DETEKSI KEMIRIPAN SOURCE CODE DENGAN METODE FINGERPRINT BASED DISTANCE DAN LEVENSHTEIN DISTANCE

Jimmy[1], Viny Christanti M.[2], Agus Budi D.[3]
Program Studi Teknik Informatika, Fakultas Teknologi Informasi, Universitas Tarumanagara,
Jln. Letjen S. Parman No. 1, Jakarta, 11440, Indonesia
E-mail: [1]jimmymsz@ymail.com, [2]viny@untar.ac.id, [3]agusd@fti.untar.ac.id

Abstrak

Kemudahan mengakses informasi membuat semakin mudahnya melakukan plagiarisme. Plagiarisme tidak hanya terjadi pada makalah atau tulisan ilmiah, tetapi juga pada program komputer. Dalam paper ini akan dijelaskan metode untuk mendeteksi kemiripan setiap pasang source code secara otomatis. Metode yang digunakan adalah fingerprint based distance dan Levenshtein Distance. Hasil diukur dengan akurasi, precision, dan recall. Hasilnya, pada kumpulan data tertentu, Levenshtein Distance lebih baik daripada fingerprint based distance.

Kata kunci*— Deteksi Kemiripan Source Code, Fingerprint Based Distance, Levenshtein Distance*

Abstract

The ease of accessing information makes easier to do plagiarism. Plagiarism not only applied to written essay, but these day copying other people's program is also considered as plagiarism. This paper will researching an method to automatically calculating the similarity of pair of source code. The method used are fingerprint based distance and Levenshtein Distance. The result will be measured in accuracy, precision, and recall. The result is in some dataset, levenshtein distance is better than fingerprint based distance for detecting plagiarism in student's code especially beginner in C++.

Keywords—*Fingerprint Based Distance, Levenshtein Distance, Source Code Similarity Detection*

1. PENDAHULUAN

Kemudahan mencari informasi melalui internet menyebabkan masalah dalam masyarakat, salah satunya plagiarisme.Ini menjadi masalah serius khususnya dalam bidang akademis.Plagiarisme tidak hanya terjadi dalam bentuk makalah atau karya tulis, tetapi juga dalam bentuk *source code* dari suatu program komputer. Deteksi kemiripan source code secara manual akan sangat memakan waktu jika jumlahnya banyak, karena itu dibutuhkan suatu metode yang tepat untuk mendeteksi kemiripan dari beberapa berkas source code secara otomatis.Metode untuk mendeteksi kemiripan*source code* berbeda dengan pendeteksian plagiarisme teks biasa.

Ada beberapa penelitian yang relevan. Metode sederhana untuk mengenali kemiripan dalam bahasa alami digunakan untuk source code. Teknik ini sangat sederhana, yaitu dengan mencocokkan huruf dan kata yang digunakan, dan bukan secara semantik [1]. Keuntungannya adalah mendeteksi kemiripan antar teks lebih cepat dan mudah diimplementasikan [2].

A Machine Learning Based Tool for Source Code Plagiarism Detection" yang dilakukan oleh Upul Bandara, dan Gamini Wijayarathna. Penelitian ini menggunakan algoritma klasifikasi Multinomial Naive Bayes, Bernoulli Naïve Bayes, dan k-Nearest Neighbor, lalu dikombinasikan ke Adaboost Meta-learning dengan tujuan memperkuat metode yang kurang baik. Penelitian yang dilakukan mengarah ke identifikasi penulis source code sebenarnya. Hasil dari penelitian menunjukkan akurasi 86.6 persen untuk dataset berjumlah 741 berkas [3].

Metode lainya dibuat oleh Gilad Mishne dan Maarteen de Rijke dari University of Amsterdam. Penelitiannya adalah membuat *source code retrieval* dengan menggunakan *conceptual graph.* Hasilnya tergolong baik, tetapi mempunyai kelemahan dalam kompleksitas. Proses yang dilakukan sangat banyak, jadi akan membutuhkan waktu proses yang sangat lama [4].

