Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN: 2355-7699
Vol. 13, No. 1, Februari 2026, hlm. 11-20 e-ISSN: 2528-6579

PEMANFAATAN PEMBELAJARAN MESIN UNTUK KLASIFIKASI KEBUTUHAN PERANGKAT LUNAK

Wahyu Fajar Setiawan[*1], Ilham Putra Ariatama[2], Umi Laili Yuhana[3], Muhammad Alfian[4]

[1,2,3,4]Institut Teknologi Sepuluh Nopember, Surabaya
Email: [1]6025241020@student.its.ac.id, [*2]6025241001@student.its.ac.id, [3]yuhana@its.ac.id,
[4]7025221023@student.its.ac.id
[*]Penulis Korespondensi

(Naskah masuk: 02 Desember 2024, diterima untuk diterbitkan: 07 Mei 2025)

Abstrak

Klasifikasi kebutuhan perangkat lunak merupakan salah satu langkah terpenting dalam rekayasa perangkat lunak. Klasifikasi ini membantu pengembang untuk mengategorikan kebutuhan fungsional atau *functional requirement* (*FR*) dan kebutuhan non-fungsional atau *non-functional requirement* (*NFR*). Klasifikasi ini sangat penting untuk memastikan bahwa setiap aspek kebutuhan perangkat lunak terpenuhi dengan efisien sehingga perangkat lunak yang dikembangkan akhirnya dapat memenuhi harapan penggunanya. Namun, klasifikasi manual memerlukan waktu lama dan rentan terhadap kesalahan manusia, terutama pada proyek skala besar. Sehingga pada penelitian ini kami bertujuan mengotomatisasi proses klasifikasi kebutuhan perangkat lunak menggunakan beberapa algoritma pembelajaran mesin seperti *Logistic regression, SVM, Multinomial Naive Bayes, KNN, Random Forest*, dan *Decision Tree* dengan ekstraksi fitur seperti *TF-IDF, BoW,* dan *BERT* menggunakan *dataset PROMISE_exp* yang berisi *969* kebutuhan perangkat lunak *(*444 *FR* dan 525 *NFR)*, untuk mengetahui kombinasi terbaik antara metode ekstraksi fitur dengan algoritma klasifikasi. Hasil eksperimen menunjukkan bahwa *Logistic regression* dengan fitur *TF-IDF* menghasilkan akurasi tertinggi sebesar 97% di antara semua pendekatan. Model ini juga cukup seimbang dalam hal *precision, recall,* dan *F1-Score.* Model tersebut terbukti menjadi pilihan yang sangat andal untuk mengklasifikasikan kebutuhan perangkat lunak. *Decision Tree* yang dikombinasikan dengan *BERT* ternyata memiliki kinerja yang lebih buruk, yang menyatakan bahwa model ini kurang mampu menangani fitur-fitur yang kompleks dari *BERT*. Kontribusi utama penelitian ini adalah pembuktian empiris bahwa model klasifikasi sederhana (*Logistic Regression + TF-IDF*) dapat mengungguli pendekatan kompleks berbasis transformer (*BERT*) untuk tugas klasifikasi kebutuhan perangkat lunak, memberikan panduan praktis bagi pengembang dalam memilih pendekatan otomatisasi yang efektif dan efisien.
Kata kunci: *klasifikasi kebutuhan perangkat lunak, pembelajaran mesin, logistic regression, TF-IDF, BERT*

UTILIZING MACHINE LEARNING FOR SOFTWARE REQUIREMENTS CLASSIFICATION

Abstract

Software requirement classification is one of the most important steps in software engineering. This classification helps developers categorise functional requirements (FR) and non-functional requirements (NFR). This classification is very important to ensure that every aspect of software requirements is met efficiently so that the developed software can ultimately meet user expectations. However, manual classification is time-consuming and prone to human error, especially in large-scale projects. Therefore, in this study, we aim to automate the software requirement classification process using several machine learning algorithms such as Logistic regression, SVM, Multinomial Naive Bayes, KNN, Random Forest, and Decision Tree with feature extraction such as TF-IDF, BoW, and BERT using the PROMISE_exp dataset containing 969 software requirements (444 FR and 525 NFR), to determine the best combination of feature extraction methods with classification algorithms. The experimental results show that Logistic regression with TF-IDF features produces the highest accuracy of 97% among all approaches. This model is also quite balanced in terms of precision, recall, and F1-Score. The model proved to be a very reliable choice for classifying software requirements. Decision Tree combined with BERT turned out to have poorer performance, indicating that this model is less capable of handling the complex features of BERT. The main contribution of this research is the empirical proof that a simple classification model (Logistic Regression + TF-IDF) can outperform complex transformer-based approaches (BERT) for software requirement classification tasks, providing practical guidance for developers in choosing effective and efficient automation approaches.
Keywords: *software requirement classification, machine learning, logistic regression, TF-IDF, BERT*

1. PENDAHULUAN

Perangkat lunak meresap ke hampir semua aspek kehidupan kontemporer, dari sistem bisnis hingga perawatan kesehatan dan pendidikan. Pengembangan perangkat lunak sangat bergantung pada pemahaman dan pengelolaan yang tepat atas kebutuhan perangkat lunak yang diberikan oleh para pemangku kepentingan. Kebutuhan perangkat lunak secara umum dapat diklasifikasikan ke dalam dua kelas penting yaitu Kebutuhan Fungsional atau *functional requirement* (*FR)* dan Kebutuhan Non-Fungsional atau *non-functional requirement (NFR)*. *FR* mendefinisikan apa yang dimaksudkan untuk dilakukan oleh perangkat lunak, misalnya, memproses data atau memverifikasi identitas pengguna, sementara *N*FR mendefinisikan atribut kualitas yang berkaitan dengan keandalan, keamanan, dan kinerja yang memastikan bahwa perangkat lunak akan bekerja dengan baik untuk melayani tujuan yang dimaksudkan dalam lingkungannya (Ramadhani dkk, 2015).

