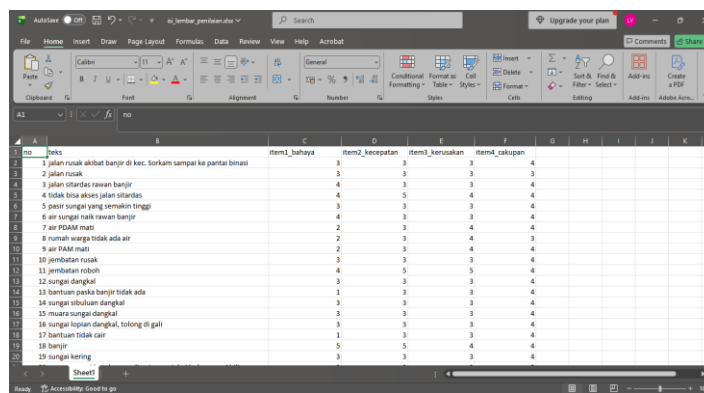


LAMPIRAN

A. Dokumentasi



Gambar A.1 Wawancara Kepala Desa



no	desk	Item1_bahaya	Item2_kesepakatan	Item3_kemudahan	Item4_rakupan
1	jalan rusak akibat banjir di kec. sorkam sampai ke pantai binasi	3	3	3	4
2	jalan rusak	3	3	3	3
3	jalan standar rawan banjir	4	3	3	4
4	tidak bisa akses jalan standar	4	5	4	4
5	pasir sungai yang semakin tinggi	3	3	3	4
6	air sungai rusak rawan banjir	4	3	3	4
7	air PDAM mati	2	3	4	4
8	rumah warga tidak ada air	2	3	4	3
9	air PDAM mati	2	3	4	4
10	jembatan rusak	3	3	3	4
11	jembatan roboh	4	5	5	4
12	sungai dangkal	3	3	3	4
13	bantuan paska banjir tidak ada	1	3	3	4
14	sungai ributuan dangkal	3	3	3	4
15	muara sungai dangkal	3	3	3	4
16	sungai lopan dangkal, tolong di gali	3	3	3	4
17	bantuan tidak cair	1	3	3	4
18	banjir	5	5	4	4
19	sungai kering	3	3	3	4

Gambar A.2 Isi Pelabelan Dataset

B. Kode Sistem Laravel

B.1. PengaduanController.php

```
<?php
namespace App\Http\Controllers;
use App\Models\User;
use App\Models\Dusun;
use App\Models\Kategori;
use App\Models\Pengaduan;
use App\Services\NotifikasiService;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Http;
class PengaduanController extends
Controller
{
    public function create()
    {
        if (Auth::check()) {
            $pengaduanAktif =
Pengaduan::where('user_id', Auth::id())
->whereNotIn('status', [
                'Selesai',
                'Ditolak Dusun',
                'Ditolak Desa',
                'Ditutup Otomatis',
                'Tidak Ditindaklanjuti'
            ])
->exists();
            if ($pengaduanAktif) {
                return redirect('/dashboard')
->with(
                    'error',
                    'Anda masih memiliki
pengaduan yang sedang diproses.'
                );
            }
        }
        $dusuns = Dusun::all();
        $kategoris = Kategori::all();
        return view('pengaduan.create',
compact(
    'dusuns',
    'kategoris'
));
    }
    public function store(Request $request)
    {
        if (Auth::check()) {
            $request->validate([
                'dusun_id' => 'required',
                'kategori_id' => 'required',
                'alamat_pengadu' => 'required',
                'judul_pengaduan' => 'required',
                'deskripsi_pengaduan' =>
'required',
                'lokasi_pengaduan' => 'required',
                'foto_lokasi' =>
'nullable|image|mimes:jpg,jpeg,png|max:2
048',
            ]);
            $user = Auth::user();
            $namaPengadu = $user->name;
            $nomorHp = $user->phone;
        } else {
            $request->validate([
                'nama_pengadu' => ['required',
'regex:/^[pL\s]+$/u'],
                'nomor_hp' => 'required',
                'dusun_id' => 'required',
            ]);
        }
    }
}
```

```

        'kategori_id' => 'required',
        'alamat_pengadu' => 'required',
        'judul_pengaduan' => 'required',
        'deskripsi_pengaduan' =>
'required',
        'lokasi_pengaduan' => 'required',
        'foto_lokasi' =>
'nullable|image|mimes:jpg,jpeg,png|max:2
048',
    ]);
    $user = User::where('phone',
$request->nomor_hp)->first();
    if (!$user) {
        $user = User::create([
            'name' => $request-
>nama_pengadu,
            'phone' => $request-
>nomor_hp,
            'email' => 'warga' . time() .
'@desa.local',
            'password' =>
Hash::make($request->nomor_hp),
            'role' => 'user',
        ]);
    }
    $namaPengadu = $request-
>nama_pengadu;
    $nomorHp = $request->nomor_hp;
}
$teksPengaduan =
$request->judul_pengaduan . ' ' .
$request->deskripsi_pengaduan;
$prioritas = 'Normal';
try {
    $response = Http::timeout(10)-
>post(