Pada penelitian ini, akan digunakan metode *fingerprint based distance*dan Levenshtein Distance. Metode *fingerprint based distance*diperkenalkan oleh Sandhya Narayanan dan Simi dalam penelitiannya yang berjudul *Source Code Plagiarism Detection and Performance Analysis Using Fingerprint Based Distance Measure Method* di tahun 2012. Penelitian yang dilakukan Sandhya Narayanan menyimpulkan bahwa metode fingerprint based distance mempunyai akurasi lebih dari 97% dan nilai presisi lebih dari 77%[4].

Penelitian ini bertujuan untuk mengetahui pengaruh fitur dari source code dalam mendeteksi kemiripan source code dan membandingkan performa dari *fingerprint based distance* dan Levenshtein Distance. Pada penelitian akan digunakan beberapa skenario nilai bobot (*weight*) pada setiap fitur*fingerprint based distance*. Data yang akan digunakan adalah *source code* C++ yang diperoleh dari pekerjaan mahasiswa kelas pemrograman C++ lanjut Universitas Tarumanagara semester genap 2016/2017 yang kode soalnya adalah "Bayar atau Kabur".

2. METODE PENELITIAN

2.1 Pengumpulan Data

Data yang dikumpulkan berupa *source code* C++. Dengan menggunakan data yang sudah dikumpulkan, diambil 15 yang dipercaya penulis sebagai tidak plagiat. Berdasarkan data tersebut, dibuatlah 35 file yang memplagiati file aslinya dengan berbagai macam jenis plagiat. Setiap pasang source code kemudian diberikan label “Plagiat” dan “Tidak Plagiat”. Jumlah pasangan source code adalah 1225, yaitu setiap kombinasi 2 file dari 50 file yang tersedia. Jumlah pasangan yang dianggap plagiat berjumlah 61, dan sisanya 1164 diberikan label “Tidak Plagiat”.

2.2Fingerprint Based Distance

Fingerprint based distance dapat menentukan nilai *similarity* antar *source code*.Fitur yang digunakan dalam algoritma ini berdasarkan pada penelitian Halstead, mengenai properti yang dapat digunakan untuk membedakan suatu kode program dengan kode program lainnya. Untuk menghitung nilai *similarity*, diperlukan nilai dari setiap fitur terlebih dahulu. Anggap nilai fitur program A adalah f(A) dan nilai fitur program B adalah f(B). Sehingga rasio δ dapat dihitung dengan rumus berikut[4].

$$\delta_i = \begin{cases} \frac{f_i(A)}{f_i(B)} \; jika \;\; f_i(B) > f_i(A) \\ \frac{f_i(B)}{f_i(A)} \; jika \;\; f_i(A) \geq f_i(B) \end{cases} \quad (1)$$

δ adalah rasio dari fitur antara program A dan program B, yang sudah dihitung di tahap atas. W_i adalah nilai bobot dari fitur.*Similarity score* atau angka kemiripan (direpresentasikan dengan Δ) dapat dihitung dengan rumus berikut[4].

$$\Delta = \frac{\sum_{i=1}^{n} w_i * \delta_i}{n} \quad (2)$$

2.3 Levenshtein Distance

Levenshtein Distance merupakan salah satu metode yang cukup populer untuk mencari jarak terpendekantara dua *string*. Metode ini dibuat oleh Vladimir Levensthein tahun 1965. Metode ini juga dikenal sebagai *edit distance* (ED). Jika diketahui terdapat *string* s1 dan s2, maka jaraknya adalah operasi penyuntingan minimal yang dibutuhkan untuk mengubah s1 menjadi s2. Operasi yang dimaksud adalah menyisipkan satu karakter, menghapus satu karakter, dan mengganti (substitusi) satu karakter menjadi karakter lain[5]. Secara matematis, Levenshtein Distance antara dua *string*, **a** dan **b**, adalah sebagai berikut.

