BAB I
PENDAHULUAN

1.1 Latar Belakang

Kualitas dan keandalan dari sebuah software sangat berhubungan erat dengan keberhasilan dalam menghapus *bug* di dalam software. Salah satu cara konvensional untuk dapat menemukan *bug* pada software yaitu dengan melakukan testing dan review (Ding and Xing 2020). Kualitas dari sebuah software diukur berdasarkan kemampuan software tersebut untuk memenuhi persyaratan fungsional serta non-fungsional. Beberapa faktor yang memengaruhi kualitas software yaitu *Correctness, Reliability, Efficiency, Integrity*, dan *Usability* (Galin 2004).

Menemukan dan memperbaiki *bug* pada suatu software dengan melakukan software testing membutuhkan biaya dan tenaga kerja tinggi dalam proses pengembangan software (Jones and Bonsignour 2012). Perbaikan *bug* pada software setelah pengiriman membutuhkan biaya lebih mahal dibandingkan pada masa pengembangan, proses pengujian software menghabiskan waktu yang cukup lama. Menurut penelitian, terlihat bahwa penemuan *bug* yang dilakukan dengan software testing ternyata tidak efektif dalam menemukan *bug*. Penemuan *bug* yang dilakukan secara manual hanya dapat mendeteksi 35% hingga 60% dari *bug* yang ada (Shull et al. 2002). Penemuan secara manual dilakukan dengan melihat beberapa baris kode yang dilakukan oleh anggota dari review team. Dapat ditarik kesimpulan bahwa penemuan dan perbaikan *bug* software dengan cara manual software review dan software testing tidak efektif dan tidak efisien.

Karena pendekatan software testing dan review yang digunakan pada saat ini masih kurang efektif dan tidak efisien, para peneliti kemudian mencari cara supaya proses penemuan dan perbaikan *bug* bisa lebih efektif dan efisien dilakukan. Salah satu cara yang saat ini banyak dilakukan para peneliti dan menjadi tren topik penelitan di bidang *software engineering* adalah dengan prediksi *bug* software. Prediksi *bug* software adalah sebuah proses yang memprediksi bagian mana pada kode yang memiliki *bug* (Siers and Islam 2015). Prediksi otomatis untuk

menemukan *bug* pada tahap awal dapat sangat membantu pengembang software dalam memperbaiki kualitas kode tanpa membutuhkan biaya yang besar (Giray et al. 2023). Prediksi *bug* software merupakan salah satu cara untuk melakukan validasi dari sebuah sistem software (Öztürk 2017). Prediksi *bug* software membangun sebuah defect prediction models dengan melakukan mining pada repositori software dan membangun sebuah model yang dapat mengidentifikasi potensi defect pada projek baru (Chen et al. 2019). (Menzies, Greenwald, and Frank 2007) menyatakan bahwa dengan menggunakan metode prediksi, 71% *bug* software dapat ditemukan. Inilah yang membuat prediksi *bug* software saat ini menjadi salah satu tren penelitian penting di bidang software engineering (Song et al. 2011). Penelitian yang sudah dilakukan tentang prediksi *bug* software memanfaatkan berbagai metode data mining dan machine learning untuk membuat model prediksi *bug*. Metode klasifikasi yang telah digunakan seperti decision tree (Siers and Islam 2015), Naïve Bayes (Arar and Ayan 2017), random forest (Xia et al. 2015), nearest neighbor (Jing et al. 2015), neural network (Zhu et al. 2021), logistic regression (Chen et al. 2018), dan ensemble methods (Alazba and Aljamaan 2022) (Xu et al. 2019).

Salah satu kelemahan yang dimiliki oleh beberapa metode klasifikasi yang digunakan yaitu model yang *uninterpretable* atau tidak dapat mudah dipahami. (Moeyersoms et al. 2015). Walaupun metode klasifikasi dapat menghasilkan hasil yang akurat, namun data yang dihasilkan tidak mudah untuk dapat dipahami. Sehingga metode klasifikasi yang interpretable akan menjadi metode yang lebih preferred untuk digunakan dibandingkan metode yang uninterpretable.

Salah satu metode klasifikasi yang memiliki karakter model yang dihasilkan lebih *interpretable* dan mudah dipahami adalah dengan menggunakan algoritma decision tree (Juneja 2019). Algoritma decision tree merupakan algoritma klasifikasi yang berkembang dari menggunakan konsep split criteria berbasis gini index (CART) (Breiman et al. 1984), information gain (ID3) (Quinlan 1986), dan kemudian diperbaiki dengan Information Gain Ratio (C4.5) (Quinlan 1993). Penggunaan C4.5 memiliki tingkat kesuksesan tinggi pada penelitian mengenai prediksi *bug* software (Laradji, Alshayeb, and Ghouti 2015). Metode ini dipilih karena dengan menggunakan algoritma C4.5, model yang dihasilkan akan dapat lebih mudah dan cepat dipahami. Dengan begitu saat akan lebih mudah untuk menentukan dimana konsentrasi software testing akan dilakukan.

