UNIVERSITAS INDONESIA

PENGEMBANGAN WEB CRAWLER UNTUK PENDETEKSIAN FRAUD PADA SITUS MERCHANT PT NUSA SATU INTI ARTHA

SKRIPSI

BAHY HELMI HARTOYO PUTRA
1606918124

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI SISTEM INFORMASI
DEPOK
BULAN TAHUN

UNIVERSITAS INDONESIA

PENGEMBANGAN WEB CRAWLER UNTUK PENDETEKSIAN FRAUD PADA SITUS MERCHANT PT NUSA SATU INTI ARTHA

SKRIPSI
Diajukan sebagai salah satu syarat untuk memperoleh gelar
S.Kom

BAHY HELMI HARTOYO PUTRA
1606918124

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI SISTEM INFORMASI
DEPOK
BULAN TAHUN

HALAMAN PERSETUJUAN

Judul	:	Pengembangan Web Crawler untuk Pendeteksian Fraud pada Situs Merchant PT Nusa Satu Inti Artha
Nama	:	Bahy Helmi Hartoyo Putra
NPM	:	1606918124

Laporan Skripsi ini telah diperiksa dan disetujui.

Tanggal Bulan Tahun

Adila Alfa Krisnadhi S.Kom., M.Sc	Pembimbing Kedua Anda
Pembimbing Skripsi	Pembimbing Skripsi

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Bahy Helmi Hartoyo Putra**
NPM	:	**1606918124**
Tanda Tangan	:	
Tanggal	:	**Tanggal Bulan Tahun**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Bahy Helmi Hartoyo Putra
NPM	:	1606918124
Program Studi	:	Sistem Informasi
Judul Skripsi	:	Pengembangan Web Crawler untuk Pendeteksian Fraud pada Situs Merchant PT Nusa Satu Inti Artha

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar S.Kom pada Program Studi Sistem Informasi, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing 1	:	Adila Alfa Krisnadhi S.Kom., M.Sc	(	)
Pembimbing 2	:	Pembimbing Kedua Anda	(	)
Penguji 1	:	Penguji Pertama Anda	(	)
Penguji 2	:	Penguji Kedua Anda	(	)

Ditetapkan di	:	Depok
Tanggal	:	Tanggal Bulan Tahun

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Bahy Helmi Hartoyo Putra
NPM	:	1606918124
Program Studi	:	Sistem Informasi
Fakultas	:	Ilmu Komputer
Jenis Karya	:	Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

Pengembangan Web Crawler untuk Pendeteksian Fraud pada Situs Merchant PT Nusa Satu Inti Artha

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyatan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : Tanggal Bulan Tahun
Yang menyatakan

(Bahy Helmi Hartoyo Putra)

KATA PENGANTAR

Pendahuluan. Ucapan Terima Kasih:

1. Pembimbing.
2. Dosen.
3. Instansi.
4. Orang tua.
5. Sahabat.
6. Teman.

Penulis menyadari bahwa laporan Skripsi ini masih jauh dari sempurna. Oleh karena itu, apabila terdapat kesalahan atau kekurangan dalam laporan ini, Penulis memohon agar kritik dan saran bisa disampaikan langsung melalui *e-mail* `emailanda@mail.id`.

Depok, Tanggal Bulan Tahun

Bahy Helmi Hartoyo Putra

ABSTRAK

Nama	:	Bahy Helmi Hartoyo Putra
Program Studi	:	Sistem Informasi
Judul	:	Pengembangan Web Crawler untuk Pendeteksian Fraud pada Situs Merchant PT Nusa Satu Inti Artha

Isi abstrak.

Kata kunci:
Keyword satu, kata kunci dua

ABSTRACT

Name	:	Bahy Helmi Hartoyo Putra
Program	:	Sistem Informasi
Title	:	Development of Web Crawler for Fraud Detection on PT Nusa Satu Inti Artha Merchant Site

Abstract content.

Key words:
Keyword one, keyword two

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE PROGRAM

BAB 1
PENDAHULUAN

Pada bab ini, akan dijelaskan tentang latar belakang dan permasalahan yang diselesaikan pada penelitian ini.

1.1 Latar Belakang

Financial technology atau yang biasa disebut *fintech* merupakan industri yang tengah berkembang pesat di Indonesia. Kemajuan yang signifikan ini ditunjukkan oleh dua sektor dominan yang merajai pasar *fintech* Indonesia, yaitu sektor pembayaran dan pendanaan. Berdasarkan data yang dihimpun oleh Asosiasi Fintech Indonesia, laju pertumbuhan nilai transaksi meningkat sebanyak 15% dari Juni 2018 ke Juni 2019 pada sektor pembayaran. Sedangkan pada sektor pendanaan, laju pertumbuhan meningkat secara drastis sebanyak 97.6% dari Desember 2018 ke Juni 2019 [1].

Berkembanganya layanan *fintech* mendorong perusahaan-perusahaan baru bermunculan dan ikut serta dalam menyelenggarakan layanan *fintech*. Pada Desember 2019, tercatat sebanyak 61 perusahaan telah tergabung pada sektor pembayaran dan 164 perusahaan pada sektor pendanaan [1]. Meskipun sektor pembayaran bukan merupakan sektor *fintech* dengan pemain dan pertumbuhan terbesar saat ini, namun potensi untuk berkembang pada sektor ini sangat memungkinkan untuk terjadi.

Sebuah riset yang dikeluarkan oleh Metra Digital Innovation Venture dan Mandiri Capital memprediksi bahwa pada tahun 2020 angka *Gross Transaction Volume* (GTV) industri pembayaran *mobile* di Indonesia akan mencapai US$30 miliar. Hal ini menunjukkan kenaikan yang cukup signifikan sebesar 159% dari tahun 2016. Selain itu, berkembangnya industri pembayaran *mobile* ini juga diprediksi akan berkontribusi terhadap total *Gross Domestic Product* (GDP) Indonesia sebanyak 3% [2].

Dari sekian banyak perusahaan *fintech* yang bergerak pada sektor pembayaran, salah satu yang cukup lama berada di sektor ini adalah PT Nusa Satu Inti Artha, atau lebih dikenal dengan DOKU, DOKU berdiri pada tahun 2007 dan sejak saat itu fokus perusahaan adalah mengembangkan dan menyediakan layanan pembayaran secara *online* kepada pelaku bisnis dan masyarakat. Saat ini, DOKU telah terkoneksi dengan lebih dari 20 bank dan institusi finansial di Indonesia. Layanan pembayaran DOKU meliputi penyediaan metode pembayaran bagi *merchant* dan pelanggannya (*payment gateway*) dan layanan transfer melalui berbagai *channel* seperti kartu kredit, transfer bank, *e-wallet* dan

juga pembayaran *offline* di beberapa toko-toko pilihan.

Saat ini DOKU telah digunakan oleh lebih dari 100.000 *merchant* dalam kedua layanannya, yaitu *payment gateway* dan juga *transfer service*. Untuk dapat bergabung menjadi *merchant* yang menggunakan layanan DOKU, pemiliki *merchant* perlu melalui serangkaian proses registrasi. Proses registrasi tersebut dinamakan *Know Your Business* (KYB). Dalam mengoptimalkan proses registrasi ini, DOKU membentuk sebuah tim khusus untuk mengurus proses *end-to-end* registrasi, yang dinamakan dengan tim EDU atau *Early Detection Unit*. Tugas utama dari tim ini adalah mengevaluasi *merchant* yang melakukan proses registrasi dan memastikan bahwa *merchant* tersebut telah memenuhi syarat & ketentuan yang ditentukan oleh DOKU. Jika *merchant* dianggap telah sesuai, maka *merhcant* dapat melanjutkan ke proses integrasi sistem pembayaran, namun apabila tidak sesuai maka *merchant* akan diberikan peringatan dan diminta keterangan lebih lanjut.

Salah satu hal penting dalam proses pemeriksaan ini adalah kelengkapan komponen situs *merchant* yang bersangkutan. Ada beberapa syarat yang ditetapkan oleh tim EDU untuk sebuah situs agar dapat dianggap sebagai situs yang valid. Diantaranya adalah pemeriksaan ketersediaan tautan utama (*contact us, about us, terms & condition*), jumlah tautan tidak aktif, ketersediaan informasi kontak, dan juga adanya *policy* terkait pengembalian barang dan uang. Keseluruhan proses ini masih dilakukan secara manual oleh anggota tim EDU. Menurut keterangan pihak DOKU[1], *rate* pemeriksaan sebuah situs yang berada pada *waiting list* registrasi berkisar antara 35 - 50 situs/hari, atau sekitar 5 situs/jam[2]. Selain itu, untuk mencapai *rate* tersebut dibutuhkan 2 orang tim EDU yang ikut serta dalam pengecekan. Rate pengecekan ini dianggap masih terlalu lambat dan masih melibatkan terlalu banyak sumber daya manusia. Di satu sisi diketahui juga bahwa rata-rata *merchant* yang mendaftar mencapai 100 *merchant* per hari. Oleh karena itu, pada tahun 2020 tim DOKU mencoba untuk melakukan otomasi terhadap proses ini, dengan harapan *rate* pemeriksaan harian dapat meningkat dan beban pekerjaan yang dimiliki tim EDU dapat terbantu.

Dengan adanya *web crawler*, rata-rata pengecekan satu buah situs hanya memakan waktu sekitar 15 detik[3]. Dengan *rate* 15 detik/situs, maka dalam satu jam *web crawler* dapat melakukan pengecekan sebanyak 240 situs. Sehingga, jika dikonversikan ke *rate* harian, dengan menggunakan *web crawler*, tim DOKU dapat mengecek sebanyak 1620 situs. Angka ini jauh meningkat dibandingkan pengecekan manual. Sehingga, efisiensi

[1]Diwakili oleh Pak Reza Farasdak Abdat, Product Owner DOKU

[2]Dengan perhitungan 9 jam kerja dan dengan asumsi bahwa seluruh jam kerja staf yang melakukan pengecekan didedikasikan untuk ini

[3]Diambil dari rata-rata percobaan yang dilakukan bersama tim DOKU terhadap 100 situs *merchant* yang disampel secara *random* dengan kecepatan internet 20 MBps

rate pengecekan dapat ditingkatkan sebesar 4700%[4].

Pengecekan yang dilakukan oleh *web crawler* meliputi pengecekan kelengkapan komponen situs dan prediksi kategori situs (*fraud* atau tidak *fraud*). Akurasi model *classifier* yang dimiliki oleh *web crawler* juga sudah cukup baik dengan angka 0.864 dan AUC sebesar 0.953.

1.2 Permasalahan

Sebutkan permasalahan penelitian Anda dari latar belakang tersebut.

1.2.1 Definisi Permasalahan

Berikut ini adalah rumusan permasalahan dari penelitian yang dilakukan:

- Bagaimana cara membuat sebuah web crawler untuk melakukan ekstraksi data pada situs *merchant*?
- Fitur apa saja yang dapat digunakan untuk mendeteksi situs *merchant* yang *fraud*?
- Bagaimana performa model yang dibuat dalam mendeteksi situs *merchant* yang *fraud*?

1.2.2 Batasan Permasalahan

Berikut ini adalah asumsi yang digunakan sebagai batasan penelitian ini:

- Situs *merchant* yang digunakan dalam penelitian adalah situs yang terdaftar dalam proses registrasi *online merchant* PT Nusa Satu Inti Artha.
- Output dari klasifikasi adalah dua *class* yaitu *fraud* dan tidak *fraud* beserta probabilitasnya.

1.3 Tujuan Penelitian

Berikut ini adalah tujuan penelitian yang dilakukan:

- Membuat sebuah *web crawler* yang bisa melakukan ekstraksi data pada sebuah situs *merchant*.
- Mendapatkan fitur-fitur yang bisa digunakan untuk mendeteksi sebuah situs *merchant* yang *fraud*.

[4] Asumsi *best case*, dengan *rate* 15 detik/situs

- Melihat performa model yang dibuat dalam mendeteksi situs *merchant* yang *fraud*.

1.4 Tahapan Penelitian

Tahapan-tahapan yang dilakukan penulis dalam melakukan penelitian adalah sebagai berikut:

1. Studi literatur
   Pada tahap ini, dipelajari teori-teori yang terkait dengan penelitian ini untuk mendapatkan konsep dasar yang dibutuhkan dalam mencapai tujuan penelitian.

2. Desain dan implementasi
   Pada tahap ini, dilakukan desain dari aplikasi *web crawler* yang akan dibuat dan pemanfaatan *machine learning* yang digunakan untuk melakukan klasifikasi hasil akhir. Hasil prediksi akan dianalisa dan ditarik sebuah kesimpulan dari keseluruhan penelitian ini.

1.5 Sistematika Penulisan

Sistematika penulisan laporan adalah sebagai berikut:

- Bab 1 PENDAHULUAN
  Bab ini mencakup latar belakang, cakupan penelitian, dan pendefinisian masalah.

- Bab 2 STUDI LITERATUR
  Bab ini mencakup pemaparan terminologi dan teori yang terkait dengan penelitian berdasarkan hasil tinjauan pustaka yang telah dilakukan.

- Bab 3 DESAIN DAN IMPLEMENTASI
  Bab ini menjelaskan desain dari aplikasi *web crawler* yang dibuat oleh penulis dan proses implementasinya pada perusahaan.

- Pecah bab 3 menjadi dua bab (desain dan implementasi)

- Bab 4 EKSPERIMEN DAN ANALISIS
  Bab ini menjelaskan hasil eksperimen yang dilakukan untuk menguji kebeneran implementasi dari *web crawler* yang dibuat.

- Bab 5 PENUTUP
  Bab ini mencakup kesimpulan akhir penilitian dan saran terkait pengembangan selanjutnya.

BAB 2
STUDI LITERATUR

Untuk memulai penelitian, dibutuhkan kerangka berpikir yang sesuai untuk permasalahan yang ingin dipecahkan. Untuk membentuk kerangka berpikir yang sesuai, perlu dikaitkan dengan hasil studi literatur yang telah dilakukan. Oleh karena itu, pada bab ini, akan dijelaskan hasil studi literatur yang telah dilakukan yang telah dikaitan dengan kerangka kerja untuk penelitian ini.

2.1 *Financial Technology*

Financial technology atau biasa disingkat *fintech* adalah sebuah industri finansial yang mengaplikasikan teknologi untuk kemajuan kegiatan finansialnya [3]. Dalam definisi lain, *fintech* juga dapat disebut sebagai sebuah aplikasi, proses, produk, atau model bisnis baru dalam industri finansial, yang terdiri dari satu atau lebih layanan keuangan dan dilaksanakan melalui proses *end-to-end* melalui internet [4]. Perusahaan yang bergerak pada bidang *fintech* dapat terdiri dari berbagai macam perusahaan, mulai dari *startup*, perusahaan jasa keuangan, ataupun perusahaan teknologi yang menyediakan solusi bagi perusahaan jasa keuangan yang telah ada.

2.1.1 *Payment Financial Technology*

Industri *fintech* sangat luas dengan berbagai macam layanan yang ditawarkan. Sehingga *fintech* dapat dikategorikan menjadi beberapa segmen, salah satunya adalah *payment* [5]. Industri *fintech payment* atau sebagian menyebutnya *digital payment* berfokus pada pelayanan dan penyediaan sistem pembayaran berbasis digital. Contoh dari layanan ini adalah layanan dompet digital, *payment gateway*, dan *transfer services*.

2.1.2 *Know Your Business*

Know Your Business merupakan suatu rangkaian proses yang bertujuan untuk melakukan verifikasi terhadap suatu bisnis [6]. KYB biasa diterapkan pada sebuah perusahaan *fintech* dalam melakukan *screening* terhadap calon mitra kerjanya, baik itu instansi finansial ataupun *merchant*. Hal ini merupakan sebuah protokol standar pada jenis bisnis B2B. KYB dilakukan untuk menghindari terjadinya kerjasama dengan calon mitra yang memiliki tendensi kriminal seperti penipuan identitas (*identity fraud*), bisnis palsu (*fake*

merchant fraud), pencucian uang, pendanaan teroris, perusahaan tempurung, dan lain-lainnya.

2.2 *Web Crawler*

Web crawler atau dalam terminologi singkat disebut *crawler* adalah sebuah bot yang memanfaatkan internet untuk melakukan penelusuran terhadap *World Wide Web* secara sistematis [7]. Pemanfaatan *web crawler* dapat meliputi berbagai kebutuhan, seperti *web indexing*, otomasi *maintenance* situs, dan pengekstrakan data situs (*web scraping*).

2.2.1 *Crawl Frontier*

Crawl Frontier merupakan sebuah bagian dari sistem *crawling* yang mengatur *logic*, urutan, dan ketetapan yang akan diterapkan oleh sebuah *web crawler* dalam mengunjungi sebuah situs. Hal pertama yang akan dilakukan oleh sebuah *crawl frontier* adalah mengumpulkan URL yang akan dituju. Kumpulan dari URL ini dinamakan *seeds* [8]. Seeds selanjutnya dapat dimanfaatkan kedepannya sesuai dengan *logic* yang telah ditentukan oleh *frontier*.

