INTEGRASI METODE RESAMPLING DAN K-NEAREST NEIGHBOR PADA PREDIKSI CACAT SOFTWAR APLIKASI ANDROID

Rimbun Siringoringo
Manajemen Informatika, Universitas Methodist Indonesia
E-mail : ringorbnsrg@gmail.com

ABSTRACT

efects are actual errors that can be found in the software. The existence of defects can make the software does not work optimally even crashes. The existence of defect is something that must be eliminated in order to produce high-quality software. Testing Techniques conventional software is cracking failure is flawed searches on a case by case basis. This technique is done by flawed search case by case basis. To ensure the quality of the software, we need models and effectively defect testing methods.Method of k-Nearest Neighbor (k-NN) is one of the popular classification method that is widely applied to build predictive models. K-NN method is very susceptible to errors caused by the imbalance of the class., because it results in a low level of accuracy. class imbalance produces a low level of accuracy. To address the class imbalance, resampling methods applied at the pre-processing stage. Experiments done by comparing the results that obtained with and without resampling method. To validate the superiority of the model, the experimen results compared to other classification methods, namely J48 and decision table. The results showed that the use of resampling methods produce significant performance improvements to the k-NN. The proposed model is better than the method J48 and decision table.

Keywords : *k-Nearest Neighbor, resampling, software defects, android aplication*

ABSTRAK

Cacat merupakan kesalahan aktual yang dapat ditemukan pada perangkat lunak. Keberadaan cacat dapat mengakibatkan perangkat lunak tidak berfungsi maksimal bahkan mengalami crash. Keberadaan cacat merupakan hal yang harus dihilangkan dalam rangka menghasilkan software berkualitas tinggi. Teknik Pengujian software yang konvensional adalah tracking failure. Teknik ini dilakukan dengan cara penelusuran cacat secara kasus per kasus. Untuk menjamin kualitas software, dibutuhkan model dan metode pengujian cacat yang efektif. Metode K-nearest neighbor merupakan salah satu metode yang populer dan banyak diterapkan dalam membangun model prediksi cacat pada aplikasi android berbasis klasifikasi. Metode k-NN *dikenal sangat rentan terhadap ketidak seimbangan kelas, karena hal tersebut menghasilkan tingkat akurasi yang rendah. Untuk mengatasi ketidak seimbangan kelas tersebut, metode resampling diterapkan pada tahap pre-processing. Eksperimen dilakukan dengan membandingkan hasil yang diperoleh dengan resampling dan tanpa resampling . Untuk memvalidasi superioritas model, hasil penelitian dibandingkan dengan metode klasifikasi lain yaitu J48 dan decision table. Hasil penelitian menunjukkan bahwa penerapan resampling mengakibatkan perbaikan performa yang signifikan terhadap* k-NN*, model yang diusulkan pada penelitian ini lebih baik dari hasil yang dicapai dengan metode J48 dan decision table.*

Kata kunci : *k-Nearest Neighbor, resampling, cacat software, aplikasi android*

PENDAHULUAN

Software berkualitas tinggi adalah *software* yang tidak mengandung cacat pada saat pegujian, pemeriksaan dan implementasi.[1] Keberadaan cacat merupakan ukuran kualitas perangkat lunak.[2]

Cacat atau *fault* merupakan merupakan suatu kesalahan aktual yang dapat ditemukan pada perangkat lunak. Keberadaan cacat pada kode program dapat mengakibatkan perangkat lunak tidak berfungsi sebagaimana yang diharapkan. Cacat sering tidak terdeteksi pada saat proses *debugging* atau *testing*, keberadaan cacat baru terdeteksi pada saat di implementasikan. Keberadaan cacat pada perangkat lunak merupakan hal yang harus dihilangkan karena hal tersebut dapat meningkatkan biaya dan waktu perawatan bahkan mengancam keselamatan pengguna.[3]

Perkembangan teknologi telepon cerdas (*smartphone*) merupakan salah satu terobosan yang sangat berdampak pada banyak hal. Telepon cerdas mengubah cara manusia untuk berkomunikasi dan saling bertukar informasi bahkan menikmati hiburan. Selama sepuluh tahun terakhir ini telah terjadi peningkatan pengunaan smartphone yang sangat signifikan. Hal tersebut dilatar belakangi meningkatnya kuantitas aplikasi *mobile* dan kuantitas *smartphone* yang diproduksi. Analis industri memperkirakan bahwa terdapat sekitar 250.000 aplikasi *mobile* yang tersedia pada AppStore dan Google Play. [4] Selain peningkatan kuantitas aplikasi yang tersedia, salah satu hal yang tidak kalah pentingnya adalah mejamin kelayakan kualitas aplikasi. Kuantitas aplikasi yang dimaksud dapat berupa ketiadaan cacat atau *zero-defect.*
Pengujian *software* merupakan salah satu tahapan yang sangat krusial dalam rangka mengetahui kelayakan suatu perangat lunak. Tahapan ini bertujuan untuk menemukan sebanyak mungkin *flaws* sebelum dilanjutkan ke tahap produksi [5]. Metode pengujian yang konvensional adalah *tracking faiure* yaitu dengan menelusuri kesalahan kasus demi kasus. Metode ini membutuhkan waktu dan biaya yang banyak, dimana sekitar 50% dari waktu pengembangan perangkat lunak digunakan untuk metode ini [6].

Selain dari metode pemeriksaan cacat yang dijelaskan di atas, metode pencegahan cacat atau *defect prediction* merupakan pendekatan terbaik yang dapat dilakukan [7]. Metode ini tidak membutuhkan biaya dan waktu yang besar serta prosesnya dapat diulang dikemudian hari sesuai kebutuan.

