SOFTWARE DEFECT PREDICTION MENGGUNAKAN ALGORITMA K-NN YANG DIOPTIMASI DENGAN PSO

Taufik Hidayat, Ahmad Faqih Habibi, Umi Laili Yuhana
Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember Surabaya
Email: taufikhidayat.19051@mhs.its.ac.id

Abstrak. *Untuk menjamin kualitas dari perangkat lunak, kita perlu meminimalisir defect yang terjadi pada perangkat lunak. Salah satu bidang penelitian dalam penentuan kualitas perangkat lunak adalah Software defect prediction (SDP) untuk prediksi kemungkinan terjadinya defect pada perangkat lunak yang akan dikembangkan sehingga dapat membantu pengembang untuk mengetahui apakah pada perangkat lunak yang dikembangkan berpeluang terjadi defect atau tidak sekaligus menghemat biaya. Metode yang digunakan adalah metode klasifikasi menggunakan algoritma k-NN yang dioptimasi menggunakan algoritma PSO untuk mendapatkan nilai akurasi maksimum. Dari hasil penelitian didapatkan akurasi terendah pada dataset MC2 dengan hasil 71,05% dan tertinggi pada dataset PC2 dengan hasil 99,15%.*

Kata Kunci: Software defect Prediction, k-Nearest Neighbor, Particle Swarm Optimization.

Salah satu aspek penting dalam proses pengembangan perangkat lunak adalah nilai kegunaan dari perangkat lunak itu sendiri. Untuk menghasilkan perangkat lunak yang memiliki kualitas yang baik perlu meminimalisir *defect* pada perangkat lunak. Ada beberapa jenis *defect* pada perangkat lunak, yaitu *defect* kebutuhan (*requirement defect*), *defect* desain (*design defect*), dan *defect* kode (*code defect*)[1]. Salah satu bidang penelitian dalam penentuan kualitas perangkat lunak adalah *Software defect prediction* (SDP) untuk memprediksi kemungkinan terjadinya *defect* pada perangkat lunak yang akan dikembangkan.

Telah banyak penelitian yang berfokus pada SDP seperti pada[1], penelitian tersebut menggunakan metode pendekatan *Data Mining* yaitu *Association Rule Mining* pada *dataset* NASA MDP CM1 yang menghasilkan tingkat akurasi 82,35%, kemudian pada [6], menggunakan metode *Siamese Parallel fully-connected Neural Network* (SPFCNN) yang menghasilkan performa yang lebih baik daripada *deep learning* untuk *dataset* berdimensi tinggi namun masih belum dapat dipakai untuk jenis *dataset Software defect* yang lain.

Selain itu, juga terdapat banyak penelitian yang menggunakan berbagai macam *dataset*, *framework* dan metode. Setidaknya ada 7 metode *machine learning* yang paling sering digunakan yaitu *Logistic Regression* (LR), *Naïve Bayes* (NB), *k-Nearest Neighbor* (k-NN), *Neural Network* (NN), *Decision Tree* (DT), *Support Vector Machine* (SVM) dan *Random Forest* (RF). Dengan NB, DT, NN dan RF adalah 4 metode yang paling sering digunakan.[8]

Algoritma *k-Nearest Neighbor* (k-NN) adalah salah satu algoritma klasifikasi yang sederhana dan dapat dilakukan pengoptimasian nilai k untuk menghasilkan akurasi yang maksimum[9], oleh karena itu pada penelitian ini, penulis mengusulkan metode k-NN yang dioptimasi menggunakan algoritma *Particle Swarm Optimization* (PSO).

Software defect

Sebuah *defect* merupakan ketidaksempurnaan dari sebuah *software*, yang mana *defect* tersebut mengurangi kualitas ataupun kegunaan suatu *software* tersebut. *Software defect* sering didapati pada program komputer, data, metode yang digunakan dalam pengembangan dan pengelolaan *software*. Sebuah *defect* lebih sering disebut *fault* atau *bug* ketika ditemukan di dalam *executable code*. *Fault* tetap berada di dalam *software* himgga *software* tersebut dijalankan[2].