Secara tradisional, *FR* dan *NFR* diklasifikasikan secara manual. Hal ini, sangat memakan waktu dan membutuhkan banyak sumber daya manusia. Untuk proyek besar yang melibatkan beberapa ribu baris kebutuhan, beberapa kesalahan manusia hampir tidak dapat dihindari. Dalam dua tahun terakhir, telah terjadi peningkatan eksponensial untuk mengatasi keterbatasan ini. Oleh karena itu, para peneliti mengeksplorasi pendekatan berbasis Pembelajaran Mesin (*ML*) untuk mengotomatisasi klasifikasi kebutuhan perangkat lunak (Zhao dkk, 2021). Algoritma *ML* telah menunjukkan hasil yang menjanjikan dalam tugas klasifikasi teks, yang dapat dioptimalkan lebih lanjut sehubungan dengan data yang tersedia. Otomatisasi klasifikasi meningkatkan efisiensi tidak hanya dalam proses pengembangan perangkat lunak tetapi juga dalam pengurangan biaya selama operasinya.

Berbagai studi penelitian mengeksplorasi berbagai pendekatan *ML* untuk klasifikasi *FR* dan *NFR*. Penelitian oleh (Baskoro dkk, 2021) mengklasifikasikan kebutuhan perangkat lunak dengan menggunakan kombinasi *BoW* dan *SVM*, yang memperoleh akurasi yang relatif tinggi. Algoritma lain seperti *Logistic Regression*, *Random Forest*, *Multinomial Naive Bayes*, dan *Decision Tree* juga telah dicoba pada beberapa kumpulan data yang tersedia untuk umum seperti *PROMISE_exp* untuk memeriksa kinerjanya pada tugas klasifikasi persyaratan pada penelitian sebelumnya (Al-fraihat dkk, 2024), (Binkhonain dan Zhao, 2019) dan (Canedo dan Mendes, 2020). Teknik ekstraksi fitur ini, seperti *TF-IDF* dan *BoW*, telah menunjukkan hasil yang menjanjikan dengan model seperti *Decision Tree* dan *KNN* dengan memberikan kinerja yang baik dengan efisiensi komputasi yang wajar (Umar dkk, 2023).

Namun, metode *ML* ini masih memiliki beberapa celah. Sebagian besar penelitian sebelumnya memfokuskan studi mereka pada ruang ekstraksi fitur kecil atau satu model pembelajaran mesin tunggal, yang dapat membuat model ini kurang efektif dalam menangani kumpulan data yang kompleks. Lebih jauh, beberapa studi sebelumnya telah mengeksplorasi penggunaan beberapa algoritma pembelajaran mesin secara massal dalam mengklasifikasikan kebutuhan perangkat lunak, meskipun hal itu dapat membantu memperkuat akurasi dan stabilitas klasifikasi.

Oleh karena itu, penelitian ini mengusulkan model yang membandingkan beberapa model *ML* dengan beragam teknik ekstraksi fitur untuk representasi terbaik dari kebutuhan perangkat lunak. Berbagai model pembelajaran mesin yang digunakan dalam studi tersebut meliputi *Logistic Regression, Random Forest, Multinomial Naive Bayes, KNN, SVM, dan Decision Tree.* Teknik ekstraksi fitur yang digunakan adalah *TF-IDF, BoW,* dan *BERT*, sedangkan kumpulan data *PROMISE_exp* berfungsi sebagai bahan uji utama. Tujuan dari penelitian ini adalah untuk berkontribusi terhadap otomatisasi klasifikasi kebutuhan perangkat lunak sehingga mempercepat siklus hidup pengembangan perangkat lunak tercapai tanpa mengorbankan kualitas

2. TINJAUAN PUSTAKA

Penelitian dalam mengklasifikasikan kebutuhan perangkat lunak telah berkembang pesat seiring dengan meningkatnya minat dalam otomatisasi proses rekayasa kebutuhan menggunakan teknik pembelajaran mesin. Di sini akan dibahas dalam dua kategori utama penelitian terkait: metode ekstraksi fitur yang digunakan dalam pemrosesan data teks kebutuhan dan algoritma klasifikasi yang digunakan untuk memprediksi kelas kebutuhan perangkat lunak, baik *FR* maupun *NFR*.

2.1. Penelitian Terkait

Salah satu tugas utama dalam rekayasa perangkat lunak melibatkan pengklasifikasian kebutuhan perangkat lunak ke dalam *FR* dan *NFR*, yang memungkinkan pengembang untuk memastikan masalah fungsional dan kualitas perangkat lunak yang sedang dikembangkan. Biasanya, klasifikasi manual merepotkan dan rawan kesalahan, terutama pada proyek besar. Tantangan semacam itu telah memotivasi para peneliti untuk menyelidiki berbagai pendekatan pembelajaran mesin yang mengklasifikasikan kebutuhan perangkat lunak ke dalam *FR* dan *NFR*.

(Canedo dan Mendes, 2020) melakukan studi perbandingan tentang metode ekstraksi fitur teks dan algoritma *ML* untuk klasifikasi kebutuhan perangkat lunak. Para penulis menguji *BoW, TF-IDF*, dan *CHI2* yang dikombinasikan dengan algoritma *LR, SVM, MNB,* dan *KNN*. Mereka melaporkan bahwa *TF-IDF* yang dikombinasikan dengan *LR* menghasilkan

kinerja klasifikasi terbaik, yang mencapai *F-measure* sebesar 0,91 dalam tugas klasifikasi biner.

Dalam studi yang lebih baru, (Althunibat dkk, 2023) mengembangkan skema kategorisasi kebutuhan perangkat lunak yang selanjutnya disempurnakan dengan pendekatan *ML* pada kumpulan data *Arabic PROMISE*. Mereka mencoba algoritma *ML* standar, model pembelajaran mendalam, dan model berbasis *transformer* dengan menggunakan protokol pemrosesan bahasa rumit yang sesuai untuk bahasa Arab. Penelitian mereka menggarisbawahi seberapa efektif model *transformer* dalam menangani struktur linguistik rumit yang ada dalam kebutuhan perangkat lunak.

Namun, (Baskoro dkk, 2021) mengevaluasi performa algoritma *ML* dan teknik *vectorization* teks dalam mengklasifikasikan kebutuhan perangkat lunak berdasarkan *dataset* yang relevan. Studi ini menyoroti pentingnya pemilihan *dataset* yang tepat dan metode *preprocessing* yang sesuai untuk meningkatkan akurasi klasifikasi.