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & jika \min(i,j) = 0 \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(ai \neq bj)} \end{cases} & selain\ itu \end{cases} \quad (3)$$

Pengubahan dari nilai *distance* menjadi *similarity* dapat dihitung dengan rumus sebagai berikut.

$$sim_{a,b} = \frac{\max(|A|,|B|) - lev_{a,b}}{\max(|A|,|B|)} \quad (4)$$

$Sim_{a,b}$ adalah nilai kemiripan antara a dan b. |A| adalah panjang dari string A. $Lev_{a,b}$ adalah hasil perhitungan Levenshtein Distance sebelumnya.

2.4 Evaluasi

Evaluasi merupakan langkah terakhir yang diperlukan untuk menyatakan bahwa sistem yang dibuat sudah berjalan sesuai dengan yang diharapkan. Evaluasi penelitian ini akan menggunakan nilai akurasi, *precision* dan *recall*. Masing-masing dari pasangan tersebut diketahui keadaan sebenarnya apakah orisinil atau hasil jiplakan. Setiap dari hasil uji akan diberi label, yang akan dicocokkan pada keadaan sebenarnya.

Apabila hasil deteksi menyatakan P (plagiat), dan sebenarnya adalah P (plagiat), maka dapat diberi label TP (*True Positive*). Apabila hasil deteksi menyatakan P (plagiat), dan sebenarnya adalah NP (non-plagiat), maka dapat diberi label FP (*False Positive*). Apabila hasil deteksi menyatakan NP (non-plagiat), dan keadaan sebenarnya adalah P (plagiat), maka dapat diberi label FP (*False Negative*). Apabila hasil deteksi menyatakan NP (non-plagiat), dan sebenarnya adalah (non-plagiat), maka dapat diberi label TN (*True Negative*).

Precision dan *recall* adalah teknik untuk mengukur keefektifan suatu algoritma, terutama di bidang pengenalan pola dan *information retrieval*. Dalam penelitian ini, *precision* adalah nilai perbandingan antara jumlah berkas yang plagiat dan dideteksi sebagai plagiat dengan jumlah berkas yang terdeteksi sebagai plagiat, sedangkan *recall* adalah nilai perbandingan antara jumlah berkas yang plagiat dan terdeteksi sebagai plagiat dengan jumlah dokumen yang sebenarnya plagiat. Rumusan untuk menghitung nilai *precision, recall*, dan akurasi adalah sebagai berikut.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6}$$

$$\text{akurasi} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{6}$$

Setiap paragraf diawali dengan 1 cm ke dalam (*first line*). Sub bab diberi nomor dan rata kiri. Organisasi naskah meliputi pendahuluan, metode, eksperimen/hasil dan pembahasan, kesimpulan, dan referensi. Lampiran (jika ada) ditulis setelah kesimpulan dan sebelum referensi dan tidak bernomor. Jarak antara paragraf adalah satu spasi.

3. HASIL DAN PEMBAHASAN

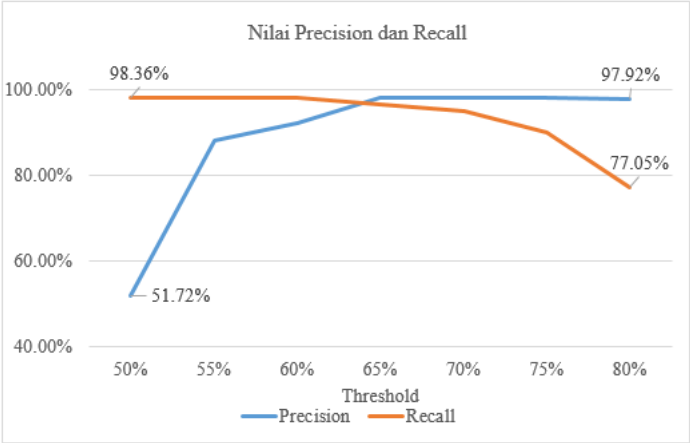
Pengujian pada program yang sudah dirancang dibagi menjadi 2 (dua) bagian, yaitu pengujian metode Leveshstein distance, serta pengujian metode *fingerprint based distance* dengan bobot yang sudah ditentukan pada Tabel 4.

