IMPLEMENTASI MACHINE LEARNING DALAM DETEKSI BUG OTOMATIS PADA KODE SUMBER OPEN SOURCE

Hery Pitunas, Wahyu Teja Kusuma*, Ahsanun Naseh Khudori
Informatika, Institut Teknologi, Sains, dan Kesehatan RS.DR. Soepraoen Kesdam V/BRW
Jl. S. Supriadi No. 22, Sukun, Kec. Sukun, Kota Malang, Indonesia
wtkusuma@itsk-soepraoen.ac.id

ABSTRAK

Dalam pengembangan software, menghilangkan bug penting untuk menjaga kualitas aplikasi. Developer sering menghabiskan banyak waktu untuk mengatasi dan menemukan solusi bug yang dialami. Beragamnya jenis bug membuat proses ini menjadi semakin kompleks dan memakan waktu. Berdasarkan hal tersebut, klasifikasi bug menjadi solusi penting. Data penyelesaian bug yang tersedia secara public dapat dimanfaatkan untuk klasifikasi dan prediksi otomatis menggunakan machine learning. Penelitian ini menerapkan KNN untuk mendeteksi bug otomatis pada kode sumber *open source*. Hasilnya menunjukkan bahwa parameter k dalam KNN dan k-fold pada *cross-validation* berpengaruh signifikan terhadap performa model. Nilai k yang kecil membuat model lebih sensitive terhadap noise dan rentan overfitting, sedangkan nilai k ynag besar meningkatkan stabilitas dan generalisasi meskipun akurasi menurun. K-fold yang lebih besar menghasilkan model yang lebih stabil dengan akurasi yang lebih tinggi. Akurasi terbaik yang diperoleh adalah 0,9025 dengan presisi, recall, dan *f-measure* yang perlu dioptimalkan.

Kata kunci : *Bug, Software, Machine learning, KNN, Cross-validation*

1. PENDAHULUAN

Bug kode sumber merupakan bagian yang tidak terpisahkan dalam pengembangan software [1].

Menemukan dan menghilangkan bug menjadi aspek penting yang harus dilakukan dalam mengembangkan software untuk memastikan kualitas aplikasi yang dikembangkan [2].

Penelitian yang telah ada menunjukkan bahwa para developer menghabiskan lebih dari sebagian waktu mereka untuk memperbaiki bug dan hanya sebagian kecil waktu mereka gunakan untuk penambahan fitur di aplikasi mereka yakni hanya sekitar 36% [3].

Memperbaiki bug meliputi penentuan kenapa aplikasinya error, bagian mana yang error, kemudian memperbaiki baggian yang error tersebut. Developer yang memperbaiki bug menghasilkan perubahan pada kode sumber yang dapat diidentifikasi dengan jelas sebagai perubahan perbaikan bug aplikasi. Namun, proses identifikasi penyebab bug menjadi lebih sulit [3].

Ketika terjadi bug dalam aplikasi yang dikembangkan, developer biasanya mencari bug serupa dengan bug yang dialaminya sebagai inspirasi perbaikan bug. Saat ini telah banyak repositori yang menyediakan data histori perbaikan bug. Namun, sebagian besar histori perbaikan bug yang tersedia menggunakan pola perbaikan berulang sehingga semakin banyak waktu yang diperlukan developer untuk membaca dan mempelajari pola perbaikan bug untuk mendapat solusi yang paling tepat untuk menyelesaikan permasalahannya [4].

Tindakan perbaikan bug ini berkaitan erat dengan jenis penyebab bug. Sehingga jika histori perbaikan bug diklasifikasikan berdasarkan penyebabnya dan terdapat pola perbaikan disetiap jenis penyebab bug, efisiensi perbaikan bug dapat ditingkatkan [5].

Sebelumnya dijelaskan bahwa setiap sumber kode dari histori bug yang diperbaiki menyimpan banyak pengetahuan empiris seperti pola perbaikan. Salah satu pendekatan yang dapat digunakan untuk mengenali pola pada histori bug ialah machine learning. Machine learning memiliki kemampuan untuk mengenali pola dari data histori untuk melakukan prediksi dimasa yang akan datang [6].

Teknik machine learning supervised learning yang paling banyak dan umum diterapkan untuk prediksi bug software dan perkembangan terkini machine learning telah berkontribusi meningkatkan kinerja dan kemampuan model machine learning untuk memprediksi bug software [7].

