Deteksi Plagiarisme pada Kode Bahasa Pemrograman Java menggunakan XGBoost

Tomy Widjaja, Andre Gunawan, Liliana
Program Studi Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) - 8417658
E-mail: tomywid77@gmail.com, andre.gunawan@petra.ac.id, lilian@petra.ac.id

ABSTRAK

Dengan adanya kemudahan akses informasi dan teknologi cloud server memudahkan siapa saja untuk mengakses data *code*. Ditambah lagi dengan zaman industri 4.0 sehingga jumlah mahasiswa informatika yang semakin banyak. Hal tersebut membuat tindakan plagirisme *code* khususnya di lingkungan akademis semakin mudah dilakukan. Faktanya, proses pengecekan plagiarisme *source code* secara manual merupakan tugas yang repetitif, sulit, dan memerlukan waktu yang lama. Dengan demikian, otomasi untuk deteksi plagiarisme *source code* yang memiliki kualitas tinggi sangat dibutuhkan.

Dataset yang digunakan untuk penelitian ini dikumpulkan dari kelas Dasar Pemrograman Universitas Kristen Petra. Setelah itu kode akan melewati tahapan preprocessing tokenisasi menggunakan grammar Java. Lalu, algoritma akan menghitung *pairwise features* dengan menggunakan 3 algoritma utama, yaitu levenshtein *distance*, greedy string tiling, dan bigram yang akan menghasilkan 12 *features* dan kumpulan *feature* statistik. Di langkah akhir, *features* akan digunakan untuk proses *training* maupun *inference* pada model XGBoost.

Hasil pengujian menunjukkan bahwa menggunakan *features* yang diajukan beserta *preprocessing* memiliki performa metrik yang lebih baik dari penelitian sebelumnya, yaitu f1-score sebesar 99%. Penerapan *preprocessing* juga dapat meningkatkan performa metrik pada *features* yang diajukan di di penelitian sebelumnya.

Kata Kunci: deteksi plagiarisme kode, pemrosesan teks, *pairwise features*, XGBoost, Levenshtein *Distance*, *greedy string tiling*

ABSTRACT

With the ease of access to information and cloud server technology, it makes it easier for anyone to access the code data. Coupled with the industry 4.0 era, the number of informatics students is also increasing rapidly. This makes code plagiarism easier to do, especially in academic environment Manual checking of plagiarism is repetitive, difficult, and time-consuming task. Therefore, automation for high quality source code plagiarism detection is needed.

The dataset used in this research was collected from "Dasar Pemrograman" class at Petra Christian University. After that the code will continue to tokenization preprocessing using java grammar stage. Then, the algorithm will calculate pairwise features using 3 main algorithms, namely levenshtein distance, greedy string tiling, and bigram which will produce 12 features and a collection of statistic features. Finally, the features will be used for the training and inference process on the XGBoost model.

The test result shows that the proposed features have better performance metrics than previous research, it has f1-score of 99%. Implementation of preprocessing can also improve performance metrics on the features proposed in this study and in previous research.

Keywords: *code plagiarism detection, text processing, pairwise features, XGBoost, Levenshtein Distance, greedy string tiling*

1. PENDAHULUAN

Plagiarisme merupakan tindakan yang meniru hasil pekerjaan orang lain secara langsung maupun tidak langsung. Di lingkungan akademis, plagiarisme yang biasanya sering terjadi adalah pada dokumen tekstual seperti *essay*, laporan, dan bahkan penelitian. Akan tetapi, plagiarisme tidak hanya berlaku pada dokumen tekstual tetapi juga dokumen *source code*. Plagiarisme *source code* di dunia akademis biasanya terjadi ketika mahasiswa meniru kode milik mahasiswa lain dan melakukan *submission* seolah-olah seperti pekerjaan milik mahasiswa tersebut [6].

Permasalahan yang biasanya timbul di lingkungan akademis yang berkaitan dengan ilmu komputer adalah terjadinya plagiarisme di tugas-tugas pemrograman. Apalagi pada masa ini yang menggunakan konsep belajar secara *online*, maka kesempatan untuk berbuat plagiat semakin luas karena tidak adanya pengawasan secara langsung dari pengajar. Dengan demikian, otomasi untuk deteksi plagiarisme *source code* yang memiliki kualitas tinggi sangat dibutuhkan untuk memenuhi kebutuhan karena jumlah mahasiswa yang terus berkembang.

Terdapat berbagai penelitian yang telah dilakukan untuk mengatasi masalah plagiarisme kode. Salah satunya penelitian yang dilakukan menggunakan metode Levenshtein *distance* untuk mengukur *similarity* antara dua buah dokumen *source code* [12]. Kekurangan dalam penelitian tersebut adalah hanya mengandalkan *levenshtein distance* yang kurang *robust* jika menghitung similarity antara *source code* yang panjangnya berbeda dan penggunaan *grammar* yang terlalu spesifik.