3.1 Pengujian Levenshtein Distance

Pengujian yang dilakukan bertujuan untuk menghasilkan tingkat keakuratan metode Levenshtein Distance dalam menentukan plagiarisme source code. Hasil dari perhitungan berupa angka kemiripan. Hasil ini akan diberi label, jika melewati *threshold*, maka akan diberi label "Plagiat". Setiap hasil dari metode Levenshtein Distance akan dicocokkan ke kelas sebenarnya, dan akan dihitung jumlah *True Positive*, *True Negative*, *False Positive*, dan *False Negative*. Setelah itu, dihitunglah nilai *precision, recall,* dan akurasi dari setiap kasus uji. Hasil Pengujian dapat dilihat pada Tabel 1dan Gambar 1.

Tabel 1 Hasil Pengujian Levenshtein Distance dengan Threshold Tertentu

Keterangan	***Threshold***						
	50%	**55%**	**60%**	**65%**	**70%**	**75%**	**80%**
TP	60	60	60	59	58	55	47
TN	1108	1156	1159	1163	1163	1163	1163
FP	56	8	5	1	1	1	14
FN	1	1	1	2	3	6	1
Precision	51.72%	88.24%	92.31%	98.33%	98.31%	98.21%	97.92%
Recall	98.36%	98.36%	98.36%	96.72%	95.08%	90.16%	77.05%
Accuracy	95.35%	99.27%	99.51%	99.76%	99.67%	99.43%	98.78%

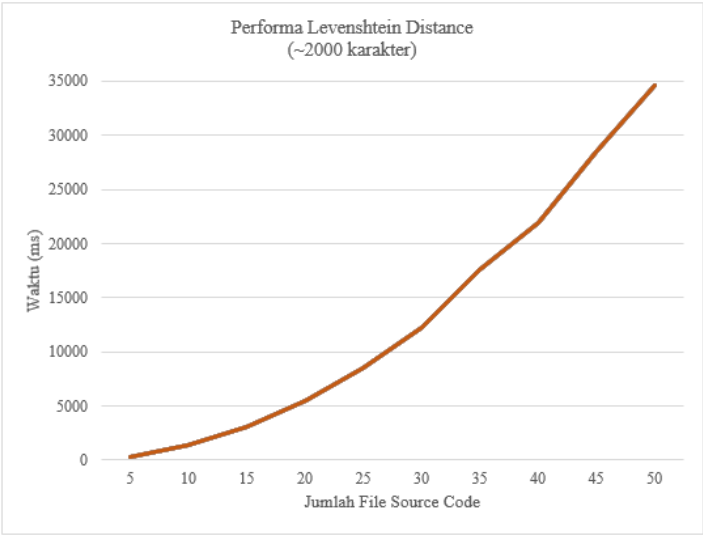


Gambar 1 Hasil Pengujian Levenshtein Distance

Hal lain yang dilakukan pada pengujian ini adalah pengukuran performa dari Levenshtein Distance. Hal yang dijadikan variabel bebas adalah jumlah *source code* yang digunakan. Variabel kontrolnya adalah setiap *source code* dibuat sama yaitu kurang lebih 2000 karakter sebelum dihapus komentar dan spasinya oleh sistem. Varibel terikatnya adalah waktu eksekusi yang dibutuhkan sistem. Hasil dapat dilihat pada Tabel 2 dan Gambar 2.

Tabel 2 Perbandingan Waktu Eksekusi Levenshtein

Jumlah file source code	Jumlah Perbandingan	Waktu (ms)
5	10	292
10	45	1400
15	105	3117
20	190	5460
25	300	8482
30	435	12286
35	595	17533
40	780	21878
45	990	28434
50	1225	34610



Gambar 2 Perbandingan Waktu Eksekusi Levenshtein

3.2 Pengujian Fingerprint Based Distance

Pengujian kedua ini menggunakan data uji yang sama dengan pengujian pertama. Pengujian ini bertujuan menentukan nilai bobot (*weight*) dari *fingerprint based distance* yang cocok dalam menentukan plagiarisme source code. Setiap hasil yang dihasilkan metode ini akan dicocokkan ke kelas sebenarnya yang sudah ditentukan. Terdapat 13 macam skenario uji yang akan dilakukan. Nilai bobot setiap fitur yang dijadikan skenario uji dapat dilihat pada Tabel 4. Nilai *threshold* yang digunakan akan menggunakan nilai *threshold* yang cukup optimal dari percobaan sebelumnya, yaitu 65%.Tabel 3 menunjukkan ciri dari source code yang dijadikan fitur dari *fingerprint based distance*.Tabel 4 menunjukkan bobot dari setiap fitur di setiap percobaan.