Penelitian sebelumnya menggunakan machine learning random forest untuk menganalisis dan mendeteksi laporan bug aplikasi dengan menggunakan kumpulan data bug pada perangkat lunak yang diperoleh dari system pelacakan bug pada open source. Teknik random forest yang digunakan menghasilkan akurasi sebesar 0,75 [8].

Penelitian lain yang dilalukan Sharma [9] melakukan analisis terhadap teknik machine learning untuk analisis sumber kode. Hasil yang diperoleh penelitian ini menunjukkan machine learning banyak diterapkan dalam rekayasa perangkat lunak. Berdasarkan hal tersebut, penerapan machine learning dalam rekayasa perangkat lunak memiliki peluang dan tantangan tersendiri. Penelitian ini menyajikan berbagai tantangan dan tantangan dilapangan dalam penerapan machine learning.

Penelitian lain oleh Mauricio [10] meneliti tentang efektifitas algoritma machine learning untuk memprediksi refaktori perangkat lunak. Penelitian ini

menggunakan beberapa teknik machine learning dengan dataset yang diperoleh dari beberapa sumber seperti Apache, Github, dan lain-lain. Hasil yang diperoleh yakni teknik random forest memberikan hasil yang terbaik dalam memprediksi refaktori perangkat lunak.

Beberapa penelitian tersebut menunjukkan bahwa machine learning berpeluang diintegrasika dalam rekayasa perangkat lunak. Berdasarkan hasil penelitian-penelitian tersebut dan peluang yang tersedia, peneliti terdorong untuk melaksanakan penelitian dengan mengimplementasikan machine learning untuk deteksi bug otomatis pada sumber kode open source. Teknik machine learning yang akan digunakan yakni k-nearest neighbor (kNN). KNN ialah teknik supervised learning yang sangat popular digunakan untuk penugasan klasifikasi [11].

Model KNN yang dibangun, nantinya untuk meminimalisir terjadinya overfitting, akan dilakukan evaluasi menggunakan teknik k-fold *cross-validation*. Diharapkan penelitian ini dapat berkontribusi memperluas wawasan terkait penerapan machine learning dalam rekayasa perangkat lunak.

2. TINJAUAN PUSTAKA

2.1. Bug Pada Perangkat Lunak

Bug merupakan kesalahan atau cacat dalam kode program yang dibangun yang mengakibatkan perangkat lunak tidak berfungsi sebagaimana mestinya [12].

Terdapat banyak jenis dari bug kode program diantaranya syntax bug, logic bug, runtime bug, memory leak, dan security bug. Logic bug dan syntax bug merupakan jenis bug yang paling umum ditemui pada kode program. Dalam menuliskan kode program, sangat penting memiliki pemahaman logis alur dari kode program yang dibangun. Ketika terdapat kesalahan logis, kode program dapat memberikan hasil yang tidak sesuai dengan alur yang semestinya [4].

2.2. Machine Learning

Definisi machine learning menurut Arthur Samuel ialah bidang studi yang memberikan kemampuan computer untuk belajar mandiri tanpa harus deprogram secara eksplisit [13].

Ketika berhadapan dengan data dalam jumlah besar, manusia kesulitan untuk menggali informasi yang terdapat pada data, disinilah machine learning berperan. Machine learning diajari bagaimana cara menangani data dalam jumlah besar tersebut dan mengekstrak beragam informasi yang tersirat dalam data [14].

Machine learning (ML) banyak digunakan untuk mengatasi berbagai permasalahan dibidang klasifikasi, regresi, dan klustering tergantung pada jenis data yang digunakan untuk melatih model ML [15].

ML terdiri dari dua jenis pembelajaran secara umum yakni pembelajaran yang diawasi (supervised) dan pembelajaran tanpa pengawasan (unsupervised) [16].

Supervised learning berguna untuk mengatasi data yang terstruktur yang telah diklasifikasikan dalam kategori atau kelas tertentu, sedangkan unsupervised learning lebih digunakan untuk mengatasi data yang tidak terstruktur [17].

Supervised learning memiliki beberapa teknik yang dapat diterapkan untuk mengatasi penugasan klasifikasi, regresi, dan klasifikasi, salah satunya yakni k-nearest neighbor (KNN).