2.2.2 *Web Driver*

Web Driver adalah sebuah *interface* dan protokol yang dapat digunakan untuk mengatur behavior sebuah *web browser*. WebDriver biasa dimanfaatkan untuk melakukan *automatic testing* dengan menuliskan serangkaian instruksi menggunakan sebuah bahasa pemrograman. Bahasa pemrograman yang didukung meliputi Java, .Net, PHP, Python, Perl, dan Ruby. Instruksi yang telah ditulis akan dieksekusi dan akan menjalankan *web browser* yang dipilih sesuai dengan *logic* yang dituliskan. Beberapa *web browser* yang dapat dikontrol melalui WebDriver adalah Google Chrome, Mozilla Firefox, Internet Explorer, Safari, Opera, dan GhostDriver [9].

2.2.3 *HTTP Programming*

HyperText Trasnfer Protocol (HTTP) adalah sebuah *application-level protocol* yang digunakan untuk sistem informasi terdistribusi dan kolaboratif [10]. Salah satu kegunaan HTTP yang sering digunakan adalah HTTP *request*. Cara kerja sebuah HTTP *request* adalah dengan mengirimkan sebuah *request* ke server yang dituju, dimana data tersebut disimpan, lalu server mengembalikan respon berupa *status code* dan konten (data) yang

diminta. Siklus seperti ini disebut sebagai *request-response cycle*[1].

Untuk dapat melakukan *request*, pengguna harus menuliskan sebuah program komputer yang dapat mememungkinkan proses komunikasi antara komputer pengguna dan server dapat terjadi. Hal ini disebut sebagai *HTTP Programming*. Saat ini ada banyak *library* yang dapat dimanfaatkan untuk melakukan hal ini, salah satunya *library requests*[2] yang dibangun di atas bahasa pemrograman Python.

2.3 *Web Scraping*

Web scraping merupakan sebuah terminologi yang digunakan untuk menggambarkan proses esktraksi data dari sebuah situs [11]. *Web scraping* dapat digunakan untuk beberapa kebutuhan seperti *information extraction*, *content analysis*, *content ranking*, *price engine*, dan banyak hal lainnya. Proses *scraping* dapat dilakukan secara manual oleh pengguna atau disematkan secara otomatis di dalam sebuah *web crawler*.

Web scraping yang dilakukan secara manual oleh pengguna biasanya dilakukan untuk keperluan tidak berulang (*single use*) dan memang spesifik terhadap suatu situs. Sedangkan, untuk melakukan ekstraksi data secara berulang dan dalam skala yang besar, maka proses *scraping* harus dijalankan menggunakan bantuan *bot* atau biasa disebut *web crawler*.

Web crawler berfungsi sebagai sebuah *bot* yang akan membentuk sebuah jaring-jaring, jalan, atau peta, untuk mengunjungi berbagai macam-macam halaman yang ada di situs tersebut [12]. Setiap *crawler* mengunjungi sebuah halaman, maka proses *scraping* akan dijalankan. Informasi ataupun data yang ada dalam halaman tersebut akan diekstrak dan disimpan atau langsung diproses sesuai dengan kebutuhan pengguna. Halaman web biasanya dibangun di atas sebuah *text-based mark-up language* seperti HTML dan XHTML. Karena hal ini, tidak semua informasi yang diekstrak biasanya dapat langsung dimengerti dan diinterpretasikan oleh pengguna. Sehingga perlu ada proses lanjutan lainnya yang dapat menyempurnakan proses *scraping* yaitu dengan melakukan *HTML parsing*.

2.3.1 *HTML Parsing*

Objek yang diterima dalam sebuah *HTTP request* merupakan sebuah entitas *HTTP response message*. Di dalam *HTTP response message*, terdapat beberapa komponen yang termuat, diantaranya *status code*, *header*, dan *body* [10]. *Status code* merepresentasikan status balasan dari *request* yang dikirimkan oleh pengguna. *Status code* yang lazim ditemui dalam melakukan proses *crawling* dapat dilihat pada Tabel 2.1

[1]Menurut situs pembelajaran daring *w3schools*, https://www.w3schools.com/whatis/whatis_http.asp
[2]https://pypi.org/project/requests/

Status Code	Deskripsi
0	Blocked by Robots.txt/DNS Lookup Failed/-Connection Timeout/Connection Refused/No Response
200	OK
301	Moved Permanently
302	Moved Temporarily
400	Bad Request
403	Forbidden
404	Page Not Found
410	Removed
429	Too Many Requests
500	Internal Server Error

Tabel 2.1: Contoh *response code* yang dikembalikan *HTTP response message*

Terkait konten ataupun informasi utama dari halaman yang dituju, akan dikembalikan di dalam *body*. Untuk dapat melakukan ekstraksi informasi yang ada pada *body* dari *HTML response* maka perlu dilakukan teknik *HTML parsing*. Salah satu *library* yang dapat dimanfaatkan dalam melakukan *parsing* adalah BeautifulSoup[3]. BeautifulSoup memungkinkan untuk mengubah dokumen HTML yang kompleks menjadi objek Python yang berbentuk *tree* dan dapat dengan mudah dinavigasikan oleh pengguna dengan cara mengakses tag-tag yang ada pada *tree* HTML tersebut.

2.3.2 *Regular Expression*

Regular expression atau biasa disebut *regex* merupakan sebuah konstruksi teks yang disusun untuk mencocokkan sebuah pola tertentu. Regex bisa digunakan untuk melakukan operasi *matching* (pencocokan), *locating* (pelokasian), dan manipulating (manipulasi), pada sebuah *string*. Pola (*pattern*) yang dapat dibuat menggunakan regex sangat banyak, beberapa contoh yang paling sering digunakan tertulis dalam Tabel **??** di bawah ini.

[3]https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Regex	Contoh Syntax	Contoh String	Penjelasan
\d	urutan_\d	urutan_5	Mencocokan satu digit angka 0-9
[abc]	karakter_[abc]	karakter_a	Mencocokan karakter ”a” atau ”b” atau ”c” sebanyak 1 kali
[abc]+	karakter_[abc]+	karakter_aaa	Mencocokan karakter ”a” atau ”b” atau ”c” sebanyak 1 kali sampai tidak terbatas
[^abc]	karakter_[^abc]	karakter_s	Mencocokan karakter selain ”a” atau ”b” atau ”c” sebanyak 1 kali
\s	satu\sdua	satu dua	Mencocokan karakter *whitespace* (*spasi, tab, newline*)
[a-zA-z]	karakter_[a-zA-z]	karakter_Y	Mencocokan karakter dari *range* ”a” sampai ”z” atau ”A” sampai ”Z” sebanyak 1 kali
(word1|word2)	ini (word1|word2)	ini word1	Mencocokan kata ”word1” atau ”word2” sebanyak 1 kali

Tabel 2.2: Contoh penggunaan syntax *regex*

Pada Tabel 2.2, yang ditampilkan hanya merupakan beberapa *regex* yang biasa dan sering digunakan dan termuat dalam *common regex usage* di beberapa situs. Penggunaan *syntax* regex dalam bahasa pemrograman Python sendiri dapat dilakukan menggunakan *built-in library* yang dimiliki, yang dinamakan *re*[4]. Ada beberapa metode yang dapat dilakukan menggunakan *re*, diantaranya sebagai berikut.

- **re.match()**: Menentukan apakah regex yang dibuat dapat menemukan *pattern* yang cocok pada awal string yang diberikan.
- **re.search()**: Melakukan *scanning* di keseluruhan string, mencari apakah ada *pattern* yang sesuai dengan regex yang ditentukan.

[4]https://docs.python.org/3/howto/regex.html

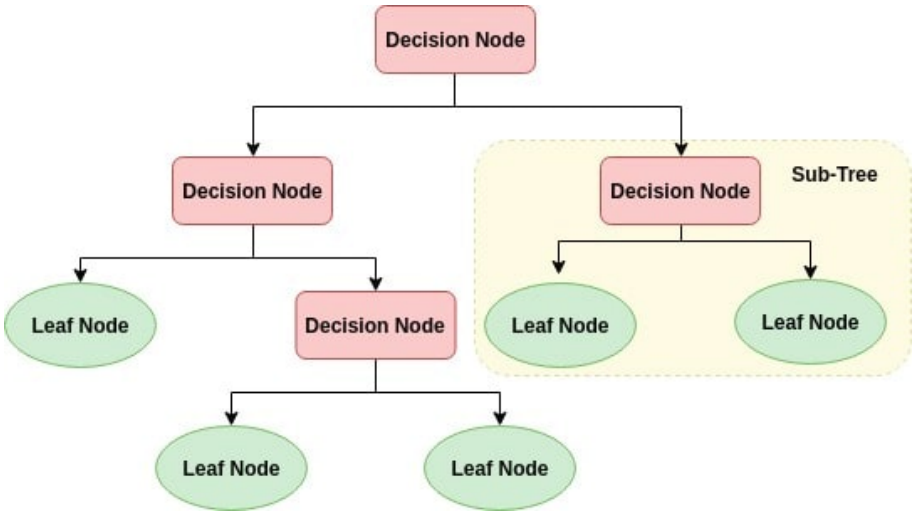
- **re.findall**(): Mencari semua substring dalam sebuah string dimana *pattern* regex dapat ditemukan. Fungsi ini mengembalikan semua substring dalam bentuk list.

2.4 *Classifier*

Classifier merupakan sebuah algoritma yang memanfaatkan atau mempelajari input (*training data*) untuk melakukan prediksi terhadap label kategori (*class*) dari sebuah *data point* yang diberikan (*testing data*) [13]. *Classifier* banyak dimanfaatkan untuk banyak kasus di dunia nyata, seperti klasifikasi *spam email*, klasifikasi transaksi *fraud*, dan banyak hal lainnya yang merupakan implementasi dari *pattern recognition*[5].

2.4.1 Decision Tree Classifier

Decision Tree Classifier merupakan salah satu model prediktif yang biasa digunakan dalam *data mining*, *machine learning*, dan statistik. *Decision tree* menerapkan algoritma *non-parametric unsupervised learning* dan dapat digunakan untuk persoalan regresi maupun klasifikasi. *Classifier* jenis ini akan membangun sebuah *tree* yang akan melakukan partisi secara rekursif, menciptakan sebuah *tree* baru (*sub-tree*), sampai homogenitas dari setiap node terminimalisir. Strukturnya yang mudah dimengerti dan menyerupai *flowchart* memudahkan penggunanya untuk mengerti dan menginterpretasikan hasil klasifikasinya. Struktur dari *decision tree* dapat dilihat lebih jelas pada Gambar 2.1 dibawah ini.



Gambar 2.1: Struktur *tree* pada Decision Tree Classifier, diambil dari DataCamp Community

[5]Lebih lengkapnya tentang *pattern recognition* dapat diakses pada tautan berikut https://deepai.org/machine-learning-glossary-and-terms/pattern-recognition

Pada sebuah *decision tree*, *node* yang berada paling atas disebut dengan *root node*. *Root node* merupakan partisi pertama dari keseluruhan data yang ada. Berikutnya, untuk setiap *node* yang berada di sisi dalam (*internal node*), merepresentasikan atribut (*feature*) yang digunakan pada model. Lalu, setiap percabangan (*branch*) merepresentasikan *decision rule* yang dibuat dan setiap *node* terluar (*leaf node*) merepresentasikan hasil akhir (*outcome*) dari partisi data.

$$Gini\ impurity = \sum_{i=1}^{k} p_i(1 - p_i) cross - entropy = -\sum_{i=1}^{k} p_i log(p_i)$$

Gambar 2.2: Cara menghitung *gini impurity* pada Decision Tree Classifier

Dalam kasus *outcome categorical*, salah satu cara yang paling umum dalam melakukan *splitting* adalah dengan menghitung *gini impurity*. Pada persamaan di Gambar 2.2, *k* mewakili jumlah *class* (kategori) yang ada pada *dataset*. Sedangkan p_i merepresentasikan proporsi *case* (data) yang merupakan bagian dari *class* i. Cara untuk melakukan *splitting* adalah dengan menghitung *gini impurity* pada *initial node* dan pada kedua *child node* yang dihasilkan. Jika hasil *weighted sum gini impurity* dari kedua *child node* lebih kecil daripada *gini impurity* yang ada pada *initial node*, maka *split* dianggap menghasilkan *improvement* untuk lebih mempartisi data dan *split* akan dilakukan [14].

Untuk menggunakan Decison Tree Classifier dalam Python, *scikit-learn*[6] telah menyediakan sebuah *library* yang dapat digunakan secara komprehensif untuk melakukan *training*, *tuning*, dan *predicting*. Ada beberapa parameter penting yang perlu dipahami dalam menggunakan *library* ini, diantaranya adalah:

- **criterion** (“gini”, “entropy”, default=”gini”)
  Menentukan standar hitung yang digunakan untuk melakukan *split*. Untuk *split* berdasarkan *gini impurity*, dapat menggunakan ”gini”. Sedangkan untuk menghitung berdasarkan *information gain*, dapat menggunakan ”entropy”.

- **max_depth** (”int”, default=None)
  Menentukan kedalaman maksimum dari *tree* yang dibuat. Jika dibiarkan None (*default*), maka *tree* akan terus berekspansi sampai semua *leaf* sudah murni atau jumlah *dataset* pada *leaf* sudah lebih sedikit dari jumlah yang ditetnukan pada *min_samples_split*

- **min_samples_split** (int, float, default=2)
  Jumlah minimum data pada sebuah *leaf* yang dibutuhkan untuk sebuah *leaf* agar

[6]https://scikit-learn.org/

dapat melakukan *split*. Jika yang dimasukkan adalah *integer*, maka angka tersebut akan menjadi angka minimum data yang dimaksud. Sedangkan apabila nilai *float* yang dimasukkan, maka akan ditransalasikan menjadi presentase minimum sampel.

- **min_samples_leaf** (int, float, default=1)
  Jumlah minimum data yang harus ada pada sebuah *leaf*. *Split* hanya dapat dilakukan apabila pada jumlah data pada *child node* yang dihasilkan berisi minimum sejumlah/di atas jumlah yang ditetapkan.

2.4.2 Random Forest Classifier

Random Forest Classifier adalah sebuah *tree-based* model yang dikembangkan dari algoritma dari Decision Tree Classifier. Faktor yang membedakan diantara keduanya adalah jumlah *tree* yang dibangun saat proses *fitting*. Jika pada Decision Tree Classifer *tree* yang dibangun hanya berjumlah satu, pada Random Forest Classifier, seperti namanya, *forest*, algoritma ini akan membangun lebih dari satu *tree*. Pada metode ini, proses *training* dilakukan menggunakan metode *bagging*. Ide umum dari metode *bagging* adalah dengan menggabungkan berbagai macam *learner* untuk menciptakan hasil keseluruhan yang lebih baik. Dengan adanya diversitas *tree* dan fitur-fitur yang dipilihnya, secara umum, performa yang dihasilkan model akan menjadi lebih baik.

Model ini disebut *random* dikarenakan ada sebuah perilaku acak yang terjadi dalam membangun kumpulan *tree*nya. Alih-alih memilih fitur terpenting dari seluruh fitur yang ada dalam melakukan *splitting* pada sebuah *node*, *random forest* akan menentukan terlebih dahulu subset fitur yang akan digunakan secara *random* baru melakukan pemilihan fitur dengan kualitas split terbaik dari subset tersebut [15].

Sebuah *tree* akan dibangun sesuai parameter yang ditentukan layaknya yang terjadi pada Decision Tree Classifier. Jadi pada dasarnya, parameter yang dapat digunakan untuk kedua model ini dapat dibilang sama. Namun, model ini tidak akan berhenti pada satu *tree*, melainkan akan kembali membangun *tree* berikutnya dengan skema pemilihan fitur acak, sampai jumlah *tree* yang diinginkan tercapai. Saat semua *tree* sudah dibangun, maka nantinya proses *predicting* akan melalui proses *voting*. Proses *voting* adalah proses pemilihan suara terbanyak dari keseluruhan hasil prediksi *tree* yang ada. Misalkan, pada sebuah proses *fitting* Random Forest Classifier menghasilkan 10 *tree* dan masing-masing *tree* telah memiliki hasil prediksinya tersendiri terhadap sebuah *test data*. Jika 8 dari 10 *tree* memprediksi bahwa *test data* tersebut milik sebuah *class* 1 dan 2 sisanya memprediksi *class* 0, maka model akan mengembalikan hasil prediksi dengan *vote* terbanyak, yaitu *test data* milik *class 1*.

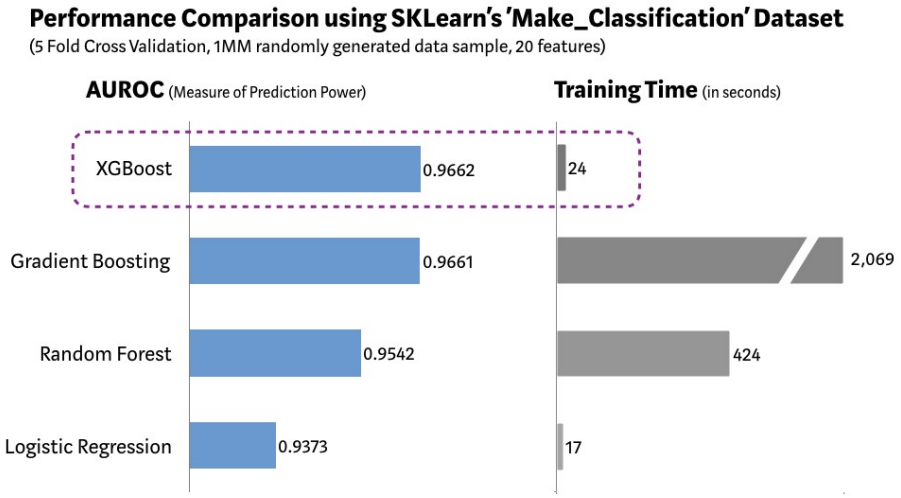
Pemilihan dataset dalam proses *fitting* Random Forest Classifier juga dapat dilakukan

dengan teknik *bootstrap*. Bootstrap merupakan sebuah metode *sampling* statistika, yaitu dengan cara mengambil sebuah proporsi sampel pada dataset secara iteratif dan acak dengan menerapkan *replacement* [16]. Salah satu keunggulan dari metode ini adalah bisa dihasilkannya lebih banyak data dalam melakukan estimasi tanpa harus mengumpulkan lebih banyak *training data*.