Ada beberapa kategori kesalahan yang dapat digolongkan sebagai *software defect* diantaranya *User Interface Error*, merupakan *error* ketika tampilan dari software berbeda

dari spesifikasi, *Calculation Error*, merupakan kesalahan pada perhitungan matematika dan logika pemrograman, *Error in Interpreting Data,* kesalahan akibat melewatkan dan adanya konversi data antar sistem, dan *Testing Error*, terjadi karena tester membuat kesalahan selama percobaan.

Seperti yang disebutkan sebelumnya, *software defect* adalah salah satu faktor penting yang mempengaruhi kualitas perangkat lunak. Kualitas dari perangkat lunak harus ditingkatkan dengan mencegah munculnya *defect* melalui perbaikan aksi yang mungkin menghasilkan *defect* pada proses pengembangan perangkat lunak[5]. Selain itu, pencegahan *software defect* juga berdampak pada pengurangan varian proyek dan peningkatkan stabilitas proses perangkat lunak. Faktor-faktor penyebab cacat bervariasi sesuai dengan atribut yang berbeda dari suatu proyek, termasuk pengalaman pengembang, kompleksitas produk, alat pengembangan dan lain lain [7].

Pencegahan *defect* dapat dilakukan dengan prediksi cacat pada tahap awal pengembangan perangkat lunak, yang juga akan memberikan keuntungan tambahan pada perencanaan sumber daya[2]. Prediksi Cacat dalam perangkat lunak dipandang sebagai salah satu operasi yang paling berguna dan efisien biaya[3]. Prediksi cacat perangkat lunak telah menyebabkan kekhawatiran luas di kalangan peneliti rekayasa perangkat lunak, yang bertujuan untuk mendirikan prediktor cacat perangkat lunak menurut data historis[6]. Prediksi cacat perangkat lunak memerlukan pengembangan teknik baru yang bertujuan untuk memprediksi secara akurat modul yang rusak dalam perangkat lunak[4].

k-Nearest Neighbor

Algoritma *k-Nearest Neighbor* (k-NN) merupakan salah satu algoritma klasifikasi yang sederhana dan sering digunakan. Pengklasifikasian dilakukan dengan menghitung jarak dari setiap fitur dari data testing dan memilih data training sejumlah k dengan jarak terdekat. Hasil klasifikasi bergantung pada, 1) nilai k yang merepresentasikan jumlah data training terdekat. Untuk menghindari hasil klasifikasi yang memenuhi dua label atau lebih, k harus berupa angka ganjil. Jika k terlalu kecil, maka dapat menyebabkan *overfitting* dan lebih sensitif terhadap noise dalam data, sedangkan nilai k yang besar dapat menyebabkan bias yang lebih tinggi dan akurasi yang lebih rendah, 2) jumlah dan label data training, 3) *distance metric* yang digunakan misalnya *Euclidean distance, Minkowski distance, Chebyshev distance* dan lain lain[8]. *Distance metric* yang biasanya digunakan adalah jarak *Euclidean* yang dirumuskan pada persamaan,

$$d(x_i, x_j) = \sum_{k=1}^{d}(x_{ik} - x_{jk})^2 \qquad (1)$$

di mana d (x_i, x_j) merupakan jarak antara dua sampel x_i dan x_j, (x_i, x_j) $\in R^m$, $x_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}$, dan m adalah dimensi dari sampel[9].