2.3. K-Nearest Neighbor

K-nearest neighbor yang merupakan bagian dari teknik ML *simple* dengan performa yang sangat bagus untuk tugas klasifikasi dan regresi [18].

KNN memiliki data tahan yang kuat terhadap data latih yang mengandung noise dan sangat bagus digunakan untuk data latih dalam jumlah besar [19].

Cara kerja dari KNN dengan mencari tetangga terdekat berdasarkan jarak tertentu menggunakan nilai k dan menggunakannya untuk menentukan hasil klasifikasi [20].

Dalam pembelajarannya, kNN menggunakan perbandingan data sebelumnya dengan data yang terbaru dan dalam menentukan jarak terdekat, kNN menggunakan rumus Euclidean distance sebagai metode paling umum digunakan pada KNN [21].

Euclidean distance dapat menghasilkan akurasi lebih tinggi daripada metode perhitungan jarak yang lain seperti Cosine distance, Hamming, dan lain sebagainya [22].

Rumus Euclidean distance sebagaimana berikut.

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad (1)$$

n disini merupakan banyaknya atribut dalam dataset, x menyatakan vector atribut real data, y menyatakan vector atribut dari hasil perhitungan pada data, dan d(x,y) merupakan jarak Euclidean dari y ke y.

2.4. K-Fold *Cross-validation*

Salah satu teknik evaluasi yang digunakan untuk mengevaluasi kinerja model machine learning ialah *cross-validation* [23].

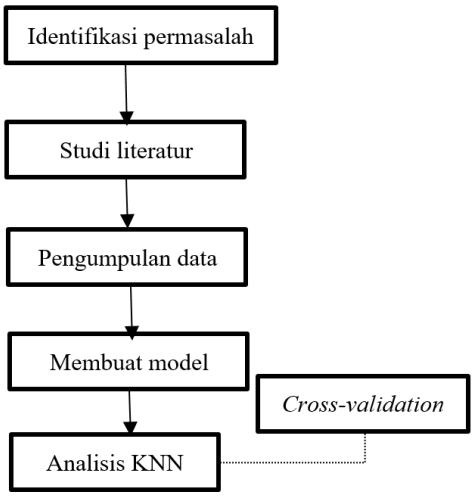
Cross-validation bagus untuk megevaluasi sejauh mana model mengalami overfit [24].

Konsep kerja dari *cross-validation* membagi setiap kumpulan data kedalam subset dan menjadikan setiap subset sebagai data validasi untuk mengevaluasi model secara bergantian sebanyak jumlah nilai k yang ditentukan, umumya peneliti menentukan nilai k sebesar 10 [24].

Teknik *cross-validation* membagi dataset menjadi beberapa subset yang sama besar sesuai dengan nilai k yang ditentukan untuk memastikan model tidak mengalami overfitting ataupun underfitting. Kinerja dari *cross-validation* ini, model dilatih sebanyak k kali dimana disetiap subset secara bergantian sebagai data uji dan sisa subset yang tidak

digunakan sebagai data uji digunakan sebagai data latih.

3. METODE PENELITIAN



Gambar 1. Tahapan penelitian

Gambar 1 menyajikan alur yang peneliti lakukan selama penelitian. Penelitian ini diawali dengan mengidentifikasi permasalahan dibidang perangkat lunak terutama bug pada pengembangan software dan melakukan studi literature untuk menggali wawasan dari penelitian sebelumnya dibidang rekayasa perangkat lunak, machine learning, penerapan machine learning dalam rekayasa perangkat lunak, dan teknik k-fold *cross-validation*. Setelah mendapat wawasan dan peluang penelitian dari penelitian terdahulu, langkah selanjutnya yang dilakukan yakni mulai mengumpulkan data. Tahapan pengumpulan data dengan mengambil dataset yang telah tersedia secara publik untuk digunakan melati model KNN yang akan dibangun menggunakan python. Setelah model KNN berhasil dibuat, kemudian dilakukan analisis terhadap model menggunakan teknik *cross-validation*. Nilai k yang ditentukan pada saat evaluasi ini sebesar 10. Performa teknik ML umumnya ditentukan berdasarkan akurasi yang dihasilkan. Namun, akurasi yang dihasilkan oleh model tidak selalu dapat digunakan sebagai acuan dalam menentukan tingkat kualitas KNN terutama jika data yang dilatih tidak seimbang. Sehingga performa KNN juga diukur berdasarkan nilai presisi, *recall*, dan *f measure* [25], [26].

