

PENINGKATAN KINERJA PREDIKSI CACAT SOFTWARE DENGAN HYPERPARAMETER TUNING PADA ALGORITMA KLASIFIKASI DEEP FOREST

Emma Andini¹, Mohammad Reza Faisal^{*2}, Rudy Herteno³,

Radityo Adi Nugroho⁴, Friska Abadi⁵, Muliadi⁶

^{1 2 3 4 5 6} Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam,
Universitas Lambung Mangkurat
reza.faisal@ulm.ac.id

ABSTRAK

Prediksi cacat *software* adalah salah satu studi pada bidang Rekayasa Perangkat Lunak yang telah diteliti oleh banyak peneliti. Tujuan dari studi ini adalah untuk mencari tahu algoritma yang dapat memberikan kinerja prediksi cacat *software* yang lebih baik. Salah satu penelitian yang telah dilakukan adalah melakukan prediksi cacat *software* dengan menggunakan algoritma berbasis pohon seperti *Decision Tree*, *Random Forest* dan *Deep Forest*. *Deep Forest* adalah algoritma klasifikasi berbasis pohon yang baru yang merupakan perbaikan dari algoritma *Random Forest*. Namun implementasi *Deep Forest* dalam penelitian terdahulu masih belum memberikan kinerja yang maksimal. Hasil pada penelitian terdahulu menunjukkan bahwa kinerja algoritma *Deep Forest* masih ada yang lebih rendah dibandingkan algoritma berbasis pohon yang lain. Pada penelitian ini berfokus pada peningkatan kinerja algoritma berbasis pohon dengan melakukan normalisasi pada dataset dan *hyperparameter tuning* pada algoritma klasifikasi dengan menggunakan *Grid Search*. Dataset yang digunakan adalah 3 dataset dari ReLink yaitu Apache, Safe, dan Zxing. Setiap model prediksi divalidasi dengan *stratified 10-Fold Cross Validation* dan kinerja dievaluasi menggunakan AUC. Rata-rata nilai AUC yang dihasilkan *Decision Tree* sebesar 0.69, sedangkan rata-rata nilai AUC yang dihasilkan *Random Forest* sebesar 0.76 dan rata-rata nilai AUC yang dihasilkan *Deep Forest* sebesar 0.79. Hasil eksperimen ini menunjukkan bahwa pendekatan yang diusulkan dapat bekerja lebih baik dalam memprediksi cacat *software* dibandingkan algoritma berbasis pohon lainnya.

Keyword : *decision tree, random forest, deep forest, prediksi cacat software, klasifikasi*

1. PENDAHULUAN

Pada era komputerisasi sekarang ini di mana penggunaan *software* telah digunakan pada banyak aspek kehidupan [1]. *Software* bertanggung jawab atas semua sistem yang berjalan di komputer. Para pengembang *software* dalam memelihara sistem perlu melakukan pencegahan untuk menghindari kesalahan yang tidak terduga saat mengembangkan sebuah *software*. Kompleksitas pada sistem *software* yang terus meningkat sangat rentan terhadap cacat. Cacat *software* terjadi karena ketidakmampuan *software* untuk menjalankan proses yang seharusnya di mana sistem tidak berjalan sebagaimana fungsinya sehingga menyebabkan kualitas *software* menurun [2]. Turunnya kualitas *software* menjadi suatu kerugian tersendiri. Sangat penting untuk melakukan prediksi cacat *software* sebagai upaya menangani masalah *software* sekaligus dapat meningkatkan kualitas *software* [3]. Prediksi cacat *software* dilakukan dengan cara menganalisis metrik *software* kemudian dilanjutkan dengan membangun model untuk memprediksi cacat. Pendeteksian cacat dalam modul *software* dilakukan berupa klasifikasi. Ini melibatkan modul *software* dikategorikan sebagai cacat atau tidak cacat menggunakan algoritma klasifikasi seperti yang umumnya telah dilakukan banyak penelitian [4] [5] [6].

Penelitian [7] menunjukkan bahwa telah dilakukan perbandingan menggunakan berbagai algoritma klasifikasi untuk memprediksi cacat *software*. Terdapat penelitian yang membandingkan *Naïve Bayes*, *Multi-Layer Perceptron*, *Radial Basis Function*, *Support Vector Machine*, *K Nearest Neighbor*, *Decision Tree* dan *Random Forest* menggunakan dataset NASA. Hasilnya adalah kinerja klasifikasi dengan nilai AUC tertinggi dihasilkan *Random Forest* sebesar 0.94. Pada penelitian [8] dilakukan perbandingan beberapa algoritma klasifikasi yaitu *Deep Forest*, *Deep Belief Networks*, *Random Forest*, *Naive Bayes*, *Logistic Regression* dan *Support Vector Machine* untuk memprediksi cacat *software* pada dataset NASA, PROMISE, AEEEM, ReLink. Hasil penelitian ini menunjukkan bahwa *Deep Forest* memiliki kinerja terbaik pada dataset NASA, PROMISE, AEEEM dengan nilai berturut-turut adalah sebesar 0.92, 0.89, 0.86. Sedangkan di beberapa dataset ReLink menunjukkan *Random Forest* lebih baik dibanding yang lain dengan AUC tertinggi adalah 0.75. Penelitian [9] membandingkan teknik deteksi *manifold nonlinier* dan teknik seleksi fitur pada algoritma klasifikasi *Bayesian Belief Network*, *Decision Tree-J48*, *Naive Bayes* dan IBK menggunakan dataset ReLink. Hasilnya menunjukkan teknik terbaik dimiliki oleh LLE (*Locally Linier Embedding*) dan LPP (*Linearity*

Preserving Projection) dengan nilai AUC sebesar 0.80 pada dataset Safe dan algoritma yang paling efektif digunakan adalah *Decision Tree-J48* dan IBK.