2.4.3 Extreme Gradient Boost Classifier

Extreme Gradient Boost atau biasa dikenal dengan XGBoost adalah sebuah algoritma yang baru-baru ini dikembangkan dan mendominasi pada berbagai bidang *applied machine learning* dan di beberapa kompetisi-kompetisi daring [17]. XGBoost sendiri merupakan sebuah implementasi dari algoritma Gradient Boosted Decision Tree yang telah dimodifikasi sedemikian rupa sehingga memiliki performa yang lebih baik dalam perihal kecepatan dan performa. Model ini pertama kali dikenalkan oleh seorang *researcher* yang memiliki spesialisasi dalam bidang *machine learning*, Tianqi Chen[7].

XGboost merupakan sebuah metode *ensemble learning* [18]. Terkadang, bergantung pada satu model *machine learning* tidak cukup. *Ensemble learning* menawarkan sebuah solusi yang sistematis untuk menggabungkan kemampuan prediksi dari berbagai macam *learner*. Hasil dari penggabungan ini adalah sebuah model tunggal yang memberikan keluaran yang diaggregasikan dari beberapa model (*learner*). Dengan melakukan aggregasi dan penggabungan ini, maka hasil prediksi yang dibuat akan menjadi lebih baik dan memiliki variansi yang rendah.



Gambar 2.3: *Training time* dan performa XGBoost dibandingkan model lain

XGBoost memiliki dukungan dari komunitas Data Scientist yang kuat. Saat ini *open*

[7]https://tqchen.com/

source projek XGBoost telah memiliki lebih dari 350 kontributor dan 3600 *commit* pada *repository* GitHub[8]. Konsekuensi dari pengembangan dan kontribusi yang luas dari algoritma ini adalah:

- Dapat digunakan dalam berbagai macam permasalahan: Dapat digunakan pada permasalah regresi, klasifikasi, dan *ranking*.
- Portabilitas: Dapat dijalankan dengan lancar di berbagai macam OS (Windows, Linux, Mac OS) dan di berbagai macam *cloud server* seeprt AWS, Azure, dan ekosistem lain seperti Spark, Flink.
- Bahasa pemrograman: Dapat dijalankan di berbagai macam bahasa pemrograman diantaranya C++, Python, R, Java, Scala, dan Julia
- Kecepatan dan Ketepatan: Dapat dilihat pada Gambar 2.3 bahwa menggunakan *dataset* yang sama, XGBoost dapat menampilkan hasil yang lebih baik dengan nilai 0.96 hanya dengan waktu training selama 24 detik

2.4.4 Bernoulli Naive Bayes Classifier

Metode-metode Naive Bayes adalah sekumpulan metode *supervised learning* yang didasari pada penerapan teorema Bayes. Teorema Bayes menerapkan asumsi *naive*, yaitu adanya asumsi independensi yang kondisional diantara satu fitur dengan fitur lainnya diketahui nilai dari *class variable*. Teorema Bayes dapat dinyatakan sebagai berikut:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Pada teorema Bayes, y merupakan *class label* yang diketahui dan x1 sampai dengan xn merupakan *feature vector* yang dependen. Teorema ini berlaku pada keseluruhan Naive Bayes Classifiers yang ada, umumnya yang membedakan antara satu jenis Naive Bayes Classfier dan lainnya adalah asumsi dari masing-masing jenis terhadap distribusi dari $P(x_i \mid y)$[19].

Walaupun Naive Bayes dikatakan sebagai algoritma yang sangat sederhana, namun algoritma ini telah banyak terbukti cukup baik dalam menghadapi berbagai macam permasalahan nyata di dunia, seperti pada kasus *document classification* dan *spam filtering* [20]. Jumlah *training data* yang dibutuhkan algoritma ini tidak perlu banyak untuk dapat melakukan prediksi yang baik. Selain itu, algoritma ini juga cepat dalam melakukan proses *training* dan *predicting* dibandingkan algoritma rumit lainnya.

[8]https://github.com/dmlc/xgboost/

Spesifik terhadap salah satu jenis dari Naive Bayes, yaitu Bernoulli Naive Bayes algoritma ini merupakan algoritma yang menerapkan cara Naive Bayes melakukan *training* dan *classification* terhadap data-data yang terdistribusikan secara Bernoulli multivariat. Maksudnya dari Bernoulli multivariat disini adalah saat ada berbagai macam fitur-fitur yang dijadikan input dan setiap fitur yang dimiliki tersebut bersifat *binary-valued* (Bernoulli, *boolean*). Jika fitur-fitur tersebut tidak bersifat biner, maka binerisasi akan dilakukan terhadap nilai-nilai dari fitur-fitur yang tidak bersifat biner tersebut.

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

Decision rule untuk Bernoulli Naive Bayes ditampilkan pada persamaan di atas. Dimana pada Bernoulli Naive Bayes, kemunculan dari fitur *i* tidak diacuhkan seperti pada Multinomial Naive Bayes. Pada algoritma Bernoulli, ketidakmunculan fitur *i* diberikan penalti namun tetap dimasukan kedalam perhitungan.

Pengimplementasian algoritma Bernoulli Naive Bayes dapat menggunakan model BernoulliNB[9] pada *scikit-learn*. Ada beberapa parameter penting yang perlu dipahami dalam menggunakan *library* ini, diantaranya adalah:

- **alpha** (float, default="1.0")
  Additive (Laplace/Lidstone) *smoothing* parameter pada *decision rule* (0 untuk tidak ada *smoothing* sama sekali)

- **binarize** (float, None, default=0.0)
  Threshold yang dapat ditentukan untuk melakukan binerisasi dari *sample features* yang bukan merupakan *binary-valued*. Jika None, maka diasumsikan fitur yang diberikan sudah merupakan fitur biner.

- **fit_prior** (boolean, default=True)
  Menentukan apakah model perlu mempelajari probabilitas *class prior* atau tidak. Jika False, maka *prior* uniform untuk dua *class* akan digunakan.

2.5 Validasi model

Untuk mengukur kualitas dari sebuah model yang telah dibuat, maka sebuah validasi harus dilakukan. Validasi model merupakan sebuah kunci untuk dapat mengembangkan model yang telah dibuat secara iteratif. Dalam melakukan validasi, ada beberapa teknik yang dapat diadopsi, diantaranya adalah dengan melakukan *nested cross validation* dan melihat metrik evaluasi suatu model.

[9]https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html

2.5.1 *Nested Cross Validation*

Nested Cross validation merupakan salah satu teknik validasi model yang dikembangkan dari teknik *cross validation* pada umumnya. *Cross validation* sendiri merupakan sebuah teknik validasi yang digunakan untuk mengestimasi kualitas model secara tidak bias. Hal ini dikarenakan *cross validation* dapat menghindari *overfitting* dan *selection bias* pada model [21]. Metode ini bekerja dengan membagi dataset menjadi k bagian. Dari k bagian yang ada, akan diambil 1 bagian sebagai *test set* dan sisanya (k-1 bagian) lainnya akan menjadi *train set*. Pembagian ini akan dilakukan *looping* sebanyak k kali, sehingga semua bagian data berkesempatan menjadi *test set*.

Sedangkan pada *nested cross validation*, proses k-fold *cross validation* yang biasa dilakukan hanya menjadi salah satu bagian dari metode ini. K-fold pada metode ini akan dilakukan pada dua *loop*, yaitu pada *outer loop* dan *inner loop*. Tujuan dari pembagian ini adalah agar *test data* tetap terjaga *pure* pada *fold* di *outer loop* karena CV dilakukan pada *inner loop* di *training set outer loop* tersebut [22]. Algoritma dari *nested cross validation* secara lengkap dapat dijabarkan sebagai berikut:

1. Bagi keseluruhan dataset menjadi K *folds* secara random
2. Untuk setiap fold k = 1,2,...,K: merupakan *outer loop* digunakan untuk evaluasi model dengan hyperparameter yang terpilih
   (a) Jadikan *fold k* sebagai *test*
   (b) Jadikan seluruh data selain yang ada di *fold k* sebagai *trainval*
   (c) Secara *random*, lakukan *split* pada *trainval* menjadi L *folds*
   (d) Untuk setiap fold l = 1,2,...,L: merupakan *inner loop* digunakan untuk *hyperparameter tuning*
      i. Jadikan *fold l* sebagai val
      ii. Jadikan seluruh data selain yang ada pada *test* dan *val* sebagai *train*
      iii. Lakukan *training* menggunakan setiap kombinasi *hyperparamter* pada *train* lalu evaluasi pada *val*. Catat *performance metric* dari *hyperparameter tersebut*
   (e) Untuk setiap kombinasi *hyperparameter*, lakukan kalkulasi rata-rata *metric score* pada L *folds* yang ada lalu pilih kombinasi yang terbaik
   (f) Train sebuah model menggunakan *hyperparamter* terbaik pada *trainval*. Evaluasi performanya pada *test* lalu simpan *score* untuk *fold k*
3. Lakukan kalkulasi rata-rata skor yang ada pada seluruh K *folds*

2.5.2 Metrik Evaluasi

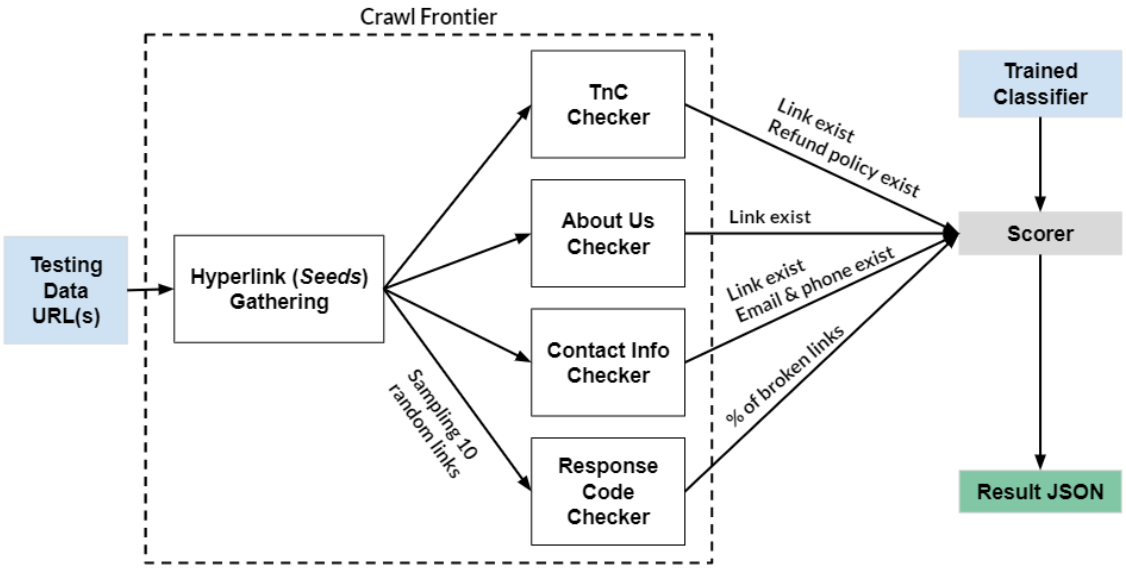
Untuk mengukur kualitas prediksi dari model, sebuah metrik harus ditentukan sebagai acuan. Ada berbagai macam metrik yang dapat dijadikan sebagai acuan penilaian untuk sebuah model *classifier*, diantaranya adalah Confusion Matrix, Akurasi, AUC, Precision, Recall, dan F1 Score.

- **Confusion matrix**: Sebuah matriks untuk menampilkan visualisasi dari performa model dalam melakukan klasifikasi. Setiap kolom pada *confusion matrix* merepresentasikan jumlah data yang terprediksi pada *class* yang bersangkutan dan setiap baris merepresentasikan jumlah data aktual pada *class* yang bersangkutan atau dapat pula terjadi sebaliknya [23]. Adapun 4 klasifikasi elemen yang ada pada *confusion matrix* dapat didefinisikan sebagai berikut:
  - True Positive (TP): Jumlah data positif yang diprediksi positif.
  - False Positive (FP): Jumlah data negatif yang diprediksi positif.
  - True Negative (TN): Jumlah data negatif yang diprediksi negatif.
  - False Negative (FN): Jumlah data positif yang diprediksi negatif.
- **AUC**: Probabilitas sebuah model bisa mengurutkan probabilitas data positif lebih tinggi dibandingkan data negatif. Dengan AUC = 1, jika dibuat sebuah garis bilangan horizontal dari 0 - 1, maka semua probabilitas prediksi data positif akan berada di sebelah kanan data negatif.
- **Akurasi**: Rasio prediksi benar (*class* positif dan *class* negatif) dibanding keseluruhan data. Dapat dihitung dengan rumus (TP+TN) / (TP+FP+FN+TN).
- **Precision**: Berapa proporsi benar dari prediksi *class* positif yang model lakukan. Precision dapat dihitung dengan rumus TP / (TP+FP).
- **Recall**: Berapa proporsi benar dari keseluruhan data positif yang ada yang dapat diprediksi oleh model. Recall dapat dihitung dengan rumus TP / (TP+FN).
- **F1 Score**: Perbandingan rata-rata Precision dan Recall yang dibobotkan. Dapat dihitung dengan rumus 2 * (Recall * Precission) / (Recall + Precission).

BAB 3
DESAIN DAN IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai desain dan implementasi *web crawler* untuk melakukan pendeteksian *fraud*. Desain dan implementasi mencakup arsitektur *web crawler*, desain proses pelatihan model, implementasi modul *helper*, implementasi modul ekstraksi fitur, dan *model scorer* pada *crawler*.

3.1 Arsitektur Web Crawler



Gambar 3.1: Arsitektur Web Crawler

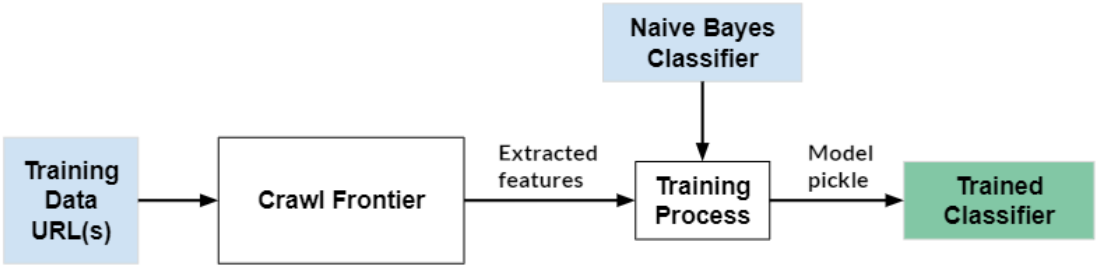
Web crawler dibangun di atas bahasa pemrograman Python menggunakan sebuah *framework microservice* yang ringkas dan mudah diimplementasikan bernama Flask[1]. *Web crawler* dapat diakses melalui sebuah *endpoint*[2] yang menerima sebuah *request* GET berupa URL atau sekumpulan URL situs *merchant* yang telah dikompilasi ke dalam satu *file* CSV (*comma-seperated values*). URL tersebut akan melalui serangkaian *logic* (*process*) yang diatur di dalam sebuah *crawl frontier*. Proses tersebut meliputi pengumpulan *hyperlink* yang terdapat pada halaman muka situs *merchant* (*hyperlink gathering*) dan pengecekan kelengkapan atribut situs, secara bergantian, dengan mengunjungi tautan-tautan pada situs tersebut yang telah terkumpul dari proses *hyperlink gathering*.

[1]https://flask.palletsprojects.com/en/1.1.x/
[2]Demo untuk *endpoint* ini dapat diakses pada http://demodemotest.com

Setelah menjalani proses pengecekan yang terdapat pada *crawl frontier*, keluaran dari setiap proses tersebut akan dikumpulkan menjadi serangkaian fitur. Fitur kemudian dijadikan sebagai *input* bagi *classifier* pada proses *scoring*. Hasil dari *scoring* dan fitur-fitur yang telah diekstrak oleh *crawler* akan dijadikan satu di dalam sebuah JSON dan dikembalikan sebagai *response* dari penembakan *endpoint*.

3.2 Proses Pelatihan Model

Pada Gambar 3.1 terdapat sebuah komponen yang dinamakan *trained classifier*. Proses yang terjadi untuk menghasilkan *trained classifier* tersebut digambarkan pada Gambar 3.2 berikut.



Gambar 3.2: Proses Training

Proses *training* dimulai dengan mengakses sebuah *endpoint*[3] yang menerima *input* berupa URL atau sekumpulan URL. URL tersebut merupakan URL yang ingin dijadikan sebagai *training data*. URL kemudian akan diproses oleh *crawl frontier* dan menghasilkan fitur-fitur yang telah diekstrak. Fitur-fitur tersebut kemudian akan menjadi *input* bagi *classifier* yang dipilih, yaitu Naive Bayes Classifier. Naive Bayes Classifier yang digunakan pada proses ini adalah Bernoulli Naive Bayes. Hal ini didasarkan pada kemampuannya yang baik dalam mengolah input fitur yang bersifat *binary* untuk melakukan prediksi.