Prediksi cacat difokuskan pada beberapa hal yaitu (i) memperkirakan jumlah cacat yang ada pada sistem *software* (ii) menemukan asosoasi antar cacat dan (iii) mengklasifikasi cacat atas dua kondisi atau *class* yaitu tanpa cacat dan cacat.[8] Penelitian ini dilakukan dengan menekankan pada aspke ke tiga tersebut.

Keseimbangan kelas atau *class imbalanced* merupakan hal yang sangat penting dalam rangka menghasilkan data *training* atau *training set* yang baik. Hampir semua algoritma klasifikasi menunjukkan performa yang sangat buruk ketika bekerja pada data dengan kelas yang sangat tidak seimbang atau *imbalance*. Pada hakekatnya data *real* adalah tidak seimbang. Hal tersebut menyulitkan metode klasifikasi dalam melakukan generalisasi pada proses *machine learning*. Metode *k-Nearest Neighbor* (k-NN) sebagai metode klasifikasi konvensional tidak delengkapi dengan kemampuan untuk menangani masalah ketidak seimbangan kelas.

Strategi *resampling* merupakan metode yang telah banyak digunakan untuk menangani masalah ketidak seimbangan kelas dengan cara (i) *undersampling* yaiu mengeliminasi sebahagian data dari kelas yang mayoritas dan (ii) *oversamping* yaitu menambahkan data pada data dengan kelas minoritas atau (iii) *hybrid* yaitu gabungan *under-sampling* dengan *over-samping*. [9] Penelitian ini menggunakan *resampling* untuk menangani masalah ketidakseimbangan kelas pada *dataset* prediksi cacat perangkat lunak apikasi android.

Penelitian dalam menerapkan metode *resampling* dalam rangka meningkatkan akurasi klasifikasi. Penelitian tersebut diterapkan pada 100 jenis *dataset* yang dikelompokkan atas dua *pool* yaitu *real dataset* dan *artificial dataset* . *Dataset* diuji dengan tiga jenis metode klasifikasi yaitu *Decision Tree*, k-Nndan logR (logR(*Logistic Regression*). Hasil penelitian menunjukkan bahwa penerapan *resampling* untuk menangani ketidakseimbangan kelas berhasil meningkatkan akurasi klasifikasi. Pada penelitian tersebut, tidak ditemukan metode klasifikasi yang dominan pada semua *dataset*.[10]

Penelitian dalam menerapkan metode *resampling* untuk mengatasi ketidak seimbangan kelas pada empat jenis *dataset credit scoring* yaitu *German Credit, Australian Credit, Japan Credit* dan *UCSD* (*Universal Credit San Diego*). *Dataset* diuji dengan metode klasifikasi SVM (*Support Vector Machines*) dan logR (*Logistic Regression*). Hasil penelitian menunjukkan bahwa penerapan *resampling* berhasil meningkatkan *ranking* kinerja klasifikasi. Tanpa *resampling* , *ranking* performa klasifikasi AUC (*Area Under Curve*) diperoleh sebesar 7, 708 dan 7,083. Dengan menerapkan *resampling*, *ranking* performa AUC naik menjadi 2,333 dan 3,667.[11]

Penelitian tentang *resampling* dalam rangka mengatasi masalah ketidakseimbangan kelas pada 17 *dataset* pada UCI *repository*. Dengan menerapkan metode klasifikasi k-NN, SVC (*Support Vector Clasifier*), NBC (*Naive Bayes Clasifier*), *Decision Tree* (J48) dan RBF (*Radial Basis Function*). Performa klasifikasi didasarkan atas Tprate, TNrate dan Gn (*Geometric mean*). Hasil penelitian menunjukkan bahwa metode *resampling* memiliki pengaruh yang sangat signifikan pada peningkatan akurasi pada kelas minoritas (TPrate).[12]

Algoritma k-NN

Agoritma k-NN disebut juga dengan Agoritma k-Nearest Neighbor (k-NN). Menurut [13], metode *K-Nearest Neighbor* (k-NN) merupakan metode klasifikasi klasik yang paling sederhana. k-NN melakukan klasifikasi terhadap objek berdasarkan data pembelajaran yang jaraknya paling dekat (*nearest*) dengan objek yang duji. Metode k-NN menggunakan prinsip ketetanggaan *(neighbor)* untuk memprediksi *class* yang baru. Jumlah tetangga yang dipakai adalah sebanyak K tetangga.

Prosedur k-NN dapat dijelaskan sebagai berikut. Misalkan terdapat sebuah data uji $z = (x', y')$ dan data latih $d = (x', x)$, dengan x' ada atribut data uji, dan y' adalah label *class* data uji yang belum diketahui. Selanjutnya dihitung jarak antara data uji ke setiap data latih, kemudian mengambil k tetangga terdekat pertama. Setelah mengambil k tetangga terdekat pertama kemudian dihitung jumlah data yang mengikuti kelas yang ada dari k tetangga tersebut. Kelas dengan data terbanyak yang mengikutinya menjadi kelas pemenang yang diberikan sebagai label kelas pada data uji y'. [13]

Algoritma k-Nearest Neighbor :

1. z = (x',y'), adalah data uji dengan vektor x' dan label kelas y' yang belum diketahui

2. Hitung jarak *eucledian* d(x',x), jarak diantara data uji z ke setiap vektor data latih menggunakan persamaan (2.1), kemudian simpan dalam D

$$d_i = \sqrt{\sum_{i=1}^{p} (x'_i - x_i)^2} \quad \ldots \qquad (1)$$

3. Pilih $D_z \subseteq D$, yaitu k tetangga terdekat dari z

4. Berikan label kelas dari data latih tetangga terdekat yang jumlahnya mayoritas atau terbesar menggunakan persamaan (2).

$$y' = \arg\ max \sum (x_i, y_i) \in D_z \qquad (2)$$

Matrik konfusi

Matrik konfusi atau *confusion matrix* adalah salah satu metode yang sering digunakan untuk mengevaluasi performa atau kinerja klasifikasi. Matrik konfusi didasarkan pada tabel yang menyatakan jumlah data uji yang benar diklasifikasikan dan jumlah data uji yang salah diklasifikasikan. Tabel 1 merupakan tabel matrik konfusi untuk klasifikasi biner.