(Handa, Sharma dan Gupta 2022) juga mengeksplorasi penggunaan fitur semantik pada pernyataan kebutuhan perangkat lunak untuk tujuan klasifikasi. Dengan menggunakan pengetahuan linguistik dan teknik *ML*, makalah ini berupaya memberikan makna yang efektif untuk meningkatkan akurasi pengklasifikasian atribut dalam sistem perangkat lunak sehubungan dengan faktor kualitas.

Semua penelitian ini bersama-sama menunjukkan kelayakan teknik *ML* dalam mengotomatiskan klasifikasi kebutuhan perangkat lunak. Namun, masalah masih tetap ada terkait aspek-aspek seperti kemampuan beradaptasi model di seluruh bahasa dan metode ekstraksi fitur tingkat lanjut. Dalam konteks ini, makalah ini mencoba memenuhi upaya awal tersebut dengan membandingkan berbagai model *ML* dengan berbagai teknik ekstraksi fitur, yaitu *TF-IDF, BoW,* dan *BERT*, untuk mengklasifikasikan kebutuhan perangkat lunak secara optimal.

2.2. Ekstraksi Fitur

Ekstraksi fitur merupakan salah satu tugas paling mendasar dalam *Natural Language Processing* (*NLP)*, dan memiliki dampak yang mendalam pada kinerja model klasifikasi. Selama klasifikasi kebutuhan perangkat lunak, fitur yang diekstraksi dari teks kebutuhan perangkat lunak yang sering digunakan untuk mewakili pengetahuan utama mengenai kategori *FR* dan *NFR*. Beberapa metode ekstraksi fitur umum diterima secara luas dalam literatur:

Term* Fr*equency-Inverse Document* Fr*equency (TF-IDF)

TF IDF merupakan metodologi ekstraksi fitur yang didasarkan pada pentingnya kata yang ditentukan berlawanan dengan jumlah kata yang ada di seluruh dokumen dalam korpus tertentu. Memang, teknik ini telah banyak digunakan dalam sebagian besar klasifikasi teks karena kemudahan dalam menghitung bobot untuk kata-kata yang lebih relevan. Karya-karya sebelumnya, seperti yang dilakukan oleh (Khurshid dkk, 2022) dan (Rashme, 2024), telah dieksekusi menggunakan *TF-IDF* bersama dengan algoritma klasifikasi seperti *Support Vector Machine* dan *K-Nearest Neighbors* untuk klasifikasi kebutuhan perangkat lunak. Mungkin kelemahan terpenting dari *TF-IDF* adalah ketidakmampuannya untuk menangkap konteks semantik tentang makna kata-kata yang diungkapkan dalam teks (Dewi dkk, 2023).

Bag of Words (BoW)

Bag of words merupakan salah satu teknik representasi yang menyederhanakan dokumen menjadi kumpulan kata-kata unik yang tidak berurutan. Dalam *BoW*, setiap kata tunggal direpresentasikan sebagai satu vektor tunggal dan vektor-vektor ini digunakan untuk melatih model pembelajaran mesin. (Al-fraihat dkk, 2024) dan (Umar dkk, 2023) menunjukkan bahwa *BoW*, meskipun memberikan kinerja yang dapat diterima dalam kasus-kasus tertentu, memiliki kemampuan yang sangat terbatas untuk merepresentasikan hubungan kontekstual antar kata. Akibatnya, *BoW* sering kali gagal memberikan kinerja yang optimal untuk dokumen teks yang kompleks, seperti dokumen spesifikasi kebutuhan perangkat lunak.

Bidirectional Encoder Representations* fr*om Transformers (BERT)

Dalam beberapa tahun terakhir di antara metode ekstraksi fitur yang didasarkan pada arsitektur Transformer, *BERT* mungkin baru-baru ini muncul untuk merepresentasikan semacam standar bagi banyak tugas *NLP*. *BERT* berbeda dari *TF-IDF* dan *BoW*, misalnya, dengan mempertimbangkan setiap kata dalam kalimat dalam konteks penuh, tidak hanya melihat bagian kiri tetapi juga bagian kanan kata tersebut. (Canedo dan Mendes, 2020), (Subahi, 2023) dan (Hassan dkk, 2024) telah menunjukkan dalam karya mereka bahwa *BERT* meningkatkan kinerja klasifikasi kebutuhan perangkat lunak dibandingkan dengan metode ekstraksi fitur lainnya. Faktanya, penggunaan *BERT* memberi model pembelajaran mesin kemampuan untuk menangkap konteks semantik yang lebih dalam, sangat relevan dalam bekerja dengan teks kebutuhan perangkat lunak yang sering kali ambigu.

2.3. Algoritma Klasifikasi

Selain metode ekstraksi fitur, algoritma klasifikasi memegang peranan penting dalam menentukan akurasi model dalam memprediksi kelas kebutuhan perangkat lunak. Sejumlah algoritma klasifikasi telah dieksplorasi dalam penelitian terkait, seperti:

Support Vector Machine (SVM)

SVM merupakan salah satu algoritma klasifikasi terkenal yang paling populer dan digunakan dalam klasifikasi teks, termasuk

klasifikasi kebutuhan perangkat lunak. Algoritma ini pada dasarnya bekerja dengan mencari *hyperplane* antara dua kelas data yang mampu memisahkan kelas data tersebut dengan margin maksimum. Penggunaan *SVM* dengan *TF-IDF* dieksplorasi dalam sebuah penelitian oleh (Umar dkk, 2023) dan (Rahman dkk, 2023) yang mencoba mengklasifikasikan kebutuhan perangkat lunak ke dalam kelas FR dan NFR dengan akurasi yang sangat baik. Keunggulan utama *SVM* adalah efisiensinya dalam *dataset* berdimensi tinggi; namun, algoritma ini memiliki kelemahan terkait efisiensi komputasi pada *dataset* yang sangat besar.

Logistic Regression (LR)

LR merupakan algoritma klasifikasi yang paling umum digunakan dalam pemodelan hubungan antara variabel independen dan variabel dependen biner. Dalam konteks masalah klasifikasi kebutuhan perangkat lunak, *LR* dapat dianggap sebagai dasar yang dapat digunakan untuk membandingkan kinerja algoritma yang lebih kompleks. Teknik *BoW* telah digabungkan dengan algoritma *LR*, untuk klasifikasi kebutuhan perangkat lunak. (Umar dkk, 2023) menunjukkan bahwa meskipun algoritma ini cukup sederhana, *LR* menghasilkan hasil yang kompetitif.