Tabel 3 Fitur dari Fingerprint Based Distance

No	Fitur
1	jumlah input
2	jumlah output
3	operator yang muncul
4	operand yang muncul
5	operator yang unik
6	operand yang unik
7	volume
8	jumlah baris
9	komentar

Tabel 4 Bobot Fitur Setiap Percobaan

Percobaan	Bobot fitur								
	F1	F2	F3	F4	F5	F6	F7	F8	F9
Percobaan 1	-	-	-	-	-	-	1	1	-
Percobaan 2	-	-	1	1	1	1	-	-	-
Percobaan 3	1	1	-	-	-	-	-	-	-
Percobaan 4	1	1	-	-	-	-	1	1	-
Percobaan 5	1	1	1	1	1	1	-	-	-
Percobaan 6	-	-	1	1	1	1	1	1	-
Percobaan 7	1	1	1	1	1	1	1	1	-
Percobaan 8	1	1	0.8	1	0.8	1	1	1	-
Percobaan 9	1	1	1	0.8	1	0.8	1	1	-
Percobaan 10	1	1	0.6	0.6	0.6	0.6	0.5	1	-
Percobaan 11	1	1	1	1	1	1	1	1	0.6
Percobaan 12	1	1	1	1	1	1	1	-	-
Percobaan 13	1	0.1	1	0.9	0.7	1	0.1	1	-

Bobot dari setiap fitur dapat diatur bebas dalam rentang 0 sampai 1. Jika bobot fiturnya 0, maka dapat dianggap fitur tersebut diabaikan dan tidak perlu dihitung. Dengan mengggunakan bobot fitur pada tabel di atas, diperoleh hasil pengujian. Hasil pengujian dapat dilihat pada Tabel 5.

Tabel 5 Hasil Pengujian Fingerprint Based Distance (Bagian I)

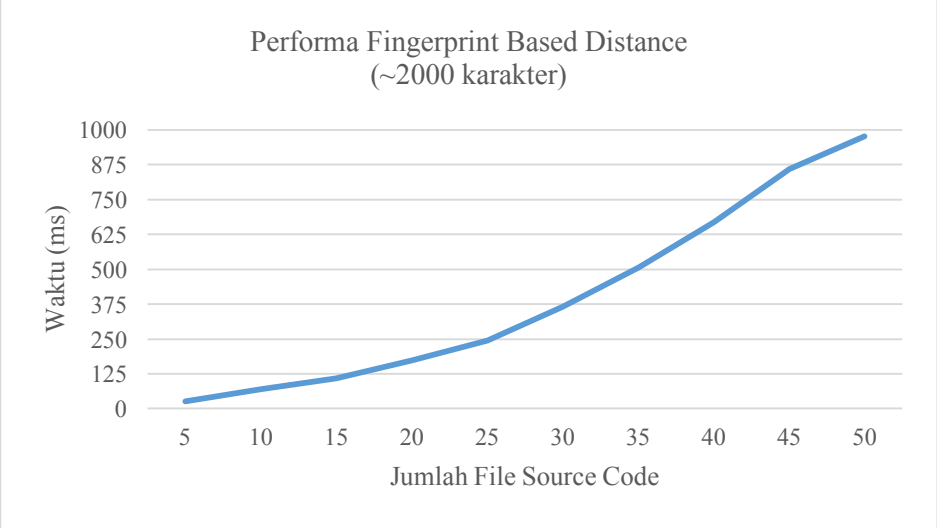
Keterangan	Skenario						
	1	**2**	**3**	**4**	**5**	**6**	**7**
TP	60	57	60	60	57	57	57
TN	322	599	503	358	605	500	541
FP	842	565	661	806	559	664	623
FN	1	4	1	1	4	4	4
Precision	6.65%	9.16%	8.32%	6.93%	9.25%	7.91%	8.38%
Recall	98.36%	93.44%	98.36%	98.36%	93.44%	93.44%	93.44%
Accuracy	31.18%	53.55%	45.96%	34.12%	54.04%	45.47%	48.82%