```

```

        config('services.ml_api.url'),
        [
            'teks' => $teksPengaduan
        ]
    );
    if ($response->successful()) {
        $hasilML = $response-
>json('prioritas');
        if (in_array($hasilML, [
            'Urgent',
            'Medium',
            'Normal'
        ])) {
            $prioritas = $hasilML;
        }
    } catch (\Exception $e) {
        $prioritas = 'Normal';
    }
    $fotoPath = null;
    if ($request->hasFile('foto_lokasi')) {
        $fotoPath = $request-
>file('foto_lokasi')
        ->store('pengaduan', 'public');
    }
    if ($prioritas == 'Urgent') {
        $deadline = now()->addHours(3);
    } elseif ($prioritas == 'Medium') {
        $deadline = now()->addDay();
    } else {
        $deadline = now()->addDays(3);
    }
    $pengaduan = Pengaduan::create([
        'user_id' => $user->id,

```

```

        'dusun_id' => $request->dusun_id,
        'kategori_id' => $request-
>kategori_id,
        'nama_pengadu' =>
$namaPengadu,
        'nomor_hp' => $nomorHp,
        'alamat_pengadu' => $request-
>alamat_pengadu,
        'judul_pengaduan' => $request-
>judul_pengaduan,
        'prioritas' => $prioritas,
        'prioritas_ml' => $prioritas,
        'dikoreksi_manual' => false,
        'deskripsi_pengaduan' => $request-
>deskripsi_pengaduan,
        'lokasi_pengaduan' => $request-
>lokasi_pengaduan,
        'foto_lokasi' => $fotoPath,
        'status' => 'Menunggu Dusun',
        'sla_deadline' => $deadline,
    ]);
    NotifikasiService:: kirimKeKepalaDusun(
        $pengaduan,
        'Pengaduan Baru',
        'Terdapat pengaduan baru dari
masyarakat yang memerlukan verifikasi.',
        '/kepala-dusun/pengaduan'
    );
    if (Auth::check()) {
        return redirect('/pengaduan')
        ->with(
            'success',
            'Pengaduan berhasil dikirim.'
        );
    }
}

```

```

return redirect('/')
->with(
    'success',
    'Pengaduan berhasil dikirim.
Gunakan Nama dan Nomor HP untuk login
melihat status pengaduan.'
);
}
public function followUp($id)
{
    $pengaduan =
Pengaduan::findOrFail($id);
    if ($pengaduan->user_id != Auth::id())
    {
        abort(403);
    }
    if ($pengaduan->status != 'Menunggu
Follow Up') {
        return back()->with(
            'error',
            'Pengaduan belum dapat
dilakukan Follow Up.'
        );
    }
    if ($pengaduan->follow_up) {
        return back()->with(
            'error',
            'Follow Up sudah pernah dikirim.'
        );
    }
    $pengaduan->follow_up = true;
    $pengaduan->follow_up_at = now();
    $pengaduan->status = 'Menunggu
Dusun';
    if ($pengaduan->prioritas == 'Urgent')
    {

```

```

        $pengaduan->sla_deadline = now()-
>addHours(3);
    } elseif ($pengaduan->prioritas ==
'Medium') {
        $pengaduan->sla_deadline = now()-
>addDay();
    } else {
        $pengaduan->sla_deadline = now()-
>addDays(3);
    }
    $pengaduan->save();
    NotifikasiService::kirimKeKepalaDus
un(
        $pengaduan,
        'Follow Up Pengaduan',
        'Masyarakat melakukan Follow Up.
Mohon segera lakukan verifikasi.',
        '/kepala-dusun/pengaduan'
    );
    NotifikasiService::kirim(
        Auth::id(),
        'Follow Up Berhasil',
        'Follow Up berhasil dikirim dan
pengaduan telah dikirim kembali ke Kepala
Dusun.',
        '/pengaduan'
    );
    return back()->with(
        'success',
        'Follow Up berhasil dikirim.'
    );
}
public function index()
{
        $pengaduans =
Pengaduan::with('dusun')