Random Forest (RF)

RF adalah algoritma berbasis *ensemble*, *RF* mengembangkan sejumlah besar pohon keputusan dan menggunakan hasil prakiraannya untuk membuat prediksi akhirnya. Bahkan, (Al-Fraihat dkk, 2024) memberikan presentasi bahwa *RF* berjalan sangat baik dalam menangani kumpulan data variabel dan tidak seimbang saat mengoperasikan tugas klasifikasi kebutuhan perangkat lunak. Keuntungan dari algoritma ini adalah dalam menangani masalah *overfitting* dan menunjukkan kinerja yang baik pada kumpulan data berdimensi tinggi. Namun, kelemahan utama untuk beberapa kasus adalah waktu pelatihan yang lebih lama dibandingkan dengan beberapa algoritma lainnya.

Multinomial Naive Bayes (MNB)

Naive Bayes adalah algoritma *probabilistik* berdasarkan teorema Bayes. Versi *multinomial* sangat sesuai dengan kebutuhan tugas klasifikasi teks. Faktanya, sangat jelas dari penelitian yang dilakukan oleh (Abdulmajeed and Younis 2021) dan (Khurshid dkk, 2022) menunjukkan bahwa *Multinomial Naive Bayes (MNB)* bekerja dengan baik dalam tugas klasifikasi kebutuhan *NFR* berdasarkan teks spesifikasi. Meskipun kesederhanaan MNB merupakan aset yang hebat, ia tidak memiliki fungsionalitas untuk menangani fitur yang sangat bergantung, yang sering kali menjadi salah satu masalah dalam dokumen kebutuhan perangkat lunak yang kompleks.

K-Nearest Neighbors (KNN)

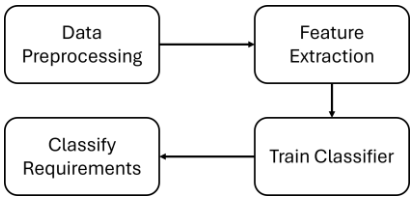
Dalam algoritma berbasis *instance*, *KNN* mengklasifikasikan data baru tergantung pada suara mayoritas oleh *K nearest neighbors*. Penelitian oleh (Canedo dan Mendes, 2020) menyatakan bahwa *KNN* dapat bekerja dengan baik dalam kumpulan data berukuran kecil atau sedang, sementara ia berkinerja buruk ketika diterapkan pada kumpulan data berukuran besar. Selain itu, *KNN* memiliki kelemahan dalam efisiensi komputasi karena setiap prediksi perlu menghitung jarak ke seluruh data pelatihan.

Decision Tree (DT)

DT adalah algoritma klasifikasi yang membuat bagian-bagian, pada kumpulan data tertentu, yang dipisahkan sebisa mungkin menurut beberapa metrik, biasanya *entropi* atau indeks *Gini*. *DT* cukup intuitif untuk dipahami dan mengalami *overfitting* jika pohon tersebut tumbuh terlalu dalam. (Althunibat dkk, 2023) *DT* berkinerja kompetitif untuk klasifikasi kebutuhan perangkat lunak yang dikombinasikan dengan metode regularisasi, termasuk pemangkasan.

3. METODE PENELITIAN

Metodologi yang digunakan dalam penelitian ini bertujuan untuk mengotomatisasi klasifikasi kebutuhan perangkat lunak. Meliputi detail kumpulan data yang digunakan, teknik *preprocessing*, metode ekstraksi fitur, algoritma klasifikasi, dan metrik pengukuran kinerja untuk menentukan efektivitas setiap model. Metode penelitian yang digunakan pada penelitian ini dapat dilihat pada gambar 1.



Gambar 1. Metode Penelitian.

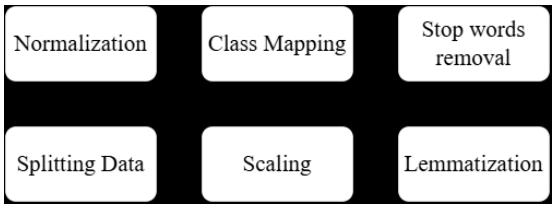
3.1. *Dataset*

Penelitian ini menggunakan *dataset PROMISE_EXP* dari penelitian (Lima dkk, 2019) yaitu versi yang diperluas dari *dataset TERAPROMISE* yang tersedia secara publik. *Dataset* asli berisi 250 *FR* dan 370 *NFR*, sehingga totalnya menjadi 625 kebutuhan. Dalam versi *PROMISE_EXP* yang diperluas, jumlah *FR* meningkat menjadi 444 dan *NFR* menjadi 525, dengan total keseluruhan 969 kebutuhan.

3.2. *Preprocessing*

Tahap *preprocessing* dilakukan untuk memastikan bahwa data siap digunakan oleh algoritma pembelajaran mesin. Pada penggunaan

dataset ada beberapa fitur yang dipakai pada tahap ini seperti *RequirementText* dan *_class_*.



Gambar 2. Langkah-langkah *preprocessing* kebutuhan.

Langkah-langkah yang diterapkan secara rinci pada Gambar 2. sebagai berikut.

1. *Normalization*: Semua huruf diubah menjadi huruf kecil untuk konsistensi. Hapus tanda baca dan simbol yang tidak berguna, menggunakan pustaka *string Python*.
2. *Class Mapping*: Kelas dipetakan ke *FR* atau *N*FR dari kategori asli, seperti *PE (Performance), A (Availability), LF (Look and feel), L (Legal), O (Operability), FT (Fault Tolerance), US (Usability), PO (Portability), MN (Maintainability), SE (Scurity), SC (Scalability)* yang dipetakan dalam kelas *NFR* sedangkan F (Functional) masuk kedalam kelas *FR.*
3. *Stop words removal:* Melalui pustaka *nltk*, ia menghapus kata-kata umum yang tidak membawa informasi berguna seperti *"the", "and", "is",* dll.
4. *Lemmatization*: Ini melibatkan pengurangan variasi kata dengan mengubahnya ke bentuk dasarnya. Misalnya, kata "*running*" diubah menjadi "*run*".
5. *Splitting Data*: Seluruh data akan dibagi menjadi dua bagian: 80% akan menjadi bagian dari data pelatihan dan 20% dari data akan menjadi bagian dari data pengujian. Fungsi *train_test_split* berfungsi sebagai cara yang adil untuk memperkirakan kinerja generalisasi suatu model dengan membagi data menjadi kumpulan data pelatihan dan pengujian.
6. *Scalling*: Penskalaan fitur numerik sehingga semua fitur berada dalam rentang yang seragam (biasanya setelah data diubah ke bentuk *numerik*).