Pada penelitian ini, kami melakukan prediksi cacat *software* yang membandingkan *Decision Tree*, *Random Forest* dan *Deep Forest* menggunakan dataset ReLink. Ketiga algoritma tersebut sama-sama berbasis pohon. Selain itu, mereka memiliki keterkaitan satu sama lain di mana pada *Random Forest* mengandung banyak *Decision Tree* dan *Deep Forest* mengandung banyak *Random Forest*. Penelitian sebelumnya membuktikan bahwa *Decision Tree* mampu membuat data yang kompleks menjadi sederhana dalam bentuk pohon dengan memberi interpretasi yang baik untuk seluruh data [10]. Berbeda dengan *Decision Tree* yang hanya membentuk satu pohon, penelitian sebelumnya membuktikan bahwa *Random Forest* membentuk banyak pohon melalui subset dari data yang diberikan dan memilih subset secara acak sehingga mampu menganalisa data dengan jumlah yang besar [11]. Adapun penelitian sebelumnya juga membuktikan bahwa *Deep Forest* mampu berkinerja dengan baik tanpa mengatur parameter terlalu banyak [12]. Namun algoritma ini masih belum bekerja baik pada dataset ReLink seperti yang telah dikerjakan pada penelitian sebelumnya.

Berdasarkan masalah yang telah dipaparkan di atas maka dilakukan riset dengan algoritma berbasis pohon untuk prediksi cacat *software* pada dataset Relink dengan melakukan *hyperparameter tuning*. Algoritma berbasis pohon pada eksperimen ini adalah *Decision Tree*, *Random Forest* dan *Deep Forest*. Pada riset ini digunakan rentang parameter yang lebih lebar dari penelitian terdahulu dengan tujuan saat melakukan *hyperparameter tuning* akan didapat peningkatan kinerja prediksi.

2. TINJAUAN PUSTAKA

2.1. Z-score Normalization

Normalisasi merupakan suatu teknik untuk memetakan data ke dalam skala yang beragam. Berbagai jenis teknik normalisasi salah satunya adalah *z-score*. Normalisasi *z-score* juga disebut sebagai normalisasi data di antara rata-rata di mana data dinormalisasi berdasarkan rata-rata dan standar deviasi artinya ukuran nilai ditentukan oleh seberapa jauh titik data dari rata-rata dalam satuan standar deviasinya [13].

2.2. Grid Search

Grid search atau pencarian grid merupakan metode *hyperparameter tuning* yang paling sederhana. Keunggulan metode ini adalah parameter kandidat dihasilkan secara berurutan atau sistematis sehingga setiap kumpulan data akan dievaluasi menggunakan parameter kandidat yang sama. Dengan demikian, strategi yang dilakukan *grid search* dapat menelusuri satu persatu dan mengecek

pada kombinasi mana yang menghasilkan nilai terbaik. *Grid search* memberikan jaminan kepastian karena di setiap kombinasinya mempengaruhi setiap proses yang berjalan [14].

2.3. Decision Tree

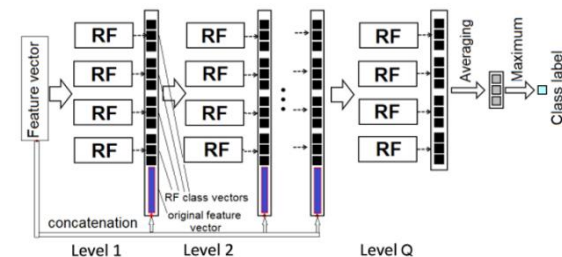
Decision Tree merupakan algoritma sederhana yang terdiri dari kumpulan simpul keputusan dalam struktur pohon. Pohon mencapai klasifikasi melalui pemisahan cabang-cabang pohon. Pemisahan cabang ini berlanjut hingga level terakhir tercapai. Ketika semua *tuple* milik satu kelas maka algoritma berhenti mempartisi lebih lanjut. Akhirnya, simpul daun memprediksi hasil [15].

2.4. Random Forest

Random Forest merupakan sebuah *ensemble* (kumpulan) menggunakan *Decision Tree* sebagai *base classifier* yang dibangun dan dikombinasikan dengan cara *majority vote*. Sesuai dengan namanya, algoritma ini menciptakan sebuah hutan (*forest*) dengan sejumlah pohon (*tree*) [16].

2.5. Deep Forest

Deep Forest disebut sebagai alternatif *Deep Neural Network*. Algoritma ini memiliki struktur lapis demi lapis berupa *Random Forest*, lapisan tersebut bernama *cascade forest* [17].



Gambar 1. Arsitektur *cascade forest* [18].

Pada Gambar 1 terlihat bahwa algoritma dilapis dan dilapis lagi di setiap tingkat lapisan. Lapisan pertama menerima input dari atribut atau fitur pada dataset asli yang akan diproses bersama *Random Forest* pada lapisan selanjutnya. Lapisan akan berhenti apabila proses yang dihasilkan *Random Forest* sudah tidak ada peningkatan atau hasil pada lapisan yang diberikan mengalami penurunan. Dari setiap tingkat lapisan yang ada, algoritma akan merata-ratakan hasil dari lapisan demi lapisan sampai di lapisan yang terakhir. Meskipun membutuhkan waktu melebihi *Random Forest* dalam prosesnya, *Deep Forest* dapat bekerja lebih baik apabila dihadapkan pada data dengan skala yang kecil [18].

