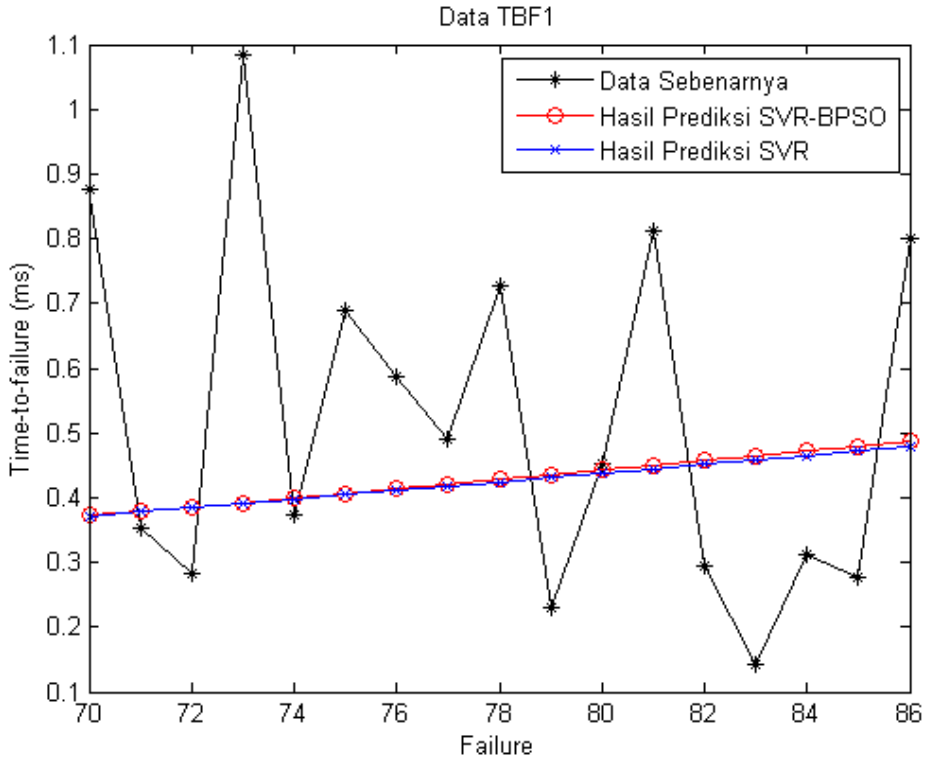
Hasil prediksi yang didapatkan ini kemudian dinormalisasi kembali dengan skala yang sesuai dengan data asli FC3. Dari nilai normalisasi ini, dapat dihitung keakuratan AE. Metode SVR-BPSO memiliki nilai AE sebesar 1.88% dan metode SVR memiliki nilai AE sebesar 4.34%. Dari kedua nilai AE ini, terlihat bahwa metode SVR-BPSO memiliki keakuratan yang lebih baik setelah hasil prediksi dinormalisasi seperti skala aslinya.

4.1.4. Evaluasi Hasil Pengujian pada data TBF1

Setelah dilakukan percobaan sebanyak 5 kali, didapatkan hasil prediksi untuk metode SVR-BPSO dan SVR untuk data TBF1. Hasil rata-rata prediksi untuk data TBF1 dapat dilihat pada Gambar 4.4.

Dari Gambar 4.4, terlihat bahwa metode SVR-BPSO dan SVR tidak dapat mendeteksi fluktuasi data dengan baik. Nilai MSE rata-rata yang didapatkan untuk metode SVR-BPSO adalah 0.083, sedangkan nilai MSE rata-rata untuk metode SVR adalah 0.083. Dari nilai MSE ini, metode SVR-BPSO memiliki keakuratan yang lebih baik daripada metode SVR.

Hasil prediksi yang didapatkan ini kemudian dinormalisasi kembali dengan skala yang sesuai dengan data asli TBF1. Dari nilai normalisasi ini, dapat dihitung keakuratan AE. Metode SVR-BPSO memiliki nilai AE sebesar 95.18% dan metode SVR memiliki nilai AE sebesar 93.77%. Dari kedua nilai AE ini, terlihat bahwa metode SVR memiliki keakuratan yang lebih baik setelah hasil prediksi dinormalisasi seperti skala aslinya. Namun, hasil ini menunjukkan bahwa kedua metode memiliki nilai kesalahan yang besar untuk data ini.



Gambar 4.4. Hasil prediksi SVR-BPSO dan SVR untuk Data TBF1

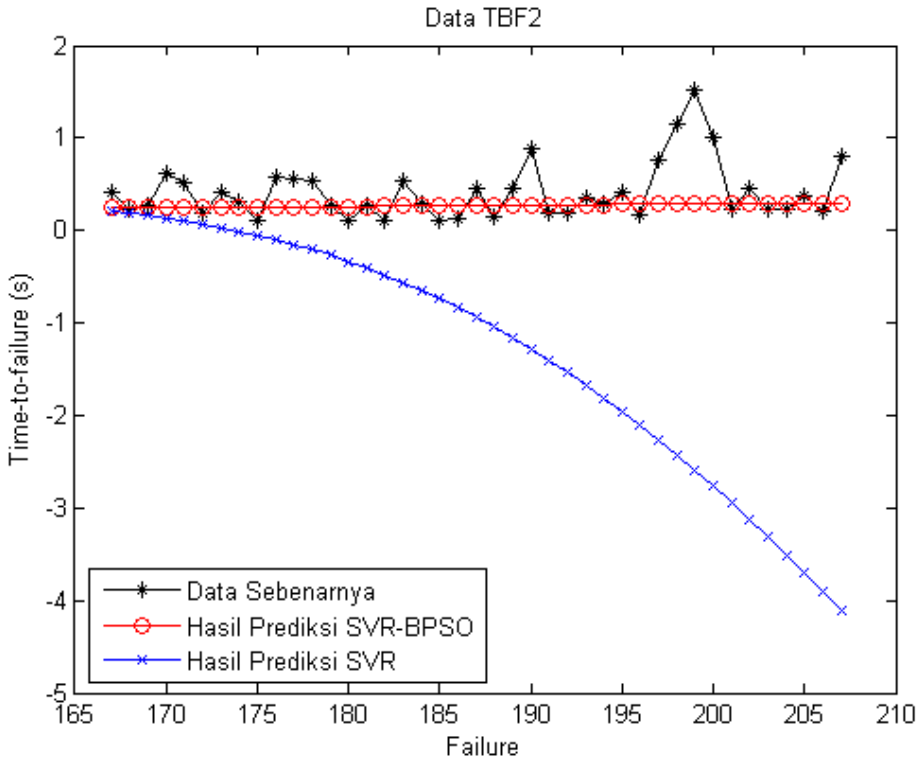
4.1.5. Evaluasi Hasil Pengujian pada data TBF2

Setelah dilakukan percobaan sebanyak 5 kali, didapatkan hasil prediksi untuk metode SVR-BPSO dan SVR untuk data TBF2. Hasil rata-rata prediksi untuk data TBF2 dapat dilihat pada Gambar 4.5.