Selanjutnya beberapa penelitian mulai menggunakan metode *machine learning* untuk menentukan plagiarism kode [3, 15]. Di penelitian [15] menggabungkan beberapa algoritma machine learning dan menggunakan features yang bersifat metrik dengan tujuan untuk mengklasifikasikan identitas penulis kode. Kekurangan dalam penelitian tersebut adalah fitur *metrics* yang digunakan tidak cukup untuk menggambarkan kemiripan antara dua buah dokumen *source code*. Sedangkan penelitian [3] merumuskan 6 pairwise features dan menguji SVM serta XGBoost. Penelitian [3] menggunakan algoritma jaccard *similarity* di salah satu fiturnya untuk mengukur kemiripan alur dua dokumen kode. Kekurangan dalam penelitian tersebut adalah tidak terdapat *preprocessing* tokenisasi *grammar* sehingga tidak *robust* terhadap pergantian-pergantian yang dilakukan oleh *plagiarist*. Selain itu penelitian tersebut juga tidak melakukan *feature selection* dan menjelaskan fitur-fitur mana saja yang penting untuk tujuan

mendeteksi plagiarisme *source code*. Penelitian ini akan menerapkan pairwise features seperti di penelitian sebelumnya [3] dan juga menambahkan preprocessing grammar Java. Pairwise features akan menggunakan 3 algoritma utama yaitu, Levenshtein *distance*, greedy string tiling, dan bigram. Dengan demikian, dapat menghasilkan model yang lebih robust terhadap berbagai macam serangan *plagiarist*.

2. TINJAUAN STUDI

Penelitian ini menggunakan beberapa penelitian terkait sebagai tinjauan studi.

Munif, et al. [12] mendeteksi plagiasi pada kode dengan bahasa C++. Metode yang digunakan adalah parser bahasa C++ dengan ANTLR sehingga terbentuk terbentuk *Abstract Syntax Tree* (AST). Lalu menggunakan Levenshtein *Distance* untuk mengukur kemiripan antara *submission*. Hasil dari penelitian terkait adalah pada uji coba 1 menghasilkan akurasi sebesar 98% dan *recall* sebesar 50%. Sedangkan untuk uji coba 2 menghasilkan akurasi sebesar 97,62% dan *recall* sebesar 95,90%.

Priya, et al. [15] mendeteksi plagiasi suatu *code* berdasarkan ciri-ciri tekstual dari tulisan *code*. Metode yang digunakan adalah kombinasi 3 algoritma machine learning, yaitu Naïve Bayes Classifier, K-Nearest Neighbor, dan AdaBoost Meta Learning. AdaBoost digunakan sebagai meta learner untuk mengkombinasikan weak learners yang ada. Fitur-fitur yang digunakan di dalam penelitian ini adalah fitur-fitur yang bersifat *metrics* dari source code seperti frekuensi sebuah kata, frekuensi komen, frekuensi *access controls*, dll. Hasil dari penelitian terkait adalah adalah model dapat mengidentifikasi penulis dari sebuah *code*. Pada dataset contoh, terdapat 4 program file yang berbeda beserta dengan representasi fiturnya. 3 *program file* merupakan training set dan 1 *program file* dijadikan untuk testing. Hasil klasifikasi menunjukkan program untuk *testing* teridentifikasi paling mirip dengan *writer* 2.

Awale, et al. [3] mendeteksi plagiasi *code* bahasa C++ dengan menggunakan konsep *pairwise features*. Metode yang digunakan adalah SVM dan XGBoost yang di train menggunakan 6 *pairwise features* antara dokumen-dokumen source code. Pairwise features menggunakan algoritma Jaccard *similarity* dan Karp-Rabin *hash*. Hasil dari penelitian terkait adalah model SVM memiliki akurasi sebesar 90%. Sedangkan model XGBoost memiliki akurasi sebesar 94%. Confusion Matrix XGBoost menunjukkan nilai recall sebesar 81,5%.

3. TINJAUAN PUSTAKA

3.1 Source Code Plagiarism Detection

Plagiarisme *source code* terjadi ketika seorang murid/mahasiswa menggunakan ulang *source code* milik orang lain sehingga melanggar peraturan untuk suatu tugas pemrograman [6]. Secara umum metode untuk mendeteksi plagiarism *source code* dapat dibagi menjadi dua yaitu menggunakan pendekatan representasi statistik dari dokumen/*attribute based* dan representasi tekstual dari dokumen/*structure based* [7]. Metode deteksi plagarisme kode secara struktural merupakan metode yang lebih natural dan sesuai dengan perspektif manusia [11].

Tipe plagiarisme dalam *source code* bisa dibagi menjadi 4 subtipe [5]. Subtipe yang pertama disebut sebagai *manipulation from vicinity plagiarism*. Di subtipe tersebut, seorang *developer* memanipulasi program dengan menambahkan, menghapus, atau mengganti bagian-bagian tertentu dari *source code* milik orang lain. Subtipe yang kedua disebut sebagai *reordering structure plagiarism*. Di subtipe tersebut, seorang developer mengganti urutan dari *statements* dan *functions* dari *source code* milik orang lain. Subtipe yang ketiga disebut sebagai *no change plagiarism*. Di subtipe tersebut, seorang *developer* tidak melakukan perubahan sama sekali atau melakukan perubahan pada *whitespace* dari *source code* milik orang lain. Subtipe yang keempat disebut sebagai *language switching plagiarism*. Di subtipe tersebut, seorang *developer* mengubah bahasa dari *source code* milik orang lain.