Presisi mengukur seberapa banyak prediksi yang positif benar terhadap semua data yang diprediksi positif oleh model. *Recall* mengukur seberapa banyak data yang sebenarnya positif diklasifikasikan dengan benar oleh model terhadap semua data yang memang positif. F *measure* merupakan rata-rata antara presisi dan *recall*. F *measure* ini berguna untuk menjaga keseimbangan antara presisi dan *recall* [27].

Rumus yang digunakan untuk menghitung nilai akurasi, presisi, *recall*, dan f *measure* sebagaimana berikut.

$$\text{Akurasi} = \frac{TP+TN}{TP+TN+FP+FN} \quad (2)$$

$$\text{Presisi} = \frac{TP}{TP+FP} \quad (3)$$

$$\textit{Recall} = \frac{TP}{TP+FN} \quad (4)$$

$$\text{F } \textit{measure} = \frac{presisi*recall}{presisi+recall} \quad (5)$$

TP adalah *true positif* yang berarti nilai prediksi dan nilai aktualnya sama-sama bernilai positif. FP adalah *false positive* yang berarti bahwa banyaknya data negative yang oleh model ML diklasifikasikan sebagai positif. FN adalah *false negative* yang berarti banyaknya data positif yang oleh model ML diklasifikasikan sebagai *negative*. TN adalah *true negative* yang berarti banyaknya data negative diklasifikasikan dengan benar bernilai negative oleh model.

4. HASIL DAN PEMBAHASAN

4.1. Pengumpulan Data

Dataset yang digunakan diambil dari Github (https://github.com/PrudhviGNV/AI-models-for-bug-prediction/tree/main). Dataset yang diperoleh, dilakukan pemrosesan dengan mengunakan python. Dataset yang digunakan sebanyak 2109 dengan 22 atribut. Salah satu atribut data yang berlabel "*defects*" merupakan atribut target yang nilainya terdiri 2 kategori yakni "*false*" dan "*true*". Dari 2109 data, terdapat 1783 data yang masuk dalam kategori false dan sebanyak 326 data yang masuk dalam kategori true.

4.2. Membuat Model

Pembuatan model KNN untuk deteksi bug otomatis pada kode sumber open source dilakukan dengan beberapa tahapan yang harus dilakukan diantaranya.

4.2.1. Preprocessing Data

Pada tahap pengumpulan data diperoleh data sebanyak 2109 data dengan 1783 data berkategori false dan 326 data berkategori true. Hal ini menunjukkan telah terjadi ketidakseimbangan pada dataset sehingga perlu dilakukan oversampling untuk menyeimbangkan data sebelum data digunakan untuk melatih model KNN. Untuk menyeimbangkan dataset, peneliti menggunakan teknik SMOTE. SMOTE merupakan teknik untuk melakukan oversampling yang banyak digunakan untuk menangani ketidakseimbangan pada data yang digunakan untuk klasifikasi [28].

Setelah melakukan oversampling pada data, langkah selanjutkan melakukan preprocessing terlebih dahulu dilakukan dengan cara mngubah tipe data atribut defects dari Boolean ke int dan mengubah nilai false menjadi 0, true menjadi 1. Tahapan ini juga dilakukan pengecekan data yang kosong untuk

dihapus. Selama tahap ini tidak ditemukan data yang kosong sehingga data telah siap digunakan untuk menguji model.

4.2.2. Normalisasi Data (Standarisasi)

KNN menggunakan jarak Euclidean untuk mengukur kedekatan antara sampel data, sehingga data perlu dinormalisasikan menggunakan StandardScaler yang mengubah skala standar dengan rata-rata 0 dan standar deviasi 1 untuk memastikan semua fitur berpengaruh sama dalam perhitungan jarak.

4.2.3. Pembagian Data Latih dan Uji

Dataset yang telah dinormalisasikan, dibagiu menjadi 2 bagian. Sebanyak 80% digunakan sebagai data latih untuk melatih model dan sisanya sebanyak 20% sebagai data uji untuk menguji model setelah dilatih. Pembagian ini menggunakan *train_test_split()* dengan stratifikasi pada variabel target untuk menjaga distribusi kelas tetap seimbang.