Dari Gambar 4.5, terlihat bahwa metode SVR-BPSO dan SVR tidak dapat mendeteksi fluktuasi data dengan baik pada data TBF2. Nilai MSE rata-rata yang didapatkan untuk metode SVR-BPSO adalah 0.11, sedangkan nilai MSE rata-rata untuk metode SVR adalah 4.95. Dari nilai MSE ini, metode SVR-BPSO memiliki keakuratan yang lebih baik daripada metode SVR.

Hasil prediksi yang didapatkan ini kemudian dinormalisasi kembali dengan skala yang sesuai dengan data asli TBF2. Dari nilai normalisasi ini, dapat dihitung keakuratan AE. Metode SVR-BPSO memiliki nilai AE sebesar 241.13% dan metode SVR memiliki nilai AE sebesar 1539.69%. Dari kedua nilai AE ini, terlihat bahwa metode SVR memiliki keakuratan yang lebih baik setelah hasil

prediksi dinormalisasi seperti skala aslinya. Hasil ini juga menunjukkan bahwa kedua metode memiliki nilai kesalahan yang besar untuk data yang bersifat fluktuatif. Namun, terlihat pada data TBF2, metode SVR-BPSO cenderung lebih dapat mendeteksi *trend* data yang lebih baik daripada metode SVR.

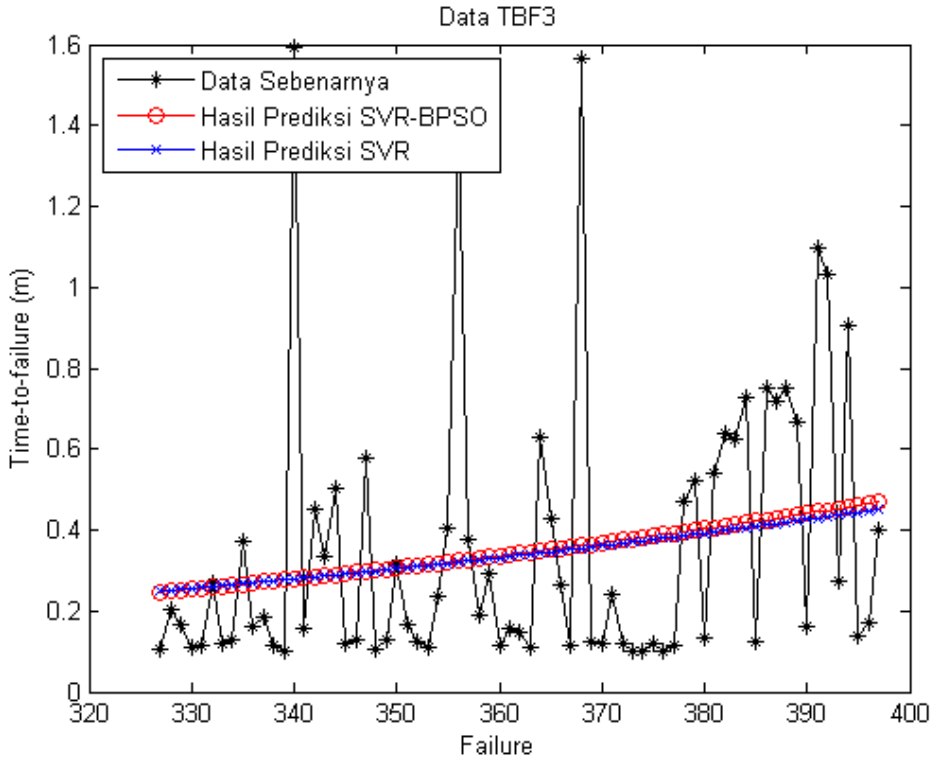


Gambar 4.5. Hasil prediksi SVR-BPSO dan SVR untuk Data TBF2

4.1.6. Evaluasi Hasil Pengujian pada data TBF3

Setelah dilakukan percobaan sebanyak 5 kali, didapatkan hasil prediksi untuk metode SVR-BPSO dan SVR untuk data TBF3. Hasil rata-rata prediksi untuk data TBF3 dapat dilihat pada Gambar 4.6.

Dari Gambar 4.6, terlihat bahwa metode SVR-BPSO dan SVR tidak dapat mendeteksi fluktuasi data dengan baik pada data TBF3. Nilai MSE rata-rata yang didapatkan untuk metode SVR-BPSO adalah 0.11, sedangkan nilai MSE rata-rata untuk metode SVR adalah 0.11. Dari nilai MSE ini, metode SVR-BPSO memiliki keakuratan yang lebih baik daripada metode SVR.



Gambar 4.6. Hasil prediksi SVR-BPSO dan SVR untuk Data TBF3

Hasil prediksi yang didapatkan ini kemudian dinormalisasi kembali dengan skala yang sesuai dengan data asli TBF3. Dari nilai normalisasi ini, dapat dihitung keakuratan AE. Metode SVR-BPSO memiliki nilai AE sebesar 2996.01% dan metode SVR memiliki nilai AE sebesar 2921.77%. Dari kedua nilai AE ini, terlihat bahwa metode SVR memiliki keakuratan yang lebih baik setelah hasil prediksi dinormalisasi seperti skala aslinya. Hasil ini juga menunjukkan bahwa kedua metode memiliki nilai kesalahan yang besar untuk data yang bersifat fluktuatif, terlebih lagi jika dilakukan pada data yang memiliki jumlah yang besar.