Setelah proses *fitting* model dilakukan dan hasil evaluasi performa model dirasa sudah tepat, maka model yang sudah melalui proses *training* tersebut akan disimpan ke dalam bentuk *pickle*. Pickle memungkinkan untuk menyimpan sebuah *python object* menjadi sebuah *binary file* [24]. *Binary file* tersebut nantinya dapat kembali diakses dan digunakan kembali sebagai *python object* walaupun dijalankan dalam *session* dan waktu yang berbeda.

[3]Demo untuk *endpoint* ini dapat diakses pada http://demodemotest.com

3.3 Modul *Helper*

Dalam menjalankan *web crawler*, terdapat beberapa fungsi yang memang digunakan secara berulang-ulang di keseluruhan implementasi. Hal ini dikarenakan fungsi ini merupakan fungsi-fungsi yang bersifat umum dan memang didesain untuk memudahkan modul-modul spesifik untuk dapat berjalan. Fungsi-fungsi ini dinamakan fungsi *helper* dan diantaranya adalah *reset browser*, *url format handler*, *hyperlink gatherer* dan *paragraph extractor*.

3.3.1 Inisiasi & Atur Ulang Browser

Dalam mengunjungi halaman-halaman yang telah ditentukan, ada dua cara umum yang dapat dilakukan oleh *crawler*, yaitu dengan mengirimkan *request* dan melakukan *parsing response* atau mengunjungi halaman dengan *WebDriver* dan melakukan *parsing* terhadap konten di halaman tersebut. Untuk cara kedua, yaitu menggunakan *WebDriver*, maka untuk setiap sesi *crawling* perlu dilakukan inisiasi dari *WebDriver* itu sendiri. *Selenium WebDriver* digunakan untuk bernavigas dan *browser* yang digunakan pada *web crawler* ini adalah Google Chrome (*ChromeDriver*[4]).

```
## Initiate ChromeDriver (WebDriver) for the 1st time
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--ignore-certificate-errors')
driver = webdriver.Chrome(chrome_options=chrome_options)
hyperlinks_dynamic = False
dynamic_links = []
dynamic_texts = []

## Reset ChromeDriver function
def reset_browser():
    ''' Browser will be reset if an exception occurs when running di WebDriver
        and will be reset after one crawling job finished '''

    global driver
    global hyperlinks_dynamic
    global dynamic_links
    global dynamic_texts

    ## Quit driver
    driver.close()
    driver.quit()

    chrome_options = webdriver.ChromeOptions()
    chrome_options.add_argument('--ignore-certificate-errors')

    driver = webdriver.Chrome(chrome_options=chrome_options)
    hyperlinks_dynamic = False
```

[4]https://chromedriver.chromium.org/getting-started

```
28      dynamic_links = []
29      dynamic_texts = []
```

Kode 3.1: Fungsi inisiasi dan atur ulang *browser*

Pada 7 baris pertama di luar fungsi *reset_browser* yang termuat pada Kode 3.1, ditampilkan bagaimana *ChromeDriver* diinisiasi untuk pertama kali pada awal sesi. *ChromeDriver* dapat menerima argumen-argumen yang dapat dimasukkan ke dalam parameter *chrome_options*. Parameter yang penulis gunakan adalah ”–ignore-certificate-errors”. Parameter ini berfungsi untuk mengabaikan *SSL Certificate Error* yang kerap terjadi pada situs-situs yang tidak memiliki kelengkapan sertifikasi yang baik. Jika tidak diabaikan, maka situs-situs tersebut tidak dapat diakses dan diekstrak kontennya. Terdapat 3 variabel penting yang diinisiasi pada bagian ini juga, yaitu *hyperlinks_dynamic* (*boolean*), *dynamic_links* (*list*), dan *dynamic_texts* (*list*). Kegunaan *hyperlinks_dynamic* adalah sebagai *flag* apakah *ChromeDriver* sudah pernah dijalankan untuk mengekstrak konten ataukah belum. Jika sudah, maka *flag* akan berubah menjadi True sehingga apabila *ChromeDriver* dijalankan untuk kedua kalinya pada sesi yang sama, proses *gathering hyperlinks* dan *paragraph extractor* yang dilakukan dengan cara dinamis tidak perlu dijalankan kembali. Hal ini dilakukan dengan tujuan efisiensi waktu dan *bandwith* karena konten pada *running* pertama sudah berhasil diekstrak dan disimpan sehingga *crawler* hanya perlu menggunakan ulang konten tersebut.

Fungsi *reset_browser* sendiri memiliki tujuan untuk melakukan atur ulang (*resetting*) terhadap *browser* yang sedang berjalan. Yang terjadi adalah *browser* akan menutup dirinya sendiri dan melakukan inisiasi ulang seperti yang terjadi pada awal sesi. Hal seperti ini diperlukan saat terjadi *TimeoutExceptionError* pada *ChromeDriver* ataupun saat sesi *crawling* untuk satu situs telah berakhir. Mengatur dan menginisiasi ulang *ChromeDriver* pada dua kondisi tersebut memastikan *browser* sudah *clean* dan bisa digunakan kembali untuk mengeksekusi situs ataupun perintah yang berjalan berikutnya.

3.3.2 *URL Format Handler*

Fungsi ini merupakan sebuah fungsi singkat yang melakukan *handling* terhadap *input* URL dari pengguna yang tidak sesuai dengan *library-library* yang digunakan. Library *request* ataupun *ChromeDriver* yang digunakan pada *web crawler* memiliki format tertentu yang dapat diterima sebagai masukan. Salah satu contohnya adalah URL yang diterima hanya URL yang memiliki skema, yaitu yang diawali dengan *http://* ataupun *https://*. Sedangkan, pengguna terkadang malas ataupun lupa untuk memasukkan skema tersebut ke dalam URL yang diinput. Hal ini akan menyebabkan *error* dan *library* tidak akan dapat berjalan. Adapun variasi kasus masukan yang ditangani pada fungsi ini dapat dilihat

lebih lengkap pada Tabel 3.1 berikut.

No	Masukan	Keluaran	Deskripsi
1	” https://facebook.com ”	”https://facebook.com”	Menghilangkan *whitespace* di belakang dan depan URL
2	”http://facebook.com”	”https://facebook.com”	Mengubah seluruh URL http:// menjadi https://
3	”https://facebook.com/”	”https://facebook.com”	Menghilangkan *backslash* di belakang URL

Tabel 3.1: Konversi masukan URL menjadi keluaran yang sesuai

3.3.3 *Hyperlink Gatherer*

Pada proses *hyperlink gathering*, yang terjadi adalah *crawler* akan melakukan ekstraksi pada seluruh *hyperlink* yang ada di halaman antarmuka situs tersebut. Pada implementasinya, terdapat dua cara untuk melakukan ekstraksi *hyperlink*, bergantung pada jenis halaman utama yang dimiliki oleh situs yang dituju. Untuk situs dengan halaman statis, maka *crawler* akan mengirimkan *HTTP request* dan melakukan *parsing* terhadap *response body* yang didapatkan untuk mengekstraksi *tag link* (<a href>) yang ada pada *response* tersebut. Sedangkan untuk melakukan ekstraksi pada halaman muka yang dinamis[5], *crawler* akan menggunakan *WebDriver* untuk mengekstraksi *tag link* yang terdapat pada halaman dinamis tersebut.

3.3.3.1 *Static Hyperlink Gatherer*

Dalam implementasi kali ini, penulis memanfaatkan sebuah *library* yang memungkinkan *http programming* menjadi lebih mudah di Python, yaitu *requests-html*[6]. Pada dasarnya *requests-html* merupakan pengembangan dari *library requests* yang dikeluarkan oleh Python, hanya saja terdapat beberapa fungsi olahan lanjutan yang diimplementasikan oleh *library ini* sehingga *code* dapat ditulis dan dijalankan dengan lebih efisien.

```
1 def get_hyperlinks(url):
```

[5]Halaman dinamis didefinisikan sebagai halaman yang memerlukan *javascript* untuk melakukan *rendering* terhadap keseluruhan kontennya

[6]https://github.com/psf/requests-html

```
    """Return all absolute hyperlinks within the home url"""

    base_url = url_format_handler(url)
    session = HTMLSession()
    r = session.get(base_url, headers = {'User-Agent':
    np.random.choice(user_agent_list)}, timeout=30)
    res = list(r.html.absolute_links)

    ## If r-html anchor failed, concate manually
    res_final = [""]
    for url in res:
        if not str(url).startswith("http"):
            res_final.append(str(base_url + url))
        else:
            res_final.append(str(url))

    ## Check if domain expired/redirects
    domain = base_url.split("//")[1]
    if any(domain in url for url in res_final):
        pass
    else:
        res_final = [""]

    ## If website does not return hyperlink
    if len(res) == 0 or len(res_final) == 0:
        res = [""]
    else:
        res = res_final

    return res
```

Kode 3.2: Fungsi *hyperlink gathering*

Fungsi *get_hyperlinks* yang terdapat pada Kode 3.2 menerima input berupa *string* URL dari halaman muka sebuah situs *merchant*. Link tersebut kemudian akan dimasukkan kedalam sebuah *helper function* bernama *url_format_handler* untuk melakukan normalisasi URL agar sesuai dengan format yang bisa diterima library *requests-html*. Kemudian sebuah *session* akan dibuat dengan situs tersebut. Pada *crawler* ini, dalam setiap pembuatan *session*, *user-agent* yang digunakan pada *header* akan selalu diubah secara *random*. Hal ini dilakukan untuk menghindari *traffic* yang berasal dari *crawler* dianggap sebagai suatu serangan DDoS[7].

Setelah *session* terbuat dan *response* didapatkan, maka *hyperlink* dapat diekstrak dari *entity response* dengan memanfaatkan *parser* dari *requests-html* dengan memanggil *method html.aboslute_links*. *Hyperlink* yang telah diekstrak akan dikumpulkan dalam bentuk *list.*

Apabila *library* gagal mengembalikan *hyperlink* dengan *anchor*,[8] maka pengga-

[7]Distributed Denial-of-Service terjadi saat seorang peretas membanjiri jaringan lalu lintas data sebuah server sehingga mengganggu operasi sistem tersebut

[8]URL yang dikembalikan tidak mengandung *base URL*, seperti misalnya hanya /contact-us bukan nama-

bungan akan dilakukan secara manual. Apabila situs tidak mengirimkan *response*, maka fungsi ini akan mengembalikan sebuah *list* kosong. List yang mengandung *hyperlink* tersebut akan dimanfaatkan dalam beberapa proses lainnya.

3.3.3.2 *Dynamic Hyperlink Gatherer*

Untuk situs-situs yang dianggap memiliki konten dinamis, maka fungsi ini akan dipanggil. Fungsi ini merupakan fungsi yang akan dijalankan pada tahap awal pengumpulan *hyperlink* apabila menggunakan fungsi *get_hyperlink* tidak dapat mengembalikan *link* apapun dari proses ekstraksi statisnya. Fungsi ini juga akan dipanggil pada modul-modul *checker* apabila *crawler* tidak dapat menemukan *link-link* untuk 3 fitur utama situs (*tnc*, *contact us*, *about us*). Hal ini dikarenakan pada fungsi *get_hyperlinks_dynamic* ini, *WebDriver* akan mengecek juga pada *text* dari sebuah link tidak hanya pada URL nya, sehingga apabila *link* tidak mengandung kata kunci dari halaman yang dituju, maka URL tetap dapat terdeteksi. Contohnya adalah apabila *hyperlink* dari halaman *Contact Us* "https://situssaya.com" bukan "http://situssaya.com/contact-us" atau sejenisnya, melainkan merupakan URL yang *generic* seperti "http://situssaya.com/content=01".

```
def get_hyperlinks_dynamic(url):
    global hyperlinks_dynamic
    global dynamic_links
    global dynamic_texts

    try:
        ## Do not gather (again) if has been gathered before
        if hyperlinks_dynamic == True:
            print("- Using previously gathered hyperlinks")
            links, texts = dynamic_links, dynamic_texts
        else:
            print("- Gathering hyperlinks dynamically")
            driver.set_page_load_timeout(60)
            driver.get(url)
            elems = driver.find_elements_by_xpath("//a[@href]")
            links = []
            texts = []
            for elem in elems:
                links.append(elem.get_attribute("href"))
                texts.append(elem.text)

            ## Set to true, collect hyperlinks
            hyperlinks_dynamic = True
            dynamic_links = links
            dynamic_texts = texts
    except Exception as e:
        reset_browser()
        print(e)
```

situs.com/contact-us.

```
29          links = []
30          texts = []
31
32      return links, texts
```

Kode 3.3: Fungsi *hyperlink gathering dynamic*

Fungsi *get_hyperlinks_dynamic* pada Kode 3.3 dimulai dengan memeriksa apakah fungsi ini sudah pernah dijalankan sebelumnya. Variabel *hyperlinks_dynamic* menandakan hal ini, apabila nilainya True maka fungsi ini sudah pernah dijalankan sehingga *dynamic_links* dan *dynamic_texts* tidak perlu diekstrak kembali dan cukup langsung dikembalikan. Sedangkan apabila nilainya False, maka *ChromeDriver* akan dijalankan untuk mengakses halaman muka yang dituju. *Browser* ini memiliki batasan *timeout* 60 detik[9] untuk memuat seluruh konten yang ada pada halaman yang dituju. Jika *loading* belum selesai dalam rentang waktu tersebut, maka *TimeOutException* akan terjadi dan *browser* akan diinisiasi ulang tanpa ada konten yang terekstrak. Sedangkan apabila dalam 60 detik halaman berhasil dimuat, maka ekstraksi elemen akan dilakukan dengan melakukan pencarian berdasarkan *xpath*. *ChromeDriver* akan mencari elemen dengan *tag link* (<a href>) dan mengekstrak *attribute* (URL yang dituju pada *property href*) serta mengekstrak teks yang ditampilkan oleh *tag* tersebut. Setelah diekstrak, maka akan disimpan ke dalam variabel *links* dan *texts* dan akan dikembalikan di akhir fungsi.

3.3.4 *Paragraph Extractor*

Setiap *crawler* butuh untuk melakukan ekstraksi informasi, maka fungsi *paragraph extractor* akan. Tujuan utama fungsi ini adalah untuk melakukan proses *scraping* pada sebuah halaman. Ekstraksi informasi yang dilakukan ini nantinya akan dibutuhkan oleh 3 modul *checker* pada *crawler*, yaitu pada *TnC Checker*, *About Us Checker*, dan *Contact Info Checker*. Fungsi ini membantu untuk melakukan ekstraksi semua konten teks yang ada pada halaman-halaman yang bersangkutan. Ada dua jenis fungsi ekstraksi yang diimplementasi pada *crawler*, yaitu *paragraph_extractor* dan *paragraph_extractor_dynamic* dengan masing-masing tujuan yang sama namun cara yang berbeda. Fungsi *paragraph_extractor* digunakan sebagai esktraktor konten statis yang ada pada sebuah halaman dengan mengirim *HTTP request* dan melakukan *parsing* terhadap *body response* yang dikembalikan. Sedangkan sebaliknya, *paragraph_extractor_dynamic* berfungsi sebagai ekstraktor konten dinamis dengan memanfaatkan *ChromeDriver* seperti pada fungsi *get_hyperlinks_dynamic*.

[9]Rentang waktu ini dipilih berdasarkan hasil diskusi dengan tim DOKU dan dianggap sebagai waktu tunggu yang ideal

3.3.4.1 Static Paragraph Extractor

Static paragraph extractor fungsi *default* yang akan dijalankan saat ingin melakukan ekstraksi informasi. Fungsi ini memanfaatkan *library request* untuk mengirimkan sebuah *HTTP request* ke halaman yang dituju dan menampung *response content*nya. Elemen-elemen yang diperlukan untuk diekstrak adalah semua jenis teks yang ada pada halaman tersebut. Hal ini dilakukan dengan memanfaatkan *library beautifulsoup4* untuk melakukan *parsing* dari *response content* yang didapatkan.

```
def paragraph_extractor(url):
    ''' Extract all paragraph (texts) found in a given URL '''

    try:
        ## Send request
        page = requests.get(url, headers = {'User-Agent':
    np.random.choice(user_agent_list)}, timeout=30, verify=False)

        ## Parse content
        soup = bs(page.content, 'html.parser')
        all_ps = soup.find_all("p") + soup.find_all("em") + soup.find_all("li") +
    soup.find_all("address")\
        + soup.find_all("h1") + soup.find_all("h2") + soup.find_all("h3") +
    soup.find_all("h4") + soup.find_all("h5")\
        + soup.find_all("h6") + soup.find_all("a") + soup.find_all("strong")

        ## CloudFare email getter
        if re.search('data-cfemail', str(soup)) is not None:
            email_code = re.search('data-cfemail="(.+?)"', str(soup)).group(1)
            email = str(decode_email(email_code))
        else:
            email = ""

        ## Metadata getter
        meta_property = soup.find("meta", property="og:description")
        meta_name = soup.find("meta", {"name":"description"})

        if meta_property is not None:
            meta_property = soup.find("meta", property="og:description")["content"]
        else:
            meta_property = ""
        if meta_name is not None:
            meta_name = soup.find("meta", {"name":"description"})["content"]
        else:
            meta_name = ""

        ## Div with class:address getter
        div_address = str(soup.find("div", {"class":'address'}))
        if div_address is None:
            div_address = ""

        list_p = []
        for p in all_ps:
            list_p.append(unidecode.unidecode(p.getText()) + "\n")

```

```
43          ## Join all paragraphs element
44          paragraf = "".join(list_p)
45          paragraf += meta_property + meta_name + email + div_address
46          paragraf = paragraf.lower()
47
48      except Exception as e:
49          print(e)
50          paragraf = ""
51
52      return paragraf
```

Kode 3.4: Fungsi *paragraph extractor*

Pada Kode 3.4, yang akan dilakukan pertama kali adalah mengirimkan *request* ke URL terkait dengan beberapa parameter. Parameter tersebut adalah *user agent* pada *headers*, *timeout*, dan *verify*. *User agent* yang digunakan pada setiap *request* akan dirotasi seperti yang telah dijelaskan pada Subbab 3.3.3.1. Sedangkan *timeout* yang ditentukan untuk setiap *request* adalah 30 detik, jika melebihi waktu tersebut maka *ConnectionError* akan dimunculkan. Parameter *verify* digunakan untuk melakukan *bypass* terhadap situs-situs yang memiliki sertifikasi tidak baik agar tetap bisa dilakukan *crawling*. Hal ini sama seperti opsi ”–ignore-certificate-error” yang ada pada *ChromeDriver*.