Salah satu kelemahan dari algoritma C4.5 adalah ketika bertemu dengan suatu dataset yang memilih class yang tidak seimbang seperti pada dataset prediksi *bug* software NASA MDP (Wahono and Herman 2014). Pada saat pengklasifikasian data, modul dikategorisasikan menjadi dua yaitu *defect-prone* dan *non defect-prone* (Mesquita et al. 2016) Hal ini menunjukkan bahwa kasus *bug* hanya terjadi secara minoritas dibandingkan dengan tanpa *bug*, yang mengakibatkan adanya ketidakseimbangan class pada dataset. Walaupun angka class minoritas rendah, tetapi pengklasifikasian yang benar sangat penting dilakukan agar dapat menghindari kesalahan prediksi yang dapat mengakibatkan pembangunan software dengan kualitas yang buruk serta tingginya biaya testing (Malhotra and Kamal 2019). Ketidakseimbangan class merupakan problem yang sangat sering dibahas pada prediksi *bug* software. Hal ini dikarenakan pada algoritma *machine learning* konvensional, diasumsikan bahwa jumlah dari *class minority* dan *class majority* selalu sebanding (Khoshgoftaar, Gao, and Seliya 2010). Sehingga apabila terdapat contoh yang *defective* maka contoh tersebut akan diabaikan dan akan dilakukan prediksi dimana semua kemungkinan akan muncul sebagai *non-defective* (Feng, Keung, Yu, Xiao, Bennin, et al. 2021).

Permasalahan class yang tidak seimbang atau *imbalance class* ditangani dengan menggunakan 2 tipe sampling teknik, yaitu *oversampling dan undersampling*. *Oversampling* dilakukan dengan menghasilkan data baru yang ditambahkan ke dalam *class minority*, sedangkan *undersampling* dilakukan dengan menghilangkan beberapa data yang berada di dalam *class majority* (Feng, Keung, Yu, Xiao, Bennin, et al. 2021). *Oversampling* sudah menjadi teknik yang populer digunakan dalam menangani permasalahan *imbalance class*. Salah satu metode *oversampling* terkini yang banyak digunakan pada prediksi *bug* software adalah Synthetic Minority Oversampling Technique (SMOTE) (Feng, Keung, Yu, Xiao, and Zhang 2021).

SMOTE melakukan approach oversampling dimana *class minorty* di *oversampled* dengan membuat sample sintetis dibandingkan *oversampling* dengan melakukan penggantian sampel. (Chawla et al. 2002). SMOTE relatif berhasil memecahkan masalah ketidakseimbangan class pada dataset, akan tetapi proses penambahan sample sintetis pada class minoritas mengakibatkan *noisy data* meningkat dan terjadinya *overfitting* pada *class minority* (Fernández et al. 2018). Permasalahan *overfitting* dan *noisy attribute* tersebut dapat menyebabkan turunnya kinerja dan performansi dari model prediksi *bug* software (Wahono and Herman 2014).

Noisy data dapat menyebabkan *learning algorithm* mengalami kesulitan dalam memisahkan kelas dan juga menyebabkan model mengalami *overfitting* untuk dapat mengakomodasi data

points yang salah (Khoshgoftaar, Van Hulse, and Napolitano 2011). Sebuah metode *feature selection* biasanya digunakan saat dataset yang digunakan memiliki atribut yang *noisy*. Metode *Feature Selection* diklasifikasi berdasarkan 2 pendekatan: *filter*, dan *Wrapper* method. Pendekatan *Filter* adalah pendekatan dimana fitur dipilih berdasarkan relevansi fitur tersebut terhadapt target variabel berdasarkan tes statistik (Turabieh, Mafarja, and Li 2019). Pendekatan *Wrapper* merupakan pendekatan dimana fitur subset dipilih berdasarkan evaluasi dan training oleh algoritma machine learning pada fitur subset yang berbeda (Guyon and De 2003). Teknik yang sudah sering digunakan dalam *Wrapper* method yaitu *Forward Selection* dan *Backward Elimination* (Han and Kamber 2011). *Forward Selection* memiliki keunggulan dibandingkan *backward elimination* karena lebih efisien, memliki fleksibilitas yang tinggi dan kemungkinan terjadinya *overfitting* lebih kecil (Witten, Frank, and Hall 2011). Beberapa metode yang dapat digunakan untuk menangani permasalahan *overfitting* yaitu *Cross-Validation, Early Stopping, Pruning, Regularization* dan *Ensemble method* (Ying 2019). Metode *ensemble* yang populer digunakan yaitu *bagging*, *boosting*, *AdaBoost*, dan *Random Forests* (He and Ma 2013). *Bagging* memiliki keunggulan dalam menangani *overfitting* dibandingkan boosting ketika data memiliki noise, (Wahono and Suryana 2013).