Tabel 1. *Confusion matrix* untuk klasifikasi biner

Kelas Aktual	Kelas prediksi: yes	Kelas prediksi: no	Total
yes	TP	FN	P
no	FP	TN	N
Total	P'	N'	P+N

Keterangan :

True Positive (TP)	:	Jumlah dokumen dari kelas 1 yang benar diklasifikasikan sebagai kelas 1.
True Negative (TN)	:	Jumlah dokumen dari kelas 0 yang benar diklasifikasikan sebagai kelas 0
False Positive (FP)	:	Jumlah dokumen dari kelas 0 yang salah diklasifikasikan sebagai kelas 1.
False Negative (FN),	:	Jumlah dokumen dari kelas 1 yang salah diklasifikasikan sebagai kelas 0.

Selanjutnya berdasarkan tabel di atas dapat dibuat beberapa formula untuk menentukan performa klasifikasi atau prediksi yaitu *accuracy* , *error rate*, *sensitifity*, *sfesifity* dan *precision* [13],

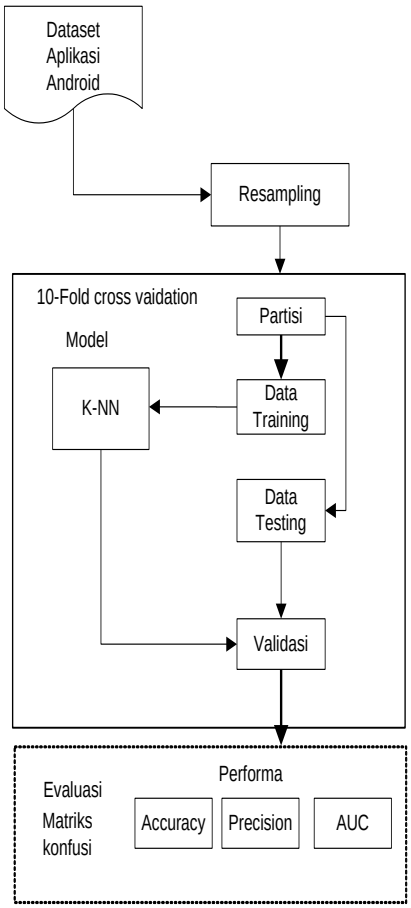
Tabel 2. Pengukuran performa prediksi dan klasifikasi

Kriteria	Formula	
Accuracy , recognition rate	$\frac{TP + TN}{P + N}$	(3)
Error rate, misclassification rate	$\frac{FP + FN}{P + N}$	(4)
Sensitivity, true positive rate	$\frac{TP}{P}$	(5)
Specificity, true negative rate	$\frac{TN}{N}$	(6)
precision	$\frac{TP}{TP + FP}$	(7)

METODE PENELITIAN

Kerangka Pemikiran

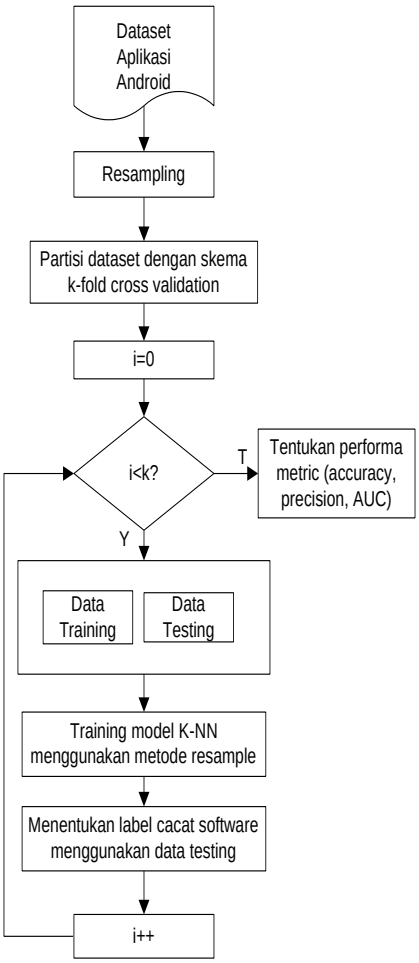
Metode yang diusulkan pada penelitian ini dapat disajikan pada gambar 1berikut ini



Gambar 1. Kerangka Pemikiran

Dataset aplikasi *mobile* dipisahkan menjadi data *training* dan data *testing* kemudian divalidasi menggunakan skema 10-*fold cross validation*. Model yang diterapkan untuk melakukan prediksi adalah k-NN. Pengaruh *resampling* terhadap kinerja k-NN dibandingkan dengan performa yang dihasilkan algoritma klasifikasi yang lain yaitu *Decision Tree* (J48) dan *Decision Table*. Perbandingan performa ke tiga model tersebut diuji berdasarkan matriks konfusi untuk menentukan *accurayc*, *precision* dan *AUC (Area Under Curve)*