Memasuki bagian *parsing*, *beautifulsoup* akan melakukan *parsing* dokumen HTML yang diterima dari proses *request* yang dikirimkan. Adapun untuk mencari seluruh teks yang ada pada halaman tersebut dapat dilakukan dengan metode *find_all* yang dimilik oleh *library beautifulsoup*. Semua teks dengan *tag-tag* yang dirinci pada kode akan diekstrak dan disimpan dalam bentuk *string* pada variabel *all_ps*. Selain teks umum pada sebuah halaman, ada beberapa elemen lain yang perlu diekstrak karena dianggap dapat menyimpan informasi yang juga penting ataupun tidak terkekstrak pada tahap pertama. Elemen pertama adalah data *email* terenkripsi jika situs tersebut menggunakan layanan *CloudFlare*[10], yang tentunya perlu didekripsi menggunakan fungsi *email_decode*. Elemen lainnya adalah *metadata* yang mengandung *description* serta *tag* yang memiliki *class address*. Hal ini penting untuk diekstrak karena beberapa informasi tersebut dibutuhkan bagi modul-modul *checker* yang mengimplementasikan fungsi ini. Fungsi ini mengembalikan variabel *paragraf* yang merupakan satu string gabungan dari keseluruhan teks yang ada pada halaman tersebut.

3.3.4.2 Dynamic Pragraph Extractor

Dalam melakukan ekstraksi, tidak selamanya fungsi *paragraph extractor* biasa berhasil melakukan ekstraksi konten yang diinginkan. Hal ini dapat terjadi karena dua hal, antara

[10]Sebuah perusahaan *internet security* yang menyediakan berbagai macam jasa keamanan untuk situs. Dikutip dari https://www.cloudflare.com

memang halaman yang dituju tidak dapat diakses, ataupun konten di halaman tersebut belum lengkap. Merujuk pada alasan kedua, ketidaklengkapan konten dapat dikarenakan halaman tersebut sebenarnya bersifat dinamis. Halaman yang bersifat dinamis tidak akan langsung melakukan *rendering* terhadap semua elemen yang sebenarnya dimiliki. Halaman seperti ini biasanya membutuhkan waktu tunggu beberapa detik agar *script* yang ada pada halamannya dapat berjalan untuk melakukan *rendering* konten halaman secara utuh. Sayangnya, *delayed content* tersebut tidak dapat didapatkan dengan mengirim *request* biasa menggunakan *library requests*. Oleh karena itu dibuat fungsi *paragraph extractor dynamic* yang memanfaatkan *ChromeDriver*, sebuah *browser* yang dapat diotomasi, untuk melakukan *rendering* dan mendapatkan konten-konten dinamis tersebut.

```
def paragraf_extractor_dynamic(url):
    ''' Extract all paragraph (texts) found in a given dynamic page URL '''

    try:
        ## Send request
        print("- Extracting paragraphs dynamically")
        driver.set_page_load_timeout(60)
        driver.get(url)

        ## Parse content
        soup = bs(driver.page_source, 'html.parser')

        all_ps = driver.find_elements_by_tag_name("p") +
    driver.find_elements_by_tag_name(
            "em") + driver.find_elements_by_tag_name("li") +
    driver.find_elements_by_tag_name("address") \
                + driver.find_elements_by_tag_name("h1") +
    driver.find_elements_by_tag_name(
            "h2") + driver.find_elements_by_tag_name("h3") +
    driver.find_elements_by_tag_name(
            "h4") + driver.find_elements_by_tag_name("h5") \
                + driver.find_elements_by_tag_name("h6") +
    driver.find_elements_by_tag_name(
            "a") + driver.find_elements_by_tag_name("strong") +
    driver.find_elements_by_tag_name("span")

        texts = []
        for p in all_ps:
            texts.append(p.text + "\n")

        ## CloudFare email getter
        if re.search('data-cfemail', str(soup)) is not None:
            email_code = re.search('data-cfemail="(.+?)"', str(soup)).group(1)
            email = str(decode_email(email_code))
        else:
            email = ""

        ## Metadata getter
        meta_property = soup.find("meta", property="og:description")
        meta_name = soup.find("meta", {"name": "description"})
```

```
35
36          if meta_property is not None:
37              meta_property = soup.find("meta", property="og:description")["content"]
38          else:
39              meta_property = ""
40          if meta_name is not None:
41              meta_name = soup.find("meta", {"name": "description"})["content"]
42          else:
43              meta_name = ""
44
45          ## Div with class:address getter
46          div_address = str(soup.find("div", {"class": 'address'}))
47          if div_address is None:
48              div_address = ""
49
50          paragraf = "".join(texts)
51          paragraf += meta_property + meta_name + email + div_address
52          paragraf = paragraf.lower()
53      except Exception as e:
54          reset_browser()
55          print(e)
56          paragraf = ""
57
58      return paragraf
```

Kode 3.5: Fungsi *paragraph extractor dynamic*

Fungsi *paragraph_extractor_dynamic* yang ada pada Kode 3.5 sebenarnya sangat mirip dengan fungsi *paragraph_extractor* yang ada pada Kode 3.4. Yang membedakan antara keduanya hanya pada proses mengirimkan *request* dan mendapatkan *response*. Pada fungsi ini, *library requests* tidak digunakan, melainkan menggunakan *ChromeDriver* yang telah diinisiasi pada awal sesi. Alasan dibalik penggunaan ini adalah, *Selenium WebDriver* yang diimplementasikan menggunakan *ChromeDriver* ini memiliki kemampuan untuk melakukan ekstraksi terhadap keseluruhan konten, yaitu yang statis maupun dinamis. *ChromeDriver* akan menunggu selama waktu *timeout* yang telah diset untuk semua konten dapat terekstrak. Apabila sampai ke waktu yang telah ditentukan halaman yang dituju masih tidak memberikan respon ataupun belum menyelesaikan proses *loading*nya, maka TimeOutException akan dikeluarkan.

Ekstraksi konten pada sekumpulan *tag* yang melambangkan teks dilakukan dengan *method find_elements_by_tag_name(). Method* ini akan mengembalikan sekumpulan WebElement[11] yang nantinya dapat diekstrak teks di dalamnya dengan memanggil *.text*. Sedangkan untuk ekstraksi konten-konten pelengkap lainnya tidak ada perbedaan dengan kode pada Kode 3.4 karena pada dasarnya *CloudFare email*, *metadata*, dan *tag div address* memang sudah merupakan elemen statis. Fungsi ini juga mengembalikan variabel *paragraf* yang merupakan satu string gabungan dari keseluruhan teks yang ada pada ha-

[11] Cara WebDriver merepresentasikan sebuah HTML *element*

laman tersebut.

3.4 Modul Ekstraksi Fitur

3.4.1 *Broken Link Checker*

Bagian ini memiliki fungsi untuk melakukan pengecekan terhadap jumlah *broken link* yang terdapat pada situs *merchant*. Fungsi ini memanfaatkan keluaran dari pengumpulan *hyperlink* yang dijalankan oleh Kode 3.2 ataupun Kode 3.3. *Library* yang digunakan pada fungsi ini adalah *grequests*[12], yang merupakan pengembangan dari *library requests* dan *gevent*[13]. *Library* ini digunakan untuk melakukan *HTTP request* secara asinkronus. *Request* secara asinkronus diperlukan untuk mempercepat proses pengecekan pada lebih dari satu URL.

```
1  def broken_link_score(df, hyperlinks):
2      """ Return score of broken link in a website
3          1 for >= 50 %
4          0 for < 50 %
5      """
6
7      print("Checking broken link...")
8
9      ## Avoid extraction of external links
10     avoid = pd.Series(hyperlinks).str.contains("wa.me") |
       pd.Series(hyperlinks).str.contains("youtube") | \
11     pd.Series(hyperlinks).str.contains("linkedin") |
       pd.Series(hyperlinks).str.contains("facebook") | \
12     pd.Series(hyperlinks).str.contains("cloudflare") |
       pd.Series(hyperlinks).str.contains("twitter") | \
13     pd.Series(hyperlinks).str.contains("github") |
       pd.Series(hyperlinks).str.contains("instagram") | \
14     pd.Series(hyperlinks).str.contains("tokopedia") |
       pd.Series(hyperlinks).str.contains("bukalapak") | \
15     pd.Series(hyperlinks).str.match("tel") |
       pd.Series(hyperlinks).str.contains("gitlab") | \
16     pd.Series(hyperlinks).str.contains("Tel") |
       pd.Series(hyperlinks).str.contains("jobstreet") | \
17     pd.Series(hyperlinks).str.contains("download") |
       pd.Series(hyperlinks).str.contains("google") | \
18     pd.Series(hyperlinks).str.contains("javaScript") |
       pd.Series(hyperlinks).str.contains("_blank")
19     hyperlinks = list(pd.Series(hyperlinks)[~avoid].values)
20
21     ## If length of hyperlinks collected > 10, do sampling
22     if len(hyperlinks) > 10:
23         hyperlinks = sample(hyperlinks, 10)
24
```

[12]https://pypi.org/project/grequests/
[13]https://pypi.org/project/gevent/

```
    ## Send request
    rs = (grequests.get(x, \
        headers = {}, \
        timeout=20, verify=False) for x in hyperlinks)
    rs_res = grequests.map(rs, size = 3)

    links = {}
    i = 0
    if len(hyperlinks) == 1 and hyperlinks[0] == "":
        links = "No hyperlinks gathered"
    else:
        ## Get all response code from sampled links
        for response in rs_res:
            try:
                links[response.request.url] = str(response)
            except Exception as e:
                print(e)
                ## No response/timeout return this
                links[hyperlinks[i]] = 'No Response/Timeout'
            i += 1

    status_not_ok = np.count_nonzero(np.array(rs_res, dtype=str) != '<Response
    [200]>')
    status_length = len(rs_res)

    try:
        ## Do scoring
        score = status_not_ok/status_length*100
    except Exception as e:
        print(e)
        score = 100

    res_df = pd.DataFrame({"merchant_name": df['merchant_name'].values[0],
    "broken_link_score": 0 if score < 50 else 1,\
                           "links_response": str(links)}, index=[0])

    print("Broken links checked.\n")

    return res_df
```

Kode 3.6: Fungsi *broken link checker*

Pada fungsi ini, yang dilakukan adalah melakukan pengecekan *response code* satu per satu terhadap kumpulan URL (*hyperlinks*) yang telah diekstrak pada proses sebelumnya. Jika *list hyperlink* mengandung lebih dari 10 *hyperlink*, maka akan dilakukan *sampling* dengan cara *random sample* sebanyak 10. Namun, jika *list* mengandung kurang dari 10 *hyperlink*, maka *sampling* tidak perlu dilakukan dan pengecekan dilakukan ke semua URL yang ada. Sebelum *request* dikirimkan ke URL yang ada pada list, URL akan terlebih dahulu disaring sehingga *link-link* eksternal yang ada pada situs tersebut tidak masuk ke dalam perhitungan fungsi[14].

[14]Link yang mengarah ke pranala luar situs bukan merupakan bagian dari *scope* pengecekan tim DOKU

List yang telah disaring dan berisi kumpulan URL tersebut akan diolah oleh *library grequests* dengan mengirimkan *request* secara asinkronus terhadap semua URL tersebut. Untuk setiap *request* akan diberikan *timout* selama 20 detik dan dalam satu detiknya *grequests* dapat mengirimkan sampai dengan 3 *request* secara bersamaan. Setelah *request* dikirimkan, *response code* dari setiap URL akan disimpan dalam sebuah *dictionary*. *Key* dari *dictionary* tersebut adalah URL dan *value*nya adalah *response code* dari URL yang bersangkutan. Setelah semua *response code* didapatkan, maka dapat dihitung presentase *hyperlink* yang berstatus OK[15] dengan membaginya dengan keseluruhan jumlah *hyperlink* yang diolah.

Pada akhirnya fungsi akan mengembalikan *dataframe* yang berisikan nama *merchant* (*merchant_name*), presentase *broken link* (*broken_link_score*) yang telah dibinerisasi[16], serta *dictionary* yang mengandung *response code* dari keseluruhan URL yang diolah (*links_response*).

3.4.2 *TnC Checker*

Bagian ini memiliki fungsi untuk melakukan pengecekan informasi syarat dan ketentuan pada sebuah situs *merchant*. Informasi yang akan dicek adalah ada/tidaknya sebuah URL yang mengarah pada halaman syarat & ketentuan serta ada/tidaknya kata kunci yang menyebutkan tentang pengembalian barang atau uang (*refund policy*) di dalam halaman tersebut. Fungsi yang digunakan pada *tnc checker* ini adalah fungsi *tnc_score*[17] untuk menjalankan keseluruhan algoritma *tnc checker* mulai dari mencari halaman syarat & ketentuan sampai mencari konten *refund policy di dalamnya*. Untuk melakukan pencarian konten *refund policy* di setiap kandidat halaman, maka ada dua fungsi *helper* lain yang dapat digunakan, yaitu *paragraph_extractor* dan fungsi *refund_policy_matcher*.

Fungsi *tnc score* menerima masukan berupa kumpulan *hyperlink* yang telah dikumpulkan pada proses sebelumnya. Fungsi ini akan melakukan pengecekan pada seluruh isi dari *list hyperlink* yang diberikan dan memeriksa apakah pada URL tersebut mengandung kata-kata yang telah ditentukan pada *keyword_tnc*. Apabila ada, maka URL-URL tersebut akan dikumpulkan ke dalam satu *list* baru dan ditandai sebagai kandidat halaman syarat & ketentuan.

Untuk mengekstraksi konten dari kandidat-kandidat halaman tersebut, maka fungsi *paragraph extractor* akan dipanggil untuk setiap halaman yang ada. Fungsi *paragraph extractor* akan mengembalikan sebuah *string* yang merupakan kumpulan teks dari halaman

[15] Didefinisikan dengan URL yang mengembalikan *response code* 200 OK

[16] Binerisasi menggunakan *threshold* 50%. Apabila jumlah *link* yang mati berada di atas atau sama dengan 50% maka akan ditransformasikan menjadi 1, sementara dibawah itu menjadi 0

[17] Dapat dilihat pada Lampiran

tersebut. Untuk setiap *string* yang diekstrak dari sebuah halaman, maka akan diperiksa eksistensi *refund policy*nya dengan menggunakan fungsi *refund_policy_matcher*. Cara kerja *refund_policy_matcher* dapat dilihat pada Kode 3.7.

Jika berhasil menemukan konten *refund policy* pada suatu halaman maka fungsi ini akan langsung mengembalikan hasilnya, yaitu sebuah *flag* yang menandakan bahwa pada situs *merchant* ini terdapat halaman syarat & ketentuan (*link_tnc_exist*: 1) dan di dalamnya terdapat kata kunci yang berhubungan dengan *refund policy* (*tnc_refund_policy_exist*: 1). Namun, apabila tidak ditemukan salah satu diantara dua komponen tersebut, *tnc_score* akan melanjutkan proses pencarian ke halaman muka dari situs tersebut, dan mencari kata kunci di halaman itu. Jika masih tidak ditemukan, maka proses pencarian akan diulang kembali namun menggunakan fungsi *paragraph_extractor_dynamic* dan *get_hyperlinks_dynamic*, dikarenakan ada kemungkinan situs tersebut sebenarnya memiliki konten dinamis yang tidak bisa diekstrak menggunakan metode statis.

```
def refund_policy_matcher(paragraf):
    keyword_refund = ['refunds', 'refund', 'refund policy', 'return', 'returns',
    'return policy', 'pengembalian', 'pengembalian dana', 'mengembalikan dana',
    'dikembalikan', 'pengembalian', 'mengembalikan', 'retur', 'tukar', 'penukaran']

    ## Tokenizing & cleaning paragraf into words
    try:
        lang = 'indonesian' if tb(paragraf).detect_language() == 'id' else
    'english'
        tokenizer = RegexpTokenizer(r'\w+')
        words = tokenizer.tokenize(paragraf)

        ## Removing stop words
        stop_words = nltk.corpus.stopwords.words(lang)

        cleaned_words = []
        for word in words:
            if word not in stop_words:
                cleaned_words.append(word)

    except Exception as e:
        print(e)
        cleaned_words = ""

    mask = pd.Series(cleaned_words).str.contains("|".join(keyword_refund))

    count_refund = np.count_nonzero(mask)
    if count_refund > 0:
        return 1
    else:
        return 0
```

Kode 3.7: Fungsi *refund policy matcher*

Fungsi *refund_policy_matcher* pada Kode 3.7 berfungsi untuk mencari kata kunci dari

masukan *string* yang diterima. Kata kunci yang dicari merupakan sekumpulan kata-kata yang berhubungan dengan proses pengembalian uang dan barang. Fungsi tersebut melakukan penyocokan kata kunci dengan sekumpulan *string* yang telah ditokenisasi pada halaman yang dicari. Tokenisasi memudahkan proses pencocokan kata karena *string* yang tadinya berbentuk paragraf telah dipecah menjadi kumpulan-kumpulan kata. Untuk melakukan *removal* terhadap kata-kata yang tidak berkaitan, pembersihan *stop words*[18] juga dilakukan. Setelah itu, akan dihitung ada berapa banyak *keyword* yang ditemukan dalam kumpulan kata-kata yang telah diolah. Jika terdapat lebih dari 1, maka fungsi akan melakukan *return* 1, yang mana dianggap sebuah *keyword* terkait *refund policy* telah berhasil ditemukan.