4.2.4. Optimasi Nilai K Terbaik dengan Cross-Validation

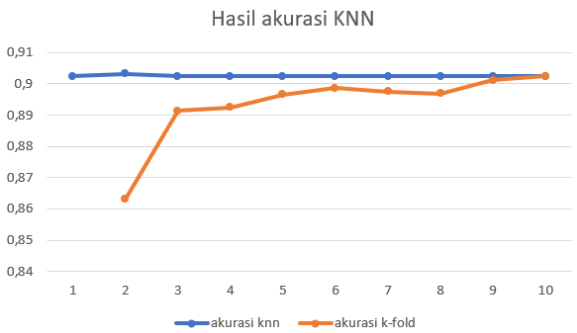
KNN sangat bergantung pada nilai k yang menentukan jumlah tetangga terdekat, sehingga penting untuk melakukan optimasi nilai k dengan beberapa langkah. Pertama, iterasi nilai k dari 1 hingga 20 (hanya bilangan ganjil) untuk menghindari kemungkinan hasil seri dalam klasifikasi. Kedua, evaluasi menggunakan cross-validation (k=10) untuk mendapatkan performa rata-rata model yang lebih stabil. Ketiga, memilih nilai k terbaik berdasarkan akurasi tertinggi dalam cross-validation.

4.2.5. Pelatihan Model KNN

Setelah nilai k terbaik diperoleh, model kemudian dilatih menggunakan *KNeighborsClassifier* dengan beberapa parameter. Pertama, *n_neighbors=best_k* (nilai k terbaik dari optimasi). Kedua, *weights='distance'* (tetangga terdekat memiliki bobot lebih besar). Ketiga, *metric-'euclidean'* (menggunakan jarak Euclidean sebagai ukuran kedekatan).

4.3. Evaluasi Model

Setelah model KNN dibangun, dilakukan pengujian dengan data uji untuk mengukur performa model berdasarkan metriks akurasi, yang mengykur persentase prediksi benar dari seluruh prediksi, presisi yang mengukur seberapa akurat model dalam mengidentifikasi bug yang benar dengan menghindari false positif, recall yang mengukur seberapa baik model dalam menemukan kasus bug yang benar dengan menghindari false negatives, f-measure yang merupakan kombinasi dari presisi dan recall. Hasil akurasi yang ditunjukkan pada masa evaluasi ini sangat bagus sebagaimana ditunjukkan pada Gambar 2.



Gambar 2. Hasil akurasi

Pengujian nilai k berguna untuk mengetahui performa KNN yang paling optimal berdasarkan nilai akurasi yang dihasilkan. Pada Gambar 2, dapat dilihat bahwa akurasi yang dihasilkan oleh peningkatan nilai k pada knn cenderung stabil, hanya di iterasi kedua terjadi peningkatan akurasi, namun tidak signifikan. Setelahnya akurasinya cenderung stabil tidak tidak mengalami peningkatan maupun penurunan. Sebaiknya, akurasi yang dihasilkan oleh nilai k-fold pada *cross-validation* cenderung terus meningkat disetiap iterasinya meskipun sempat terjadi penurunan akurasi dari iterasi ke-6 hingga ke-8, namun penurunan itu tidak signifikan.

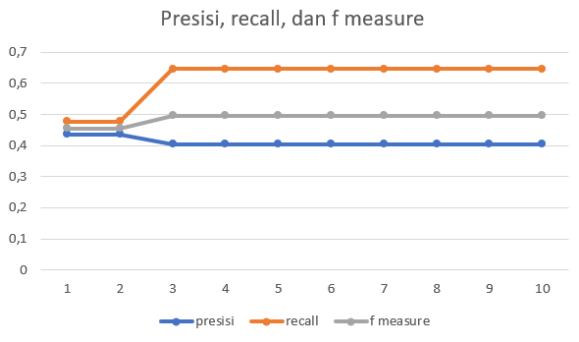
Nilai k dan k-fold memainkan peran yang sangat penting dalam mengukur performa model KNN. Semakin kecil nilai k pada KNN, model menjadi lebih sensitive terhadap noise dan cenderung terjadi overfitting pada model. Akurasi yang dihasilkan juga bisa lebih tinggi saat nilai k pada KNN kecil, namun tidak stabil dan model mungkin mengikuti pola data yang tidak relevan. Sebaliknya, semakin besar nilai k pada KNN model cenderung menjadi lebih smooth generalisasi model menjadi lebih baik. Namun, jika terlalu besar juga dapat menyebabkan underfitting karena model mempertimbangkan terlalu banyak tetangga.