Berikut *pseudo code* untuk algoritma k-NN

1. Data latih X = $\{(x_1,y_1), \ldots, (x_N,y_N)\}$, dimana $x_i \in X$ menyatakan data ke-i, y_i menyatakan label dari data ke-i, dan N menyatakan jumlah data training.
2. Tentukan nilai k
3. **for** semua data training (i = 1, 2, …, N) **do**
4. Hitung jarak antara data uji dengan data latih
5. **end for**
6. Pilih tetangga terdekat sejumlah k
7. Tentukan kelas dari data uji dengan memilih kelas dengan tetangga terbanyak.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) adalah algoritma optimasi yang pertama kali diperkenalkan oleh Dr. Eberhart dan Dr. Kennedy pada tahun 1995, optimasi pada algoritma ini meniru proses yang terjadi pada burung (*flock of bird*) dan ikan (*school of fish*) untuk bertahan hidup. Algoritma *Particle Swarm Optimization (*PSO) merupakan salah satu metode dari algoritma *Swarm Intelligence* di dalam dunia komputasi. *Particle Swarm Optimization (*PSO) memberikan solusi dalam menyelesaikan masalah manusia di segala bidang kehidupan yang terkait optimasi. Berkembangnya teknologi pada bidang kecerdasan buatan di jaman sekarang dan yang akan datang tentunya tidak bias lepas dari pemanfaatan metode algoritma *Particle Swarm Optimization (*PSO). Tidak hanya di bidang teknologi, algoritma *Particle Swarm Optimization (*PSO) juga sangat membantu dalam dunia ekonomi dan bisnis. Algoritma *Particle Swarm Optimization (*PSO) dapat diterapkan untuk mengoptimasi kinerja perusahaan, seperti memaksimalkan bidang

produksi, meminimalkan kerugian, dan mengoptimasi pengelolaan karyawan.

Algoritma PSO merupakan algoritma optimasi yang bertujuan untuk mencari solusi terdekat dengan *global minimum* atau *global maksimum*. Dimensi pencarian ditentukan oleh jumlah parameter yang diperlukan untuk optimasi. Dengan kata lain, jika PSO digunakan untuk mengoptipasi n parameter maka dimensi pencariannya adalah n, dan jumlah variabel dalam fungsi tujuan adalah n. Dalam algoritma PSO, jumlah partikel ditentukan oleh pengguna dan partikel-partikel ini ditempatkan secara acak di ruang pencarian. Posisi masing-masing partikel saat ini digunakan untuk menghitung nilai kecocokannya berdasarkan fungsi yang digunakan. Setiap partikel memiliki tiga parameter, yaitu, posisi, kecepatan, dan posisi terbaik sebelumnya. Selain itu, parameter penting lainnya adalah posisi partikel yang memiliki nilai kecocokan terbaik yang disebut sebagai global minimum atau maksimum.

Posisi setiap partikel dinyatakan pada koordinat yang mewakili titik di ruang pencarian. Selama proses pencarian, posisi dari semua partikel dievaluasi untuk menunjukkan apakah posisi saat ini lebih baik daripada posisi terbaik sebelumnya. Dengan kata lain, partikel akan menyimpan posisi yang memiliki nilai lebih baik yaitu partikel yang memiliki solusi lebih dekat ke solusi lokal atau global.[8]

Kecepatan setiap partikel dalam setiap iterasi dihitung sesuai dengan Persamaan. (2) Dari Persamaan. (2), kecepatan baru setiap partikel di ruang pencarian ditentukan oleh:

1. Gerakan saat ini atau kecepatan asli partikel tersebut.
2. Posisi posisi terbaik sebelumnya dari partikel yang disebut sebagai komponen kognitif. Istilah ini digunakan untuk mengatur kecepatan menuju posisi terbaik yang dikunjungi oleh partikel tersebut.
3. Komponen sosial yang digunakan untuk menyesuaikan kecepatan menuju posisi terbaik global di semua partikel.