***Flowchart* model yang diusulkan**



Gambar 2. *Flowchart* model yang diusulkan

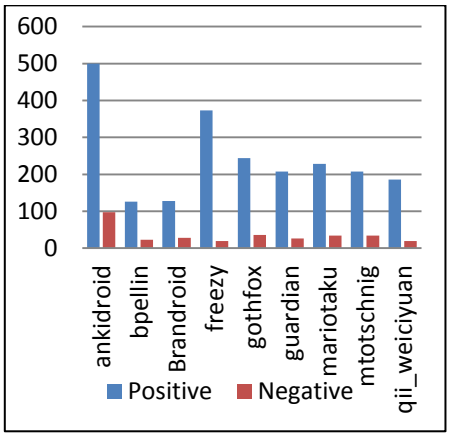
Dataset* Aplikasi *Mobile

Dataset aplikasi *mobile* yang yang digunakan pada penelitian ini terdiri atas sembilan *dataset* aplikasi *mobile* yang bersumber dari *F-Droid Repository*. Pada tabel berikut ditampilkan detail keterangan setiap *dataset* tersebut yaitu jumlah atribut (#A), jumlah *record* (#R), jumlah *record* cacat (#C) dan persentase cacat (#%C).

Tabel 3. *Dataset* Aplikasi *Mobile*

No	*Dataset*	#A	#R	#C	#%C
1	ankidroid	17	597	97	16%
2	bpellin	17	149	23	15%
3	Brandroid	17	156	28	18%
4	freezy	17	392	19	19%
5	gothfox	17	280	36	13%
6	guardian	17	233	26	11%
7	mariotaku	17	262	34	13%
8	mtotschnig	17	241	34	14%
9	qii_weiciyuan	17	205	19	19%

Dataset apikasi android yang diterapkan pada penelitian ini merupakan *dataset* dengan karakteristik ketidak seimbangan kelas, dimana jumlah *record* positif tidak sebanding dengan jumlah *record* negatif. Dalam hal ini, *record* positif adalah tidak adany cacat, dan *record* negatif adalah adanya cacat. Perbandingan ketidak seimbangan kelas tiap *dataset* diampilkan pada gambar 2 berikut ini.



Gambar 3. Perbandingan ketidak seimbangan kelas *dataset*

Dataset tersebut di atas tersedia pada file *dataset* dengan format **.arff** (*atribut relation file format*) dengan sumber F-Droid Repository. Pada gambar berikut ini ditampilkan sampel isi file *dataset* **ankidroid.arff.** Isi dari file tersebut terdiri atas daftar atribut (@attribute) dan data setiap atribut (@data).

```
@attribute FromPrev numeric
@attribute ToNext numeric
@attribute duplicate numeric
@attribute filecur numeric
@attribute fileprev numeric
@attribute LOC numeric
@attribute CYCLOMATIC numeric
@attribute COMMITS numeric
@attribute FIXCOMMITS numeric
@attribute ADD numeric
@attribute DEL numeric
@attribute TOPDIR numeric
@attribute BOTTOMDIR numeric
@attribute FILE numeric
@attribute FILEENTROPY numeric
@attribute CHURNENTROPY numeric
@attribute crashrelease {0,1}

@data
1,0,1,3,3,45136,8795,1,0,1,1,1,1,1,0
,0,0
1,1,0,3,9,45085,8789,1,0,3,1,1,1,1,0
,0,0
1,1,0,9,0,45083,8789,1,1,8,3,1,1,1,0
,0,0
0,1,0,0,3,44842,8739,0,0,0,0,0,0,0,0
,0,0
4,0,1,3,5,44841,8737,2,0,22,11,1,2,2
,1,0.967295,0
0,4,0,5,1,44796,8729,2,0,4,0,1,1,2,1
,0.811278,0
0,0,0,1,0,44776,8727,1,0,18,21,1,1,1
,0,0,0
0,0,0,0,2,44779,8727,0,0,0,0,0,0,0,0
,0,0
0,0,2,2,4,44750,8724,1,0,25,30,1,1,1
,0,0,0
5,0,0,4,2,44777,8726,5,0,77,414,1,1,
2,0.721928,0.952729,0
1,5,0,2,1,45057,8774,1,0,48,8,1,1,1,
0,0,0
0,1,0,1,5,45025,8771,1,0,4,3,1,1,1,0
,0,0
1,0,1,5,2,45024,8771,1,0,0,40,1,1,1,
0,0,0
0,1,0,2,0,45065,8772,1,0,2,2,1,1,1,0
,0,1
```

Gambar 4. Sampel File *dataset* ankidroid_Anki-Androidmeta.arff.

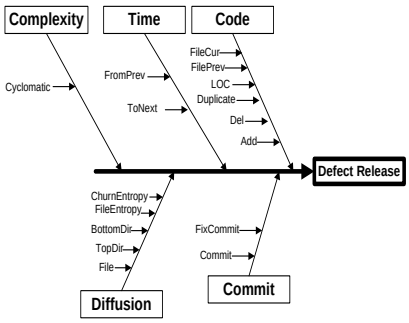
Pada tabel 4 berikut ini ditampilkan daftar atribut *dataset* aplikasi *mobile*. Keseluruhan *dataset* aplikasi *mobile* yang disajikan pada tabel di atas terdiri atas tujuh belas atribut termasuk atribut *class*.[14]