3.2 Levenshtein Distance

Levensthein *distance* adalah suatu metrik yang digunakan untuk mengukur perbedaan kedua *string*. Perhitungan levensthein *distance* menghasilkan angka 'jarak' yang menunjukkan berapa minimal suatu operasi yang diperlukan agar suatu *string* pertama berubah ke *string* kedua [13]. Perubahan yang dilakukan terdapat 3 kategori yaitu *insertion*, *deletion*, dan *substitution*. *Piecewise function* yang digunakan untuk menghitung levenshtein *distance* menggunakan matrix terdapat di persamaan (1). Beberapa variabel yang digunakan untuk menghitung levenshtein *distance* adalah a merupakan string 1, b merupakan string 2, i merupakan posisi karakter terminal dari string 1, dan j merupakan posisi karakter terminal dari string 2.

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & if\ \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{a_i \neq b_j} \end{cases} & otherwise. \end{cases} \quad (1)$$

3.3 Levenshtein Ratio

Levenshtein *ratio* merupakan levenshtein *distance* yang dinormalisasi sehingga menghasilkan nilai di antara 0 – 1 [16]. Komponen utama dari levenshtein *ratio* adalah menggunakan nilai *cost* dari perhitungan levenshtein *distance*. Berikut persamaan levenshtein ratio terdapat di persamaan (2) dimana terdiri dari komponen *cost*, len(s1), dan len(s2). Nilai *cost* untuk *delete* dan *insertion* adalah 1, sedangkan untuk *substitution* adalah 2. Sedangkan, len(s1) dan len(s2) merupakan panjang dari string 1 dan string 2.

$$Levenshtein\ Ratio(s1,s2) = 1 - \frac{Cost}{len(s1)+len(s2)} \quad (2)$$

3.4 Greedy String Tiling

Greedy String Tiling (GST) merupakan algoritma yang digunakan untuk membandingkan *substring* di antara dua dokumen. GST akan menghasilkan segmen-segmen *substring* yang sama antara satu dokumen dengan dokumen lainnya. Di GST terdapat bagian penandaan *substring* yang sudah *match* sehingga tidak terjadi *overlap* dan mengulang hasil *match* yang sama.

Di algoritma yang dirujuk dari [14] terdapat 2 bagian untuk menghasilkan *set of substring* yang sama pada dua dokumen. Pada bagian pertama, algoritma mencari match terbesar secara kontinu di antara kedua dokumen. Terdapat 3 *nested loop*, loop tingkat pertama berfungsi untuk mengiterasi seluruh token di string A, loop tingkat kedua melakukan komparasi token dengan token di string B. Lalu jika hasil komparasi identik, akan lanjut ke loop tingkat tiga yang akan melakukan matching sampai akhir. Bagian pertama ini akan menghasilkan sebuah set *common substring* yang terpanjang. Pada bagian 2 berfungsi untuk memberi tanda pada hasil match yang ada sehingga token-token yang sudah ditandai tidak akan

digunakan lagi di iterasi berikutnya untuk bagian pertama. Hasil match disini dinamakan dengan terminologi *tile*.

3.5 Bigram

Bigram adalah urutan dari dua elemen yang berdekatan di sebuah *string* atau kumpulan *token*. Bigram merupakan salah satu jenis N-Gram dimana N memiliki nilai 2. Contohnya untuk kalimat "saya sedang belajar matematika" akan memiliki 3 pasangan bigram, yaitu "saya sedang", "sedang belajar", dan "belajar matematika". Bigram merepresentasikan hubungan semantik terkecil secara beurutan di suatu teks [2].

3.6 XGBoost

XGBoost merupakan salah satu algoritma berbasis *decision tree*. *Decision tree* merupakan salah satu algoritma klasifikasi yang paling sering digunakan di dunia *machine learning, image processing*, dan *identification of patterns* [10]. XGBoost memiliki beberapa keunggulan salah satunya adalah menggunakan konsep tree *boosting*. Ide utama dari *ensemble method* ini adalah menggunakan gradient dari residual model-model sebelumnya untuk training model berikutnya sehingga dapat meminimalisir *error* [1] XGBoost dapat dijalankan dengan kecepatan 10 kali lebih cepat dibandingkan dengan metode-metode lainnya di *single machine* dan dapat diskalakan sampai jutaan *example* dengan menggunakan resource yang lebih sedikit atau secara terdistribusi [4]. Perubahan major di dalam XGBoost adalah sistem tree boosting yang bisa diskalakan, algoritma *justified weighted quantile sketch*, algoritma *sparsity aware*, dan *cache-aware block structure*. Algoritma *weighted quantile sketch* berfungsi untuk menangani weighted data dan menemukan *splitting point* yang lebih baik. Algoritma *sparsity aware* untuk menangani data yang *sparse*. Sedangkan *cache-aware* memanfaatkan penggunaan cache di CPU sehingga algoritma akan berjalan lebih efisien untuk dataset yang besar.