3.4.3 *About Us Checker*

Bagian ini memiliki fungsi pengecekan yang paling sederhana. Pada bagian ini, *crawler* hanya akan melakukan pengecekan apakah situs *merchant* memiliki URL yang mengarah pada sebuah halaman yang berisi penjelasan tentang situs tersebut (*about us*, tentang kami, dan sebagainya) atau tidak. Proses pencarian halaman informasi ini dilakukan melalui dijalankan dengan melakukan pencocokan kata kunci dengan setiap URL yang ada pada kumpulan *hyperlink*.

```
1  def about_us_check(df, hyperlinks):
2      """Return boolean of about us link existance"""
3
4      print("Checking about us...")
5      keyword_about = ["about", "tentang"]
6      avoid = ["conditioner", "termurah", "termahal", "expired", "expire", "renewal"]
7
8      about_mask =
       pd.Series(hyperlinks).str.lower().str.contains('|'.join(keyword_about)) \
9                  & ~pd.Series(hyperlinks).str.lower().str.contains('|'.join(avoid))
10     about_count = 1 if np.count_nonzero(np.array(hyperlinks)[about_mask]) >= 1
       else 0
11
12     if about_count == 1:
13         print("About Us Link: %s" % np.array(hyperlinks)[about_mask][0])
14
15     ## Check for inexact link
16     try:
17         if about_count < 1:
18             base_url = str(df['website'].values[0])
19             links, texts = get_hyperlinks_dynamic(url_format_handler(base_url))
20             if len(links) > 0:
21                 for i in range(len(links)):
22                     ## About Us Link Finder
```

[18] *Stop words* adalah sebuah kumpulan kata yang lazim digunakan di berbagai macam bahasa. *Stop words* biasanya dihilangkan agar pemrosesan dapat berfokus pada kata yang penting saja.

```
23                      if
    pd.Series(str(texts[i])).str.lower().str.contains('|'.join(keyword_about)).any():
24                          print("About Us Link: %s" % links[i])
25                          about_count = 1
26      except Exception as e:
27          print(e)
28          pass
29
30      res_df = pd.DataFrame({"merchant_name": df['merchant_name'].values[0],
        "link_about_us_exist": int(about_count)},
31                            index=[0])
32
33      print("About us checked.\n")
34
35      return res_df
```

Kode 3.8: Fungsi *about us check*

Pada Kode 3.8, fungsi *about_us_check* akan melakukan pengecekan pada setiap URL yang ada di masukan (*list hyperlinks*). Untuk setiap URL, dicek apakah terdapat kata kunci yang termuat dalam *list keyword_about*. Jika dapat ditemukan maka *flag about_count* akan diubah *value*nya menjadi 1. Apabila tidak ditemukan, maka akan dicari *hyperlinks* baru menggunakan metode pencarian dinamis.

Pencarian dinamis dilakukan dengan cara mengumpulkan ulang seluruh *hyperlink* yang ada di halaman muka dengan fungsi *get_hyperlinks_dynamic*. Apabila ternyata hasil ekstraksi *hyperlink* mengeluarkan hasil yang berbeda dan *link about us* dapat ditemukan, maka dapat disimpulkan *tag-tag* link yang ada pada halaman utama dirender dengan metode dinamis ataupun memiliki *link* yang tidak mengandung *keyword* di URL nya seperti contoh yang sudah dijelaskan pada Subbab 3.3.4.2. Dengan menerapkan pencarian *hyperlink* secara dinamis, ekstraksfi informasi menjadi lebih akurat dikarenakan adanya *double check* dan tidak hanya bergantung pada pencarian statis.

3.4.4 *Contact Info Checker*

Bagian ini berfungsi untuk melakukan pengecekan terhadap informasi kontak yang terdapat pada situs *merchant*. Informasi yang dicek adalah ada/tidaknya URL yang mengarah ke halaman kontak dan juga ada/tidaknya *email* serta nomor telepon pada situs tersebut. Pencarian *email* dan nomor telepon dilakukan pada dua *search spaces*, yaitu pada halaman kontak dan pada halaman muka situs itu sendiri. Dalam bagian ini, terdapat dua algoritma *regular expression* yang diimplementasikan, yaitu *telephone_matcher* dan juga *email_matcher* untuk mencari nomor telepon dan email pada paragraf yang diekstrak.

Untuk mencari URL yang mengarah ke halaman kontak, fungsi *contact_us_score*[19] akan menjalankan algoritma yang sama seperti yang dijalankan oleh *tnc_score* pada Sub-

[19]Dapat dilihat pada Lampiran

bab 3.4.2. Yang dilakukan adalah mengiterasi keseluruhan elemen pada masukan *list hyperlinks* dan menyocokkan dengan kata kunci *contact us* yang telah ditentukan. Apabila ditemukan, maka paragraf dari setiap halaman tersebut akan diekstrak dan dijadikan masukan bagi fungsi *telephone_matcher* dan *email_matcher*. Jika kedua fungsi tersebut dapat menemukan *email* dan telepon di halaman terkait, maka fungsi akan langsung mengembalikan hasilnya.

Jika salah satu dari ketiga elemen yang dicari (*link*, *email*, dan nomor telepon) tidak dapat ditemukan, maka situs diasumsikan memiliki konten dinamis. Saat asumsi ini diaktifkan, maka pengecekan terakhir akan dilakukan menggunakan fungsi-fungsi yang diperuntukkan untuk konten dinamis, yaitu *get_hyperlinks_dynamic* dan *paragraph_extractor_dynamic*.

```
def telephone_matcher(paragraf):
    telephone_match \
    = re.search(r'(\d{3,5}[-\.\s]??\d{3,5}[-\.\s]??\d{3,5}|\(\d{3,5}\)'
           '\s*\d{3,5}[-\.\s]??\d{3,5}|\d{3,5}[-\.\s]??\d{3,5})',\
                                    paragraf)
    if telephone_match is not None:
        print("Phone Number: %s" % telephone_match.group(0))
        return 1
    else:
        return 0
```

Kode 3.9: Fungsi *telephone matcher*

Fungsi *telephone_matcher* yang ada pada Kode 3.9 akan melakukan pencocokan terhadap *pattern* nomor telepon Indonesia di masukan *string* yang diberikan. Masukan *string* ini berupa paragraf yang diekstrak pada setiap halaman *contact us* yang ditemukan. Saat *pattern* nomor telepon tersebut ditemukan, maka *console* akan melakukan *printing* terhadap nomor telepon tersebut untuk keperluan *debugging* dan fungsi akan melakukan *return flag* 1.

```
def email_matcher(paragraf):
    email_match = re.search(r'[\w\.-]+(@|\[at])[\w\.-]+', paragraf)

    if email_match is not None:
        print("Email: %s" % email_match.group(0))
        return 1
    else:
        return 0
```

Kode 3.10: Fungsi *email matcher*

Fungsi *email_matcher* yang ada pada Kode 3.10 akan melakukan pencocokan terhadap *pattern* email yang ada pada masukan *string* yang diberikan. Masukan *string* ini, sama seperti pada Kode 3.9 merupakan paragraf yang diekstrak pada setiap halaman *contact us* yang ditemukan. Saat *pattern* email tersebut ditemukan, maka *console* akan melakukan

printing terhadap email tersebut untuk keperluan *debugging* dan fungsi akan melakukan *return flag* 1.

3.5 Model Scorer

Modul ini memiliki fungsi untuk melakukan *scoring* terhadap situs *merchant* yang telah diekstrak fitur-fiturnya. Scoring dilakukan dengan menggunakan *trained classifier* yang telah diimplementasikan dan mengeluarkan *output* berupa *prediction class* dan *prediction probability* kelas itu berasal dari *class* berlabel 1.

```
def calculate_score(features):
    """Return fraud prediction score of a website, 0-100 (Good - Bad)"""

    ## Post API URL, change the URL to the host site URL
    url = 'http://127.0.0.1:5000/api/v1/model'

    ## Process Test Data
    df = pd.DataFrame(features, index=[0])
    columns = ['broken_link_score','link_contact_us_exist', 'cu_email_exist',\
    'cu_phone_number_exist', 'link_about_us_exist', 'link_tnc_exist',\
    'tnc_refund_policy_exist']
    test_df = df[columns]
    data = test_df.to_json()

    ## Post to Model API
    headers = {'content-type': 'application/json', 'Accept-Charset': 'UTF-8'}
    req = requests.post(url, data=data, headers=headers).text
    score = float(req)

    df['prediction_prob'] = score
    df['prediction_class'] = 1 if score >= 0.5 else 0
    res = df

    return res
```

Kode 3.11: Fungsi *model scorer*

Setelah semua fitur diekstrak pada proses-proses sebelumnya, maka fungsi *calculate_score* seperti yang dapat dilihat pada Kode 3.11 akan menjalankan tugasnya untuk melakukan *scoring* berdasarkan kumpulan fitur tersebut. Masukan yang diterima merupakan sebuah *dictionary* yang berisi pengembalian hasil dari modul-modul yang telah dijalankan. Hanya 7 kolom yang akan diambil dari keseluruhan kolom yang ada, yaitu fitur-fitur yang memang menjadi input bagi model *machine learning* dan akan dibahas lebih detil pada Bab 4.

Fitur-fitur tersebut akan ditransalasikan ke dalam bentuk JSON dengan format {"fitur1": value, "fitur2": value, ..., "fiturN": value}. JSON ini akan ditembak ke *end-*

point model scoring[20] dengan POST *request*, yang mana *endpoint* tersebut akan mengembalikan *prediction probability class* 1 dari fitur-fitur tersebut. Jika *probability* berada di atas 0.5 maka akan dianggap sebagai *class* 1, jika tidak maka 0. Threshold dapat diatur secara manual pada fungsi ini apabila ada *preferensi* tersendiri setelah dilakukan evaluasi model lagi pada waktu yang akan datang.

[20]Kode *endpoint* untuk *model scoring* dapat dilihat pada Lampiran

BAB 4
EKSPERIMEN DAN ANALISIS

4.1 Alur Kerja Eksperimen

Sebelum *web crawler* siap untuk masuk ke tahap produksi, maka perlu ada evaluasi yang mendalam terhadap model yang ditanamkan pada *web crawler* tersebut. Oleh karena itu, sebuah alur kerja eksperimen disusun untuk mengevaluasi performa model secara komprehensif. Adapun langkah-langkah yang dilakukan pada bagian ini meliputi proses pengolahan data awal dan evaluasi hyperparameter model menggunakan *nested cross-validation*.

Pada penelitian ini data yang digunakan untuk membuat model yang akan diimplementasikan di *crawler* merupakan situs-situs yang terdaftar sebagai *online merchant* DOKU. Eksperimen dilakukan menggunakan metode *supervised learning* dimana model akan menerima input untuk dipetakan menjadi output dari sebuah dataset yang telah dilabel. Dataset didapatkan dari tim *Early Detection Unit* (EDU) DOKU yang mereka kumpulkan dan verifikasi secara manual. Data awal diberikan dalam bentuk CSV dengan total 292 baris dan 3 kolom. Kolom terdiri dari nama *merchant* (*merchant_name*), alamat situs (*website*), dan juga status dari situs tersebut (*class*). Struktur data awal yang berbentuk file CSV tersebut digambarkan pada Tabel 4.1.

merchant_name	**website**	**label**
Nyetak.ID	www.nyetak.id	APPROVED
YoyoMats Indonesia	www.yoyomats.shopify.com	REJECTED

Tabel 4.1: Struktur tabel yang dikirimkan oleh tim EDU

Data awal yang diberikan oleh tim EDU bukan merupakan data yang bisa langsung dijadikan input (*training data*) untuk model *machine learning*. Data ini harus melalui proses ekstraksi fitur yang telah dijelaskan di Gambar 3.2 pada Bab 3. Proses ekstraksi fitur, akan menambah kolom pada Tabel 4.1 sebanyak 8 kolom tambahan, yang merupakan fitur hasil ekstraksi *crawler*. Data yang dihasilkan memiliki dimensi 292 baris dan 14 kolom. Namun, untuk proses evaluasi dan *training*, tidak semua fitur digunakan sebagai input bagi model *machine learning*, beberapa fitur hanya merupakan fitur informatif yang diminta oleh pihak DOKU untuk ditampilkan pada JSON *response crawler*. Keseluruhan fitur yang diekstrak beserta penjelasannya dapat dilihat pada Tabel 4.2 berikut.

No	Fitur	Deskripsi	Catatan
1.	merchant_name	Nama *merchant* yang bersangkutan. **Nilai**: String	-
2.	website	Alamat situs *merchant* yang bersangkutan. **Nilai**: String	-
3.	links_response	Sebuah *dictionary* yang menampilkan *response code* dari setiap *hyperlink* yang disampel **Nilai**: link (key):*response code* (value)	-
4.	label	Label kategori dari *merchant* yang bersangkutan. **Nilai**: 0(APPROVED)/1(REJECTED)	-
5.	broken_link_score	Binerisasi persentase tautan yang mengembalikan *response code* 200 dari semua *hyperlink* yang disampel. **Nilai**: 0(0-50%)/1(51-100%)	Pengambilan sampel dilakukan dengan *random sampling* (n = 10). *Hyperlink* didefinisikan sebagai semua tautan yang berada di beranda situs web.
6.	link_contact_us_exist	Menandakan adanya tautan halaman kontak di situs *merchant*. **Nilai**: 0/1	-

7.	cu_email_exist	Menandakan apakah adanya alamat email tercantum di situs *merchant*. **Nilai**: 0/1	Ruang pencarian meliputi halaman beranda dan halaman kontak.
8.	cu_phone_number_exist	Menandakan apakah adanya nomor telepon tercantum di situs *merchant*. **Nilai**: 0/1	Ruang pencarian meliputi halaman beranda dan halaman kontak.
9.	link_about_us_exist	Menandakan adanya tautan *About Us* (atau yang sejenisnya) di situs *merchant* **Nilai**: 0/1	-
10.	link_tnc_exist	Menandakan adanya tautan halaman syarat & ketentuan di situs *merchant* **Nilai**: 0/1	-
11.	tnc_refund_policy_exist	Menandakan adanya ketentuan pengembalian barang dan/atau uang di situs *merchant* **Nilai**: 0/1	Ruang pencarian meliputi halaman beranda dan halaman syarat & ketentuan.