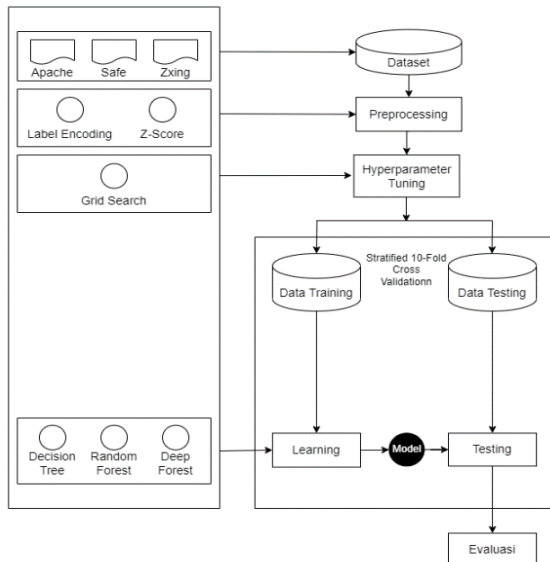
2.6. Cross Validation

Cross Validation adalah salah satu teknik pembagian data yang paling banyak digunakan di mana target adalah prediksi dan perlu untuk memperkirakan keakuratan kinerja model prediksi. *Cross Validation* membagi data orisinal menjadi

data *training* dan *testing* secara acak. Data *training* digunakan untuk melatih model klasifikasi dan data *testing* digunakan untuk menguji model yang dilatih [19].

3. METODE PENELITIAN

Alur penelitian ini dapat dilihat pada Gambar 2 berikut ini.



Gambar 2. Diagram alur penelitian

Pada Gambar 2 merupakan diagram alur pada penelitian ini. Pertama dataset ReLink akan dikumpulkan dan dilakukan *preprocessing* menggunakan *label encoding* dan normalisasi *z-score*. Setelah dataset di *preprocessing*, tahapan selanjutnya melakukan klasifikasi di mana pada penelitian ini terdapat tiga skenario yaitu klasifikasi menggunakan *Decision Tree*, klasifikasi menggunakan *Random Forest* dan klasifikasi menggunakan *Deep Forest*. Teknik validasi yang digunakan adalah *Stratified 10-Fold Cross Validation* dan di evaluasi menggunakan AUC.

3.1. Pengumpulan Dataset

Dataset yang digunakan dalam penelitian ini adalah dataset metrik *software* bernama ReLink, terdiri dari data Apache, Safe dan Zxing [20]. Dataset ini dapat diunduh pada link berikut <https://github.com/bharlow058/AEEEM-and-other-SDP-datasets/tree/master/dataset/Relink>.

Tabel 1 menunjukkan jumlah data yang berbeda-beda di setiap dataset ReLink yaitu Apache sebanyak 194 data, Safe sebanyak 56 data dan Zxing sebanyak 399 data. Kemudian Tabel 2 menjelaskan metrik *software* dataset ReLink yang dikelompokkan menjadi 2 kategori (kelompok) metrik *software* yaitu *Complexity Metric* (CPM) dan *Count Metric* (CTM). Dataset ReLink memiliki jumlah metrik *software* yang sama dan dapat dilihat pada Tabel 3.

Tabel 1. Jumlah data pada dataset ReLink

Dataset	Jumlah Modul	Jumlah Kelas Defective	Jumlah Kelas Non-defective
Apache	194	98	96
Safe	56	22	34
Zxing	399	118	281

Tabel 2. Metrik *software* dataset ReLink.

Kategori Metriks	Atribut
Complexity Metric (CPM)	<i>AvgCyclomatic</i>
	<i>AvgCyclomaticModified</i>
	<i>AvgCyclomaticStrict</i>
	<i>AvgEssential</i>
	<i>MaxCyclomatic</i>
	<i>MaxCyclomaticModified</i>
	<i>MaxCyclomaticStrict</i>
	<i>RatioCommentToCode</i>
	<i>SumCyclomatic</i>
	<i>SumCyclomaticModified</i>
	<i>SumCyclomaticStrict</i>
Count Metric (CTM)	<i>SumEssential</i>
	<i>AvgLine</i>
	<i>AvgLineBlank</i>
	<i>AvgLineCode</i>
	<i>AvgLineComment</i>
	<i>CountLine</i>
	<i>CountLineBlank</i>
	<i>CountLineCode</i>
	<i>CountLineCodeDecl</i>
	<i>CountLineCodeExe</i>
	<i>CountLineComment</i>
	<i>CountSemicolon</i>
	<i>CountStmnt</i>
	<i>CountStmntDecl</i>
	<i>CountStmntExe</i>

Tabel 3. Jumlah metrik *software* dataset ReLink

Dataset	Jumlah Metrik
Apache	26
Safe	26
Zxing	26

Pada riset prediksi cacat *software* yang sering dihadapi banyak penelitian apabila ingin menggunakan dataset metrik *software* yaitu terdapat *class imbalance*. Namun *class imbalance* pada dataset ReLink memiliki perbandingan yang tidak terlalu jauh, artinya lebih baik dari dataset lain di mana jumlah data dari modul *software* yang cacat dibandingkan dengan jumlah data dari modul *software* yang bebas cacat tidak terlalu jauh. Pada prediksi cacat *software* disini data dari modul *software* yang cacat berperan paling penting. Perhatian terhadap *class imbalance* menjadi salah satu faktor penting dalam melakukan prediksi cacat *software* karena dapat mempengaruhi kinerja dari algoritma klasifikasi [21].

3.2. Preprocessing

Seperti yang ditunjukkan pada Gambar 2, sebelum dilakukan pembagian data, data akan melalui proses *preprocessing* terlebih dahulu. *Preprocessing* dilakukan untuk menyesuaikan keperluan data terhadap algoritma klasifikasi yang dapat berguna dalam meningkatkan kinerja model klasifikasi. *Preprocessing* data yang dilalui pada penelitian ini ada dua, yaitu *label encoding* dan normalisasi.

a. Label encoding

Label Encoding merupakan proses untuk mengganti nilai label ke dalam bentuk numerik. Setiap model yang digunakan hanya bisa memproses label kelas berupa angka biner (0 dan 1), sehingga perlu dilakukan penggantian nilai label pada dataset ReLink menjadi dalam bentuk numerik. Dataset ReLink memiliki dua kelas yaitu *buggy* dan *clean*. Label kelas tersebut diubah menjadi 0 dan 1. Pada tabel 4 menunjukkan hasil *label encoding* yang telah dilakukan.