3.7 Jaccard Similarity

Jaccard *similarity* adalah metrik yang digunakan untuk mengukur kemiripan antara dua *set* data. Jaccard *similarity* melakukan kalkulasi dengan menghitung seberapa banyak data yang sama antara dua buah *set* yang dibagi dengan jumlah elemen unik di kedua set [8]. Jaccard *similarity* memiliki nilai di antara 0 – 1, semakin tinggi maka kedua set tersebut akan semakin mirip. Persamaan (3) adalah persamaan untuk menghitung jaccard *similarity*. Di persamaan tersebut terdapat 2 buah set, yaitu set A dan set B.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (3)$$

3.8 Karp-Rabin

Algoritma Karp-Rabin menggunakan konsep *hash* atau biasanya disebut *fingerprint* untuk menemukan string yang cocok [17]. Algoritma Karp-Rabin memiliki waktu yang lebih cepat untuk membandingkan dua *sequence* karena menggunakan *hash* pada tiap kata atau huruf sehingga dapat menghindari kompleksitas $O(n^2)$. Algoritma Karp-Rabin biasanya dijalankan dari kiri teks ke kanan. Selain itu fungsi *hashing* harus dapat mengkalkulasi *hash* secara efisien. Biasanya untuk mendapatkan *fingerprint*, teks akan melewati proses tokenisasi terelebih dahulu dan *hash* akan dibentuk sesuai dengan jumlah k-gram yang diinginkan. Misalkan k-grams yang terbentuk dari c_1 sampai c_k akan di *hash* dengan nilai base b. Persamaan hash value dari $H(c_1 \dots c_k)$ terdapat di persamaan (4). Hash value dari sequence berikutnya akan dihitung dengan menggunakan rolling hash seperti di persamaan (5).

$$H(c_1 \dots c_k) = c_1 * b^{k-1} + c2 * b^{k-2} + \cdots + c_{k-1} * b + c_k \quad (4)$$

$$H(c_2 \dots c_{k+1}) = (H(c_1 \dots c_k) - c_1 * b^{k-1}) * b + c_{k+1} \quad (5)$$

4. DESAIN SISTEM

Bagian ini membahas desain sistem khususnya untuk pengumpulan data dan proses sistem yang akan menghasilkan *features*.

4.1 Dataset

Pembahasan untuk *dataset* akan dibagi menjadi dua, yaitu pengumpulan data dan pembagian data. *Dataset* yang digunakan bersumber dari *files submission* kode bahasa java yang akan diproses lebih lanjut menjadi pasangan plagiat / non plagiat.

4.1.1 Pengumpulan Data

Data dikumpulkan dari mata kuliah Dasar Pemrograman Prodi Informatika, Universitas Kristen Petra. Lalu *dataset* tersebut akan dilabeli secara manual untuk pasangan-pasangan yang memiliki potensi plagiat dan divalidasi oleh salah satu dosen pengajar Informatika Universitas Kristen Petra. Pasangan-pasangan plagiat diharapkan memenuhi variasi dan batas bawah proporsi yang terdapat di Tabel 1.

Tabel 1. Variasi Plagiat dan Batas Bawah Proporsi

Variasi Plagiat	Batas Bawah Proporsi
1. Perubahan nama variabel 2. Perubahan nama fungsi 3. Perubahan isi *string* 4. Perubahan *whitespace*	50%
5. Pengacakan susunan fungsi 6. Perubahan urutan *statement* dalam suatu *line* 7. Perubahan tipe variabel 8. Perubahan angka	30%
9. Pengacakan susunan *lines statements* / struktur kode 10. Perubahan *statement loop* (misal for jadi while) 11. Perubahan *mathematical identity*	20%

Batas bawah tersebut berbanding terbalik dengan usaha yang dibutuhkan. Sebagai contoh 'perubahan *statement loop*' membutuhkan usaha lebih besar daripada 'perubahan nama variabel'. Sedangkan, untuk pasangan non plagiat akan di ambil secara acak (*random sampling*) dengan menggunakan nilai fitur CSA terendah dari kumpulan pasangan plagiat sebagai batas atas. Algoritma akan mengambil pasangan non plagiat sejumlah pasangan plagiat agar menghasilkan rasio 50% pasangan plagiat dan 50% pasangan non plagiat.

4.1.2 Pembagian Data

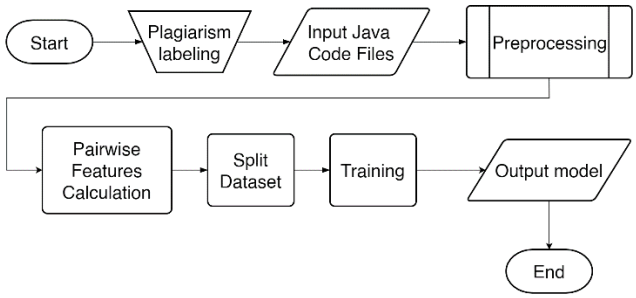
Dataset yang dikumpulkan akan dibagi menjadi *training* set dan *testing* set sesuai dengan komposisi 80%:20% (toleransi ± 2%). Pembagian data juga akan memastikan untuk mempunyai panjang *token* yang berbeda-beda agar menghasilkan data yang semakin *general* dan dapat mengurangi *bias*. Detail target minimal komposisi pembagian untuk dataset training dan testing terdapat di Tabel 2.