3.3. *Feature Extraction*

Pada penelitian ini menggunakan beberapa metode fitur ekstraksi yaitu *TF-IDF, BoW,* dan *BERT*.

Term* frequency-*Inverse Document* frequency *(TF-IDF)

Rumus untuk menghitung *TF-IDF* terdiri dari dua komponen utama: *Term frequency (TF)* dan *Inverse Document frequency (IDF*).

TF adalah frekuensi kata dalam dokumen, yang dapat dihitung menggunakan rumus:

$$TF_{i,j} = \frac{f_{i,j}}{n_j} \quad (1)$$

Di mana :

$f_{i,j}$: Jumlah kemunculan kata i dalam dokumen j.

n_j : Jumlah total kata dalam dokumen j.

IDF mengukur seberapa sering atau jarangnya sebuah kata muncul dalam dokumen. Rumusnya adalah:

$$IDF_i = log(\frac{n}{d_i}) \quad (2)$$

Di mana :

n : Jumlah total dokumen dalam korpus.

d_i : Jumlah dokumen yang berisi kata i

Setelah menghitung *TF* dan *IDF*, nilai *TF-IDF* dapat diperoleh dengan mengalikan kedua komponen:

$$TF - IDF_{i,j} = TF_{i,j} \times IDF_i \quad (3)$$

TF-IDF memberi bobot lebih tinggi pada kata-kata yang sering muncul dalam suatu dokumen, tetapi jarang muncul di dokumen lain, sehingga lebih informatif untuk klasifikasi.

Untuk mengimplementasikan pendekatan TF-IDF, kita perlu mengimpor *TfidfVectorizer* dari pustaka *sklearn.feature_extraction.text*, yang digunakan untuk mengubah teks menjadi representasi numerik berupa matriks *TF-IDF*, yang mempertimbangkan frekuensi kata dalam dokumen serta seberapa penting kata tersebut dalam seluruh korpus.

Bag of Words (BoW)

Pendekatan *BoW* merupakan salah satu pendekatan yang paling sederhana. Dalam pendekatan ini, teks direpresentasikan sebagai vektor berdasarkan jumlah frekuensi kata dalam sebuah dokumen. Dengan asumsi bahwa terdapat *M* kata unik dalam sebuah korpus, setiap dokumen akan direpresentasikan sebagai vektor berdimensi *M*, di mana setiap elemen dapat merepresentasikan jumlah frekuensi kata tertentu.

Karena teknik ini hanya menghitung frekuensi kemunculan setiap kata, tidak ada rumus khusus untuk *BoW*. Vektor *BoW* untuk sebuah dokumen dapat ditulis sebagai:

$$BoW = [f_1, f_2 \ldots\ldots f_m] \quad (4)$$

Di mana f_1 adalah jumlah kemunculan kata i dalam dokumen.

Untuk mengimplementasikan pendekatan ini, kita perlu mengimpor *CountVectorizer* dari pustaka *sklearn.feature_extraction.text*, yang digunakan untuk mengubah teks menjadi representasi numerik berupa matriks frekuensi kata dalam setiap dokumen.

Bidirectional Encoder Representations* from *Transformers (BERT)

Representasi *BERT* memiliki rumus formal yang sulit dinyatakan secara langsung karena *BERT* diimplementasikan dengan jaringan saraf berlapis-

lapis, bersama dengan mekanisme perhatian yang sangat rumit. Namun, secara umum, proses tersebut dapat diringkas dengan rumus berikut:

$$E_1 = BERT(T_1) \quad (5)$$

Di mana :

T_1 : Token i yang telah melalui proses tokenisasi.

E_1 : Representasi vektor dari token i, yang dihasilkan oleh model *BERT*.

Vektor penyisipan yang dihasilkan dapat digunakan untuk mengukur kesamaan antara dokumen atau untuk *input* ke model pembelajaran mesin lainnya.

Untuk mengimplementasikan pendekatan *BERT*, kita perlu mengimpor model *BERT* dari pustaka seperti transformers milik Hugging Face. *BERT* digunakan untuk menghasilkan representasi teks dalam bentuk vektor yang lebih kaya, dengan mempertimbangkan konteks kata dalam sebuah kalimat, bukan hanya frekuensi kata.

3.4. Algoritma Klasifikasi

Langkah ini akan terdiri dari pelatihan dan evaluasi model pembelajaran mesin menggunakan berbagai algoritma klasifikasi seperti:

1. Pemilihan Algoritma

   Multinomial Naive Bayes (MNB), Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF), K-Nearest Neighbors (KNN), dan *Decision Tree (DT)* merupakan algoritma yang digunakan. Pada masing-masing algoritma menggunakan parameter bawaan yang sudah diatur dari *library scikit-learn,*.

2. Penggunaan *Library/Tools*

   Pada penelitian ini, implementasi algoritma pembelajaran mesin dilakukan menggunakan bahasa pemrograman *Python* versi 3.12.7 dengan beberapa pustaka berikut:

   a. *Scikit-learn* versi 1.6.1 untuk implementasi algoritma klasifikasi tradisional.
   b. *Transformers* versi 4.51.3 untuk implementasi model *BERT.*
   c. *Pandas* versi 2.2.3 untuk pengolahan data.
   d. *NumPy* versi 2.2.5 untuk operasi numerik.
   e. *NLTK* versi 3.9.1 untuk *preprocessing* teks.
   f. *Matplotlib* versi 3.10.1 dan *Seaborn* versi 0.13.2 untuk visualisasi data.

3. Pelatihan Model

   Setiap model akan dilatih dengan data pelatihan yang telah diproses sebelumnya oleh berbagai metode ekstraksi fitur: *TF-IDF, BoW,* dan *BERT*.