Tabel 4. Hasil *label encoding*

Label awal	Hasil <i>label encoding</i>	Arti label
<i>buggy</i>	1	<i>Defective</i>
<i>clean</i>	0	<i>Non-defective</i>

b. Normalisasi

Normalisasi merupakan teknik pengaturan skala di mana nilai-nilai diskalakan ulang. Normalisasi yang digunakan pada penelitian ini adalah normalisasi *z-score* sehingga nilai-nilai berkisar di antara rata-rata. Jika nilainya di atas rata-rata maka *z-score* bernilai positif, dan jika nilainya di bawah rata-rata maka *z-score* bernilai negatif. Pada dataset ReLink memiliki jangkauan yang berbeda-beda tiap fitur, sehingga perlu dilakukan proses normalisasi. Bentuk data yang sudah melalui proses normalisasi dapat dilihat pada tabel 5 menunjukkan hasil normalisasi *z-score*.

Tabel 5. Hasil normalisasi *z-score*.

Atribut 1	Atribut n	Label
1.10531341	0.47128798	1
-0.96481500	-0.66040812	0
0.41527061	1.12795115	1

3.3. Hyperparameter Tuning

Hyperparameter tuning adalah proses mencari nilai parameter optimal di mana mula-mula harus menentukan daftar parameter dan rentang pencarian untuk setiap parameter [22]. *Hyperparameter tuning* ini berfungsi untuk membantu model menemukan nilai parameter yang tepat untuk tiap dataset agar mendapatkan hasil kinerja yang maksimal. Pada bagian ini dilakukan *hyperparameter tuning* menggunakan *grid search*. Tabel 6 menunjukkan

detail rentang parameter-parameter model yang akan dicari nilai optimalnya.

Tabel 6. Detail parameter model

Model	Parameter	Deskripsi	Rentang	
DT	min_samples_leaf	Jumlah minimum sampel yang diperlukan untuk berada pada simpul daun	1	10
	min_samples_split	Jumlah minimum sampel yang diperlukan untuk membagi simpul internal	2	10
	max_depth	Kedalaman maksimum pohon	1	10
	min_impurity_decrease	Ketidakmurnian	0	3
RF	n_estimators	Jumlah pohon individu	100	500
	max_depth	Kedalaman maksimum dari pohon individu	1	5
DF	n_estimators	Jumlah hutan di setiap lapisan	2	11
	n_trees	Jumlah pohon di setiap hutan	100	2000

Tahapan awal dari *grid search* yaitu menentukan rentang tiap-tiap parameter. Rentang pada *grid search* sebagai jumlah kombinasi konfigurasi parameter yang akan dijadikan sebagai kandidat dan divalidasi dengan *cross validation*. Misalnya, jika rentang parameter pertama diatur sebanyak 5 kemudian range parameter kedua diatur sebanyak 10 maka *grid search* akan menghasilkan 5x10 konfigurasi kombinasi parameter secara sistematis dan di simpan sebagai kandidat. Artinya, jumlah kandidat oleh *grid search* dihasilkan dari seberapa banyak rentang parameter yang divalidasi bersamaan dengan *cross validation*. Semakin banyak rentang nilai maka semakin banyak kandidat yang dihasilkan semakin lama waktu proses dari *grid search*.

Seperti yang terlihat pada Tabel 6, pada penelitian ini rentang nilai parameter model *Decision Tree* yaitu min_samples_leaf adalah 1 sampai 10 meliputi 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 kemudian min_samples_split adalah 2 sampai 10 meliputi 2, 4, 6, 8, 10 kemudian max_depth adalah 1 sampai 10 meliputi 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 kemudian min_impurity_decrease adalah 0 sampai 3 meliputi 0, 1, 2, 3. Sedangkan penentuan rentang parameter model *Random Forest* yaitu max_depth adalah 1 sampai 5 meliputi 1, 2, 3, 4, 5 kemudian

$n_estimators$ adalah 100 sampai 500 meliputi 100, 200, 300, 400, 500. Dan penentuan rentang parameter model *Deep Forest* yaitu n_trees adalah 100 sampai 2000 meliputi 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000 kemudian $n_estimators$ adalah 2 sampai 11 meliputi 2, 3, 5, 7, 9, 11.

3.4. Klasifikasi

Setiap dataset ReLink akan diproses oleh setiap algoritma berbasis pohon yaitu *Decision Tree*, *Random Forest* dan *Deep Forest* sesuai dengan hasil dari *hyperparameter tuning* oleh *grid search*. Model dilatih menggunakan teknik *stratified 10-fold cross validation*, teknik ini memastikan semua *instance* teruji [23]. Pembagian data dilakukan secara stratifikasi untuk mempertahankan proporsi pembagian kelas, sehingga pada data *training* dan data *testing* memiliki proporsi perbandingan kelas yang sama. Data dibagi menjadi 10 partisi, 9 partisi untuk data *training* dan 1 partisi untuk data *testing*. Setiap partisi secara bergantian digunakan sebagai data *training* dan data *testing* pada setiap percobaan. Pada percobaan pertama, partisi pertama digunakan sebagai data *testing* dan partisi lainnya digunakan sebagai data *training*, kemudian model dilatih menggunakan partisi data *training* dan divalidasi menggunakan partisi data *testing* untuk mendapatkan nilai akurasi dari percobaan pertama. Begitu seterusnya sampai semua partisi telah digunakan sebagai data *training* dan data *testing* secara bergantian atau hingga terjadi 10 kali percobaan.