Tabel 4. daftar atribut *dataset* aplikasi *mobile*

No	Atribut	Keterangan
1	FromPrev	Interval waktu (dalam hari) antara dua release, sekarang dengan sebelumnya
2	ToNext	Interval waktu (dalam hari) antara dua release, sekarang dengan sesudahnya
3	duplicate	Total file kode program yang dimodifikasi pada release sekarang dan release sebelumnya
4	filecur	Jumlah file kode program
5	fileprev	Jumlah file kode program pada aplikasi sebelumnya
6	LOC	Total baris kode program
7	CYCLOMATIC	Jumlah percabangan (*branching*) pada source code aplikasi
8	COMMITS	Total commit.
9	FIXCOMMITS	Total commit yang mengandung bug-but yang tetap
10	ADD	Jumlah baris program (line of code) yang ditambahkan pada release baru
11	DEL	Jumlah baris program (line of code) yang hapus pada release baru
12	TOPDIR	Total path induk yang terdapat pada direktory file source code
13	BOTTOMDIR	Total path anak yang terdapat pada direktory file source code
14	FILE	Total file unik yang dimodifikasi di antara dua release
15	FILEENTROPY	Distribusi dari kode program yang dimodifikasi pada release
16	CHURNENTROPY	Distribusi dari kode program yang dimodifikasi pada aplikasi
17	crashrelease {0,1}	Kelas *dataset*

Diagram *Fishbone*

Pada tabel di atas dijelaskan daftar atribut *dataset* aplikasi *mobile*. Hubungan antara atribut-atribut tersebut terhadap cacat *software* dapat dijelaskan melalui diagram *fishbone* pada gambar 3 berikut ini



Gambar 5. Diagram *fishbone*

Complexity : Jika sebuah aplikasi memiliki tigkat kompleksitas yang tinggi (misalnya jumlah aliran data yang tinggi dalam aplikasi), hal tersebut membutuhkan tingkat pemeliharaan yang sulit, sehingga membuka peluang yang besar untuk mengalami *crash*

Time : Jika interval waktu antara sebelum release dengan release sekarang berlangsung singkat, release sekarang memiliki peluang yang lebih besar untuk tidak crash.

Code : Jika sebuah rilis memiliki perubahan besar pada source code, maka ia memiliki kemungkinan lebih besar untuk mengalami cacat, yang pada gilirannya dapat menyebabkan crash. Jika rilis sekarang memiliki modifikasi yang banyak pada source code sebelumnya, maka ini adalah suatu indikasi bahwa rilis saat ini memiiki lebih banyak perbaikan.

Diffusion : Secara umum, rilis yang sangat terdistribusi atau tersebar lebih sulit untuk dipahami, dan membutuhkan lebih banyak pekerjaan untuk memeriksa semua lokasi *source kode* yang berubah. Studi prediksi cacat sebelumnya menemukan bahwa

perubahan yang tersebar adalah indikator dari cacat. Jika rilis memiliki sejumlah besar commit, rilis memiliki probabilitas tinggi untuk menjadi crash. Keyakinan ini didorong bahwa semakin banyak commit maka hal itu berarti perubahan yang lebih banyak (misalnya perbaikan bug, fungsionalitas baru, untuk aplikasi, yang dapat memperkenalkan lebih banyak masalah (misalnya, bug) pada rilis. *Pr*

Partisi *Dataset*

Teknik validasi yang diterapkan pada penelitian ini adalah teknik *k-fold cross validation* dengan jumlah *k* atau *fold* adalah 10. Setiap *dataset* aplikasi *mobile* harus dipartisi menjadi 10 partisi. Pada tahap *training* model digunakan file partisi *training* (tra) dan pada tahap pengujian atau *testing* digunakan file partisi *testing* (tst). Skema partisi *dataset* ankidroid.arff disajikan pada tabel berikut ini

Tabel 5. Skema partisi *dataset*

Fold	Partisi	Keteragan
Fold-1	ankidroid-10-1tra	*Training*
	ankidroid-10-1tst	*test*
Fold-2	ankidroid-10-2tra	*Training*
	ankidroid-10-2tst	*test*
Fold-3	ankidroid-10-3tra	*Training*
	ankidroid-10-3tst	*test*
Fold-4	ankidroid-10-4tra	*Training*
	ankidroid-10-4tst	*test*
Fold-5	ankidroid-10-5tra	*Training*
	ankidroid-10-5tst	*test*
Fold-6	ankidroid-10-6tra	*Training*
	ankidroid-10-6tst	*test*
Fold-7	ankidroid-10-7tra	*Training*
	ankidroid-10-7tst	*test*
Fold-8	ankidroid-10-8tra	*Training*
	ankidroid-10-8tst	*test*
Fold-9	ankidroid-10-9tra	*Training*
	ankidroid-10-9tst	*test*
Fold-10	ankidroid-10-10tra	*Training*
	ankidroid-10-10tst	*test*

Performa pengujian

Penelitian ini bertujuan untuk menguji model dalam rangka prediksi cacat pada aplikasi *mobile*. Performa pegujian model dievaluasi berdasarkan tiga kriteria yakni *acuracy*, *precision*dan *recall*

a. Accuracy

Accuracy didefinisikan sebagai tingkat kedekatan antara nilai prediksi cacat dan non cacat dengan nilai aktual

cy = (TP+TN) / (TP+FP+TN+FN)

b. *Precision*

*Precision*merupakan perbandingan antara jumlah *record* cacat yang di klasifikasikan dengan tepat dengan jumlah *record* yang diklasifiasikan sebagai *record* cacat *Precision* = TP / (TP+FP)

c. *AUC*

AUC atau *Area Under Curve*

AUC =(Sensitivity + Specificity) / 2

HASIL DAN PEMBAHASAN

Ekperimen dilakukan dalam sebuah platform komputer berbasis Intel Core i3-4039U @1.90GHz (4 CPUs), RAM 4 GB, dan sistem operasi Microsoft Windows 10 pro 64-bit. Sedangkan lingkungan pengembangan aplikasi Weka 3.8.1, untuk analisa hasil eksperimen menggunakan aplikasi Microsoft Excel 2007.