4. Prediksi dan Evaluasi

   Prediksi untuk data uji dibuat dengan menggunakan model yang telah dilatih, sementara label aktual digunakan untuk evaluasi kinerja dalam hal *accuracy, precision, recall,* dan *F1-score.*

5. Visualisasi Hasil

   Confusion matrix digunakan untuk menampilkan distribusi hasil klasifikasi, yang membantu dalam pemahaman kesalahan yang dilakukan.

3.5. Parameter Kinerja

Dalam mengukur kinerja dari penelitian yang telah dilakukan menggunakan parameter sebagai berikut :

1. *Confusion matrix* Untuk masalah klasifikasi, dan ini adalah matriks hasil kelas aktual dan kelas yang diprediksi. Matriks ini dibuat seperti pada Tabel 1.

Tabel 1. Confusion matrix

		Predicted Class	
		P	*N*
Actual Class	*P*	*True Positive (TP)*	*False Negative (FN)*
	N	*False Positive (FP)*	*True Negative (TN)*

TP : Nilai benar diprediksi sebagai positif
TN: Nilai benar diprediksi sebagai negatif
FP: Nilai salah diprediksi sebagai positif
FN: Nilai salah diprediksi sebagai negatif

2. *Accuracy* untuk proporsi pengamatan yang diklasifikasikan dengan benar adalah yang dimaksud dengan akurasi. Karena menentukan jumlah total kategori yang akurat, maka ini adalah ukuran yang lebih luas. Parameter ini terbukti menjadi faktor penentu yang baik ketika kelas target seimbang.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (6)$$

3. *Precision* untuk melihat tingkat ketepatan sebagai proporsi sampel yang diidentifikasi dengan benar relatif terhadap semua sampel. Rasio total klasifikasi yang benar terhadap total klasifikasi yang dilakukan digunakan untuk menghitung presisi.

$$Precision = \frac{TP}{TP+FP} \quad (7)$$

4. *Recall* akan menghitung persentase kasus positif yang diidentifikasi dengan benar. Nilai ini diperoleh dengan membagi persentase berapa kali kelas benar-benar diidentifikasi dengan persentase berapa kali kelas tersebut muncul dalam data uji.

$$Recall = \frac{TP}{TP+FN} \quad (8)$$

5. *F1-score* untuk klasifikasi cepat, menggabungkan *recall* dan *precision* menjadi metrik yang disebut *F1-score*, terbukti menguntungkan. *F1-score* adalah rata-rata harmonik dari *recall* dan presisi. Nilai yang rendah diberi bobot yang jauh lebih besar oleh rata-rata harmonik daripada oleh rata-rata konvensional, yang memberi bobot yang sama pada semua nilai lainnya. Akibatnya, untuk mendapatkan *F1-score* yang lebih baik, presisi beserta *recall* harus memiliki nilai yang lebih tinggi.

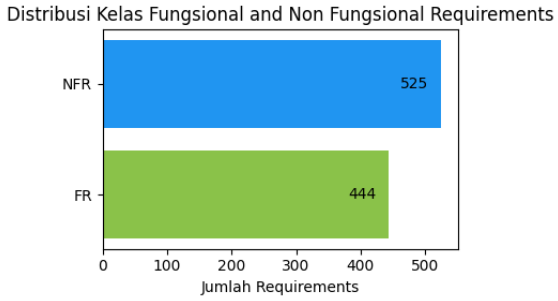
$$F1 - score = \frac{2 \times precision \times Recall}{precision + Recall} \quad (9)$$

4. HASIL DAN PEMBAHASAN

Dataset PROMISE_XP memiliki 2 kebutuhan perangkat lunak yaitu *FR* yang diidentifikasikan dengan kode *"F"* sementara *N*FR diidentifikasi dengan beberapa jenis kebutuhan sesuai dengan Tabel 2. Distribusi dari setiap kelas ini telah disajikan dalam Gambar 3.

Tabel 2. Distribusi setiap kelas

Kelas	Jenis Kebutuhan	Jumlah
FR	*F (Functional)*	444
NFR	*PE (Performance)*	67
	A (Availability)	31
	LF (Look and feel)	49
	L (Legal)	15
	O (Operability)	77
	FT (Fault Tolerance)	18
	US (Usability)	85
	PO (Portability)	12
	MN (Maintainability)	24
	SE (Scurity)	125
	SC (Scalability)	22



Gambar 3. Distribusi kelas Fungsional dan Non Fungsional

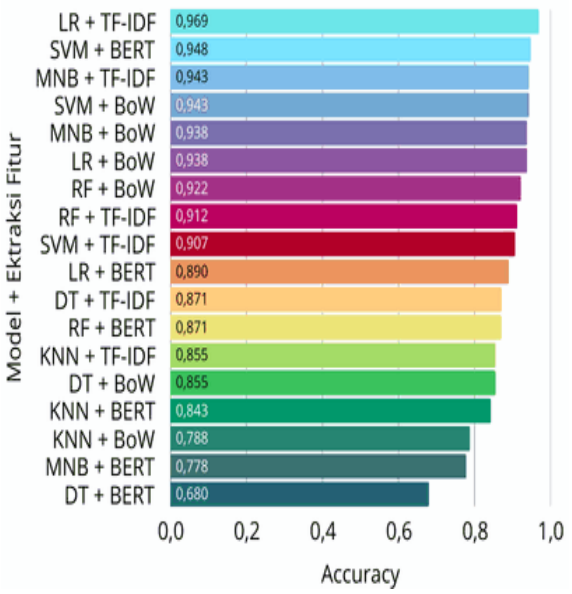
Untuk menyediakan data yang dapat diproses pada berbagai algoritma klasifikasi. pada *dataset* ini dilakukan ekstraksi fitur di mana data yang sebelumnya data *text* diubah menjadi data vektor menggunakan metode *Bag of Words* (BoW), Term frequency-Inverse *Document frequency (TF-IDF),* dan *Bidirectional Encoder Representations* from *Transformers (BERT)*.