3.5. Evaluasi

Evaluasi kinerja klasifikasi dari model *Decision Tree*, *Random Forest* dan *Deep Forest* pada masing-masing dataset ReLink menggunakan nilai AUC (Area under the ROC Curve). AUC dipilih sebagai metode evaluasi karena AUC cocok untuk mengevaluasi nilai kinerja prediksi yang menggunakan dataset dengan permasalahan kelas tidak seimbang (*class imbalance*) [24].

Tabel 7 menunjukkan pedoman umum yang digunakan untuk klasifikasi nilai AUC [25].

Tabel 7. Kategori nilai AUC

Kategori	Rentang	
Sangat Baik	0.90	1.00
Baik	0.80	0.90
Cukup Baik	0.70	0.80
Kurang Baik	0.60	0.70
Buruk	0.50	0.60

4. HASIL DAN PEMBAHASAN

4.1. Hasil

a. Kinerja Prediksi Menggunakan Z-score

Tabel 8 menunjukkan hasil evaluasi dalam nilai AUC untuk prediksi cacat *software* sebelum dan sesudah menggunakan *z-score*.

Tabel 8. Perbandingan menggunakan *z-score*

Model	Dataset	Sebelum Z-score	Sesudah Z-score
DT	Apache	0.66	0.71
	Safe	0.69	0.67
	Zxing	0.59	0.59
RF	Apache	0.79	0.79
	Safe	0.86	0.87
	Zxing	0.70	0.70
DF	Apache	0.80	0.80
	Safe	0.82	0.82
	Zxing	0.71	0.71

Seperti yang terlihat pada tabel 8 dataset menggunakan normalisasi *z-score* lebih unggul pada 2 dataset, yaitu pada dataset Apache dengan model klasifikasi *Decision Tree* dan pada dataset Safe dengan model klasifikasi *Random Forest*. Masing-masing dataset tersebut mengalami peningkatan AUC secara berturut, yaitu Apache sebesar 0.05 dan Safe sebesar 0.01. Sedangkan pada dataset dengan model klasifikasi lainnya kebanyakan tidak menunjukkan perubahan.

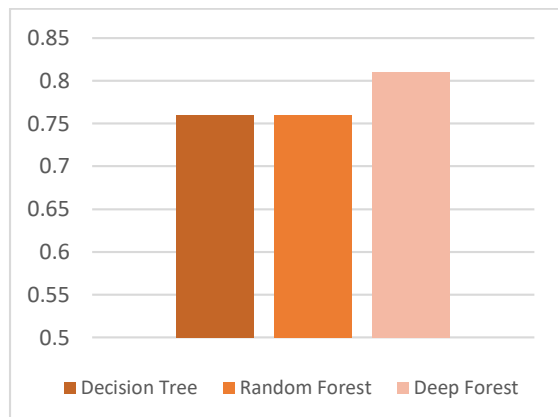
b. Kinerja Prediksi Pada Dataset Apache

Tabel 9 menunjukkan hasil evaluasi dalam nilai AUC untuk prediksi cacat *software* dari dataset Apache. Dari setiap model yang digunakan untuk melakukan prediksi pada dataset ini telah memiliki parameter optimal yang memberikan kinerja prediksi terbaik sebagai berikut: *Decision Tree* dengan nilai parameter *max_depth*: 3, *min_impurity_decrease*: 0, *min_samples_leaf*: 1, *min_samples_split*: 2. *Random Forest* dengan nilai parameter *max_depth*: 3, $n_estimators$: 200. *Deep Forest* dengan nilai parameter $n_estimators$: 5, n_trees : 400.

Tabel 9. AUC Apache

Model	Nilai AUC
<i>Decision Tree</i> (DT)	0.76
<i>Random Forest</i> (RF)	0.76
<i>Deep Forest</i> (DF)	0.81

Grafik perbandingan kinerja dari setiap algoritma untuk memprediksi dataset Apache dapat dilihat pada Gambar 3.



Gambar 3. Grafik AUC dataset Apache

Pada gambar 3 merupakan hasil evaluasi kinerja dari *Decision Tree*, *Random Forest* dan *Deep Forest* pada dataset Apache. Grafik di atas menunjukkan kinerja yang dihasilkan oleh *Decision Tree* mendapatkan nilai AUC sebesar 0.76, begitupun dengan *Random Forest* menghasilkan kinerja dengan nilai AUC sebesar 0.76. Sedangkan kinerja yang dihasilkan oleh *Deep Forest* mendapatkan nilai AUC sebesar 0.81.

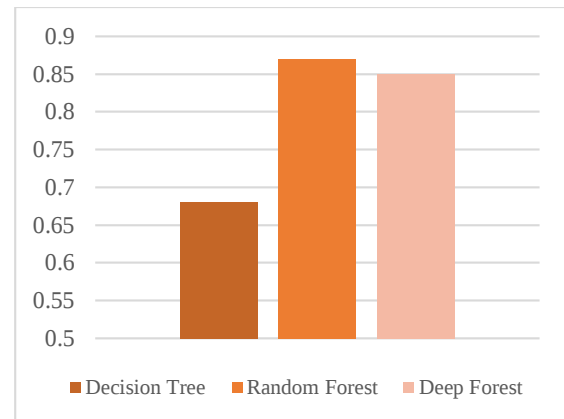
c. Kinerja Prediksi Pada Dataset Safe

Pada Tabel 10 menunjukkan hasil evaluasi prediksi cacat *software* dari dataset Safe dalam nilai AUC. Dari setiap model yang digunakan untuk melakukan prediksi pada dataset ini telah memiliki parameter optimal sebagai berikut yaitu *Decision Tree* meliputi *max_depth*: 6, *min_impurity_decrease*: 0, *min_samples_leaf*: 1, *min_samples_split*: 2. Nilai parameter yang digunakan pada *Random Forest* meliputi *max_depth*: 4, *n_estimators*: 100. Sedangkan nilai parameter optimal yang digunakan pada *Deep Forest* meliputi *n_estimators*: 9, *n_trees*: 800.