Tabel 2. Komposisi Dataset Training & Testing

Panjang Tokens Kode	Jumlah Soal	Training Set		Testing Set	
		Plag	Non Plag	Plag	Non Plag
<= 200	2	40	40	10	10
200 – 350	2	40	40	10	10
>= 350	2	40	40	10	10
Total Jumlah		**120**	**120**	**30**	**30**

4.2 Sistem Deteksi Plagiarisme Kode Bahasa Java

Proses akan bermula dari pelabelan secara manual. Lalu terdapat modul *preprocessing* yang akan menghilangkan bagian-bagian umum dari kode dan melakukan tokenisasi dengan *grammar* Java. Setelah *preprocessing* selesai, modul *pairwise features calculation* akan menghitung fitur-fitur yang dirumuskan. Tahapan terakhir adalah *training* sehingga dapat menghasilkan model yang akan digunakan di web application. Proses ini dapat dilihat dalam bentuk flowchart di Gambar 1.



Gambar 1. Flowchart Sistem Deteksi Plagiarisme

4.2.1 Preprocessing

Tahapan *preprocessing* akan dimulai dengan menghapus bagian-bagian yang cenderung statis dan sama untuk semua code seperti deklarasi *package*, deklarasi *class*, deklarasi *main function*. Hal tersebut dilakukan agar bagian yang umum tidak memengaruhi penilaian apakah suatu *source code* plagiat atau tidak plagiat. Setelah itu semua karakter *space* dan *escape sequence* akan dihapus agar dapat memperoleh *source code* yang murni. *Source code* tersebut akan di tokenisasi menggunakan *grammar* yang sudah ditentukan.

4.2.2 Tokenization

Tahapan tokenisasi akan dimulai dengan mengelompokkan tiap *token* dari *source code* menjadi 4 kategori. Kategori-kategori tersebut adalah Identifier, Keyword, Separator, Operator. Pengelompokan kategori token tersebut diambil dari spesifikasi Bahasa Pemrograman Java [9]. Setelah pengelompokan selesai, maka *token-token* tersebut akan dikonversi ke kode alfabet. Pembagian kode-kode tersebut terdapat di Tabel 3.

4.2.3 Pairwise Features Calculation

Pairwise feature adalah fitur yang didapatkan dari perhitungan suatu komponen antara dua *source code*. Untuk penelitian ini, terdapat 12 *pairwise features* + kumpulan statistic features. Detail fitur-fitur yang akan digunakan beserta perhitungannya terdapat di Tabel 4.

Tabel 3. Java Token Grammar

Kategori Token	Nama *Grammar*	Keterangan	Kode
Identifier	Variable / Function Name	Nama variabel atau fungsi	**A**
	IdentifierOthers	Methods dan library	**B**
Keyword	LoopStatement	do, while, for	**C**
	DecisionStatement	if, else, switch, case	**D**
	BasicType	boolean, byte, char, double, float, int, long, short, String	**E**
	KeywordOthers	abstract, continue, new, assert, default, package, synchronized, goto, private, this, break, implements, protected, throw, byte, else, import, public, throws, case, enum, instanceof, return, transient, catch, extends, try, final, interface, static, void, class, finally, long, strictfp, volatile, const, float, native, super	**F**
Literal	String	"…" '…'	**G**
	Number	Integer atau float	**H**
	Boolean	true, false	**I**
	Null	null	**J**
Separator	Separator	( ) { } [ ] ; , .	**K**
Operator	ArithmeticOperator	+, -, *, /, %	**L**
	AssignmentOperator	=, +=, -=, *=, /=, %=	**M**
	LogicalOperator	&&, ||, !	**N**
	ComparisonOperator	<, >, <=, =>, !=, ==	**O**
	Increment-Decrement Operator	++, --	**P**
	OperatorOthers	~ ? : -> & | ^ << >> >>> &= |= ^= <<= >>= >>>=	**Q**