```

```

->where('user_id', Auth::id())
->latest()
->get();
        return view('pengaduan.index',
compact('pengaduans'));
    }
    public function adminIndex(Request
$request)
    {
        $pengaduans =
Pengaduan::with(['dusun', 'kategori'])
->where('status', 'Menunggu Desa');
        if ($request->prioritas) {
            $pengaduans->where(
                'prioritas',
                $request->prioritas
            );
        }
        $pengaduans = $pengaduans
->latest()
->get();
        return view(
            'admin.pengaduan.index',
            compact('pengaduans')
        );
    }
    public function show($id)
    {
        $pengaduan = Pengaduan::with([
            'dusun',
            'kategori'
        ])->findOrFail($id);
        return view(
            'admin.pengaduan.show',
            compact('pengaduan')
        );
    }

```

```

}
public function riwayatAdmin()
{
    $diproses = Pengaduan::with('dusun')
        ->where('status', 'Diproses Dusun')
        ->latest()
        ->get();
    $selesai = Pengaduan::with('dusun')
        ->whereIn('status', [
            'Selesai',
            'Ditolak Dusun',
            'Ditolak Desa'
        ])
        ->latest()
        ->get();
    return view(
        'admin.pengaduan.riwayat',
        compact('diproses', 'selesai')
    );
}

public function showRiwayatAdmin($id)
{
    $pengaduan = Pengaduan::with([
        'dusun',
        'kategori'
    ])->findOrFail($id);
    return view(
        'admin.pengaduan.show-riwayat',
        compact('pengaduan')
    );
}

public function terima($id)
{
    $pengaduan =
    Pengaduan::findOrFail($id);

    $pengaduan->status = 'Diproses
    Dusun';
    $pengaduan->save();
    NotifikasiService:: kirimKeKepalaDus
    un(
        $pengaduan,
        'Pengaduan Disetujui',
        'Pengaduan telah disetujui oleh
        Kepala Desa. Silakan segera lakukan
        proses penanganan.',
        '/kepala-dusun/proses'
    );
    return redirect()->back()
        ->with(
            'success',
            'Pengaduan berhasil diterima
            Kepala Desa.'
        );
}

public function tolak(Request $request,
    $id)
{
    $request->validate([
        'alasan_penolakan' => 'required'
    ]);

    $pengaduan =
    Pengaduan::findOrFail($id);
    $pengaduan->status = 'Ditolak Desa';
    $pengaduan->alasan_penolakan =
    $request->alasan_penolakan;
    $pengaduan->save();
    NotifikasiService:: kirimKeUser(
        $pengaduan->user_id,
        'Pengaduan Ditolak',

```

```

        'Pengaduan Anda ditolak oleh
Kepala Desa. Silakan lihat alasan
penolakan pada riwayat pengaduan.',
        '/pengaduan'
    );
    return redirect()->back()
        ->with(
            'success',
            'Pengaduan berhasil ditolak.'
        );
    }
    public function koreksiPrioritas(Request
$request, $id)
    {
        $request->validate([
                                'prioritas' =>
'required|in:Urgent,Medium,Normal',
                                ]);

        $pengaduan =
Pengaduan::findOrFail($id);
        $pengaduan->prioritas = $request-
>prioritas;
        $pengaduan->dikoreksi_manual =
true;

```

B.2. NotifikasiService.php

```

<?php
namespace App\Services;
use App\Models\User;
use App\Models\Notifikasi;
use App\Models\Pengaduan;
class NotifikasiService
{
    public static function kirim(
        $userId,
        $judul,
        $pesan,

```

```

        $pengaduan->dikoreksi_oleh =
Auth::id();
        $pengaduan->dikoreksi_pada =
now();
        if ($request->prioritas == 'Urgent') {
            $pengaduan->sla_deadline = now()-
>addHours(3);
        } elseif ($request->prioritas ==
'Medium') {
            $pengaduan->sla_deadline = now()-
>addDay();
        } else {
            $pengaduan->sla_deadline = now()-
>addDays(3);
        }
        $pengaduan->save();
        return back()->with(
            'success',
            'Prioritas pengaduan berhasil
dikoreksi.'
        );
    }
}