Hasil pengujian yang dilakukan pada model KNN, akurasi yang dihasilkan cenderung stabil 0,9025 dengan sedikit peningkatan pada satu iterasi sebesar 0,9032. Hasil ini menunjukkan bahwa perubahan nilai k pada KNN tidak terlalu berdampak signifikan terhadap akurasi yang dihasilkan model. Hasil tersebut menunjukkan bahwa dataset yang digunakan sudah cukup stabil hingga tidak memberikan perubahan akurasi yang cukup besar dan dapat juga mengindikasikan bahwa model telah mencapai batas optimum atau dapat juga disebabkan pola yang terdapat pada data mudah untuk diklasifikasikan oleh model.

Hasil pengujian yang dilakukan dengan mengubah nilai k pada *cross-validation* menunjukkan bahwa akurasi yang dihasilkan nilai k kecil akurasi yang dihasilkan cenderung meningkat tajam dari 0,8633 ke 0,8924 yang menunjukkan bahwa model menjadi lebih stabil dan lebih baik jika nilai k yang ditentukan semakin besar. Dengan nilai k yang kecil jumlah data validasi menjadi terlalu sedikit disetiap

foldnya yang menyebabkan akurasi yang dihasilkan kurang stabil. Peningkatan akurasi dari nilai k-5 hingga k-8 tidak terlalu signifikan yang menunjukkan model lebih stabil dengan generalisasi yang lebih baik. Peningkatan yang tidak signifikan ini menunjukkan bahwa perubahan nilai k tidak selalu memberikan akurasi yang besar. k-fold 9 dan 10 memberikan hasil yang optimal pada k-fold 10 sebesar 0,9025. Model cukup stabil dari k-9 dan hanya memberikan sedikit perbedaan dengan k-10, yakni sebesar 0,0014. Berdasarkan hasil tersebut, jika dilakukan peningkatan nilai k-fold pada *cross-validation* tidak memberikan begitu banyak perubahan dalam performa model.

Semakin kecil nilai k menyebabkan model semakin bergantung terhadap data latih di setiap fold-nya. Sehingga semakin besar fold yang digunakan untuk melatih model, data yang digunakan juga semakin banyak dalam setiap iterasinya yang menyebabkan model menjadi stabil. Akurasi yang dihasilkan oleh model tidak dapat dijadikan acuan dalam menentukan performa dari model. Hal ini disebabkan akurasi yang dihasilkan dapat dipengaruhi oleh banyak faktor seperti kualitas dari data yang digunakan untuk melatih model, variasi atribut data yang digunakan, dan sebagainya. Sehingga presisi, *recall*, dan f *measure* juga digunakan untuk menentukan seberapa baik model yang dibangun. Hasil presisi, *recall*, dan f *measure* yang dihasilkan model KNN pada penelitian ini disajikan pada Gambar 3.



Gambar 3. Hasil presisi, *recall*, dan f *measure*

Dari Gambar 3, nilai presisi berkisar antara 0,4038 hingga 0,4366. *Recall* menunjukkan variasi yang lebih yakni dalam rentang 0,4769 hingga 0,6462, sedangkan f *measure* yang merupakan rata-rata dari presisi dan *recall* memberikan hasil antara 0,4559 hingga 0,497.