Tabel 6 Hasil Pengujian Fingerprint Based Distance (Bagian II)

Keterangan	Skenario					
	8	**9**	**10**	**11**	**12**	**13**
TP	57	57	57	57	57	55
TN	614	614	1054	716	553	1073
FP	550	550	110	448	611	91
FN	4	4	4	4	4	6
Precision	9.39%	9.39%	34.13%	11.29%	8.53%	37.67%
Recall	93.44%	93.44%	93.44%	93.44%	93.44%	90.16%
Accuracy	54.78%	54.78%	90.69%	63.10%	49.80%	92.08%

Hasil terbaik ada pada skenario-13. Hal ini terlihat pada akurasinya yaitu 92.08% dengan *precision* sebesar 37.67% dan *recall* sebesar 90.16%. Hasil dengan akurasi terkecil ada pada skenario-1, yaitu nilai akurasi 31.18% dengan *precision* 6.65% dan *recall* 98.36%.

Sama seperti metode Levenshtein Distance, dilakukan pengukuran performa dari fingerprint based distance dengan melihat waktu eksekusi yang diperlukan. Variabel kontrol dibuat sama yaitu jumlah karakter dalam setiap source code yaitu kurang lebih 2000 karakter sebelum dihapus komentar dan spasi oleh sistem. Hasil pengujian dapat dilihat pada Gambar 3.



Gambar 3 Perbandingan Waktu Eksekusi Fingerprint Based Distance

Waktu yang diperlukan *fingerprint based distance* relatif cepat jika dibandingkan dengan Levenshtein Distance, bahkan 1225 perbandingan dapat selesai kurang dari 1 detik. Lebih lengkapnya dapat dilihat pada Tabel 6.

Tabel 6 Perbandingan Waktu Eksekusi Fingerprint Based Distance

Jumlah file source code	Jumlah Perbandingan	Waktu (ms)
5	10	26
10	45	70
15	105	109
20	190	173
25	300	245
30	435	366
35	595	507
40	780	669
45	990	859
50	1225	976

Waktu yang diperlukan relatif stabil, sehingga memungkinkan untuk memproses *source code* dengan jumlah yang lebih banyak.

3.3 Pembahasan

Pembahasan yang akan dijelaskan adalah mengenai Levenshtein Distance dan fingerprint based distance dalam mendeteksi plagiarisme source code sesuai hasil pengujian yang telah dilakukan.

3.3.1 Pembahasan Levenshtein Distance

Dari hasil pengujian pada subbab sebelumnya, dapat terlihat bahwa Levenshtein Distance mampu untuk mendeteksi plagiarisme source code, dan mempunyai nilai *precision* dan *recall* yang cukup tinggi. Nilai *precision* tertinggi terdapat pada *threshold* 65% yaitu 98.33%. Sedangkan nilai *recall* tertinggi yaitu 98.36%.

Terdapat sedikit prediksi salah pada metode Levenshtein Distance. Di antaranya, Levenshtein Distance tidak mampu untuk mendeteksi apabila terjadi banyak perubahan pada struktur source code. Selain itu, perubahan ekspresi aljabar yang banyak juga akan sulit dideteksi oleh metode ini.

Salah satu kelebihan metode ini yang terlihat dari hasil pengujian adalah sedikitnya *false negative* yang dihasilkan. Ini berarti dari data yang digunakan, pasangan yang sebenarnya plagiat akan lebih sering dideteksi sebagai plagiat, dibandingkan sebagai non-plagiat.