Penelitian menggunakan 9 *dataset* aplikasi *mobile* yakni *ankidroid, bpellin, Brandroid, freezy, gothfox, guardian, mariotaku, mtotschnig dan qii_weiciyuan*. Metode yang diuji adalah k-NN. Informasi yang disajikan adalah *accuracy , precision* dan *AUC*. Pada tabel 8 ditampilkan hasil eksperimen *confusion matrix* tanpa *resampling* dan pada tabel 9 ditampilkan *confusion matrix* pada penerapan *resampling* .

Confusion Matrix

Informasi *accuracy , precision* dan *AUC* didasarkan pada *confusion matrix*. Pada tabel 8 berikut disajikan *Confusion matrix* yang dihasilkan pada pengujian tiap model.

Tabel 6. *Confusion matrix* k-NN

DATASET	TP	FP	FN	TN	TOTAL
ankidroid	421	77	79	20	597
bpellin	106	20	20	3	149
Brandroid	98	23	30	5	156
freezy	353	18	20	1	392
gothfox	216	25	28	11	280
guardian	186	21	21	5	233
mariotaku	204	25	24	9	262
mtotschnig	185	26	22	8	241
qii_weiciyuan	144	16	22	23	205
Rata-rata	**212,6**	**27,89**	**29,56**	**9,44**	**279,4**

Tabel 7. *Confusion matrix Resampling* k-NN (R+k-NN)

DATASET	TP	FP	FN	TN	TOTAL
ankidroid	468	32	28	69	597
bpellin	121	5	9	14	149
Brandroid	114	14	9	19	156
freezy	365	8	7	12	392
gothfox	233	11	11	25	280
guardian	202	5	12	14	233
mariotaku	218	10	12	22	262
mtotschnig	193	14	11	23	241
qii_weiciyuan	156	10	7	32	205
Rata-rata	**230**	**12,11**	**11,78**	**25,56**	**279,4**

Pada tabel 8 dan 9 di atas ditampilkan bahwa *resamping* mengakibatkan perbaikan yang signifikan pada *confusion matrix* k-NN. Perbaikan tersebut terindikasi dengan kenaikan rata-rata *True Positif* (TP) dan *True Negatif* (TN), serta penurunan rata-rata *False Positif* (FP) dan *False Negative* (FN). Tanpa *resampling* rata-rata TP adalah 212,6 dan rata-rata TN adalah 9,444. Dengan resamping Nilai TP dan TN adalah 230 dan 25,56. Perbaikan lainnya adalah adanya penurunan FP dan FN. Tanpa resamping, nilai FP dan FN adalah 27,89 dan 29,56. Dengan *resampling* , nilai FP dan FN turun menjadi 12,11 dan 11,78

Performa Model

Pada tabel berikut ditampilkan hasil pengujian metode k-NN dalam mengklasifiasi cacat *software* pada 9 *dataset* aplikasi android. Pada tabel tersebut diperole rata-rata *accuracy* sebesar 78,81 %, rata-rata *precision* sebesar 87,89 % dan rata-rata AUC sebesar 55,5 %

Tabel 8. Hasil Eksperimen k-NN

DATASET	ACC	PREC	SENS	SPEC	AUC
ankidroid	73,87	84,54	84,2	20,62	52,41
bpellin	73,15	84,13	84,13	13,04	48,59
Brandroid	66,03	80,99	76,56	17,86	47,21
freezy	90,31	95,15	94,64	5,263	49,95
gothfox	81,07	89,63	88,52	30,56	59,54
guardian	81,97	89,86	89,86	19,23	54,54
mariotaku	81,3	89,08	89,47	26,47	57,97
mtotschnig	80,08	87,68	89,37	23,53	56,45
qii_weiciyuan	81,46	90	86,75	58,97	72,86
Rata-rata	**78,81**	**87,89**	87,06	23,95	**55,5**

Pada tabel berikut ditampilkan hasil pengujian metode k-NN dengan *resampling* (R+k-NN). Pengujian model tersebut juga diterapkan pada 9 *dataset* aplikasi android. Penerapan *resampling* pada k-NN berhasil meningatkan nilai *accuracy* menjadi 91,09 %, precision sebesar 94,72 % dan nilai AUC sebesar 81,28 %.

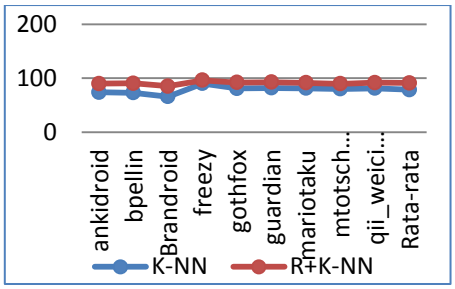
Tabel 9. Hasil Eksperimen *Resampling* + k-NN (R+k-NN)

DATASET	ACC	PREC	SENS	SPEC	AUC
ankidroid	90	93,6	94,4	68,3	81,3
bpellin	90,6	96	93,1	73,7	83,4
Brandroid	85,3	89,1	92,7	57,6	75,1
freezy	96,2	97,9	98,1	60	79,1
gothfox	92,1	95,5	95,5	69,4	82,5
guardian	92,7	97,6	94,4	73,7	84
mariotaku	91,6	95,6	94,8	68,8	81,8
mtotschnig	89,6	93,2	94,6	62,2	78,4
qii_weiciyuan	91,7	94	95,7	76,2	86
Rata-rata	**91,1**	**94,7**	94,8	67,8	**81,3**