Tabel 10. AUC Safe

Model	Nilai AUC
<i>Decision Tree</i> (DT)	0.68
<i>Random Forest</i> (RF)	0.87
<i>Deep Forest</i> (DF)	0.85

Grafik perbandingan kinerja dari setiap algoritma untuk memprediksi dataset Safe dapat dilihat pada gambar 4.



Gambar 4. Grafik AUC dataset Safe

Pada gambar 4 merupakan hasil evaluasi kinerja dari *Decision Tree*, *Random Forest* dan *Deep Forest* pada dataset Safe. Grafik di atas menunjukkan kinerja yang dihasilkan oleh *Decision Tree* mendapatkan nilai AUC sebesar 0.68. Untuk kinerja *Random Forest* menghasilkan nilai AUC sebesar 0.87. Sedangkan kinerja yang dihasilkan oleh *Deep Forest* mendapatkan nilai AUC sebesar 0.85.

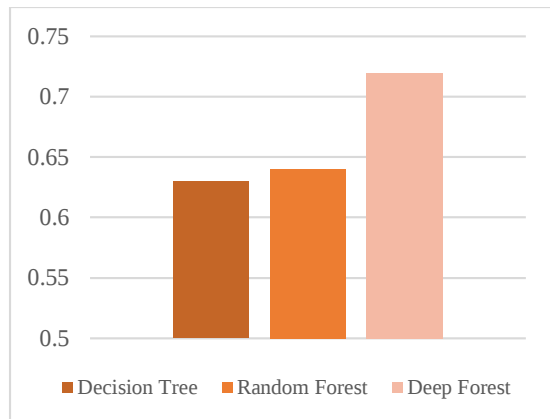
d. Kinerja Prediksi Pada Dataset Zxing

Hasil evaluasi AUC prediksi cacat *software* dataset Zxing dapat dilihat pada Tabel 11. Dari setiap model yang digunakan untuk melakukan prediksi pada dataset ini telah menggunakan parameter optimal masing-masing yaitu *Decision Tree* meliputi *max_depth*: 3, *min_impurity_decrease*: 0, *min_samples_leaf*: 3, *min_samples_split*: 2. *Random Forest* meliputi *max_depth*: 1, *n_estimators*: 100. *Deep Forest* meliputi *n_estimators*: 5, *n_trees*: 100.

Tabel 11. AUC Zxing

Model	Nilai AUC
<i>Decision Tree</i> (DT)	0.63
<i>Random Forest</i> (RF)	0.64
<i>Deep Forest</i> (DF)	0.72

Grafik perbandingan kinerja dari setiap algoritma untuk memprediksi dataset Zxing dapat dilihat pada Gambar 5.



Gambar 5. Grafik AUC dataset Zxing

Pada gambar 5 merupakan hasil evaluasi kinerja dari *Decision Tree*, *Random Forest* dan *Deep Forest* pada dataset Zxing. Grafik di atas menunjukkan kinerja yang dihasilkan oleh *Decision Tree* mendapatkan nilai AUC sebesar 0.63, hampir sama dengan *Random Forest* namun kinerja yang dihasilkan oleh *Random Forest* sedikit lebih tinggi yaitu mendapatkan nilai AUC sebesar 0.64. Sedangkan kinerja yang dihasilkan oleh *Deep Forest* mendapatkan nilai AUC sebesar 0.72.

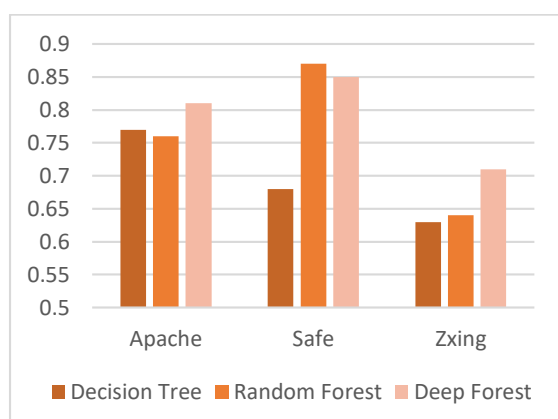
e. Kinerja Prediksi Seluruh Dataset

Tabel 12 menunjukkan hasil evaluasi AUC seluruh dataset dari setiap model prediksi cacat *software*.

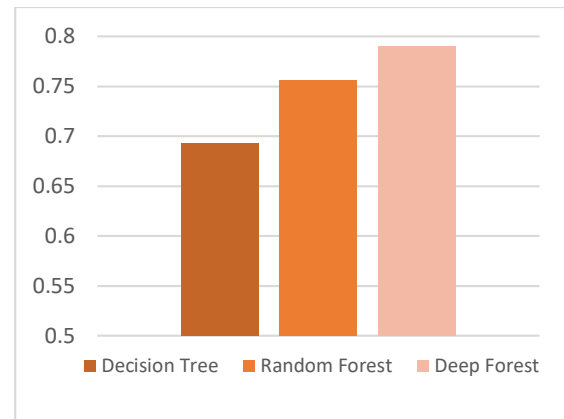
Tabel 12. AUC seluruh dataset

Dataset	DT	RF	DF
Apache	0.77	0.76	0.81
Safe	0.68	0.87	0.85
Zxing	0.63	0.64	0.72
Rata-Rata	0.69	0.76	0.79

Secara grafik, hasil keseluruhan ditunjukkan pada gambar 6.