Kelemahan dari metode ini yang terlihat dari hasil pengujian yang telah dilakukan adalah lebih sulit dalam mendeteksi perubahan nama variabel. Selain itu, perubahan urutan eksekusi program, misalnya inisialisasi variabel yang diacak-acak juga akan menurunkan nilai kemiripan. Perubahan ekspresi aljabar seperti menambahkan tanda kurung, atau mengubah alur perhitungan aritmatika juga akan menurunkan nilai kemiripan antar source code.

Terdapat beberapa alasan mengapa nilai *precision* dan *recall* yang didapat pada pengujian ini tinggi. Pertama, solusi dari soal yang digunakan cukup sederhana. Kedua, solusi dari soal

yang didapat terlalu pendek. Ketiga adalah pengetahuan mahasiswa yang mengerjakan hampir sama, sehingga menggunakan cara pengerjaan yang sederhana.

3.3.2 Pembahasan Fingerprint Based Distance

Dari hasil pengujian, dapat terlihat bahwa hasil dari fingerprint based distance mempunyai nilai precision dan recall yang relatif rendah jika dibandingkan dengan metode Levenshtein Distance. Nilai precision tertinggi terdapat pada skenario-10 yaitu 34.13%. Sedangkan nilai recall yang dihasilkan pada skenario yang sama termasuk tinggi, yaitu 93.44%.

Terdapat relatif banyak prediksi salah pada metode fingerprint based distance. Jika dilihat dari skenario-10, jumlah true positive adalah 57, sedangkan false positive berjumlah 110. Ini berarti terdapat banyak pasangan yang sebenarnya tidak plagiat namun teridentifikasi metode ini sebagai plagiat.

Secara teoritis, kelebihan metode ini adalah dapat mendeteksi jika terdapat perubahan nama variabel, maupun perubahan eksekusi kode. Kecilnya nilai precision dapat berarti terdapat banyak kesalahan pendeteksian dimana pasangan source code yang seharusnya tidak plagiat, dideteksi sebagai plagiat. Hal ini dapat disebabkan oleh pengetahuan mahasiswa yang mengerjakan hampir sama, sehingga menggunakan cara pengerjaan yang hampir sama pula. Karena source code ini diperoleh dari mahasiswa semester 2 (dua), maka pengetahuan mahasiswa masih terlalu sedikit untuk menjawab pertanyaan dengan cara yang tidak umum. Solusi dari soal "Bayar atau Kabur" yang dipilih sebagai data uji juga terlalu pendek pada umumnya.

Beberapa hal yang dapat dilakukan untuk memiminalisasi nilai kesalahan prediksi adalah dengan menggunakan nilai bobot yang kecil pada setiap fiturnya. Hal ini terlihat pada hasil pengujian, nilai bobot yang rendah lebih akurat untuk pengetahuan mahasiswa yang sederhana. Pada pengujian di atas menggunakan nilai threshold 65%. Mengatur nilai threshold dengan nilai yang lebih tinggi disarankan untuk meminimalisasi kesalahan prediksi.

3.3.3 Pembahasan Keseluruhan

Levenshtein Distance mempunyai akurasi yang lebih baik dikarenakan struktur program pada setiap mahasiswa yang tidak saling plagiat secara kasat mata tidak sama. Hal ini sangat mirip dengan cara pendeteksian plagiarisme secara manual, tetapi lebih cepat.

Fingerprint based distance dilihat dari hasil pengujian kurang baik jika diatur nilai threshold 65%. Penyebab utamanya adalah secara tersirat, maksud dari program adalah sama, sehingga metode ini salah prediksi dan menyebabkan pasangan source code yang tidak plagiat terdeteksi sebagai plagiat. Bobot dari setiap fitur juga sangat berpengaruh terhadap nilai precision dan recall yang dihasilkan.

Pendeteksian plagiarisme di antara 2 metode yang sudah diuji, maka dapat dikatakan Levenshtein Distance lebih akurat dibandingkan dengan fingerprint based distance. Fingerprint based distance dapat mendeteksi makna dalam program lebih baik, tetapi untuk pendeteksian tugas mahasiswa yang sederhana kurang cocok.