Posisi baru dari partikel kemudian dihitung dengan menambahkan kecepatan dan posisi saat ini dari partikel tersebut seperti dalam Persamaan. (3) Dengan kata lain, algoritma PSO menggunakan x, p, v, dan G, untuk mencari posisi yang lebih baik dan terus menggerakkan partikel menuju solusi global.

$$v^i_{(t+1)} = w v^i_{(t)} + C_1 r_1 (p^i_{(t)} - x^i_{(t)}) + C_2 r_2 (G - x^i_{(t)}) \quad (2)$$

$$x^i_{(t+1)} = x^i_{(t)} + v^i_{(t+1)} \quad (3)$$

di mana w adalah inersia, C_1 adalah faktor pembelajaran kognisi, C_2 adalah faktor pembelajaran sosial, r_1, r_2 adalah angka acak yang dihasilkan dalam kisaran [0, 1], dan p^i adalah solusi terbaik dari partikel ke-i. Karena kecepatan partikel tergantung pada variabel acak, maka partikel-partikel tersebut dipindahkan secara acak. Oleh karena itu, gerakan partikel-partikel ini disebut *random walk.*

Berikut *pseudo code* dari algoritma PSO

1. Inisialisasi posisi partikel (x_i), kecepatan (v_i), posisi terbaik sebelumnya (p_i), dan jumlah partikel N.
2. **While** (t < jumlah iterasi) **do**
3. **for** semua partikel (i) **do**
4. Hitung nilai *fitness function* ($F(x_i)$) untuk posisi saat ini
5. **if** $F(x_i) > F(p_i)$ **then**
6. $p_i = x_i$
7. **end if**
8. **if** $F(x_i) > F(G)$ **then**
9. $G = x_i$
10. **end if**
11. Hitung posisi dan kecepatan untuk iterasi selanjutnya
12. **end for**
13. **end while**

I. Metodologi

Data Preprocessing

Data yang digunakan merupakan *dataset* NASA MDP diantaranya adalah CM1, KC3, MC2, MW1, PC1, PC2, PC3 dan PC4 dengan rincian sebagai pada Tabel 1,

Tabel 1. Rincian *dataset* NASA yang digunakan

Dataset	Jumlah Modul	*Defect*	Non - *Defect*	Rasio (%)
CM1	344	42	302	12,21
KC3	200	36	164	18
MC2	128	44	84	34,33
MW1	264	27	237	10,23
PC1	759	61	698	8,03
PC2	1585	16	1569	1,01
PC3	1125	140	985	12,44
PC4	1399	178	1221	12,72

Delapan *dataset* tersebut kemudian dibagi menjadi data latih dan data uji dengan perbandingan untuk masing masing *dataset* adalah 70 % : 30 %, pembagian data latih dan data uji disajikan pada Tabel 2.

Tabel 2. Persebaran *dataset*

Dataset	Pembagian	*Defect*	Non - *Defect*	Total
CM1	Latih	30	211	241
	Uji	12	91	103
KC3	Latih	26	114	140
	Uji	10	50	60
MC2	Latih	31	58	89
	Uji	13	25	38
MW1	Latih	19	165	184
	Uji	8	72	80
PC1	Latih	43	488	531
	Uji	18	210	228
PC2	Latih	12	1098	1110
	Uji	4	471	475
PC3	Latih	98	689	787
	Uji	42	296	338
PC4	Latih	125	854	979
	Uji	53	367	420

Inisialisasi Parameter

Pada tahap ini, parameter dari PSO ditentukan seperti banyaknya partikel, kecepatan maksimum dan minimum, posisi awal, dan kecepatan awal. Algoritma PSO akan mencari nilai k yang optimum untuk melakukan klasifikasi.