Tabel 10. Perbandingan Akurasi k-NN dengan R+k-NN

Dataset	k-NN (%)	R+k-NN (%)
ankidroid	73,87	89,95
bpellin	73,15	90,6
Brandroid	66,03	85,26
freezy	90,31	96,17
gothfox	81,07	92,14
guardian	81,97	92,7
mariotaku	81,3	91,6
mtotschnig	80,08	89,63
qii_weiciyuan	81,46	91,71
Rata-rata	**78,81**	**91,09**

Pada tabel 10 di atas ditampilkan perbandingan *accuracy* antara model k-NN dengan model R+k-NN. Informasi yang tertera pada tabel tersebut menunjukkan bahwa penerapan metode *resampling* pada k-NN memberikan hasil *accuracy* yang lebih baik. Pada tabel tersebut juga ditampilkan bahwa metode R+k-NN unggul pada semua *dataset* aplikasi android. Dengan metode k-NN, rata-rata *accuracy* yang dihasilkan untuk 9 *dataset* adalah 78, 81 %, dengan metode R+k-NN rata-rata *accuracy* keseluruhan *dataset* adalah 91,09 %. Pada gambar 4 berikut ditampilkan grafik perbandingan *accuracy* yang dihasilkan dengan model k-NN dengan R+k-NN



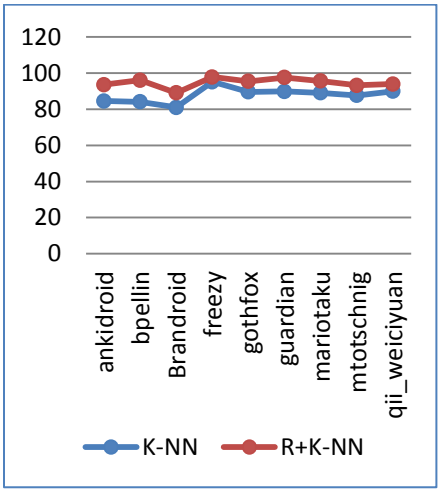
Gambar 4. Grafik perbandingan *accuracy* k-NN dengan R+k-NN

Seain aspek *accuracy* , pada tabel berikut ini ditampilkan kinerja *precision* antara model k-NN dengan R+k-NN. Sama halya dengan aspek *accuracy* , penerapan metode *resampling* pada k-NN juga memperbaiki kinerja klasifikasi.

Tabel 11. Perbandingan *Precision* k-NN dengan R+k-NN

Dataset	k-NN (%)	R+k-NN (%)
ankidroid	84,54	93,6
bpellin	84,13	96,03
Brandroid	80,99	89,06
freezy	95,15	97,86
gothfox	89,63	95,49
guardian	89,86	97,58
mariotaku	89,08	95,61
mtotschnig	87,68	93,24
qii_weiciyuan	90	93,98
Rata-rata	**87,89**	**94,72**

Pada tabel 11 tersebut juga ditampilkan bahwa metode R+k-NN unggul pada semua *dataset* aplikasi android. Dengan metode k-NN, rata-rata *precision* yang dihasilkan untuk 9 *dataset* adalah 87,89 %, dengan metode R+k-NN rata-rata *precision* keseluruhan *dataset* adalah 94, 72 %. Dalam bentuk grafik, perbandingan *precision* ditampilkan pada gambar 5 berikut.



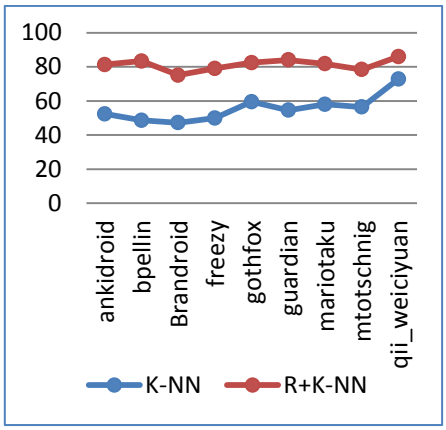
Gambar 6. Grafik perbandingan *precision* k-NN dengan R+k-NN

Tabel 12. Perbandingan *AUC* k-NN dengan R+k-NN

DATASET	k-NN (%)	R+k-NN (%)
ankidroid	52,41	81,34
bpellin	48,59	83,38
Brandroid	47,21	75,13
freezy	49,95	79,06
gothfox	59,54	82,47
guardian	54,54	84,04
mariotaku	57,97	81,77
mtotschnig	56,45	78,39
qii_weiciyuan	72,86	85,95
Rata-rata	**55,50**	**81,28**

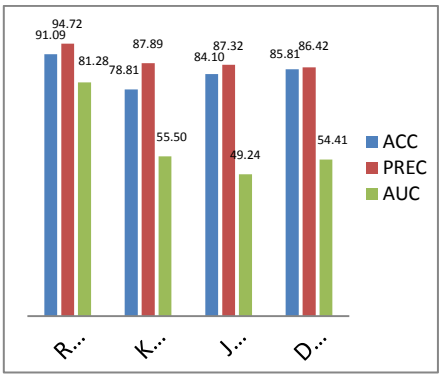
Hasil perbandingan nilai AUC pada model k-NN dengan model R+k-NN disajikan pada tabel di atas. Model R+k-NN memiliki nilai AUC yang lebih baik dibandingkan dengan k-NN. Perbaikan nila AUC terdapat pada keseluruhan *dataset* yang diuji. Rata-rata Nilai AUC

dengan k-NN adalah 55, 50 %. Nilai AUC dengan model R+k-NN adalah 81,28%. Dalam bentuk grafik, perbandingan nilai AUC ke dua model tersebut ditampilkan pada gambar berikut.