4.2. Analisa Hasil Pengujian

Dari hasil pengujian yang didapatkan, proses analisa dapat dilakukan dengan meninjau hasil pengujian tersebut. Dalam penelitian ini, penting untuk menganalisa keakuratan prediksi metode SVR-BPSO dengan melihat hasil MSE dan AE untuk masing-masing data. Hasil MSE dan AE yang didapatkan dapat ditampilkan secara ringkas pada Tabel 4.1.

Tabel 4.1. Hasil MSE dan AE Keseluruhan

Data	Metode SVR-BPSO		Metode SVR	
	MSE	AE(%)	MSE	AE(%)
FC1	0.00095	2.68	0.002	3.03
FC2	0.00077	2.92	0.0072	8.30
FC3	0.00037	1.88	0.0034	4.34
TBF1	0.083	95.18	0.083	93.77
TBF2	0.11	241.13	4.95	1539.69
TBF3	0.11	2996.01	0.11	2921.77

Pada Tabel 4.1, perbandingan keakuratan untuk kedua metode dapat lebih mudah dilakukan untuk semua data. Analisa dari hasil yang didapatkan adalah sebagai berikut.

4.2.1. Hasil prediksi FC1

Seperti yang telah diulas sebelumnya, metode SVR-BPSO dapat menghasilkan prediksi yang lebih akurat daripada metode SVR untuk data FC1. Hal ini terlihat dari nilai MSE dan AE metode SVR-BPSO yang lebih kecil daripada metode SVR. Jika dilihat pada posisi partikel terbaik yang dihasilkan, seperti yang secara lengkap terdapat pada Lampiran, prediksi pada data FC1 dapat secara optimal dilakukan dengan menggunakan data x_{i-6}, x_{i-5}, x_{i-4}, x_{i-3}, dan x_{i-2} (dari posisi partikel "00111110"). Data x_{i-8}, x_{i-7}, dan x_{i-1} tidak digunakan karena posisi partikel bernilai nol pada data tersebut. Pemilihan data masukan dengan proses *model mining* ini terbukti dapat memberikan pengaruh yang lebih baik untuk keakuratan prediksi pada data FC1.

4.2.2. Hasil prediksi FC2 dan FC3

Pada data FC2 dan FC3, metode SVR-BPSO dapat menghasilkan prediksi yang lebih akurat daripada metode SVR. Dapat juga diamati bahwa metode SVR menghasilkan prediksi yang membentuk kurva menurun pada Gambar 4.2 dan Gambar 4.3. Hal ini dapat disebabkan karena pemilihan parameter SVR yang dipakai pada penelitian ini pada dasarnya masih menggunakan cara tradisional,

yaitu dengan menelusuri kemungkinan dari jangkauan nilai yang telah ditentukan sebelumnya seperti yang terlihat pada Gambar 3.5. Cara pencarian ini sebenaranya adalah cara yang umum dilakukan pada penelitian SVR lainnya, namun tidak dapat dipungkiri bahwa terdapat metode lain untuk mencari parameter SVR dengan hasil yang lebih optimal. Hal ini memberikan ruang untuk penelitian selanjutnya dengan memberikan perbaikan pada pemilihan parameter SVR untuk menghasilkan nilai keakuratan yang lebih baik untuk metode SVR-BPSO.

Selain itu, dari hasil yang didapatkan untuk data FC1, FC2, dan FC3, dapat diamati bahwa metode SVR-BPSO menghasilkan prediksi yang akurat dengan nilai AE di bawah 10%. Hal ini menunjukkan bahwa metode ini memiliki tingkat kesalahan yang rendah pada jangkauan nilai data sebenarnya (tanpa *scaling*). Hal ini dapat dikaitkan dengan sifat data FC yang secara umum bersifat stabil dan tidak fluktuatif, sehingga SVR-BPSO dapat secara baik memodelkan hubungan pada data tersebut untuk menghasilkan prediksi yang akurat. Secara keseluruhan, terlihat bahwa penggunaan *model mining* penting untuk diterapkan agar reliabilitas perangkat lunak dapat diprediksi dengan lebih baik.

4.2.3. Hasil prediksi TBF1 dan TBF3

Pada data TBF1 dan TBF3, metode SVR-BPSO memiliki nilai MSE yang lebih kecil daripada metode SVR, namun memiliki nilai AE yang lebih besar daripada metode SVR. Hal ini menunjukkan bahwa metode SVR-BPSO yang dirancang telah dapat memodelkan prediksi pada data TBF1 dan TBF3 secara lebih baik, namun memiliki kesalahan yang lebih besar setelah hasil prediksi diproyeksikan pada jangkauan data aslinya. Hal ini tentunya merupakan kelemahan metode SVR-BPSO yang tidak dapat menghasilkan prediksi yang lebih akurat.

Pengamatan lain juga dapat dilakukan pada nilai AE yang besar untuk kedua data, dimana data TBF1 memiliki nilai AE > 90% untuk kedua metode dan data TBF3 memiliki nilai AE > 2900%. Hal ini menunjukkan bahwa metode SVR-BPSO memiliki tingkat kesalahan yang besar untuk kedua data. Hal ini dapat dikaitkan dengan hasil yang terlihat pada Gambar 4.4 dan Gambar 4.6, dimana

terlihat kedua metode tidak dapat menghasilkan prediksi yang dapat mengikuti fluktuasi data. Hal ini dapat dipengaruhi oleh sifat data TBF dimana *time-between-failure* tiap kegagalan tidak hanya dipengaruhi oleh urutan kegagalan tersebut, sehingga diperlukan data pendukung lainnya untuk membuat prediksi yang lebih akurat, misalnya data modul perangkat lunak yang terkait. Dengan kata lain, data TBF kurang cocok untuk dijadikan bahan uji coba metode yang diusulkan karena kriteria data yang tidak sesuai dengan asumsi waktu dan kegagalan yang terdapat pada metode SVR-BPSO.