Evaluasi Akurasi

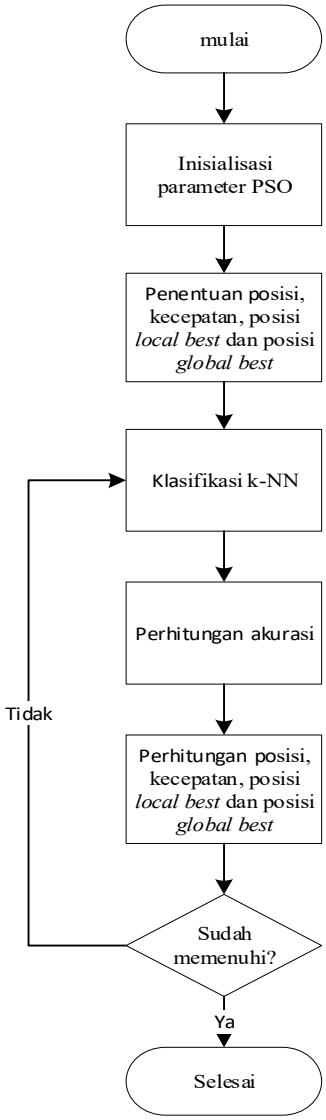
Pada algoritma PSO-kNN setiap data tes diuji untuk setiap posisi partikel pada PSO yang menyatakan nilai k, kemudian posisi dari setiap partikel akan dievaluasi berdasarkan akurasi dari hasil klasifikasi, akurasi pada klasifikasi dihitung menggunakan rumus

$$Akurasi = \frac{TP+}{TP+TN+FP+FN} \quad (4)$$

Dimana TP adalah True Positive, TN adalah True Negative, FP adalah False Positive, FN adalah False Negative.

Flowchart

Flowchart algoritma PSO k-NN ditunjukkan pada Gambar 1.



Gambar 1. *Flowchart* algoritma PSO k-NN

Langkah pertama adalah inisialisasi parameter PSO, kemudian partikel yang ditentukan dimana posisi awal pada batas minimum dan maksimum k, kecepatan awal, posisi *local best* dan posisi *global best* adalah 0, kemudian dilakukan klasifikasi menggunakan k-NN dengan nilai k adalah posisi masing-masing partikel, kemudian dilakukan perhitungan akurasi menggunakan persamaan (4), setelah didapat nilai perhitungan akurasi untuk masing-masing partikel, program akan menetukan *local best* masing-masing partikel dan *global best* dari keseluruhan partikel. Nilai *local best* dan *global best* akan dipakai untuk perhitungan posisi partikel dan kecepatan partikel

selanjutnya dengan persamaan (2) dan (3). Operasi tersebut akan diulang berkali kali hingga partikel konvergen pada posisi *global best*.

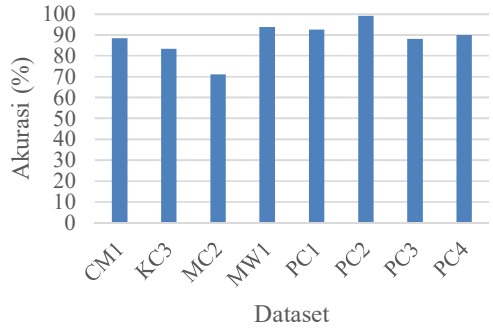
II. Hasil dan Pembahasan

Setiap partikel pada akan ditempatkan secara acak pada masing masing nilai k, kemudian masing masing partikel akan melakukan penghitungan nilai akurasi, kecepatan, dan posisi partikel pada iterasi selanjutnya. Dari perhitungan akurasi, didapat nilai maksimum local untuk tiap tiap partikel dan maksimum global dari keseluruhan partikel, parameter ini yang akan menetukan kemana arah partikel bergerak dan posisi pada iterasi selanjutnya.

Untuk parameter dari PSO, digunakan maksimal jumlah partikel = $2N_{dl} - 1$ dimana N_{dl} merupakan jumlah *defect* pada data uji untuk mencegah terjadinya ketidakseimbangan jumlah tetangga antara *defect* dan *non-defect*, kecepatan awal = 0, kecepatan minimum dan maksimum adalah -2 dan 2. Tabel 3 dan Gambar 2 menunjukkan hasil pengukuran akurasi dan nilai k dari algortima PSO-kNN untuk masing-masing *dataset*.