```

```

        $link = null
    ) {
        Notifikasi::create([
            'user_id' => $userId,
            'judul' => $judul,
            'pesan' => $pesan,
            'link' => $link,
        ]);
    }
    public static function kirimKeUser(
        $userId,
        $judul,

```

```

        $pesan,
        $link = null
    ){
        self:: kirim(
            $userId,
            $judul,
            $pesan,
            $link
        );
    }

    public static function
    kirimKeKepalaDusun(
        Pengaduan $pengaduan,
        $judul,
        $pesan,
        $link = null
    ){
        $kepalaDusun = User::where('role',
        'kepala_dusun')
        ->where('dusun_id', $pengaduan-
        >dusun_id)
        ->first();
        if ($kepalaDusun) {
            self:: kirim(
                $kepalaDusun->id,
                $judul,
                $pesan,
                $link
            );
        }
    }

    public static function kirimKeAdmin(
        $judul,

```

```

        $pesan,
        $link = null
    ){
        $admin = User::where('role', 'admin')
        ->first();
        if ($admin) {
            self:: kirim(
                $admin->id,
                $judul,
                $pesan,
                $link
            );
        }
    }

    public static function kirimKeSekretaris(
        $judul,
        $pesan,
        $link = null
    ){
        $sekretaris = User::where('role',
        'sekretaris')
        ->first();
        if ($sekretaris) {
            self:: kirim(
                $sekretaris->id,
                $judul,
                $pesan,
                $link
            );
        }
    }
}

```

C. Kode Machine Learning

C.1. praproses.py

```
import re
from indoNLP.preprocessing import (
    remove_html,
    remove_url,
    replace_slang,
    replace_word_elongation,
)
from Sastrawi.Stemmer.StemmerFactory
import StemmerFactory
from kamus_stem import KOREKSI,
LINDUNGI
_STEMMER = StemmerFactory().create_stemmer()
_INGATAN = {}
# Tanda tanya dan seru dipertahankan
sebagai token.
_TANYA = re.compile(r"\?")
_SERU = re.compile(r"!")
_BUKAN_HURUF = re.compile(r"^[a-z0-9\s]")
_SPASI = re.compile(r"\s+")
def stem_kata(kata: str) -> str:
    # Mengubah satu kata menjadi bentuk
    dasar.
    if kata in _INGATAN:
        return _INGATAN[kata]
    if kata in KOREKSI:
        hasil = KOREKSI[kata]
    elif kata in LINDUNGI:
        hasil = kata
    else:
        hasil = _STEMMER.stem(kata) or kata
    _INGATAN[kata] = hasil
```

```
return hasil
def bersihkan(teks: str) -> str:
    # Membersihkan teks pengaduan
    sebelum masuk ke model.
    t = str(teks).lower()
    t = remove_url(t)
    t = remove_html(t)
    t = _TANYA.sub(" tandatanya ", t)
    t = _SERU.sub(" tandaseru ", t)
    t = _BUKAN_HURUF.sub(" ", t)
    t = _SPASI.sub(" ", t).strip()
    if not t:
        return ""
    t = replace_word_elongation(t)
    t = replace_slang(t)
    return " ".join(stem_kata(k) for k in
t.split() if k)
if __name__ == "__main__":
    for contoh in [
        "gk ada air pdam udh seminggu, tlg
pak!!",
        "RUMAH warga hanyutttt terbawa
banjir bandang",
        "Jln rusak parah di kec. sorkam",
        "ada korban meninggal, kerusakan
berdampak ke banyak penduduk",
        "sebelum banjir petugas menangis
melihat penanganan yang lambat",
        "kapan mulai di bangun??",
    ]:
        print(f"{contoh!r}\n    ->
{bersihkan(contoh)!r}")
```

C.2. kamus_stem.py

```
# Koreksi hasil stemming Sastrawi untuk
menjaga makna kata.
KOREKSI = {
```