4.2.4. Hasil prediksi TBF2

Pada data TBF2, juga terlihat bahwa metode SVR-BPSO tidak dapat menghasilkan prediksi yang dapat mengikuti fluktuasi data dengan baik, meskipun metode ini dapat menghasilkan prediksi yang lebih akurat dari metode SVR jika dilihat dari nilai MSE dan AE yang didapatkan. Namun, metode SVR pada data TBF2 menunjukkan hasil yang bisa dibandingkan dengan data FC2 dan FC3, dimana prediksi yang dihasilkan membentuk kurva menurun pada Gambar 4.5. Hal ini juga dapat disebabkan oleh pemilihan parameter SVR yang digunakan pada penelitian ini yang dapat diperbaiki untuk mendapatkan hasil yang lebih akurat pada penelitian selanjutnya.

Selain itu, dapat diamati bahwa jumlah data mempengaruhi tingkat kesalahan untuk data TBF1, TBF2, dan TBF3, dimana jumlah data yang besar cenderung memiliki kesalahan prediksi yang besar. Hal ini dapat dikaitkan pada sifat data TBF yang berbeda dengan data FC, sehingga nilai kesalahan akan menjadi lebih besar dengan semakin banyaknya data pada proses pengujian. Dari hasil tersebut, dapat ditarik pelajaran bahwa data reliabilitas perangkat lunak yang lebih baik dipakai untuk masalah prediksi adalah data Failure Count karena dapat menghasilkan prediksi dengan kesalahan yang lebih kecil daripada data Time-Between-Failure.

[Halaman ini sengaja dikosongkan]

BAB V
PENUTUP

Bab ini membahas kesimpulan dari hasil penelitian dan saran untuk penelitian selanjutnya berdasarkan hasil dan evaluasi pengujian. Kesimpulan dan saran yang didapatkan dari penelitian ini adalah sebagai berikut.

5.1. Kesimpulan

Berdasarkan Perumusan Masalah yang telah ditentukan, dapat diambil beberapa kesimpulan sebagai berikut.

1. BPSO dapat digunakan sebagai metode *model mining* dengan menggunakan angka biner satu (“1”) dan nol (“0”) untuk mewakili data yang dipakai dan tidak dipakai sebagai masukan, dimana masukan tersebut kemudian digunakan untuk pemodelan prediksi reliabilitas perangkat lunak dengan menggunkan teknik SVR. Secara keseluruhan, metode ini disebut juga metode SVR-BPSO yang dirancang berdasarkan prinsip masing-masing untuk dapat menghasilkan prediksi kegagalan perangkat lunak yang lebih akurat.
2. Nilai keakuratan SVR-BPSO dapat diukur dengan MSE dan AE, dimana secara umum metode SVR-BPSO memiliki nilai MSE dan AE yang lebih kecil daripada metode SVR tanpa *model mining* dengan pengecualian pada nilai AE data TBF1 dan TBF3. Hal ini disebabkan karena sifat data TBF yang berbeda dengan data FC, dimana *time-between-failure* kegagalan tidak bergantung pada urutan kegagalan yang ditemukan sehingga tidak sesuai dengan asumsi yang dipakai pada metode yang diusulkan.
3. Penggunaan BPSO untuk *model mining* secara umum terbukti dapat menghasilkan prediksi yang lebih baik dibandingkan dengan metode tanpa *model mining*. Beberapa perbaikan dapat diterapkan untuk mendapatkan hasil yang lebih optimal, namun penggunaan *model mining* terbukti dapat memberikan manfaat yang nyata dalam bidang prediksi reliabilitas perangkat lunak.

5.2. Saran

Penelitian ini dapat dikembangkan lebih jauh dengan beberapa saran sebagai berikut.

1. Pemilihan parameter SVR dengan metode yang lebih baik dapat meningkatkan keakuratan prediksi metode SVR-BPSO, misalnya dengan teknik optimasi. Rancangan diagram alir baru dapat dikembangkan untuk mengintegrasikan teknik pemilihan parameter SVR ini.
2. Metode SVR-BPSO dapat diujicobakan pada data lain untuk menguji beberapa aspek dalam metode ini secara lebih jauh.

DAFTAR PUSTAKA

Adnan, W. A., & Yaacob, M. H. (1994). An Integrated Neural-Fuzzy System of Software Reliability Prediction. *Software Testing, Reliability and Quality Assurance*, (pp. 154 - 158). New Delhi.

Aljahdali, S., Sheta, A., & Rine, D. (2001). Prediction of software reliability: a comparison between regression and neural network non-parametric models . *International Conference on Computer Systems and Applications*, (pp. 470 - 473). Beirut.

American Institute of Aeronautics and Astronautics. (1993). *Recommended Practice for Software Reliability.* AIAA.

Amin, A., Grunske, L., & Colman, A. (2013). An approach to software reliability prediction based on time series modeling. *Journal of Systems and Software , 86* (7), 1923-1932.

Bai, C., Hub, Q., Xieb, M., & Ng, S. (2005). Software failure prediction based on a Markov Bayesian network model. *Journal of Systems and Software , 74* (3), 275–282.

Barghout, M. L., & Abdel-Ghaly, A. (1998). A non-parametric order statistics software reliability model. *Software Testing, Verification and Reliability , 8* (3), 113–132.

Benaddy, M., Aljahdali, S., & Wakrim, M. (2011). Evolutionary Prediction for Cumulative Failure Modeling: A Comparative Study. *8th International Conference on Information Technology: New Generations*, (pp. 41-48).

Bisi, M., & Goyal, N. K. (2012). Software Reliability Prediction using Neural Network with Encoded Input. *International Journal of Computer Applications , 47* (22), 46-52.

Cai, K.-Y., Cai, L., Wang, W.-D., Yu, Z.-Y., & Zhang, D. (2001). On the neural network approach in software reliability modeling. *The Journal of Systems and Software , 58*, 47-62.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology , 2* (3).

Eberhart, R. C., & Shi, Y. (2001). Particle swarm optimization: developments, applications and resources. *Evolutionary Computation*, (pp. 81 - 86). Seoul.

Goel, A. (1985). Software Reliability Models: Assumptions, Limitations, and Applicability . *Software Engineering , SE-11* (12), 1411-1423.

Guo, P., & Lyu, M. R. (2004). A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data. *Neurocomputing , 56*, 101-121.

Ho, C.-H., & Hsieh, S.-W. (2009). Prediction of software reliability by support vector regression with genetic algorithms. *Journal of Information & Optimization Sciences , 30* (3), 503-523.