Gambar 6. Grafik rata-rata nilai AUC



Gambar 7. Grafik rata-rata nilai AUC

Pada gambar 7 merupakan rata-rata nilai AUC dari 3 model klasifikasi pada dataset ReLink. Dapat dilihat pada grafik di atas bahwa kinerja model klasifikasi yang dihasilkan menggunakan *Deep Forest* mengungguli *Decision Tree* dan *Random Forest*. Pada penelitian ini *Deep Forest* menunjukkan rata-rata nilai AUC seluruh dataset sebesar 0.79. Kinerjanya sangat baik pada data Safe, dengan hasil mencapai 0.85. *Random Forest* juga tampil baik dengan mendapatkan rata-rata nilai AUC seluruh dataset sebesar 0.76 dan kinerja terbaik terlihat pada data Safe sebesar 0.87. Sedangkan *Decision Tree* memiliki kinerja lebih rendah dari model yang lain, rata-rata nilai AUC seluruh dataset yang dihasilkan hanya mencapai 0.69, kinerja terbaiknya hanya mendapatkan nilai AUC sebesar 0.76 pada data Apache. Dapat diketahui disini bahwa kinerja keseluruhan dari rata-rata nilai AUC yang dihasilkan *Deep Forest* adalah yang terbaik.

f. Perbandingan Dengan Penelitian Sebelumnya

Perbandingan kinerja prediksi hasil penelitian dengan menggunakan metode yang kami usulkan pada penelitian ini dengan hasil penelitian sebelumnya dapat dilihat pada Tabel 13.

Tabel 13. Perbandingan prediksi dengan penelitian sebelumnya.

Dataset	Metode penelitian sebelumnya			Metode yang diusulkan		
	DT[9]	RF[8]	DF[8]	DT	RF	DF
Apache	0.70	0.76	0.75	0.76	0.76	0.81
Safe	0.63	0.73	0.73	0.68	0.87	0.85
Zxing	0.64	0.67	0.70	0.63	0.64	0.71
Rata-Rata	0.66	0.72	0.73	0.69	0.76	0.79

Pada penelitian sebelumnya, parameter *Decision Tree* diatur dengan nilai default atau tanpa melakukan hyperparameter tuning menghasilkan nilai AUC rata-rata 0.66 [9]. Adapun penelitian lain sebelumnya, parameter *Random Forest* yang diatur hanya jumlah pohon saja dan rentang yang dicari dari 100 sampai 1000 begitu pun dengan *Deep Forest*. Namun pada *Deep Forest* digunakan

parameter tambahan yang lain untuk diatur yaitu jumlah hutan dengan rentang adalah 3, 4 dan 5. Pada penelitian tersebut nilai AUC rata-rata yang dihasilkan Random Forest sebesar 0.72 dan *Deep Forest* sebesar 0.73 [8].

4.2. Pembahasan

Dalam melakukan prediksi cacat *software*, dataset yang digunakan telah melewati proses normalisasi *z-score*. *Z-score* mampu meningkatkan kinerja klasifikasi pada sebagian metode dan dataset seperti pada Tabel 8.

Secara umum *hyperparameter tuning* yang dilakukan seluruh algoritma berbasis pohon berhasil meningkatkan prediksi cacat *software* jika dibandingkan dengan penelitian terdahulu [8] [9], hal ini terlihat dari perbandingan antara rata-rata kinerja dari setiap metode yang diusulkan lebih tinggi dari metode dari penelitian sebelumnya seperti yang terlihat pada Tabel 13.

Berdasarkan Tabel 13, untuk algoritma *Deep Forest* yang diusulkan mengalami peningkatan jika dibandingkan dengan algoritma *Deep Forest* pada penelitian sebelumnya. Namun jika dibandingkan dengan kinerja yang dihasilkan oleh algoritma *Random Forest* dengan *hyperparameter tuning*, terdapat nilai kinerja yang sedikit lebih rendah yaitu pada saat melakukan prediksi untuk dataset Safe. Walaupun begitu, nilai rata-rata kinerja dari algoritma ini adalah yang terbaik.

Penentuan rentang nilai parameter menjadi lebih panjang untuk digunakan pada proses tuning menjadi salah satu solusi untuk menemukan nilai parameter yang optimal. Namun hal ini membuat pencarian nilai optimal dengan *grid search* menjadi lebih lama karena makin banyaknya kombinasi yang mungkin untuk digunakan. Selain itu karakteristik algoritma *Deep Forest* yang lebih rumit sehingga waktu pencarian parameter optimal lebih lama dibandingkan dengan algoritma berbasis pohon yang lain.

Untuk permasalahan kelas tidak seimbang pada dataset Safe dan Zxing, algoritma *Deep Forest* dengan metode yang diusulkan tetap dapat bekerja lebih baik sehingga menghasilkan kinerja prediksi dalam kategori bagus sesuai nilai rentang pada poin 3.5.

5. KESIMPULAN DAN SARAN

Prediksi cacat *software* pada dataset ReLink dengan membandingkan algoritma klasifikasi berbasis pohon seperti *Decision Tree*, *Random Forest* dan *Deep Forest* menghasilkan kinerja yang berbeda-beda. Berdasarkan hasil perbandingan yang didapatkan, *Deep Forest* memiliki kinerja lebih baik dibandingkan algoritma klasifikasi berbasis pohon yang lain.

Pada penelitian selanjutnya akan dilakukan pengujian algoritma *Deep Forest* pada dataset cacat *software* yang memiliki rasio ketidakseimbangan

lebih tinggi. Tujuannya untuk mengetahui kehandalan kinerja algoritma ini dalam memprediksi cacat *software* pada kasus data tidak seimbang.

Penelitian lanjutan lainnya adalah melakukan eksperimen terhadap metode pencarian parameter optimal agar proses *hyperparameter tuning* dapat bekerja lebih cepat dibandingkan metode *grid search*. Tujuannya agar rentang parameter yang lebih lebar bisa digunakan dengan harapan didapatkan nilai kinerja klasifikasi yang lebih baik.