Pada penelitian ini, metode *forward selection* akan digunakan untuk seleksi atribut yang tidak relevan pada dataset yang dihasilkan metode SMOTE, sedangkan masalah *overfitting* akan diselesaikan dengan *ensemble learning* yaitu *bagging*. Metode *hybrid* SMOTE-*bagging*-*forward selection* diusulkan pada penelitian ini, untuk prediksi *bug* software dengan menggunakan algoritma C4.5 yang dapat menghasilkan model prediksi yang *interpretable*, lebih akurat, dan terbebas dari noisy atribut serta masalah *overfitting*.

Pada penelitian ini, akan dilakukan analisis serta komparasi kepada beberapa metode yang ada untuk penanganan class yang tidak seimbang. Serta akan dilakukan penanganan atribut yang tidak relevan. Kedua metode terbaik yang telah didapat lalu akan diintegrasikan, sehingga dapat meningkatkan akurasi dan performansi dari prediksi *bug* software dengan menggunakan algoritma C4.5. Sehingga ketika dilakukan prediksi *bug* software yang menggunakan decision tree, hasil yang didapat akan lebih akurat.

1.2 Rumusan Masalah

Berdasarkan uraian pada latar belakang dapat diidentifikasi beberapa permasalahan yaitu:

- Metode SMOTE dapat menangani ketidakseimbangan class pada prediksi *bug* software dengan algoritma C4.5, akan tetapi proses penambahan sample sintetis pada dataset di kelas minoritas yang dilakukan, menimbulkan *noisy attribute* pada dataset dan *overfitting* pada model yang dihasilkan, sehingga akurasi prediksi menjadi rendah

1.3 Pertanyaan Penelitian

Berdasarkan rumusan masalah, pertanyaan penelitian yang diangkat pada penelitian ini adalah:

- Bagaimana peningkatan akurasi prediksi *bug* software dengan algoritma C4.5, apabila masalah *overfitting* dan *noisy attribute* yang dihasilkan pada metode SMOTE diperbaiki dengan penambahan *feature selection* dengan metode *forward selection* dan *ensemble learning* dengan metode *bagging*?

1.4 Tujuan Penelitian

Tujuan proposal tugas akhir ini berdasarkan latar belakang yang telah diuraikan adalah sebagai berikut:

- Melakukan perbaikan metode SMOTE pada prediksi *bug* software dengan algoritma C4.5, dengan penambahan metode forward selection untuk menangani masalah ketidakrelevanan atribut dan metode bagging untuk menangani masalah overfitting pada model yang dihasilkan, sehingga akurasi prediksi *bug* software dapat meningkat.

1.5 Kontribusi Pengetahuan dari Penelitian

Kontribusi pengetahuan dari penelitian ini adalah pengembangan metode baru pada prediksi *bug* software dengan melakukan penggabungan metode algoritma C4.5 dengan pendekatan level data SMOTE serta pendekatan level algortima Bagging dengan seleksi fitur Forward Selection. Hasil penelitian ini diharapkan dapat memberikan kontribusi sebagai pengembangan atau perbaikan metode C4.5 dalam mengatasi ketidakseimbangan kelas serta ketidakrelevanan atribut pada prediksi *bug* software.

1.6 Manfaat Penelitian

Hasil penelitian ini dimanfaatkan untuk pengembangan metode prediksi *bug* software yang dapat digunakan untuk mempermudah pendeteksian *bug* pada software dalam masa pengembangan. Selain itu, penelitian ini diharapkan dapat memberikan sumbangan ilmu yang berhubungan dengan pengembangan dan penerapan model untuk menangani ketidakseimbangan kelas serta ketidakrelevanan atribut pada prediksi *bug* software.

1.7 Sistematika Penulisan

Pada tugas akhir ini terdapat lima bab dan setiap bab akan dibagi lagi menjadi beberapa subbab sesuai dengan topik yang dibahas. Sistematika pada penulisan ini adalah:

BAB 1 PENDAHULUAN

Pada bab ini akan diuraikan mengenai latar belakang masalah, rumusan masalah, pertanyaan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB 2 KAJIAN PUSTAKA

Pada bab ini akan dibahas mengenai kajian pustaka berupa penelitian terkait yang dilakukan peneliti lain dalam bidang prediksi *bug* software serta landasan teori.

BAB 3 METODOLOGI PENELITIAN

Pada bab ini akan dibahas mengenai metode penelitian berupa tahapan penelitian, analisis masalah dan tinjauan pustaka, pengumpulan dataset serta metode yang diusulkan dan pengujian metode.

BAB 4 HASIL DAN PEMBAHASAN

Pada bab ini akan dibahas mengenai hasil dari penelitian dan pembahasannya. Hasil pada bab ini akan menyajikan hasil dari kinerja metode yang diusulkan dan dibandingkan dengan metode sebelumnya.

BAB 5 KESIMPULAN DAN SARAN

Pada bab ini akan menyajikan kesimpulan dari hasil penelitian dan saran untuk penelitian lebih lanjut.