Ho, S. L., Xie, M., & Goh, T. N. (2003). A study of the connectionist models for software reliability prediction. *Computers & Mathematics with Applications , 46* (7), 1037-1045.

Hu, Q., Dai, Y., Xie, M., & Ng, S. (2006). Early Software Reliability Prediction with Extended ANN Model . *30th Annual International of Computer Software and Applications Conference (COMPSAC '06)*, (pp. 234 - 239). Chicago.

Jeske, D. R., Zhang, X., & Pham, L. (2005). Adjusting software failure rates that are estimated from test data. *Reliability , 54* (1), 107-114.

Jin, C. (2011). Software reliability prediction based on support vector regression using a hybrid genetic algorithm and simulated annealing algorithm. *IET Software , 5* (4), 398 - 405 .

Jin, C., & Jin, S.-W. (2014). Software reliability prediction model based on support vector regression with improved estimation of distribution algorithms. *Applied Soft Computing , 15*, 113-120.

Jin, C., Jin, S.-W., Ye, J.-M., & Zhang, Q.-G. (2009). Software Reliability Prediction Based on Discrete Wavelet Transform and Neural Network. *Computational Intelligence and Software Engineering*, (pp. 1 - 4). Wuhan.

Kanoun, K., Martini, M., & Souza, J. (1991). A method for software reliability analysis and prediction application to the TROPICO-R swicthing system. *Software Engineering , 17* (4), 334-344.

Karunanithi, N., & Whitley, D. (1992). Prediction of Software Reliability Using Feedforward and Recurrent Neural Nets. *International Joint Conference on Neural Networks*, (pp. 800-805). Baltimore.

Karunanithi, N., Malaiya, Y. K., & & Whitley, D. (1991). Prediction of software reliability using neural networks. *International Symposium on Software Reliability Engineering*, (pp. 124-130).

Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992b). Prediction of Software Reliability Using Connectionist Models. *Software Engineering , 18* (7), 563-574.

Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992c). Using Neural Networks in Reliability Prediction. *Software , 9* (4), 53-60.

Kennedy, J., & Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. *International Conference on Systems, Man, and Cybernetics*, (pp. 4104 - 4108). Orlando.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Neural Networks*, (pp. 1942 - 1948). Perth.

Kennedy, J., Eberhart, R. C., & Shi, Y. (2001). *Swarm Intelligence*. San Diego: Academic Press.

Kiran, N. R., & Ravi, V. (2008). Software reliability prediction by soft computing techniques. *Journal of Systems and Software , 81* (4), 576-583.

Li, K., Zhao, K., & Liu, W. (2013). Neural Network Ensemble Based on K-Means Clustering Individual Selection and Application for Software Reliability Prediction. *Fourth World Congress on Software Engineering (WCSE)*, (pp. 131-135).

Liu, M. C., Kuo, W., & Sastri, T. (1995). An Exploratory Study of a Neural Network Approach for Reliability Data Analysis. *Quality and Reliability Engineering International , 11*, 107-112.

Lo, J.-H. (2010). Early Software Reliability Prediction Based on Support Vector Machines with Genetic Algorithms. *5th Industrial Electronics and Applications (ICIEA)*, (pp. 2221-2226). Taichung.

Lyu, M. (1996). *Handbook of Software Reliability Engineering.* Hightown, NJ, USA: McGraw-Hill.

Lyu, M. (2007). Software Reliability Engineering: A Roadmap. *Future of Software Engineering*, (pp. 153-170). Minneapolis.

Musa, J. D. (1979). *Software Reliability Data.* Data and Analysis Center for Software. New York: Rome Air Development Center.

Noekhah, S., Hozhabri, A. A., & Rizi, H. S. (2013). Software reliability prediction model based on ICA algorithm and MLP neural network. *7th Intenational Conference on e-Commerce in Developing Countries: With Focus on e-Security (ECDC)*, (pp. 1-15).

Pai, P.-F., & Hong, W.-C. (2006). Software reliability forecasting by support vector machines with simulated annealing algorithms. *Journal of Systems and Software , 79* (6), 747-755.

Qin, L.-N. (2011). Software reliability prediction model based on PSO and SVM. *International Conference on Consumer Electronics, Communications and Networks (CECNet)*, (pp. 5236 - 5239). XianNing.

Robinson, D. G., & Dietrich, D. (1987). A New Nonparametric Growth Model. *Reliability , R-36* (4), 411 - 418.

Roy, P., Mahapatra, G. S., Rani, P., Pandey, S. K., & Dey, K. N. (2014). Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction. *Applied Soft Computing , 22*, 629-637.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels.* Cambridge: MIT Press.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural computation , 13* (7), 1443-1471.

Schölkopf, B., Smola, A. J., Williamson, R. C., & Bartlett, P. L. (2000). New support vector algorithms. *Neural computation , 12* (5), 1207-1245.

Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. *Annual Conference on Evolutionary Programming*, (pp. 591-600).

Singh, Y., & Kumar, P. (2010). Prediction of Software Reliability Using Feed Forward Neural Networks. *Computational Intelligence and Software Engineering*, (pp. 1 - 5). Wuhan.

Sitte, R. (1999). Comparison of Software-Reliability-Growth Predictions: Neural Networks vs Parametric-Recalibration. *IEEE Transactions on Reliability , 48* (3), 285-292.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing , 14* (3), 199-222.

Su, Y.-S., & Huang, C.-Y. (2007). Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *The Journal of Systems and Software , 80*, 606-615.

Tamura, Y., & Yamada, S. (2005). Comparison of software reliability assessment methods for open source software". *International Conference on Parallel and Distributed Systems (ICPADS 2005)*, (pp. 488-492). Los Almitos.

Tian, L., & Noore, A. (2005c). Dynamic Software Reliability Prediction: An Approach Based On Support Vector Machines. *International Journal of Reliability, Quality and Safety Engineering , 12* (4), 309-321.

Tian, L., & Noore, A. (2005). Evolutionary neural network modeling for software cumulative failure time prediction. *Reliability Engineering & system safety , 87* (1), 45-51.

Tian, L., & Noore, A. (2005b). On-line prediction of software reliability using an evolutionary connectionist model. *Journal of Systems and Software , 77* (2), 173-180.