Tabel 3. Hasil pengukuran akurasi

Dataset	k	Akurasi (%)
CM1	9	88,34
KC3	17	83,33
MC2	5	71,05
MW1	19	93,75
PC1	11	92,54
PC2	7	99,15
PC3	11	88,16
PC4	3	90



Gambar 2. Grafik pengukuran akurasi

Pada Tabel 3 disajikan data akurasi untuk masing-masing dataset. Dapat dilihat bahwa, nilai akurasi bervariasi mulai dari 71,05% pada dataset MC2 hingga 99,15% pada dataset PC2. Dataset MC2 memiliki akurasi yang rendah dikarenakan jumlah modul yang terdapat pada dataset MC2 memiliki jumlah yang sedikit yaitu total hanya 128 modul, jumlah modul yang sedikit menyebabkan terjadinya underfitting, yaitu fenomena yang terjadi akibat kurangnya data latih yang dipakai untuk memodelkan permasalahan. Dataset PC2 memiliki nilai akurasi yang tinggi dikarenakan memiliki jumlah modul yang banyak sehingga mesin dapat memodelkan permasalahan dengan baik, namun masih terdapat kekurangan yaitu jumlah rasio modul data defect terhadap total jumlah modul yang terlalu kecil.

III. Kesimpulan

Penelitian ini membahas tentang prediksi kemungkinan terjadinya *defect* pada perangkat lunak menggunakan algoritma klasifikasi k-NN. Klasifikasi dioptimasi menggunakan algoritma PSO untuk mendapat nilai akurasi yang maksimum. Hasil penelitian menunjukkan akurasi paling rendah terjadi pada *dataset* MC2 dengan akurasi 71,05 % dan paling tinggi pada PC2 dengan akurasi 99,15 %. Nilai akurasi cenderung lebih rendah pada *dataset* dengan jumlah modul yang lebih sedikit.

Saran untuk penelitian selanjutnya adalah optimasi tidak hanya dilakukan pada nilai akurasi namun juga pada parameter lainnya seperti presisi.

IV. Daftar Pustaka

[1] R. S. Perdana and U. L. Yuhana. (2015). Prediksi Code Defect Perangkat Lunak Dengan Metode Association Rule Mining dan Cumulative Support Thresholds. *J. Buana Inform.*, vol. 6, no. 2, pp. 113–120.

[2] T. M. Khoshgoftaar and J. C. Munson. (1990). Software Complexity Metrics. vol. 8, no. 2.

[3] I. Arora, V. Tetarwal, and A. Saha. (2015). Open issues in Software defect prediction. Procedia Comput. Sci., vol. 46, no. Icict 2014, pp. 906–912.

[4] S. Ghosh, A. Rana, and V. Kansal. (2018). ScienceDirect A Nonlinear

Manifold Detection based Model for Software defect Prediction. Procedia Comput. Sci., vol. 132, pp. 581–594.

[5] V. Boehm, B. (2001). Bassili, *Software defect Reduction Top 10 List*.

[6] L. Zhao, Z. Shang, L. Zhao, T. Zhang, and Y. Yan. (2019). Neurocomputing Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks. vol. 352, pp. 64–74.

[7] C. P. Chang and C. P. Chu. (2007). Defect prevention in software processes: An action-based approach. J. Syst. Softw., vol. 80, no. 4, pp. 559–570.

[8] R. S. Wahono, N. S. Herman, and S. Ahmad. (2014). A comparison framework of classification models for Software defect prediction. *Adv. Sci. Lett.*, vol. 20, no. 10–12, pp. 1945–1950.

[9] A. Tharwat, H. Mahdi, M. Elhoseny, and A. E. Hassanien. (2018). Recognizing human activity in mobile crowdsensing environment using optimized k-NN algorithm," *Expert Syst. Appl.*, vol. 107, pp. 32–44.