Tabel 4. Pairwise Features

Nama Fitur	Penjelasan
CSS (Code Structure Similarity)	$CSS = 1 - \frac{C}{s1 + s2}$ C = *edit cost (levenshtein distance)* s1 = panjang *token* dari *source code* 1 s2 = panjang *token* dari *source code* 2
CLTS (Code Line Tiles Similarity)	$CLTS = \frac{N}{\min(l1, l2)}$ N = jumlah *tiled lines* (GST) l1 = jumlah *line source code* 1 yang sudah di-*preprocess* l2 = jumlah *line source code* 2 yang sudah di-*preprocess*
CSA (Code Similarity Average)	Rata-rata dari CSS dan CLTS.
BS (Braces Similarity)	*Brace* akan dibagi menjadi 4 kategori: 1. *Brace* di sebelah paling kiri *code* 2. *Brace* di sebelah paling kanan *code* 3. *Brace* di tengah-tengah *code* 4. *Brace* di line sendiri 4 Kategori di atas akan mewakili 1, 2, 3, 4. Lalu tiap karakter *brace* akan diambil beserta dengan kategori nya misal "{4{2}4}4". Nilai *feature* akan didapatkan dari jumlah tiles GST dibagi dengan panjang sequence terpendek.
WS (Whitespace Similarity)	*Whitespace* akan dibagi menjadi 2 kategori: 1. Indent 2. Newline 2 Kategori di atas akan mewakili 1, 2. Nilai *feature* akan didapatkan dari jumlah tiles GST dibagi dengan panjang sequence terpendek.
CS (Comment Similarity)	*Comment* akan dibagi menjadi 3 kategori: 1. Comment sendiri di newline 2. Comment setelah kode 3. Multi line comment 3 Kategori di atas akan mewakili 1, 2, 3. Nilai *feature* akan didapatkan dari jumlah tiles GST dibagi dengan panjang sequence terpendek.
CSSA (Code Style Similarity Average)	Rata-rata dari BS, WS, dan CS
CLN (Common Line Normalized)	$CLN = \frac{N}{\min(l1, l2)}$ N = jumlah *line* yang sama di antara dua *source code* l1 = jumlah *line source code* 1 yang sudah di-*preprocess* l2 = jumlah *line source code* 2 yang sudah di *preprocess*
CBLN (Common Bigram Line Normalized)	$CBLN = \frac{N}{min\ (l1, l2)}$ N = jumlah *bigram line* yang sama di antara dua *source code* l1 = jumlah *bigram line source code* 1 yang sudah di-*preprocess* l2 = jumlah *bigram line source code* 2 yang sudah di-*preprocess*
CBLN80 (Common Bigram Line Normalized 80% Ratio)	$CBLN80 = \frac{N}{\min\ (l1, l2)}$ N = jumlah *bigram line* yang mirip (>80% *levenshtein ratio*) di antara dua *source code* l1 = jumlah *bigram line source code* 1 yang sudah di-*preprocess* l2 = jumlah *bigram line source code* 2 yang sudah di-*preprocess*
TCA (Token Count Average)	Rata-rata jumlah *token* di kedua dokumen.
TCD (Token Count Difference)	Perbedaan jumlah *token* antara kedua dokumen.
Fitur Statistik (Percentile 95,90,85)	Fitur akan merepresentasikan Boolean. Jika True / 1 maka nilai dari fitur tersebut memiliki nilai lebih dari percentile N. Jika False / 0 maka nilai dari fitur tersebut memiliki nilai kurang dari percentile N.

5. PENGUJIAN SISTEM

5.1 Pengujian Performa Model XGBoost

Pengujian awal adalah melakukan *training* pada *model* dengan menggunakan *features* yang diajukan di penelitian ini. *Training model* XGBoost menggunakan konfigurasi *default* akan dilakukan dengan dua variasi yaitu tanpa *preprocessing* dan dengan *preprocessing*. Jumlah features yang digunakan adalah sebanyak 46 *features*. Hasil menunjukkan bahwa *model* tanpa *preprocessing* memiliki metrik yang lebih rendah dibandingkan dengan model yang menggunakan *preprocessing*. Berikut Tabel 5 merupakan hasil performa antara kedua variasi model.

Tabel 5. Perbandingan Penggunaan Preprocessing dengan Tanpa Preprocessing

Variasi Model	Accuracy	Precision	Recall	F1-Score
Tanpa *Preprocessing*	93.8%	88.9%	**100%**	94.1%
Dengan *Preprocessing*	**99%**	**98%**	**100%**	**99%**

Tanpa *preprocessing* akan menyebabkan bagian-bagian yang sering terulang / *template* dalam suatu *code* akan memiliki efek yang lebih mendominasi dibandingkan dengan perubahan yang dilakukan. Hal tersebut memungkinkan jadi salah satu penyebab

nilai *feature* menjadi lebih tinggi dan pada akhirnya akan membuat beberapa *false positive*.

False positive terjadi pada soal yang tergolong kategori kode pendek, sehingga kemungkinan submission memiliki solusi yang sama juga semakin tinggi (meskipun tidak mencontek). Selain itu pengambilan pasangan yang tidak plagiat juga dilakukan secara *random sampling* dengan batas atas menggunakan nilai feature CSA terkecil. Hal tersebut dapat menyebabkan beberapa nilai *feature* untuk pasangan yang tidak plagiat secara *random* dapat mendekati nilai *feature* yang plagiat.

Pengujian dilanjutkan dengan menggunakan *features* dari penelitian sebelumnya oleh Awale, et al. [3]. Jumlah *features* yang digunakan adalah 4 *features*. *Training model* XGBoost menggunakan konfigurasi default dan akan dilakukan dengan 2 variasi, yaitu dengan *preprocessing original* dan *preprocessing* yang diajukan di penelitian ini. Berikut Tabel 6 merupakan hasil performa antara kedua variasi model.

Tabel 6. Perbandingan Penggunaan Preprocessing Original dengan Preprocessing Baru di penelitian [3]

Variasi Model (Features dari penelitian [3])	Accuracy	Precision	Recall	F1-Score
Preprocessing original	90.6%	89.8%	91.7%	90.7%
Preprocessing baru	**94.8%**	**95.7%**	**93.8%**	**94.7%**

Secara keseluruhan performa metrik meningkat ketika training *model* menggunakan kombinasi *features* yang diajukan di penelitian ini dibandingkan dengan penelitian sebelumnya [3]. *Preprocessing* juga dapat meningkatkan performa di *features* yang diajukan maupun *features* dari penelitian sebelumnya. Ringkasan perbandingan performa beserta *features* yang d0069gunakan terdapat di Tabel 7.