Vapnik, V. (1995). *The nature of statistical learning theory.* New York: Springer-Verlag.

Vapnik, V., Golowich, S. E., & Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing. *Advances in neural information processing systems* , 281-287.

Wang, C.-H., & Chen, K.-Y. (2005). Prediction of Software Reliability by Support Vector Regression. *Neural Networks and Brain.*

Wiper, M. P., Palacios, A. P., & Marín, J. M. (2012). Bayesian Software Reliability Prediction Using Software Metrics Information. *Quality Technology and Quantitative Management , 9* (1), 35-44.

Wood, A. (1996). Predicting Software Reliability. *Computer , 29* (11), 69-77.

Wu, Y., & Yang, R. (2011). Study of Software Reliability Prediction Based on GR neural network. *International Conference on Reliability, Maintainability and Safety*, (pp. 688 - 693). Guiyang.

Xu, K., Xie, M., Tang, L. C., & Ho, S. L. (2003). Application of neural networks in forecasting engine systems reliability. *Applied Soft Computing , 2* (4), 255-268.

Yang, B., & Li, X. (2007). A Study on Software Reliability Prediction Based on Support Vector Machines. *Industrial Engineering and Engineering Management*, (pp. 1176-1180). Singapore.

Yang, B., Li, X., Xie, M., & Tan, F. (2010). A generic data-driven software reliability model with model mining technique. *Reliability Engineering and System Safety , 95*, 671-678.

Zemouri, R., & Zerhouni, N. (2012). Autonomous and adaptive procedure for cumulative failure prediction. *Neural Computing and Applications , 21* (2), 319-331.

Zheng, J. (2009). Predicting software reliability with neural network ensembles. *Expert systems with applications , 36* (2), 2116-2122.

[Halaman ini sengaja dikosongkan]

LAMPIRAN

LAMPIRAN 1

Lampiran 1 ini menampilkan hasil uji coba masing-masing data yang dilakukan sebanyak 5 kali untuk tiap data. Hasil yang didapatkan tersebut adalah sebagai berikut.

A. FC1 percobaan 1
Partikel terbaik: 00111110

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	1000	0.7	0.001	0.000946	2.685764
SVR	4800	0.4	0.002	0.002006	3.034060

B. FC1 percobaan 2
Partikel terbaik: 00111110

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	1000	0.7	0.001	0.000946	2.685764
SVR	4800	0.4	0.002	0.002006	3.034060

C. FC1 percobaan 3
Partikel terbaik: 00111110

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	1000	0.7	0.001	0.000946	2.685764
SVR	4800	0.4	0.002	0.002006	3.034060

D. FC1 percobaan 4
Partikel terbaik: 00111110

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	1000	0.7	0.001	0.000946	2.685764
SVR	4800	0.4	0.002	0.002006	3.034060

E. FC1 percobaan 5
Partikel terbaik: 00111110

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	1000	0.7	0.001	0.000946	2.685764
SVR	4800	0.4	0.002	0.002006	3.034060

F. FC2 percobaan 1
Partikel terbaik: 11110111

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	3000	0.2	0.003	0.000378	2.106877
SVR	400	0.1	0.005	0.007177	8.296628

G. FC2 percobaan 2

Partikel terbaik: 10001111

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	3600	0.3	0.004	0.000873	3.119314
SVR	400	0.1	0.005	0.007177	8.296628

H. FC2 percobaan 3

Partikel terbaik: 10001111

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	3600	0.3	0.004	0.000873	3.119314
SVR	400	0.1	0.005	0.007177	8.296628

I. FC2 percobaan 4

Partikel terbaik: 10001111

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	3600	0.3	0.004	0.000873	3.119314
SVR	400	0.1	0.005	0.007177	8.296628

J. FC2 percobaan 5

Partikel terbaik: 10001111

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	3600	0.3	0.004	0.000873	3.119314
SVR	400	0.1	0.005	0.007177	8.296628

K. FC3 percobaan 1

Partikel terbaik: 01000011

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	600	1.0	0.009	0.000383	1.906108
SVR	4800	1.0	0.009	0. 003396	4.338233

L. FC3 percobaan 2

Partikel terbaik: 01000011

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	600	1.0	0.009	0.000383	1.906108
SVR	4800	1.0	0.009	0. 003396	4.338233

M. FC3 percobaan 3

Partikel terbaik: 01000011

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	600	1.0	0.009	0.000383	1.906108
SVR	4800	1.0	0.009	0. 003396	4.338233

N. FC3 percobaan 4

Partikel terbaik: 01111010

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	200	0.7	0.009	0.000359	1.835523
SVR	4800	1.0	0.009	0. 003396	4.338233

O. FC3 percobaan 5

Partikel terbaik: 01111010

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	200	0.7	0.009	0.000359	1.835523
SVR	4800	1.0	0.009	0. 003396	4.338233

P. TBF1 percobaan 1

Partikel terbaik: 00011111

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	200	0.1	0.001	0.083313	95.119252
SVR	200	0.1	0.001	0.083313	95.119252

Q. TBF1 percobaan 2

Partikel terbaik: 00011111

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	200	0.1	0.001	0.083313	95.119252
SVR	200	0.1	0.001	0.083313	95.119252

R. TBF1 percobaan 3

Partikel terbaik: 01000000

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	800	0.1	0.001	0.083244	96.557456
SVR	200	0.1	0.001	0.083666	92.869071

S. TBF1 percobaan 4

Partikel terbaik: 01111100

Metode	Parameter SVR			MSE	AE
	C	σ	ε		
SVR-BPSO	200	0.1	0.001	0.083355	94.553783
SVR	200	0.1	0.001	0.083666	92.869071

T. TBF1 percobaan 5

Partikel terbaik: 01111100

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	200	0.1	0.001	0.083355	94.553783
SVR	200	0.1	0.001	0.083666	92.869071

U. TBF2 percobaan 1

Partikel terbaik: 01000000

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	400	0.1	0.009	0.110245	241.133394
SVR	4800	0.9	0.004	4.951107	1539.6876

V. TBF2 percobaan 2

Partikel terbaik: 01000000

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	400	0.1	0.009	0.110245	241.133394
SVR	4800	0.9	0.004	4.951107	1539.6876