Setelah proses ekstraksi fitur dilakukan, kami menerapkan berbagai metode klasifikasi, yaitu *Logistic Regression* (*LR*), *Support Vector Machine* (*SVM*), *Random Forest (RF), Multinomial Naive Bayes (MNB), K-Nearest Neighbors (KNN),* dan *Decision Tree (DT)*. Evaluasi terhadap *dataset PROMISE_EXP* dengan pembagian 80% data latih dan 20% data uji disajikan pada Tabel 3 dan Gambar 4 yang menyajikan metrik evaluasi untuk setiap metode ekstraksi fitur dan kinerja model klasifikasi.

Table 2. Performa Seluruh Model

Model	Feature	Acc	P	R	F1
DT	BERT	0.680	0.677	0.663	0.670
SVM	**BERT**	**0.948**	**0.938**	**0.957**	**0.947**
KNN	BERT	0.843	0.810	0.905	0.854
LR	BERT	0.890	0.893	0.884	0.888
MNB	BERT	0.778	0.758	0.800	0.777
RF	BERT	0.871	0.914	0.805	0.857
DT	BoW	0.855	0.852	0.852	0.852
KNN	BoW	0.788	0.780	0.810	0.794
LR	BoW	0.938	0.947	0.926	0.936
SVM	**BoW**	**0.943**	**0.956**	**0.926**	**0.941**
MNB	BoW	0.938	0.919	0.957	0.938
RF	BoW	0.922	0.928	0.891	0.919
RF	TF-IDF	0.912	0.953	0.868	0.908
LR	**TF-IDF**	**0.969**	**0.978**	**0.957**	**0.968**
KNN	TF-IDF	0.855	0.852	0.852	0.852
DT	TF-IDF	0.871	0.833	0.852	0.866
MNB	TF-IDF	0.943	0.922	0.957	0.940
SVM	TF-IDF	0.907	0.935	0.868	0.900

Oleh karena itu, hasil ini relevan dan disajikan pada Tabel 3, yang menunjukkan kinerja *Logistic Regression* yang lebih unggul dibandingkan dengan model lainnya. *Logistic Regression* unggul karena bekerja dengan cara memodelkan hubungan linier antara fitur dan probabilitas kelas. Model ini menghitung kemungkinan kelas berdasarkan bobot yang diberikan ke setiap fitur, yang memungkinkan LR untuk menangani data dengan dimensi tinggi secara efisien. Kemampuannya untuk menghindari overfitting, terutama dengan data yang besar atau rumit, membuatnya tetap stabil dan menghasilkan prediksi yang akurat. Untuk memudahkan pemahaman mengenai model terbaik, penelitian ini juga menampilkan urutan pada Gambar 4.



Gambar 4. Visualisasi Hasil Perbandingan Model berdasarkan Akurasi

Pada Tabel 3 menampilkan metrik evaluasi hasil klasifikasi dari *confusion matrix* yang dihasilkan melalui penggunaan berbagai algoritma. Tabel ini memberikan gambaran tentang seberapa baik setiap algoritma dalam mengklasifikasikan data dengan akurat. Metode *Logistic Regression* (*LR*) dengan ekstraksi fitur *TF-IDF* menunjukkan tingkat keandalan tinggi dalam klasifikasi, dengan akurasi yang jauh lebih baik dibandingkan algoritma lain. Oleh karena itu, hasil dari algoritma ini dijadikan sebagai acuan untuk evaluasi lebih lanjut. Untuk contoh, *confusion matrix* dari *Logistic Regression* dengan ekstraksi fitur *TF-IDF* dapat dilihat pada Tabel 4.

Table 3. Distribusi *confusion matrix*

	FR	*NFR*
FR	91	4
NFR	2	97

Logistic Regression mampu memberikan hasil kinerja yang cukup baik dalam melakukan klasifikasi kebutuhan perangkat lunak. Peforma setiap kelasnya dapat dilihat pada *classification report* yang disajikan pada Tabel 5.

Table 5. Performa model LR

	Precision	Recall	F1-score
FR	0.98	0.96	0.97
NFR	0.96	0.98	0.97

Performa model *Logistic Regression* memperlihatkan terdapat adanya kesalahan dalam melakukan klasifikasi yang disajikan pada Tabel 4. Dimana *Confusion matrix* pada model *Logistic Regression* memperlihatkan terdapat kelas *Functional Requirements* dan *Non-functional Requirements* yang salah diklasifikasikan sesuai kelasnya. Tabel 6 memperlihatkan *text requirements* yang salah diklasifikasikan sesuai kelasnya.

Table 6. *Requirement Text* yang salah diklasifikasikan

No	Requirement Text	Actual	Predicted
1	*system search feature support use regular expression search.*	*FR*	*NFR*
2	*dispute application shall interface letter application allow dispute application request letter part dispute initiation dispute follow process letter request must sent print letter utility application.*	*NFR*	*FR*
3	*system function controlled using administrative interface.*	*FR*	*NFR*
4	*product shall display grid within circle view periscope.*	*NFR*	*FR*
5	*system shall allow text search user may use find mail message.*	*FR*	*NFR*
6	*view available product compare make choice purchasing product.*	*FR*	*NFR*

Kesalahan Kesalahan pengklasifikasian terhadap *requirement text* pada Tabel 6 dapat terjadi dikarenakan keterbatasan kata kunci yang dapat diklasifikasikan oleh model. Sebagai contoh, jika sebuah kalimat pada Tabel 6 seharusnya diklasifikasikan sebagai FR (*Functional Requirement*), namun diklasifikasikan sebagai NFR (*Non-Functional Requirement*), maka kesalahan ini menunjukkan bahwa kata kunci yang ada dalam kalimat tersebut tidak dikenali atau dipetakan dengan benar oleh model ke kelas yang sesuai. Salah satu solusi yang efektif untuk meningkatkan akurasi klasifikasi teks kebutuhan perangkat lunak adalah melalui peningkatan data pelatihan. Dengan menambahkan lebih banyak contoh untuk *FR* dan *NFR* terutama yang mencakup variasi kalimat dan penggunaan kata kunci yang beragam, model dapat belajar lebih baik dalam menangkap pola perbedaan antara kedua kelas. Penambahan data yang seimbang dan representatif memungkinkan model memiliki referensi yang cukup untuk memahami konteks kata-kata ambigu dan struktur kalimat yang berbeda.