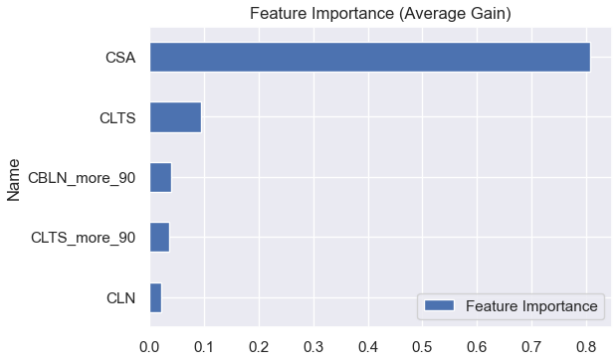
Tabel 7. Ringkasan Perbandingan Performa

Model	Accuracy	Precision	Recall	F1-*Score*
Perhitungan Features dari Penelitian ini	**99%**	**98%**	**100%**	**99%**
Perhitungan Features dari Penelitian [3]	90.6%	89.8%	91.7%	90.7%

5.2 Feature Selection

Dari 46 *features* yang digunakan, model hanya membutuhkan 11 *features*. *Features* tersebut diketahui dari *feature importance* di dalam modul XGBoost. *Feature importance* yang digunakan adalah berdasarkan average *information gain*. Idenya adalah jika sebuah *feature* tidak signifikan / tidak terlalu penting, maka *feature* tersebut dapat dibuang agar model dapat menjadi lebih *simple* dan efisien dalam melakukan *inference* maupun *training*.

Feature selection dilakukan dengan menyeleksi 11 *features* terbaik satu per satu. Dari proses *feature selection* ditemukan bahwa penggunaan 5 *features* terbaik menghasilkan metrik yang sama persis dengan saat menggunakan semua *features*. *Features* yang digunakan adalah CSA, CLTS, CBLN_more_90, CLN, CLTS_more_90. Hasil feature importance pada Gambar 2 menunjukkan CSA dan CLTS berkontribusi terhadap *information gain* terbesar, sehingga memiliki pengaruh paling dominan untuk keputusan *model* tersebut.



Gambar 2. Feature Importance setelah Feature Selection

5.3 Pengujian data submission terbaru

Hasil dari *web app* akan digunakan untuk menguji data *submission* terbaru yang terdapat di Universitas Kristen Petra. Pengujian dilakukan dengan membandingkan catatan dari dosen untuk pasangan-pasangan yang plagiat dengan hasil pasangan plagiat yang diambil dari *web app*. Jumlah pasangan yang ada di catatan dosen dan di *web app* akan menjadi *true prediction rate* untuk data submission tersebut. Sedangkan sisa dari hasil pasangan plagiat akan di-*sampling* secara acak sebanyak 15 pasangan, yang akan dicek kembali secara manual dan divalidasi oleh Dosen Informatika Universitas Kristen Petra. *True prediction rate* juga dapat dijadikan ekspektasi seberapa banyak dari sisa pasangan tersebut yang dapat menjadi potensi plagiat. Hasil dari pengujian data terbaru terdapat di Tabel 8.

Tabel 8. Pengujian Data Submission Terbaru

Sumber *Submission*	Jumlah Pasangan Plagiat di catatan Dosen	*True Prediction Rate*	Jumlah Potensi Plagiat / Jumlah Hasil Sisa (sudah di-*sampling*)
UTS DP 2021 2022 Genap – No 1	1	100%	1/1
UTS DP 2021 2022 Genap – No 2	4	75%	7/8
UAS DP 2021 2022 Ganjil – No 3	22	68.2%	12/15
UTS DP 2021 2022 Ganjil – No 4	-	-	13/15
UTS DP 2021 2022 Ganjil – No 5	7	71.4%	12/15

6. KESIMPULAN

Preprocessing yang diterapkan (menghilangkan beberapa komponen yang umum dan tokenisasi *grammar*) memiliki manfaat untuk menaikkan performa metrik pada perhitungan features

penelitian ini maupun penelitian [3]. Penggunaan *preprocessing* tokenisasi dengan *grammar* akan mempercepat proses perhitungan feature karena jumlah huruf yang diproses menjadi lebih sedikit.

Pengujian model XGBoost dengan menggunakan kombinasi features yang diajukan di penelitian ini juga memiliki performa metrik yang lebih baik dibandingkan di penelitan [3], yaitu *accuracy* 99%, *precision* 98%, *recall* 100%, dan f1-*score* 99%.

Proses feature selection dapat menurunkan jumlah feature yang digunakan hingga 5 features dengan mempertahankan performa yang sama pada *data testing*. Feature yang paling berpengaruh adalah CSA dan CLTS dengan akumulasi average information gain sebesar ≈ 90%. Kedua *feature* CSA dan CLTS memiliki pengaruh yang sangat tinggi karena kemungkinan kedua *feature* ini yang paling merepresentasikan kemiripan *code* secara strukur. Sedangkan, *features* yang berhubungan dengan *code style* juga tidak berkontribusi pada model. Sehingga dapat disimpulkan juga bahwa *code style* tidak terlalu penting dalam menentukan sebuah pasangan plagiat dalam dataset di Universitas Kristen Petra.