5. KESIMPULAN DAN SARAN

Penentuan nilai k dalam KNN dan jumlah nilai k-fold dalam *cross-validation* berpengaruh signifikan dalam performa model KNN dalam mendeteksi bug pada kode sumber. Hasil pengujian yang dilakukan menunjukkan semakin besar nilai k-fold *cross-validation* semakin stabil model dan generalisasinya yang semakin baik. Puncak optimal akurasi yang dihasilkan terjadi pada k-fold=10 dengan hasil akurasi sebesar 0,9025. Selain itu, tuning nilai k pada KNN juga berperan dalam menentukan keseimbangan bias dan variasi dari model, dimana penentuan nilai k yang optimal dapat meningkatkan akurasi dan stabilitas dari model. Teknik SMOTE yang digunakan untuk oversampling data guna menyeimbangkan data dapat meningkatkan kualitas dari model yang dibangun meskipun dalam hal ini perlu dilakukan optimasi lebih lanjut untuk memperbaiki hasil presisi, *recall*, dan f *measure*. Dengan demikian, kombinasi pemilihan parameter yang optimal, teknik resampling, dan teknik validasi model yang tepat berpengaruh signifikan dalam membangun model prediksi bug yang lebih akurat dan dapat diandalkan. Meskipun akurasi yang dihasilkan tinggi, namun presisi, recall, dan f-measure masih rendah yang menunjukkan model masih sering salah dalam mendeteksi bug. Saran untuk penelitian selanutnya yakni dapat dilakukan eksplorasi menggunakan teknik machine learning yang lain seperti SVM atau deep learning, optimasi feature angineering seperti analisis sintaksis kode atau menggunakan NLP untuk memahami deskripsi bug.

DAFTAR PUSTAKA

[1] A. Wicaksono, "Prediksi dan Deteksi Bug Pada Software Menggunakan Pendekatan Machine Learning: Machine Learning," *J. SIGN J. Ilm. Sist. Inf. dan Inform.*, vol. 2, no. 2, pp. 14–17, 2023.

[2] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, *A public unified bug dataset for java and its assessment regarding metrics and bug prediction*, vol. 28, no. 4. 2020. doi: 10.1007/s11219-020-09515-0.

[3] G. Rodríguez-Pérez, G. Robles, A. Serebrenik, A. Zaidman, D. M. Germán, and J. M. Gonzalez-Barahona, "How bugs are born: a model to identify how bugs are introduced in software components," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1294–1340, 2020, doi: 10.1007/s10664-019-09781-y.

[4] Z. Ni, B. Li, X. Sun, T. Chen, B. Tang, and X. Shi, "Analyzing bug fix for automatic bug cause classification," *J. Syst. Softw.*, vol. 163, p. 110538, 2020, doi: 10.1016/j.jss.2020.110538.

[5] M. Soltani, F. Hermans, and T. Bäck, "The significance of bug report elements," *Empir. Softw. Eng.*, vol. 25, no. 6, pp. 5255–5294, 2020, doi: 10.1007/s10664-020-09882-z.

[6] N. Tabassum, A. Namoun, T. Alyas, A. Tufail, M. Taqi, and K. H. Kim, "Classification of Bugs in Cloud Computing Applications Using Machine Learning Techniques," *Appl. Sci.*, vol. 13, no. 5, 2023, doi: 10.3390/app13052880.

[7] N. A. A. Khleel and K. Nehez, "Comprehensive Study on Machine Learning Techniques for Software Bug Prediction," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 8, pp. 726–735, 2021, doi: 10.14569/IJACSA.2021.0120884.

[8] H. M. Tran, S. T. Le, S. Van Nguyen, and P. T. Ho, "An Analysis of Software Bug Reports

Using Machine Learning Techniques," *SN Comput. Sci.*, vol. 1, no. 1, 2020, doi: 10.1007/s42979-019-0004-1.

[9] T. Sharma, "A Survey on Machine Learning Techniques for Source Code Analysis," vol. 0, no. 0, 2022.

[10] V. H. S. D. Maur´ıcio Aniche, Erick Maziero, Rafael Durelli, "The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring," vol. 5589, no. c, pp. 1–19, 2020, doi: 10.1109/TSE.2020.3021736.

[11] E. Y. Boateng, J. Otoo, and D. A. Abaye, "Basic Tenets of Classification Algorithms K-Nearest-Neighbor, Support Vector Machine, Random Forest and Neural Network: A Review," *J. Data Anal. Inf. Process.*, vol. 08, no. 04, pp. 341–357, 2020, doi: 10.4236/jdaip.2020.84020.

[12] D. G. Widder and C. Le Goues, "What Is a 'Bug'?," *Commun. ACM*, vol. 67, no. 11, pp. 32–34, 2024, doi: 10.1145/3662730.

[13] B. Mahesh, "Machine Learning Algorithms - A Review," *Int. J. Sci. Res.*, vol. 9, no. 1, pp. 381–386, 2020, doi: 10.21275/art20203995.