Tabel 4.2: Fitur yang diekstrak dari proses *feature engineering*

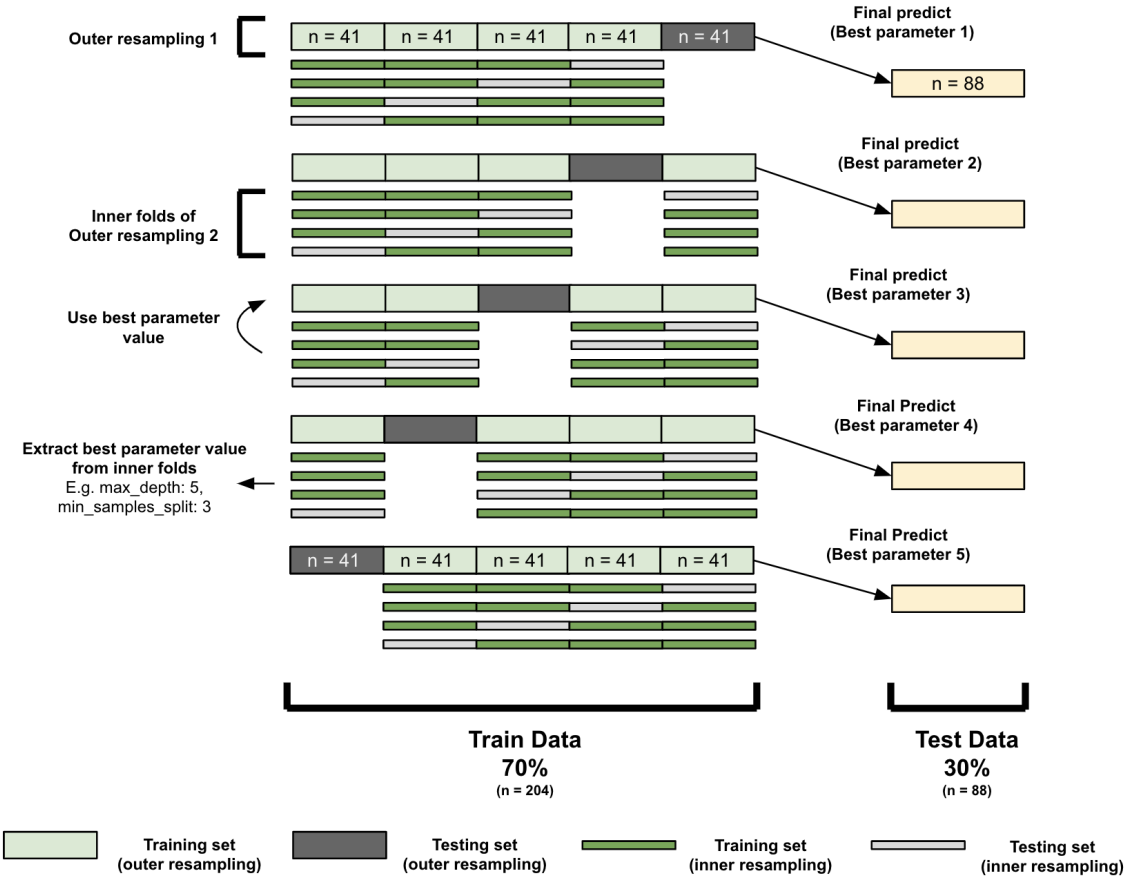
Dari Tabel 4.2, hanya akan ada 8 kolom yang diproses. Kolom-kolom tersebut adalah satu kolom kategori (*label*) dan 7 kolom fitur dari proses *feature engineering* (*broken_link_score, link_tnc_exist, tnc_refund_policy_exist, link_about_us_exist, link_contact_us_exist, cu_email_exist, cu_telephone_number_exist*). Delapan kolom ini akan digunakan sebagai masukan (*training data*) untuk model-model *machine learning* yang akan dievaluasi. Sedangkan ketiga kolom lainnya (*merchant_name*, *website*, *links_response*) hanya akan ditampilkan pada JSON *response crawler* sebagai informasi pelengkap.

4.2 Evaluasi Kinerja Model

Pada tahapan evaluasi kinerja model, ada empat jenis model *classifier* yang digunakan untuk bereksperimen. Diantaranya adalah *decision tree classifier*, *bernoulli naive bayes classifier*, *extreme gradient boost classifier*, dan *random forest classifier*. Training dan evaluasi setiap model dilakukan pada sebuah *notebook* dengan menggunakan bantuan library *scikit-learn*. Metode yang digunakan untuk memilih model terbaik beserta *hyperparameter*nya adalah *nested cross-validation*.

Pada setiap model, akan dipilih beberapa kandidat parameter beserta nilai-nilainya. Pemilihan parameter didasarkan pada pemahaman penulis akan pengaruh parameter yang bersangkutan pada proses prediksi yang akan dilakukan oleh model. Proses evaluasi ini menggunakan 292 baris data yang terdiri atas 215 data dengan label 0 dan 72 data dengan label 1. Pada langkah awal, data ini akan dibagi menjadi dua yaitu 70% *train data* dan 30% *test data*. *Test data* ini tidak akan digunakan sampai langkah terakhir evaluasi, sedangkan untuk *train data*, merujuk pada metode *nested cross-validation*, akan distribusikan dengan metode *stratified* ke dalam 5 fold pada *outer loop* dan pada setiap *train data* di setiap *fold* (*resampling*) pada *outer loop*, data akan dibagi lagi menjadi 4 fold (*resampling*) di *inner loop*. Setelah mendapatkan rata-rata skor terbaik untuk kombinasi paramater x1, x2, .., xn untuk setiap *inner loop* dengan menggunakan metode *hyperparameter tuning* Grid Search (*GridSearchCV*[1]), maka kombinasi parameter tersebut akan difit ke *training data* pada outer loop yang bersangkutan (bukan keseluruhan *training data*) dan disimpan hasilnya. Untuk memastikan stabilitas model, maka 5 kandidat kombinasi parameter terbaik dari 5 fold yang ada di *outer loop* akan difit ke keseluruhan *training data* dan dites ke *testing data* yang belum pernah digunakan. Dari situ bisa didapatkan kesimpulan skor terbaik beserta parameter terbaiknya. Metrik yang akan digunakan untuk mengevaluasi skor adalah *Area Under the Curve* (AUC). Ilustrasi terkait mekanisme ini dapat dilihat lebih lengkap pada Gambar 4.1.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html



Gambar 4.1: Skema *nested cross-validation*

4.2.1 *Decision Tree Classifier*

Pada percobaan ini, *library* yang akan digunakan adalah *DecisionTreeClassifier*[2] yang dimiliki oleh *scikit-learn*. Penggunaannya cukup langsung mengimpor model tersebut ke dalam *notebook*.

Parameter yang akan dipilih sebagai kandidat di model ini adalah *max_depth*, *min_samples_split*, dan *min_samples_leaf*. Ketiga parameter ini dipilih karena dianggap dapat memberikan variasi kombinasi kedalaman *tree* yang dibuat. Kedalaman *tree* yang dibuat akan berpengaruh pada *overfitting*/*underfitting* yang terjadi saat model melakukan *fitting* terhadap *training data*. Kumpulan nilai parameter yang dicoba pada proses ini adalah sebagai berikut:

- **max_depth**: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
- **min_samples_split**: [2, 5, 10, 15, 20, 25, 30]
- **min_samples_leaf**: [1, 3, 5, 10, 15]

[2]https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

4.2.1.1 Parameter Terbaik Hasil Grid Search pada Inner Loop

outer_resampling	max_depth	min_samples _split	min_samples _leaf	inner_avg _score
1	3.0	25.0	3.0	0.912866
2	**4.0**	**20.0**	**1.0**	**0.944893**
3	3.0	25.0	5.0	0.891810
4	4.0	20.0	5.0	0.883346
5	4.0	15.0	5.0	0.885606

Tabel 4.3: Kombinasi parameter terbaik dari *inner loop* model Decision Tree Classifier

Pada Tabel 4.3, dapat dilihat bahwa kombinasi parameter terbaik dikeluarkan oleh *inner loop* yang terdapat pada *outer resampling* (*fold*) 2. Kombinasi parameter terbaik yang dimiliki adalah *max_depth* dengan nilai 4, *min_samples_split* dengan nilai 20, dan *min_samples_leaf* dengan nilai 1. Skor AUC rata-rata yang berhasil dihasilkan oleh kombinasi parameter ini di *inner loop* pada *outer resampling* 2 sebesar **0.944893**. Sedangkan, untuk rata-rata dari keseluruhan hasil *inner loop* pada seluruh *outer resampling* adalah **0.9037042**.

4.2.1.2 Hasil Test pada Outer Loop dan Test Data

outer_resampling	outer_test_score	test_data_score
1	0.910606	0.937672
2	**0.780303**	**0.943871**
3	0.933333	0.937672
4	**0.934848**	**0.943871**
5	0.970219	0.928375

Tabel 4.4: Percobaan kombinasi parameter terbaik ke *outer loop* & *test data* model Decision Tree Classifier

Hasil yang ditampilkan pada Tabel 4.4 merupakan hasil prediksi menggunakan parameter terbaik yang telah didapatkan dari *inner loop* CV yang dilakukan di Tabel 4.3. Prediksi dilakukan ke *testing set* yang ada pada setiap *outer resampling* dan juga ke 25% *test data* yang sudah dipisahkan sejak awal proses. Hasil prediksi terbaik terhadap *testing set* pada *outer resampling* ada di kombinasi parameter 4 dengan skor AUC sebesar **0.934848**. Sedangkan untuk hasil prediksi ke *test data*, hasil tertinggi ada pada 2 kombinasi parameter, yaitu kombinasi 2 dan 4 dengan skor AUC sebesar **0.943871**. Rata-rata skor pada kolom *outer_test_score* adalah **0.9058618** dan rata-rata skor pada kolom *test_data_score*

adalah **0.9382922**.

4.2.2 *Random Forest Classifier*

Pada percobaan ini, *library* yang akan digunakan adalah *RandomForestClassifier*[3] yang dimiliki oleh *scikit-learn*. Penggunaannya cukup langsung mengimpor model tersebut ke dalam *notebook*.

Parameter yang akan dipilih sebagai kandidat di model ini sebenarnya hampir sama dengan Decision Tree. Namun ada dua parameter tambahan yang merupakan ciri khas sebuah *ensemble* model, yaitu *n_estimators* dan *bootstrap*. Parameter-paramater yang dipilih ini menentukan seberapa banyak *learner* yang akan dibangun dan seperti apa setiap *learner* tersebut akan dibangun. Kumpulan nilai parameter yang dicoba pada proses ini adalah sebagai berikut:

- **n_estimators**: [10, 25, 50, 75, 100]
- **max_depth**: [3, 5, 7, 10]
- **bootstrap**: [True, False]
- **min_samples_split**: [3, 5, 7]

4.2.2.1 Parameter Terbaik Hasil Grid Search pada Inner Loop

outer _resampling	**n_estimators**	**max_depth**	**bootstrap**	**min_samples _split**	**inner_avg _score**
1	50.0	7.0	True	7.0	0.938858
2	**10.0**	**7.0**	**True**	**3.0**	**0.954415**
3	50.0	3.0	False	7.0	0.935697
4	10.0	3.0	False	3.0	0.922688
5	10.0	3.0	False	3.0	0.917045

Tabel 4.5: Kombinasi parameter terbaik dari *inner loop* model Random Forest Classifier

Pada Tabel 4.5, dapat dilihat bahwa kombinasi parameter terbaik dikeluarkan oleh *inner loop* yang terdapat pada *outer resampling* (*fold*) 2. Kombinasi parameter terbaik yang dimiliki adalah *n_estimators* dengan nilai 10, *max_depth* dengan nilai 7, *bootstrap* dengan nilai True, dan *min_samples_split* dengan nilai 3. Skor AUC rata-rata yang berhasil dihasilkan oleh kombinasi parameter ini di *inner loop* pada *outer resampling* 2 sebesar **0.954415**. Sedangkan, untuk rata-rata dari keseluruhan hasil *inner loop* pada seluruh

[3]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

outer resampling adalah **0.9337406**.

4.2.2.2 Hasil Test pada Outer Loop dan Test Data

outer_resampling	outer_test_score	test_data_score
1	0.915152	0.942837
2	0.824242	0.931474
3	0.918182	0.943526
4	**0.966667**	**0.953168**
5	**0.959248**	**0.953168**

Tabel 4.6: Percobaan kombinasi parameter terbaik ke *outer loop* & *test data* model Random Forest Classifier

Hasil yang ditampilkan pada Tabel 4.6 merupakan hasil prediksi menggunakan parameter terbaik yang telah didapatkan dari *inner loop* CV yang dilakukan di Tabel 4.5. Prediksi dilakukan ke *testing set* yang ada pada setiap *outer resampling* dan juga ke 25% *test data* yang sudah dipisahkan sejak awal proses. Hasil prediksi terbaik terhadap *testing set* pada *outer resampling* ada di kombinasi parameter 4 dengan skor AUC sebesar **0.966667**. Sedangkan untuk hasil prediksi ke *test data*, hasil tertinggi ada pada 2 kombinasi parameter, yaitu kombinasi 4 dan 5 dengan skor AUC sebesar **0.953168**. Rata-rata skor pada kolom *outer_test_score* adalah **0.9166982** dan rata-rata skor pada kolom *test_data_score* adalah **0.9448346**.

4.2.3 *Extreme Gradient Boost Classifier*

Pada percobaan ini, *library* yang akan digunakan adalah *XGBoost Classifier*[4] untuk Python. Penggunaan dari *library* ini adalah dengan melakukan instalasi via *package manager* Python (*pip*) lalu dapat langsung diimport ke dalam *notebook*.

Karena XGBoost juga merupakan sebuah *tree-based mode*, maka parameter yang akan dipilih sebagai kandidat di model ini juga sebenarnya hampir sama dengan Decision Tree. Namun ada satu *parameter* yang meruapakan ciri khas dari algoritma *boosting*, yaitu *learning_rate*. Parameter ini membantu untuk meregularisasikan model. *Learning_rate* mereduksi pengaruh dari setiap individu *learner* (*tree*) sehingga membuka ruang bagi *tree* berikutnya untuk dapat melakukan improvisasi terhadap model [25]. Kumpulan nilai parameter yang dicoba pada proses ini adalah sebagai berikut:

- **learning_rate**: [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]

[4]https://xgboost.readthedocs.io/en/latest/python/python_api.html

- **max_depth**: [3, 4, 5, 6, 8, 10, 12, 15]
- **min_child_weight**: [1, 3, 5, 7]

4.2.3.1 Parameter Terbaik Hasil Grid Search pada Inner Loop

outer_resampling	learning_rate	max_depth	min_child _weight	inner_avg _score
1	0.10	3.0	5.0	0.924282
2	**0.05**	**3.0**	**5.0**	**0.936703**
3	0.05	3.0	5.0	0.908294
4	0.25	3.0	5.0	0.916014
5	0.20	3.0	5.0	0.911364

Tabel 4.7: Kombinasi parameter terbaik dari *inner loop* model XGBoost Classifier

Pada Tabel 4.7, dapat dilihat bahwa kombinasi parameter terbaik dikeluarkan oleh *inner loop* yang terdapat pada *outer resampling* (*fold*) 2. Kombinasi parameter terbaik yang dimiliki adalah *learning_rate* dengan nilai 0.05, *max_depth* dengan nilai 3, dan *min_child_weight* dengan nilai 5. Skor AUC rata-rata yang berhasil dihasilkan oleh kombinasi parameter ini di *inner loop* pada *outer resampling* 2 sebesar **0.936703**. Sedangkan, untuk rata-rata dari keseluruhan hasil *inner loop* pada seluruh *outer resampling* adalah **0.9193314**.

4.2.3.2 Hasil Test pada Outer Loop dan Test Data

outer_resampling	outer_test_score	test_data_score
1	0.927273	0.952479
2	**0.86818**2	**0.953168**
3	**0.921212**	**0.953168**
4	0.895455	0.945937
5	**0.949843**	**0.945937**

Tabel 4.8: Percobaan kombinasi parameter terbaik ke *outer loop* & *test data* model XGBoost Classifier

Hasil yang ditampilkan pada Tabel 4.8 merupakan hasil prediksi menggunakan parameter terbaik yang telah didapatkan dari *inner loop* CV yang dilakukan di Tabel 4.7. Prediksi dilakukan ke *testing set* yang ada pada setiap *outer resampling* dan juga ke 25% *test data* yang sudah dipisahkan sejak awal proses. Hasil prediksi terbaik terhadap *testing set* pada *outer resampling* ada di kombinasi parameter 5 dengan skor AUC sebesar **0.949843**.

Sedangkan untuk hasil prediksi ke *test data*, hasil tertinggi ada pada 2 kombinasi parameter, yaitu kombinasi 2 dan 3 dengan skor AUC sebesar **0.953168**. Rata-rata skor pada kolom *outer_test_score* adalah **0.912393** dan rata-rata skor pada kolom *test_data_score* adalah **0.9501378**.

4.2.4 *Bernoulli Naive Bayes Classifier*

Pada percobaan ini, *library* yang akan digunakan adalah *BernoulliNB*[5] untuk Python. Penggunaan dari *library* dapat langsung diimport ke dalam *notebook*.

Pada dasarnya semua algoritma Naive Bayes hampir tidak memiliki *hyperparameter* untuk dilakukan *tuning*. Hal ini dikarenakan adanya asumsi independen antara satu fitur dengan fitur lainnya dan *decision rule* yang dimilikinya. Tanpa menentukan *hyperparameter* sebenarnya Bernoulli Naive Bayes sudah akan cukup mengeneralisir hasil prediksinya. Namun ada beberapa *setup* parameter yang dapat dilakukan di awal yang dapat sedikit banyak mengubah hasil model. Parameter-parameter dan nilai-nialinya adalah sebagai berikut:

- **alpha**: [0.5, 1.0]
- **fit_prior**: [True, False]

4.2.4.1 Parameter Terbaik Hasil Grid Search pada Inner Loop

outer_resampling	alpha	fit_prior	inner_avg_score
1	1.0	True	0.923498
2	**0.5**	**True**	**0.946604**
3	1.0	True	0.931831
4	0.5	True	0.927207
5	0.5	True	0.922348

Tabel 4.9: Kombinasi parameter terbaik dari *inner loop* model Bernoulli Naive Bayes Classifier

Pada Tabel 4.9, dapat dilihat bahwa kombinasi parameter terbaik dikeluarkan oleh *inner loop* yang terdapat pada *outer resampling* (*fold*) 2. Kombinasi parameter terbaik yang dimiliki adalah *alpha* dengan nilai 0.5 dan *fit_prior* dengan nilai True. Skor AUC rata-rata yang berhasil dihasilkan oleh kombinasi parameter ini di *inner loop* pada *outer resampling* 2 sebesar **0.946604**. Sedangkan, untuk rata-rata dari keseluruhan hasil *inner loop* pada seluruh *outer resampling* adalah **0.9302976**.