7. SARAN

Berikut merupakan saran-saran yang dapat diterapkan untuk mengembangkan penelitian ini:

1. Membuat dataset yang lebih banyak dalam hal kuantitas maupun variasi serangan plagiat sehingga dapat membuat sebuah model yang dapat mengakomodasi lebih banyak jenis variasi serangan plagiat.
2. Menerapkan *multi-files plagiarism detection* untuk dapat mengakomodasi soal yang memiliki lebih dari satu *file submission*.
3. Menambahkan *compatibility* untuk bahasa-bahasa pemrograman lainnya.
4. Menguji berbagai algoritma *similarity* lainnya.
5. Menguji *character based embedding* untuk membuat *source code embedding* yang dapat digunakan untuk mengukur *similarity* sebuah dokumen *code*.
6. Menguji dengan menggunakan konsep *compiler* sehingga dapat membuat *fingerprint* atau *tokens* berdasarkan urutan eksekusi sebuah code.

8. REFERENCES

[1] Anghel, A., Papandreou, N., Parnell, T., Palma A.D., & Pozidis, H. (2019). Benchmarking and Optimization of Gradient Boosting Decision Tree Algorithms. *ArXiv, abs/1809.04559*. DOI: 10.48550/arXiv.1809.04559.

[2] Asaadi, S., Mohammad, S., & Kiritchenko, S. (2019). Big BiRD: A Large, Fine-Grained, Bigram Relatedness Dataset for Examining Semantic Composition. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*. DOI: 10.18653/v1/N19-1050

[3] Awale, N., Pandey, M., Dulal, A., & Trismina, B. (2020). Plagiarism Detection in Programming Assignments using Machine Learning. *Journal of Artificial Intelligence and Capsule Networks*, 2(3), 177-184. DOI: 10.36548/jaicn.2020.3.005

[4] Chen, T. & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. KDD '16: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. DOI: 10.1145/2939672.2939785

[5] Chowdhury, H. A. & Bhattacharyya, D. K. (2018). Plagiarism: Taxonomy, Tools and Detection Techniques. *Knowledge, Library and Information Networking, NACLIN 2016*, Assam, India. ArXiv, abs/1801.06323. DOI: 10.48550/arXiv.1801.06323

[6] Cosma, G. & Joy, M. (2008). Towards a Definition of Source-Code Plagiarism. *IEEE Transactions on Education*, 51(2), 195-200. DOI: 10.1109/TE.2007.906776

[7] Ďuračík, M., Kršák E., & Hrkút, P. (2017). Plagiarism Across Europe and Beyond 2017. *Using Concepts of Text Based Plagiarism Detection in Source Code Plagiarism Analysis*. Mendel University. ISBN: 978-80-7509-493-3

[8] Gomaa, W.H. & Fahmy, A.A. (2013). A Survey of Text Similarity Approaches. *International Journal of Computer Applications*, 68(13), 13-18. DOI: 10.5120/11638-7118

[9] Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2014). *The Java Language Specification, Java SE 8 Edition*. Addison-Wesley Professional. ISBN: 978-0-13-390069-9

[10] Jijo, B.T. & Abdulazeez, A. M. (2021). Classification Based on Decision Tree Algorithm for Machine Learning. *Journal of Applied Science and Technology Trends*, 2(1), 20-28. DOI: 10.38094/jastt20165

[11] Karnalim, O. & Sulistiani, L. (2018). Which Source Code Plagiarism Detection Approach is More Humane? *The 9th International Conference on Awareness Science and Technology*, Fukuoka, Japan. DOI: 10.1109/ICAwST.2018.8517170

[12] Munif, A., Akbar, R. J., Tantra, R.I., & Ilavi, R. (2017). Rancang Bangun Sistem E-Learning Pemrograman pada Modul Deteksi Plagiarisme Kode Program dan Student Feedback System. *JUTI: Jurnal Ilmiah Teknologi Informasi*, 15(1), 104-118. DOI: 10.12962/j24068535.v15i1.a640

[13] Pradhan, N., Gyanchandani, M., & Wadhvani, R. (2015). A Review on Text Similarity Technique used in IR and its Application. *International Journal of Computer Applications*, 120(9), 29-34. DOI: 10.5120/21257-4109

[14] Prechelt, L., Malpohl, G., & Philippsen M. (2002). Finding Plagiarisms among a Set of Programs with JPlag. *Journal of Universal Computer Science*, 8(11), 1016-1038. ISSN: 0948-6968

[15] Priya, S., Dixit, A. Das, K., & Patil, R. H. (2019). Plagiarism Detection in Source Code Using Machine Learning. International *Journal of Engineering and Advanced Technology*, 8(4), 897-901. ISSN: 2249-8958

[16] Sarkar, S., Das, D., Pakray, P., & Gelbukh, A. (2016). JUNITMZ at SemEval-2016 Task 1: Identifying Semantic Similarity Using Levenshtein Ratio. *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, 702-705. DOI: 10.18653/v1/S16-1108

[17] Shivaji, S.K. & Prabhudeva. (2015). Plagiarism Detection by using Karp-Rabin and String Matching Algorithm Together. *International Journal of Computer Applications*, 116(23), 37-41. DOI: 10.5120/20294