[14] A. Kish, "Machine Learning: A Review of Methods and Applications," *Researchgate.Net*, vol. 10, no. 07, 2023, [Online]. Available: https://www.researchgate.net/profile/Adam-Kish-2/publication/327645425_Machine_Learning_A_Review_of_Methods_and_Applications/links/5b9b679a299bf13e602d5bcd/Machine-Learning-A-Review-of-Methods-and-Applications.pdf

[15] D. Morgan and R. Jacobs, "MR50CH10_Morgan ARjats.cls Opportunities and Challenges for Machine Learning in Materials Science Keywords," *Annu. Rev. Mater. Res.*, pp. 1–33, 2020, [Online]. Available: https://doi.org/10.1146/annurev-matsci-070218-

[16] S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine Learning for Fluid Mechanics," *Annu. Rev. Fluid Mech.*, vol. 52, pp. 477–508, 2020, doi: 10.1146/annurev-fluid-010719-060214.

[17] A. Glielmo, B. E. Husic, A. Rodriguez, C. Clementi, F. Noé, and A. Laio, "Unsupervised Learning Methods for Molecular Simulation Data," *Chem. Rev.*, vol. 121, no. 16, pp. 9722–9758, 2021, doi: 10.1021/acs.chemrev.0c01195.

[18] S. Uddin, I. Haque, H. Lu, M. A. Moni, and E. Gide, "Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction," *Sci. Rep.*, vol. 12, no. 1, pp. 1–11, 2022, doi: 10.1038/s41598-022-10358-x.

[19] A. Rudiyan, A. E. Dzulkifli, and K. Munazar, "Klasifikasi Kebakaran Hutan Menggunakan Metode K-Nearest Neighbor : Studi Kasus Hutan Provinsi Kalimantan Barat," *JTIM J. Teknol. Inf. dan Multimed.*, vol. 3, no. 4, pp. 195–202, 2022, doi: 10.35746/jtim.v3i4.177.

[20] S. Zhang and J. Li, "KNN Classification With One-Step Computation," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 3, pp. 2711–2723, 2023, doi: 10.1109/TKDE.2021.3119140.

[21] A. R. Lubis, M. Lubis, and Al-Khowarizmi, "Optimization of distance formula in k-nearest neighbor method," *Bull. Electr. Eng. Informatics*, vol. 9, no. 1, pp. 326–338, 2020, doi: 10.11591/eei.v9i1.1464.

[22] Andreansyah and Grace Gata, "Analisis Sentimen untuk Program Vaksin Booster Sebagai Syarat Mudik 2022 Menggunakan Algoritma KNN," *KRESNA J. Ris. dan Pengabdi. Masy.*, vol. 2, no. 2, pp. 193–201, 2022, doi: 10.36080/jk.v2i2.48.

[23] Nti Isaac Kofi, O. Nyarko-Boateng, and J. Aning, "Performance of Machine Learning Algorithms with Different K Values in K-fold CrossValidation," *Int. J. Inf. Technol. Comput. Sci.*, vol. 13, no. 6, pp. 61–71, 2021, doi: 10.5815/ijitcs.2021.06.05.

[24] B. G. Marcot and A. M. Hanea, "What is an optimal value of k in k-fold cross-validation in discrete Bayesian network analysis?," *Comput. Stat.*, vol. 36, no. 3, pp. 2009–2031, 2021, doi: 10.1007/s00180-020-00999-9.

[25] P. Christen, D. J. Hand, and N. Kirielle, "A Review of the F-Measure: Its History, Properties, Criticism, and Alternatives," *ACM Comput. Surv.*, vol. 56, no. 3, 2023, doi: 10.1145/3606367.

[26] E. Dritsas and M. Trigka, "Supervised Machine Learning Models for Liver Disease Risk Prediction," *Computers*, vol. 12, no. 1, p. 19, 2023, doi: 10.3390/computers12010019.

[27] J. Cook and V. Ramadas, "When to consult precision-recall curves," *Stata J.*, vol. 20, no. 1, pp. 131–148, 2020, doi: 10.1177/1536867X20909693.

[28] D. Elreedy, A. F. Atiya, and F. Kamalov, "A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learning," *Mach. Learn.*, vol. 113, no. 7, pp. 4903–4923, 2024, doi: 10.1007/s10994-022-06296-4.