5. KESIMPULAN DAN SARAN

Klasifikasi kebutuhan perangkat lunak sangat krusial dalam pengembangan perangkat lunak karena membantu memastikan bahwa semua kebutuhan pengguna, baik fungsional maupun non-fungsional, diidentifikasi dan dipenuhi dengan tepat. Penelitian ini memanfaatkan algoritma *machine learning* untuk mengurangi potensi kesalahan klasifikasi, dan meminimalkan risiko tidak terpenuhinya kebutuhan pengguna. Metode klasifikasi menggunakan berbagai algoritma *machine learning,* seperti *Logistic Regression*, *Support Vector Machine*, *Random Forest*, *Multinomial Naive Bayes*, *K-Nearest Neighbors*, dan *Decision Tree*. Dikombinasikan dengan fitur ekstraksi *BoW*, *TF-IDF*, dan *BERT* memberikan hasil akurasi yang cukup baik di mana

Logistic Regression digabungkan dengan *TF-IDF* mendapatkan hasil terbaik dengan akurasi 97%.

Tingkat akurasi klasifikasi pada penelitian ini menunjukkan bahwa *machine learning* dapat digunakan untuk melakukan otomatisasi pada proses klasifikasi kebutuhan perangkat lunak. Sehingga dapat dimanfaatkan untuk sehingga meminimalkan kesalahan dan mengurangi waktu yang dihabiskan untuk klasifikasi manual pada proyek perangkat lunak berskala besar.

Penelitian ini dapat menjadi referensi untuk otomatisasi klasifikasi kebutuhan perangkat lunak. Pengembangan lebih lanjut dapat menggunakan dataset lebih besar dan beragam, serta mengeksplorasi metode seperti *Ensemble* dan *deep learning* untuk meningkatkan akurasi, terutama pada proyek dengan kebutuhan kompleks.

DAFTAR PUSTAKA

AL-FRAIHAT, D., SHARRAB, Y., AL-GHUWAIRI, A.R., ALZABUT, H., BESHARA, M. & ALGARNI, A., 2024, *'Utilizing machine learning algorithms for task allocation in distributed agile software development', Heliyon,* 10(21), 1–12.

ALTHUNIBAT, A., AMRO, R., HAWASHIN, B., ALNUHAIT, H., ALMANASRA, S. & AL-KHAWAJA, H.A., 2023, *'Automated Classification of User Requirements written in Arabic using machine learning algorithms', Applied Mathematics and Information Sciences,* 17(6), 1155–1170.

RAMADHANI, D.A., ROCHIMAH, S. & YUHANA, U.L., 2015. *Classification of non-functional requirements using semantic-Fsknn based ISO/IEC 9126*. *TELKOMNIKA*, 13(4), pp.1180–1188.

ZHAO, L., ALHOSHAN, W., FERRARI, A., LETSHOLO, K. J., AJAGBE, M. A., CHIOASCA, E.-V., & BATISTA-NAVARRO, R. T., 2022. *Natural Language Processing for Requirements Engineering. ACM Computing Surveys*, *54*(3), 1–41. doi:10.1145/3444689

BASKORO, F., ANDRAHSMARA, R.A., DARNOTO, B.R.P. & TOFAN, Y.A., 2021, *'A Systematic Comparison of Software Requirements Classification', IPTEK The Journal for Technology and Science,* 32(3), 184.

BINKHONAIN, M. & ZHAO, L., 2019, *'A review of machine learning algorithms for identification and classification of non-functional requirements', Expert Systems with Applications: X*, 10(1), 1–13.

CANEDO, E.D. & MENDES, B.C., 2020, *'Software Requirements Classification Using Machine Learning Algorithms', Entropy,* 22(9), 1–20.

DEWI, M.R., ZULFA, F., IMAM, D.K. & SIAHAAN, D.O., 2023, 'Klasifikasi Kebutuhan Perangkat Lunak Berdasarkan Kategori Iso/Iec 25000 Menggunakan Textrank Dan Svm', *Jurnal Sistem Informasi (JUNSIBI)*, 4(2), 52–58.

HANDA, N., SHARMA, A. & GUPTA, A., 2022, *'Framework for prediction and classification of non functional requirements: a novel vision', Cluster Computing,* 25(2), 1155–1173.

HASSAN, S., LI, Q., ZUBAIR, M., ALSOWAIL, R.A. & QURESHI, M.A., 2024*, 'Unveiling the Correlation between Nonfunctional Requirements and Sustainable Environmental Factors Using a Machine Learning Model', Sustainability (Switzerland)*, 16(14), 1–24.

ABDULMAJEED, A.A. and YOUNIS, Y.S., 2021. *Supporting Classification of Software Requirements system Using Intelligent Technologies Algorithms. Technium,* pp. 32–39.

KHURSHID, I., IMTIAZ, S., BOULILA, W., KHAN, Z., ABBASI, A., JAVED, A.R. & JALIL, Z., 2022*, 'Classification of Non-Functional Requirements From IoT Oriented Healthcare Requirement Document'*, *Frontiers in Public Health*, 10, 1–4.

LIMA, M., VALLE, V., COSTA, E., LIRA, F. & GADELHA, B., 2019, *Software engineering repositories: Expanding the PROMISE database*, *ACM International Conference Proceeding Series*, 427–436, Association for Computing Machinery.

RASHME, T.Y., 2024, *'Mapping Software Requirements: An Overview of Classification Strategies', International Journal of Applied Information Systems (IJAIS)*, 12(45), 1–6.

SUBAHI, A.F., 2023, *BERT-Based Approach for Greening Software Requirements Engineering Through Non-Functional Requirements*, *IEEE Access*, vol. 11, 103001–103013, Institute of Electrical and Electronics Engineers Inc.

UMAR, A., BADAMASI YA, I., UMAR ZAMBUK, F., AHMAD, A. & ALIYU AIHONG, A., 2023, *'An Ensemble Learning Approach to Software Requirement Classification with Recursive Feature Elimination and Balance Bagging Classifier'*, *Journal Of Science Technology And Education*, 11(4), 2023.

RAHMAN, M.A., NAYEM, A. & SIDDIK, S., 2023. *Non-Functional Requirements Classification Using Machine Learning Algorithms. International Journal of Intelligent Systems and Applications*, 15(3), pp.56–69.

Halaman ini sengaja dikosongkan