Dalam hal waktu yang diperlukan, metode Levenshtein Distance dapat dikatakan cukup lambat jika dibandingkan dengan metode fingerprint based distance. Hal ini disebabkan karena dalam metode Levenshtein Distance, perlu membuat matriks berukuran panjang source code A dikali panjang source code B, lalu akan diisi setiap selnya. Hal ini sangat berbeda dengan

metode fingerprint based distance yang secara langsung mencari token-token penting yang dibutuhkan.

Dengan demikian, untuk dapat mendeteksi plagiarisme source code dengan baik, sistem yang dibutuhkan adalah sistem yang dapat menghitung kemiripan semua pasang source code yang diunggah dengan menggunakan kedua metode yang tersedia sehingga dapat menunjang keputusan apakah source code yang ditulis adalah plagiat atau bukan dengan baik.

3.4 Pengujian dengan Data yang Berbeda

Selain menggunakan data yang sudah disebutkan, pengujian juga dilakukan untuk data yang berbeda. Ada 2 sumber data yang lain yaitu pekerjaan mahasiswa dengan kode soal "perkalian-kuda" (selanjutnya disebut Kumpulan Data-2) dan hasil pekerjaan pengguna Hackerrank dengan kode soal "Hackerland Radio Transmitters" (selanjutnya disebut Kumpulan Data-3).

Hasilnya, untuk kumpulan data-2, Levenshtein Distance menghasilkan akurasi 98.94%, dengan *precision* 93.75% dan *recall* 100.00%. Untuk *fingerprint based distance*, skenario dengan hasil terbaik ada pada percobaan 13, dengan akurasi 99.47%, *precision* 96.77% dan *recall* 100.00%.

Hasilnya, untuk kumpulan data-3, Levenshtein Distance menghasilkan akurasi 99.47%, dengan *precision*95.24% dan *recall* 100.00%. Untuk *fingerprint based distance*, skenario dengan hasil terbaik ada pada percobaan 13, dengan akurasi 88.95%, *precision* 48.78% dan *recall* 100.00%.

4. KESIMPULAN

Kesimpulan yang didapat dari semua pengujian yang telah dilakukan dalam penelitian ini adalah sebagai berikut.

1. Fitur yang paling berpengaruh dalam menentukan kemiripan source code adalah jumlah input, operator yang muncul, operand unik, dan jumlah baris.
2. Pada penelitian ini, metode Levenshtein Distance dengan nilai *threshold* 65% mencapai akurasi tertinggi sebesar 99.76% dengan nilai *precision* sebesar 98.33% dan *recall* sebesar 96.72%.
3. Pada penelitian ini, metode fingerprint based distance dengan nilai threshold 65% menghasilkan nilai akurasi tertinggi pada skenario-13 yaitu 92.08% dengan nilai precision sebesar 37.67% dan recall sebesar 90.16%.

Beberapa saran yang diberikan untuk pengembangan penelitian selanjutnya adalah sebagai berikut.

1. Dapat dilakukan pencarian nilai optimal untuk setiap nilai fitur yang belum dilakukan dalam penelitian ini.
2. Dapat dilakukan perbandingan nilai kemiripan dari suatu source code yang disalin ke bahasa pemrograman lain.

DAFTAR PUSTAKA

[1] J.I. Maletic and Marcus, 2000, Using latent semantic analysis to identify similarities in source code to support program understanding, *Proceedings of the 12th International Conference on 2000*

[2] Marcus and J.I. Maletic, 2001, Identification of high-level concept clones in source code, *Proceedings of the 16th International Conference on Automated Software Engineering (ASE 2001)*

[3] Bandara, Upul and Gamini Wijayarathna, 2011, A Machine Learning Based Tool for Source Code Plagiarism Detection, *International Journal of Machine Learning and Computing*, Vol.I, hal. 337

[4] Narayanan, Sandhya and Simi S., 2012, Source Code Plagiarism Detection and Performance Analysis Using Fingerprint Based Distance Measure Method, *International Conference on Computer Science & Education*, Vol. VII, hal.1066

[5] Manning, Christopher D., 2009, *An Introduction to Information Retrieval*, Cambridge University Press, Cambridge.