DAFTAR PUSTAKA

- [1] A. Faisol, N. Vendyansyah, and F. T. Industri, "Produksi Di Kalimantan Tengah Berbasis Web," vol. 4, no. 2, pp. 170–175, 2020.
- [2] R. S. Wahono, "A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks," *J. Softw. Eng.*, vol. 1, no. 1, pp. 1–16, 2007, doi: 10.3923/jse.2007.1.12.
- [3] W. Zheng, S. Mo, X. Jin, Y. Qu, Z. Xie, and J. Shuai, "Software defect prediction model based on improved deep forest and AutoEncoder by forest," *Proc. Int. Conf. Softw. Eng. Knowl. Eng. SEKE*, vol. 2019-July, no. 3, pp. 419–424, 2019, doi: 10.18293/SEKE2019-008.
- [4] R. S. Wahono, "A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks," *J. Softw. Eng.*, vol. 1, no. 1, pp. 1–16, 2015.
- [5] M. A. Mabayoje, A. O. Balogun, H. A. Jibril, J. O. Atoyebi, H. A. Mojeed, and V. E. Adeyemo, "Parameter tuning in KNN for software defect prediction: an empirical analysis," *J. Teknol. dan Sist. Komput.*, vol. 7, no. 4, pp. 121–126, 2019, doi: 10.14710/jtsiskom.7.4.2019.121-126.
- [6] R. A. Nugroho, F. Abadi, M. R. Faisal, R. Herteno, and R. Ramadhani, "Metrics Based Feature Selection for Software Defect Prediction," *J. Komputasi*, vol. 8, no. 2, 2020, doi: 10.23960/komputasi.v8i2.2670.
- [7] A. Iqbal et al., "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 5, pp. 300–308, 2019, doi: 10.14569/ijacsa.2019.0100538.
- [8] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *Inf. Softw. Technol.*, vol. 114, no. July 2018, pp. 204–216, 2019, doi: 10.1016/j.infsof.2019.07.003.
- [9] S. Ghosh, A. Rana, and V. Kansal, "A Nonlinear Manifold Detection based Model for Software Defect Prediction," *Procedia Comput. Sci.*, vol. 132, pp. 581–594, 2018, doi:

- 10.1016/j.procs.2018.05.012.
- [10] H. H. Patel and P. Prajapati, "Study and Analysis of Decision Tree Based Classification Algorithms," *Int. J. Comput. Sci. Eng.*, vol. 6, no. 10, pp. 74–78, 2018, doi: 10.26438/ijcse/v6i10.7478.
- [11] H. Esmaily, M. Tayefi, H. Doosti, M. Ghayour-Mobarhan, H. Nezami, and A. Amirabadizadeh, "A comparison between decision tree and random forest in determining the risk factors associated with type 2 diabetes," *J. Res. Health Sci.*, vol. 18, no. 2, 2018.
- [12] Z. H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 0, pp. 3553–3559, 2017, doi: 10.24963/ijcai.2017/497.
- [13] C. Saranya and G. Manikandan, "A study on normalization techniques for privacy preserving data mining," *Int. J. Eng. Technol.*, vol. 5, no. 3, pp. 2701–2704, 2013.
- [14] D. Marinov and D. Karapetyan, "Hyperparameter optimisation with early termination of poor performers," *2019 11th Comput. Sci. Electron. Eng. Conf. CEEC 2019 - Proc.*, no. September, pp. 160–163, 2019, doi: 10.1109/CEEC47804.2019.8974317.
- [15] D. Sciences, "Decision tree classifier: a detailed survey Priyanka * and Dharmender Kumar," vol. 12, no. 3, pp. 246–269, 2020.
- [16] S. Cui, Y. Yin, D. Wang, Z. Li, and Y. Wang, "A stacking-based ensemble learning method for earthquake casualty prediction," *Appl. Soft Comput.*, vol. 101, p. 107038, 2021, doi: 10.1016/j.asoc.2020.107038.
- [17] L. V. Utkin, M. S. Kovalev, and A. A. Meldo, "A deep forest classifier with weights of class probability distribution subsets," *Knowledge-Based Syst.*, vol. 173, pp. 15–27, 2019, doi: 10.1016/j.knosys.2019.02.022.
- [18] L. V. Utkin, "An imprecise deep forest for classification," *Expert Syst. Appl.*, vol. 141, p. 112978, 2020, doi: 10.1016/j.eswa.2019.112978.
- [19] D. Berrar, "Cross-validation," *Encycl. Bioinforma. Comput. Biol. ABC Bioinforma.*, vol. 1–3, no. January 2018, pp. 542–545, 2018, doi: 10.1016/B978-0-12-809633-8.20349-X.
- [20] R. Wu, H. Zhang, S. Kim, and S. C. Cheung, "ReLink: Recovering links between bugs and changes," *SIGSOFT/FSE 2011 - Proc. 19th ACM SIGSOFT Symp. Found. Softw. Eng.*, pp. 15–25, 2011, doi: 10.1145/2025113.2025120.
- [21] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?," *Softw. Qual. J.*, vol. 26, no. 2, pp. 525–552, 2018, doi: 10.1007/s11219-016-9353-3.
- [22] H. Alibrahim and S. A. Ludwig, "Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization," pp. 1551–1559, 2021, doi: 10.1109/cec45853.2021.9504761.
- [23] H. Aljamaan and A. Alazba, "Software defect prediction using tree-based ensembles," *PROMISE 2020 - Proc. 16th ACM Int. Conf. Predict. Model. Data Anal. Softw. Eng. Co-located with ESEC/FSE 2020*, pp. 1–10, 2020, doi: 10.1145/3416508.3417114.
- [24] M. Daviran, A. Maghsoudi, R. Ghezelbash, and B. Pradhan, "A new strategy for spatial predictive mapping of mineral prospectivity: Automated hyperparameter tuning of random forest approach," *Comput. Geosci.*, vol. 148, no. November 2020, p. 104688, 2021, doi: 10.1016/j.cageo.2021.104688.
- [25] R. S. Wahono, N. S. Herman, and S. Ahmad, "A Comparison Framework of Classification Models for Software Defect Prediction," no. October, 2014, doi: 10.1166/asl.2014.5640.