W. TBF2 percobaan 3

Partikel terbaik: 01000000

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	400	0.1	0.009	0.110245	241.133394
SVR	4800	0.9	0.004	4.951107	1539.6876

X. TBF2 percobaan 4

Partikel terbaik: 01000000

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	400	0.1	0.009	0.110245	241.133394
SVR	4800	0.9	0.004	4.951107	1539.6876

Y. TBF2 percobaan 5

Partikel terbaik: 01000000

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	400	0.1	0.009	0.110245	241.133394
SVR	4800	0.9	0.004	4.951107	1539.6876

Z. TBF3 percobaan 1

Partikel terbaik: 10011000

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	4200	0.1	0.001	0.111196	2963.2088
SVR	200	0.1	0.003	0. 111573	2921.7674

AA. TBF3 percobaan 2

Partikel terbaik: 11000100

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	4800	0.1	0.001	0.110990	3004.2157
SVR	200	0.1	0.003	0.111573	2921.7674

BB. TBF3 percobaan 3

Partikel terbaik: 11000100

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	4800	0.1	0.001	0.110990	3004.2157
SVR	200	0.1	0.003	0.111573	2921.7674

CC. TBF3 percobaan 4

Partikel terbaik: 11000100

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	4800	0.1	0.001	0.110990	3004.2157
SVR	200	0.1	0.003	0.111573	2921.7674

DD. TBF3 percobaan 5

Partikel terbaik: 11000100

Metode	**Parameter SVR**			**MSE**	**AE**
	C	**σ**	**ε**		
SVR-BPSO	4800	0.1	0.001	0.110990	3004.2157
SVR	200	0.1	0.003	0.111573	2921.7674

LAMPIRAN 2

Lampiran 2 ini menampilkan data yang dipakai dalam penelitian ini, yaitu data TBF1, TBF2, TBF3, FC1, FC2, dan FC3.

Data TBF1

Failure	Time-to-failure (ms)	Failure	Time-to-failure (ms)	Failure	Time-to-failure (ms)
1	479	30	613	59	1481
2	266	31	277	60	559
3	277	32	1300	61	490
4	554	33	821	62	593
5	1034	34	213	63	1769
6	249	35	1620	64	85
7	693	36	1601	65	2836
8	597	37	298	66	213
9	117	38	874	67	1866
10	170	39	618	68	490
11	117	40	2640	69	1487
12	1274	41	5	70	4322
13	469	42	149	71	1418
14	1174	43	1034	72	1023
15	693	44	2441	73	5490
16	1908	45	460	74	1520
17	135	46	565	75	3281
18	277	47	1119	76	2716
19	596	48	437	77	2175
20	757	49	927	78	3505
21	437	50	4462	79	725
22	2230	51	714	80	1963
23	437	52	181	81	3979
24	340	53	1485	82	1090
25	405	54	757	83	245
26	535	55	3154	84	1194
27	277	56	2115	85	994
28	363	57	884	86	3902
29	522	58	2037		

Data TBF2

Failure	Time-to-failure (s)	Failure	Time-to-failure (s)	Failure	Time-to-failure (s)	Failure	Time-to-failure (s)
1	39	38	129	75	14	112	85
2	10	39	4	76	11	113	240
3	4	40	4	77	41	114	178
4	36	41	35	78	210	115	34
5	4	42	5	79	16	116	102
6	5	43	5	80	30	117	9
7	4	44	22	81	37	118	146
8	91	45	36	82	66	119	59
9	49	46	35	83	9	120	48
10	1	47	121	84	16	121	25
11	25	48	23	85	14	122	25
12	1	49	33	86	24	123	111
13	4	50	48	87	12	124	5
14	30	51	32	88	159	125	31
15	42	52	21	89	89	126	51
16	9	53	4	90	118	127	6
17	49	54	23	91	29	128	193
18	44	55	9	92	21	129	27
19	32	56	13	93	18	130	25
20	3	57	165	94	2	131	96
21	78	58	14	95	114	132	26
22	1	59	22	96	37	133	30
23	30	60	41	97	46	134	30
24	205	61	12	98	17	135	17
25	5	62	138	99	1	136	320
26	129	63	95	100	150	137	78
27	103	64	49	101	382	138	39
28	224	65	62	102	160	139	13
29	186	66	2	103	66	140	13
30	53	67	35	104	206	141	19
31	14	68	89	105	9	142	128
32	9	69	90	106	26	143	34
33	2	70	69	107	62	144	84
34	10	71	22	108	239	145	40
35	1	72	15	109	13	146	177
36	34	73	19	110	4	147	349
37	170	74	42	111	85	148	274

Failure	Time-to-failure (s)	Failure	Time-to-failure (s)	Failure	Time-to-failure (s)	Failure	Time-to-failure (s)
149	82	164	3	179	78	194	90
150	58	165	39	180	3	195	149
151	31	166	63	181	83	196	30
152	114	167	152	182	6	197	317
153	39	168	63	183	212	198	500
154	88	169	80	184	91	199	673
155	84	170	245	185	3	200	432
156	232	171	196	186	10	201	66
157	108	172	46	187	172	202	168
158	38	173	152	188	21	203	66
159	86	174	102	189	173	204	66
160	7	175	9	190	371	205	128
161	22	176	228	191	40	206	49
162	80	177	220	192	48	207	332
163	239	178	208	193	126		

Data TBF3

Failure	Time-to-failure (m)	Failure	Time-to-failure (m)	Failure	Time-to-failure (m)	Failure	Time-to-failure (m)
1	60	19	23	37	10	55	9
2	30	20	1	38	12	56	45
3	540	21	104	39	14	57	68
4	67	22	16	40	33	58	36
5	40	23	9	41	9	59	50
6	23	24	10	42	4	60	81
7	5	25	12	43	66	61	89
8	53	26	4	44	0.5	62	85
9	4	27	10	45	18	63	54
10	16	28	82	46	15	64	3
11	94	29	6	47	75	65	15
12	15	30	54	48	30	66	6
13	5	31	25	49	116	67	8
14	90	32	43	50	14	68	36
15	77	33	12	51	15	69	98
16	68	34	48	52	41	70	32
17	15	35	23	53	1	71	36
18	137	36	6	54	99	72	5