Gambar 7. Grafik perbandingan AUC metode k-NN dengan R+k-NN

Untuk memvalidasi superioritas dari model R+k-NN dalam memprediksi cacat *software* aplikasi android, hasil yang diperoleh dengan metode R+k-NN dibandingan dengan metode klasifikasi (*clasifer*) lain.



Gambar 8. Grafik Perbandingan superioritas model

Metode J48 dan *Decision Table* dipiih sebagai metode pembanding. Informasi yang disajikan pada tabel tersebut adaah rata-rata *accuracy* , *precision* dan AUC dari 9 *dataset* aplikasi android. Model pembanding, decision tree dan decision tabel memiliki peforma yang hampir sama dengan k-NN. Model R+k-NN memiliki performa klasifikasi yang lebih baik dibanding ke tiga model lainnya. informasi *accuracy* , *precision* dan AUC dengan R+k-NN adalah 91,09 %, 94,72 % dan 81,28 %. Perbandingan hasil tersebut disajikan pada tabel berikut

Tabel 13. Tabel superioritas model

Performa Metrik	Model		Pembanding	
	R+k-NN	k-NN	J48	Decision Tabel
Accuracy	**91,09**	78,81	84,10	85,81
Precision	**94,72**	87,89	87,32	86,42
AUC	**81,28**	55,50	49,24	54,41

KESIMPULAN

Penelitian ini bertujuan untuk menerapkan metode *resampling* untuk memperbaiki kinerja dari metode k-NN dalam memprediksi cacat *software* aplikasi android. Berdasarkan eksperimen dengan 9 *dataset* android, model R+k-NN memiliki performa metrik yang lebih baik dibanding k-NN saja. Suprioritas model R+k-NN juga lebih baik jika dibandingkan dengan metode klasifikasi lain aitu J48 dan decision tabel. Berdasarkan eksperimen tersebut dapa disimpulkan bahwa metode *resampling* efektif dalam mengangani ketidak seimbangan kelas pada *dataset* android.

DAFTAR PUSTAKA

[1]Rianto, H. & Wahono, R., S. 2015. *Resampling* Logistic Regression untuk Penanganan Ketidakseimbangan *Class* pada Prediksi Cacat *Software* . *Journal of Software Engineerin.* 1(1) : 46-53.

[2]Pujianto, U. 2016. *Strategi Resampling Berbasis Centroid Untuk Menangani Ketidakseimbangan Kelas Pada*

Prediksi Cacat Perangkat Lunak. Jurnal TEKNO. 25 (2016) : 1-6

[3]Devi, C., A., Surendiran, B. & Kannammal, K., E. 2011. A Study of feature selection methods for *Software* fault prediction model. *Proceedings of 1st International conference on network, intelligence and computing technology* : pp. 1-5.

[4]Wasserman, A., I. 2010. *Software Engineering Issues for Mobile Application Development*. Proceedings of the FSE/SDP workshop on Future of *software* engineering research : pp. 397-400

[5]Shuai, B. Li, H., Li, M., Zhang, Q. & Tang, C. 2013. *Software Defect Prediction Using Dynamic Support Vector Machine*. Proceeding of 9th International Conference on Computational Intelligence and Security : 260-264

[6]Saifudin, A. & Wahono, R., S. 2015. *Penerapan Teknik Ensemble untuk Menangani Ketidakseimbangan Kelas pada Prediksi Cacat Software . Journal of Software Engineering*. Journal of *Software* Engineering. 1(1) : 28-39

[5]Suharso, Puguh. (2009). Metode Penelitian Kuantitatif Untuk Bisnis: Pendekatan Filosofi dan Praktek, PT Indeks Permata Puri Media, Jakarta Barat.

[7]Chang, R. & Mu., X. & Zhang, L. 2011. *Software Defect Prediction Using Non-Negative Matrix Factorization*. Journal of *software*. 6(11) : 2114-2120

[8]Song, Q., Jia, Z., Shepperd, M., Ying, S & Liu, J. 2010. *A General Software Defect-Proneness Prediction Framework*. IEEE transactions on *software* engineering, 37 (3) : 356-370.

[9]Thanathamathee,P. & Lursinsap, C. 2013. *Handling Imbalanced Data Sets with Synthetic Boundary Data Generation Using Bootstrap Re-sampling and AdaBoost Techniques*. Pattern Recognition Letters. 4(19) : 1-36

[10]Burnaev, E., Erofeev, P., Papanov, A. 2015. *Influence of Resampling on Accuracy of Imbalanced Classification.8th* International Conference on Machine Vision (ICMV 2015) : pp. 1-6

[11]Marqués, A., I., Garcia., V. & Sánchez., J., S. 2013. *On the suitability of resampling techniques for the class imbalance problem in credit scoring*. Journal of the Operational Research Society. 64 (7) : 1060-1070

[12]Garc´ıa, V., S´anchez, J., S. & Mollineda, R., A. 2010. *Exploring the Performance of Resampling Strategies for the Class Imbalance Problem*. Trends in Intelligent system : pp. 541-549

[13]Han, J., Kamber, M. & Pei, J. (2012). *Data mining tecniques and concepts*. Morgan Kaufman publisher. Watham : USA

[14]Xia, X., Shihab, E., Kamei, Y., Lo, D. & Wang, X. 2016. *Predicting Crashing Releases of Mobile Applications*. Proceedings of the 10th ACM/IEEE International Symposium on Empirical *Software* Engineering and Measurement : pp. 1-10