[5]https://scikit-learn.org/stable/modules/generated/sklearn.naive$_b$*ayes.BernoulliNB.html*

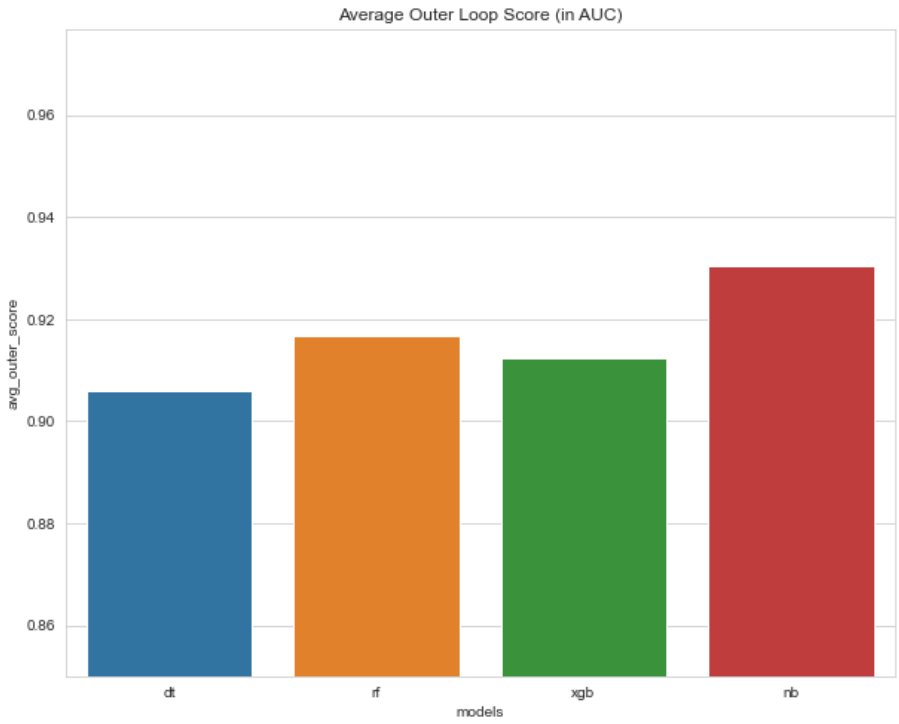
4.2.4.2 Hasil Test pada Outer Loop dan Test Data

outer_resampling	outer_test_score	test_data_score
1	**0.963636**	**0.953168**
2	0.877273	0.953168
3	0.930303	0.953168
4	0.924242	0.953168
5	0.956113	0.953168

Tabel 4.10: Percobaan kombinasi parameter terbaik ke *outer loop* & *test data* model Bernoulli Naive Bayes Classifier

Hasil yang ditampilkan pada Tabel 4.10 merupakan hasil prediksi menggunakan parameter terbaik yang telah didapatkan dari *inner loop* CV yang dilakukan di Tabel 4.9. Prediksi dilakukan ke *testing set* yang ada pada setiap *outer resampling* dan juga ke 25% *test data* yang sudah dipisahkan sejak awal proses. Hasil prediksi terbaik terhadap *testing set* pada *outer resampling* ada di kombinasi parameter 1 dengan skor AUC sebesar **0.963636**. Sedangkan untuk hasil prediksi ke *test data*, hasil tertinggi ada pada semua kombinasi parameter dengan skor AUC sebesar **0.953168**. Rata-rata skor pada kolom *outer_test_score* adalah **0.9303134** dan rata-rata skor pada kolom *test_data_score* adalah **0.953168**.

4.2.5 Kinerja Model Terbaik



Gambar 4.2: Rata-rata skor AUC yang didapatkan setiap model pada *outer loop*

Untuk dapat melihat performa model terbaik berdasarkan hasil generalisasi yang dicapai menggunakan *nested cross validation*, maka kita hanya perlu melihat hasil rata-rata dari *outer_test_score* setiap model. Pada Gambar 4.2, dapat dilihat bahwa semua model memiliki rata-rata AUC di atas 0.9. Hasil ini menunjukkan bahwa keseluruhan model sebenarnya memiliki performa yang stabil dan baik. Semua model telah berhasil mencapai probabilitas di atas 90% untuk dapat mengurutkan keluaran prediksi secara tepat.

Decision Tree Classifier merupakan model terendah dengan skor 0.9058618, disusul oleh XGBoost Classifier dengan 0.912393, lalu Random Forest Classifier dengan 0.9166982, dan skor tertinggi ada pada Bernoulli Naive Bayes Classifier dengan skor 0.9303134.

4.2.6 Kombinasi Hyperparameter Terbaik

Untuk mendapatkan *hyperparameter* terbaik dari 5 kombinasi yang dihasilkan setiap model, maka penulis melakukan *testing* menggunakan setiap kombinasi tersebut ke sebuah *test data* yang sudah dipisahkan dari awal. Data dilakukan *training* pada keseluruhan *train data* (70% dari *dataset* yang dimiliki penulis) menggunakan setiap kombinasi parameter dan dilakukan *testing* ke *test data* serta dicatat skor AUC terbaik yang dihasilkan.

Dari 4 model yang diuji, dapat dilihat bahwa nilai AUC terbaik yang diambil dari kombinasi parameter terbaik keempat model berada pada rentang yang cukup tinggi saat memprediksi *test data*. Skor paling rendah ada pada model Decision Tree Classifier yaitu 0.943871 dan skor paling tinggi dipegang oleh 3 model lainnya secara bersamaan yaitu Random Forest Classifier, XGBoost Classifier, dan Bernoulli Naive Bayes Classifier dengan skor 0.953168. Kemampuan keempat model dalam melakukan prediksi jika diukur dengan AUC, menunjukkan bahwa keempat model tersebut memiliki kemampuan yang baik dalam mengurutkan probabilitas prediksi data positif untuk berada lebih tinggi dibandingkan probabilitas prediksi data negatif. Oleh karena itu, penulis merasa perlu melihat lebih dalam kepada metrik-metrik lainnya, seperti melihat Akurasi, Presisi, Recall, dan F1 dan Confusion Matrix dari setiap model untuk dapat menarik kesimpulan model mana yang merupakan model terbaik terhadap *test data* dengan kombinasi parameter yang dimilikinya.

model	auc	accuracy	balanced _accuracy	precision	recall	f1_score
dt	0.944	0.920	0.871	0.895	0.773	0.829
rf	**0.953**	**0.909**	**0.879**	**0.818**	**0.818**	**0.818**
xgb	0.953	0.898	0.871	0.783	0.818	0.800
nb	**0.953**	**0.864**	**0.864**	**0.679**	**0.864**	**0.760**

Tabel 4.11: Ringkasan metrik terbaik keempat model

model	true_positive	true_negative	false_positive	false_negative
dt	17	64	2	5
rf	**18**	**62**	**4**	**4**
xgb	18	61	5	4
nb	**19**	**57**	**8**	**3**

Tabel 4.12: Ringkasan *confusion matrix* terbaik keempat model

Melihat ringkasan yang ditampilkan pada Tabel 4.11 dan Tabel 4.12, ada dua model yang dapat disorot, yaitu Random Forest Classifier dan Bernoulli Naive Bayes Classifier. Model pertama adalah Random Forest Classifier (kode model **rf**). Model ini memiliki nilai yang stabil dan seimbang di berbagai macam metrik. Tidak ada satupun metrik yang jatuh di bawah angka 0.8. Mengasumsikan *cost* untuk *false positive* dan *false negative* sama, model ini memiliki performa paling bagus jika dilihat dari akurasi yang telah diseimbangkan[6]. *Reliability* model dalam memprediksi label positif yang ditampilkan

[6]Balanced accuracy = (TPR+TNR) / 2

oleh *precision* juga telah menunjukkan angka yang cukup baik di 0.818. Kemampuan model untuk menangkap keseluruhan label positif juga sudah baik dengan nilai *recall* sebesar 0.818. Model ini juga memiliki jumlah kesalahan prediksi yang cukup sedikit, yaitu hanya 4 *false positive* dan 4 *false negative*. Random Forest Classifier dengan kombinasi parameter 4[7], dapat dikatakan sebagai model yang terbaik diantara keempat model yang ada dalam melakukan prediksi terhadap *test data*.

Selain model *rf*, ada juga model lain yang dapat disorot yaitu Bernoulli Naive Bayes Classifier (kode model **nb**). Pada fase awal implementasi, hasil prediksi yang dikeluarkan oleh *web crawler* nantinya akan kembali dievaluasi oleh *end-user*, yaitu tim EDU DOKU. Namun, tim EDU hanya akan melakukan evaluasi manual terhadap situs-situs *merchant* yang diprediksi sebagai *merchant fraud* oleh *web crawler*. Hal ini dilakukan karena tim EDU DOKU memiliki penilaian *cost* yang berbeda terhadap *false negative* dan *false positive* yang dikeluarkan oleh model. Mereka menganggap bahwa adanya cukup banyak *false positive* masih lebih sedikit secara *cost* dibandingkan dengan adanya *merchant fraud* yang diloloskan, walaupun sedikit, sebagai *merchant* yang tidak *fraud* (*false negative*)[8]. Dalam kata lain, *cost false negative* akan menjadi lebih tinggi dibandingkan *cost false positive*. Hal ini lumrah terjadi dalam penerapan *machine learning* untuk melakukan *decision making* di dunia bisnis. Oleh karena itu, untuk menyesuaikan preferensi tersebut, sebuah model yang memiliki *recall* tertinggi dapat dipilih, meskipun memiliki nilai *precision* yang cukup rendah, namun tetap *acceptable*. Model tersebut adalah model Naive Bayes Classifier dengan kombinasi parameter 1[9]. Dengan model ini, angka *recall* dapat mencapai 0.864 dan hanya akan memiliki 3 situs *merchant fraud* yang salah diklasifikasikan. Model ini juga mengembalikan angka *true positive* terbanyak dengan jumlah 19 prediksi positif.

Dari hasil evaluasi di atas, dapat disimpulkan bahwa terdapat dua kandidat model yang dapat dipilih untuk diimplementasikan di *web crawler*. Sejauh ini, model yang diterapkan pada *web crawler* adalah Bernoulli Naive Bayes Classifier. Hal ini dipilih berdasarkan beberapa pertimbangan yang didiskusikan dalam beberapa pertemuan bersama tim Engineering dan tim Product dari pihak DOKU. Diantaranya, model Bernoulli Naive Bayes lebih intuitif dan sederhana untuk dipahami cara bekerjanya. Selain itu, evaluasi model menggunakan 5 jenis kombinasi parameter pada *inner loop*, *outer loop*, dan *test data* juga menunjukkan hasil yang paling konsisten dibandingkan model lainnya. Rata-rata skor AUC pada *inner loop* mencapai angka 0.9302976, sedangkan pada *outer loop* sebesar 0.9303134, dan pada *test data* sebesar 0.953168. Untuk skor individualnya, Bernoulli Naive Bayes Classifier juga tidak memiliki hasil prediksi dengan skor AUC di bawah

[7]Merujuk pada Tabel 4.6

[8]Berdasarkan informasi yang disampaikan oleh Product Owner DOKU, Pak Reza Farasdak

[9]Merujuk pada Tabel 4.10

0.877273. Dengan kombinasi parameter yang cukup sederhana, model *nb* dapat menunjukkan performa yang baik dan telah berhasil mencapai tujuan awal dari pengembangan *web crawler* untuk melakukan klasifikasi situs *merchant* yang *fraud* secara tepat.

BAB 5
PENUTUP

5.1 Kesimpulan

Penelitian ini telah berhasil menciptakan sebuah aplikasi *web crawler* yang dapat digunakan untuk melakukan ekstraksi fitur pada sebuah situs *merchant* dan melakukan *scoring* terhadap situs tersebut menggunakan algoritma *machine learning*. Aplikasi dibangun dengan memanfaatkan sebuah *micro web framework* berbasis Python yang dinamakan Flask. Aplikasi ini dibuat untuk mempersingkat proses verifikasi pendaftaran *merchant* DOKU. Terdapat empat tahapan yang akan dilewati dalam penggunaan *web crawler*. Dimulai dari memasukkan URL situs *merchant*, proses ekstraksi *hyperlink*, proses ekstraksi fitur situs *merchant*, dan proses prediksi kategori (*scoring*) situs *merchant* menggunakan model *machine learning*.

Pada penelitian ini, fitur-fitur yang dapat digunakan untuk melakukan prediksi kategori sebuah situs *merchant* adalah sekumpulan 7 fitur biner. Pada dasarnya fitur-fitur ini merupakan sebuah *completeness checker*. Fitur-fitur ini disusun dan ditentukan bersama dengan tim EDU DOKU dengan mereferensikan kepada SOP pengecekan situs *merchant* yang selama ini mereka miliki. Fitur-fitur tersebut, diantaranya adalah 'broken_link_score', 'link_contact_us_exist', 'cu_email_exist', 'cu_phone_number_exist', 'link_about_us_exist', 'link_tnc_exist', 'tnc_refund_policy_exist' yang mana penjelasan lebih lengkap setiap fiturnya telah dibahas pada Bab 4.

Model *machine learning* yang digunakan pada penelitian ini adalah Bernoulli Naive Bayes Classifier. Model ini dipilih setelah melalui tahap evaluasi kinerja model menggunakan 292 data yang telah dilabel untuk kedua kategori. Model ini memiliki nilai AUC terbaik sebesar 0.953 dan akurasi sebesar 0.864. Untuk setiap situs *merchant*, rata-rata waktu yang diperlukan *web crawler* untuk melakukan prediksi adalah 20 detik. Dengan kecepatan prediksi tersebut dan ketepatan prediksi yang telah terbukti baik, adanya *web crawler* ini dapat meningkatkan efisiensi proses pengecekan situs yang dilakukan oleh tim EDU. Dalam 1 jam, *web crawler* dapat memeriksa 240 situs atau 235 situs lebih banyak dibandingkan pengecekan manual yang hanya sebanyak 5 situs. Adanya *web crawler* dapat meningkatkan efisiensi *rate* pengecekan situs per jam sebesar 4700%.

5.2 Saran

Berdasarkan hasil penelitian ini, berikut ini adalah saran untuk pengembangan penelitian berikutnya:

1. Penelitian ini menggunakan *dataset* yang sangat sedikit dan memiliki jumlah kategori yang tidak seimbang (*imbalance data*). Untuk mendapatkan performa yang lebih baik dan stabil, sebaiknya jumlah *dataset* dapat ditingkatkan terutama untuk data berlabel 1.
2. Situs *merchant* merupakan suatu *test data* yang *time-sensitive*. Sebagai contoh, situs X bisa saja pada suatu waktu pengecekan diklasifikasikan sebagai suatu situs yang tidak *fraud*, namun konten situs dapat berubah sewaktu-waktu. Sebuah *scheduler* atau mekanisme pengecekan rutin dapat dikembangkan sebagai solusi untuk selalu mendapatkan prediksi terkini.
3. Selain menggunakan pendekatan ekstraksi fitur-fitur yang statis dan *pre-determined*, metode *content analysis* juga dapat diterapkan. Word vector yang terbentuk dari transformasi *content* yang ada pada suatu halaman situs dapat dimanfaatkan sebagai fitur untuk melakukan prediksi.

DAFTAR REFERENSI

[1] A. F. Indonesia, "Fintech landscape di indonesia," Aug 2019.

[2] M. D. Innovation, "Mobile payments in indonesia," 2019.

[3] P. Schueffel, "Taming the beast: A scientific definition of fintech," *Journal of Information Management*, vol. 4, no. 4, pp. 32–54, 2016.

[4] "What is fintech?," Feb 2017.

[5] "Ini dia empat jenis fintech di indonesia," Jan 2018.

[6] "Some insights to kyc, kyb, and what they mean to businesses," Dec 2019.

[7] J. Masanès, *Web Archiving: Issues and Methods*, pp. 1–53. 02 2007.

[8] "What is a crawl frontier?," 2018.

[9] "What is selenium webdriver? difference with rc," Feb 2020.

[10] R. Fielding, J. Gettys, J. Mogul, H. Nielsen, and T. Berners-Lee, "Hypertext transfer protocol – http1.1," Jan 1997.

[11] G. Boeing and P. Waddell, "New insights into rental housing markets across the united states: Web scraping and analyzing craigslist rental listings," *Journal of Planning Education and Research*, 07 2016.

[12] F. Blog, "Web crawler 101: What is a web crawler and how do crawlers work?," 2019.

[13] "Machine learning faq," Mar 2020.

[14] L. Breiman, *Classification and regression trees*. Wadsworth International Group, 1984.

[15] G. Louppe, *Understanding Random Forests: From Theory to Practice*. PhD thesis, University of Liege, Belgium, 10 2014. arXiv:1407.7502.

[16] B. Efron, "Bootstrap methods: Another look at the jackknife," *The Annals of Statistics*, vol. 7, no. 1, p. 1–26, 1979.

[17] J. Brownlee, "A gentle introduction to xgboost for applied machine learning," Apr 2020.

[18] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research*, vol. 11, p. 169–198, 1999.

[19] V. Metsis and et al., "Spam filtering with naive bayes – which naive bayes?," in *THIRD CONFERENCE ON EMAIL AND ANTI-SPAM (CEAS*, 2006.

[20] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008.

[21] G. C. Cawley and N. L. C. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, Oct 2010.

[22] R. T. T. Hastie, J. Friedman, "The elements of statistical learning: Data mining, inference, and prediction," p. 193–224, 2001.

[23] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, p. 861–874, 2006.

[24] "pickle — python object serialization."

[25] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics amp; Data Analysis*, vol. 38, no. 4, p. 367–378, 2002.

LAMPIRAN

LAMPIRAN 1: JUDUL LAMPIRAN 1

Subbab dari Lampiran 1

Isi subbab dari lampiran 1.