Failure	Time-to-failure (m)	Failure	Time-to-failure (m)	Failure	Time-to-failure (m)	Failure	Time-to-failure (m)
73	9	112	0.5	151	1	190	25
74	60	113	61	152	21	191	175
75	34	114	118	153	104	192	5
76	16	115	20	154	8	193	12
77	164	116	3	155	23	194	18
78	123	117	11	156	1	195	70
79	19	118	87	157	6	196	3
80	19	119	5	158	141	197	10
81	126	120	249	159	3	198	114
82	36	121	28	160	21	199	213
83	54	122	44	161	3	200	212
84	15	123	31	162	18	201	4
85	16	124	3	163	0.5	202	5
86	154	125	10	164	75	203	106
87	84	126	3	165	92	204	264
88	92	127	8	166	39	205	269
89	247	128	17	167	29	206	276
90	244	129	3	168	3	207	1
91	60	130	55	169	2	208	203
92	5	131	7	170	158	209	117
93	2	132	12	171	30	210	1
94	5	133	6	172	24	211	45
95	130	134	4	173	5	212	5
96	0.5	135	169	174	92	213	110
97	233	136	30	175	7	214	18
98	50	137	4	176	33	215	10
99	54	138	38	177	20	216	179
100	52	139	7	178	16	217	66
101	57	140	4	179	292	218	1
102	1	141	4	180	3	219	106
103	2	142	13	181	9	220	2
104	5	143	10	182	12	221	19
105	1	144	40	183	18	222	117
106	17	145	57	184	9	223	30
107	14	146	22	185	75	224	130
108	2	147	37	186	15	225	31
109	87	148	127	187	46	226	28
110	19	149	12	188	9	227	4
111	29	150	8	189	94	228	302

Failure	Time-to-failure (m)	Failure	Time-to-failure (m)	Failure	Time-to-failure (m)	Failure	Time-to-failure (m)
229	362	269	350	349	98	389	1893
230	5	270	470	350	722	390	198
231	63	311	20	351	228	391	3326
232	42	312	79	352	78	392	3100
233	86	313	24	353	33	393	586
234	258	314	540	354	453	394	2686
235	294	315	1040	355	1020	395	124
236	256	316	38	356	4327	396	229
237	118	317	78	357	925	397	1008
238	13	318	41	358	302		
239	47	319	1757	359	649		
240	92	320	205	360	43		
241	343	321	2095	361	185		
242	128	322	788	362	157		
243	392	323	1	363	30		
244	90	324	2668	364	1771		
245	116	325	470	365	1088		
246	35	326	10	366	556		
247	171	327	20	367	55		
248	139	328	338	368	4892		
249	110	329	222	369	81		
250	98	330	28	370	61		
251	60	331	56	371	476		
252	90	332	561	372	63		
253	82	333	65	373	3		
254	5	334	100	374	3		
255	30	335	900	375	62		
256	35	336	212	376	1		
257	231	337	287	377	44		
258	62	338	53	378	1236		
259	158	339	3	379	1406		
260	1622	340	4973	380	109		
261	353	341	197	381	1471		
262	33	342	1174	382	1797		
263	70	343	783	383	1749		
264	35	344	1346	384	2096		
265	116	345	59	385	76		
266	809	346	98	386	2167		
267	1710	347	1594	387	2059		
268	745	348	25	388	2177		

Data FC1

CNF = Cumulative Number of Failure

Week	CNF	Week	CNF	Week	CNF	Week	CNF	Week	CNF
1	16	6	51	11	98	16	127	21	167
2	24	7	60	12	108	17	132		
3	26	8	67	13	114	18	135		
4	38	9	74	14	117	19	150		
5	45	10	89	15	124	20	164		

Data FC2

Week	CNF	Week	CNF	Week	CNF	Week	CNF	Week	CNF
1	5	12	70	23	130	34	181	45	195
2	6	13	71	24	136	35	182	46	196
3	13	14	74	25	141	36	183	47	197
4	22	15	78	26	148	37	185	48	199
5	24	16	90	27	156	38	186	49	201
6	29	17	98	28	164	39	187	50	202
7	34	18	105	29	166	40	188	51	203
8	40	19	110	30	169	41	190		
9	46	20	117	31	170	42	191		
10	53	21	123	32	176	43	192		
11	63	22	128	33	180	44	193		

Data FC3

Week	CNF	Week	CNF	Week	CNF	Week	CNF	Week	CNF
1	7	18	174	35	319	52	387	69	454
2	8	19	183	36	323	53	387	70	456
3	36	20	196	37	324	54	387	71	456
4	45	21	200	38	338	55	388	72	456
5	60	22	214	39	342	56	393	73	457
6	74	23	223	40	345	57	398	74	458
7	82	24	246	41	350	58	400	75	459
8	98	25	257	42	352	59	407	76	459
9	106	26	277	43	356	60	413	77	459
10	115	27	283	44	367	61	414	78	460
11	120	28	286	45	373	62	419	79	460
12	134	29	292	46	373	63	420	80	460
13	139	30	297	47	378	64	423	81	461
14	142	31	301	48	381	65	429		
15	145	32	302	49	383	66	440		
16	153	33	310	50	384	67	443		
17	167	34	317	51	384	68	448		

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Vika Fitratunnany Insanittaqwa, lahir di Kediri pada tanggal 15 Juli 1991, merupakan anak pertama dari tiga bersaudara. Penulis telah menempuh pendidikan dari jenjang dasar hingga menengah di kota Malang, yaitu di TK Laboratorium UM Malang (1995-1997), SD Laboratorium UM Malang (1997-2003), MTsN Malang I (2003-2006), dan SMA Negeri 1 Malang (2006-2009). Penulis memperoleh gelar sarjana setelah menempuh perkuliahan di jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya (ITS) pada tahun 2009 hingga 2013 di bidang minat Rekayasa Perangkat Lunak. Pada tahun 2014, penulis mendapatkan beasiswa *fresh graduate* dari DIKTI untuk melanjutkan studi program pascasarjana di jurusan Teknik Informatika ITS. Penulis memiliki ketertarikan pada bidang *Software Reliability*, *Software Development*, dan *Game Development.* Penulis dapat dihubungi melalui email di vika.fitra@gmail.com atau vikachew@gmail.com