Overstemming

"kerusakan": "rusak",
"perusakan": "rusak",
"merusakkan": "rusak",
"berikan": "beri",
"berikanlah": "beri",
"diberikannya": "beri",
"menangis": "tangis",
"menangisi": "tangis",
"tangisan": "tangis",
"pengerukan": "keruk",
"dikeruk": "keruk",
"pengecekan": "cek",
"pemadaman": "padam",
"dipadamkan": "padam",
"perasaan": "rasa",
"terasa": "rasa",
"dirasakan": "rasa",
"perombakkan": "rombak",
"perombakan": "rombak",
"pemetaan": "peta",
"menelan": "telan",
"ditelan": "telan",
"memuas": "puas",
"memuaskan": "puas",
"berupa": "rupa",
"menyerupai": "rupa",
"seandainya": "andai",
"andaikan": "andai",
"serakan": "serak",
"berserakan": "serak",
"beresin": "beres",
"ambulan": "ambulans",
"sesekali": "kali",
"pencarian": "cari",
"mengunjungi": "kunjung",

"pengunjung": "kunjung",
"berkunjung": "kunjung",
"mengasih": "kasih",
"dimengerti": "mengerti",
"pengertian": "mengerti",
"sekedar": "sekadar",

Salah ketik yang ditemukan pada dataset

"melaksanakan": "laksana",
"penangan": "tangani",
"disekolah": "sekolah",
"jambatan": "jembatan",
"terandam": "rendam",
"deluan": "dulu",
"kenak": "kena",
"dirubah": "ubah",
"merubah": "ubah",
"mentri": "menteri",
Keluarga kata "menangani"
"menangani": "tangani",
"penanganan": "tangani",
"ditangani": "tangani",
"tangani": "tangani",
"penanganannya": "tangani",

}

LINDUNGI = {

Kata yang maknanya berubah jika di-stemming

"meninggal",
"berdampak",
"terdampak",
"penduduk",
"kependudukan",
"pendapat",
"berpendapat",
"keadaan",

```

"anggaran",
"penganggaran",
"pengangguran",
"menganggur",
"pengalaman",
"berpengalaman",
"memadai",
"mengenai",
"mengerti",
"cekatan",
"bersalin",
"persalinan",
"terpencil",
"kelaparan",
"keracunan",
"kehilangan",
"kecelakaan",
"pengungsian",
# Kata tugas
"sebelum",
"sebelumnya",
"setelah",
"terhadap",
"menurut",
"sedangkan",
"terlebih",
"sekitar",
# Nama tempat dan istilah lokal
"kalangan",
"binasi",
"pandan",
"sitardas",
"sorkam",
"manduamas",
"pinangSORI",
"sibuluan",

```

```

"garoga",
"barus",
"kadis",
"poli",
"jadup",
# Token tanda baca dari preprocessing
"tandatanya",
"tandaseru",
}
# Beberapa kata berimbuhan tetap perlu
dikoreksi,
# meskipun kata dasarnya ada di daftar
perlindungan.
KOREKSI.setdefault("meninggalkan",
"tinggal")
KOREKSI.setdefault("ditinggalkan",
"tinggal")
__all__ = ["KOREKSI", "LINDUNGI"]

```

C.3. indikator_teks.py

```

import warnings
import numpy as np
from sklearn.base import BaseEstimator,
TransformerMixin
from sklearn.feature_extraction.text import
CountVectorizer
from sklearn.preprocessing import
KBinsDiscretizer
# Tingkat prioritas sebagai angka
berurutan.
TINGKAT = {
    "Normal": 1,
    "Medium": 2,
    "Urgent": 3,
}
NAMA_INDIKATOR = (
    "indikasi_bahaya",

```

```

        "indikasi_kerusakan_layanan",
        "indikasi_administratif",
        "skor_prioritas",
        "panjang_pengaduan",
    )
    KOLOM_PRIORITAS = len(NAMA_INDIKATOR) - 2
    KOLOM_PANJANG = len(NAMA_INDIKATOR) - 1
    # Urutan kelas untuk tiga indikator pertama.
    KELAS_INDIKATOR = (
        "Urgent",
        "Medium",
        "Normal",
    )
    # Indikator yang digunakan untuk perlindungan recall Urgent.
    INDIKATOR_BAHAYA = 0
    NILAI_MIN, NILAI_MAKS = 1, 5
    class SkorIndikator(BaseEstimator,
                        TransformerMixin):
        def __init__(self, min_df=1, alpha=1.0,
                     m_smooth=1.0):
            self.min_df = min_df
            self.alpha = alpha
            self.m_smooth = m_smooth
        def _nilai_mentah(self, daftar_teks):
            netral = self._netral()
            mentah = np.empty(
                (len(daftar_teks),
                 len(NAMA_INDIKATOR)),
                dtype=float,
            )
            for i, teks in enumerate(daftar_teks):
                teks = str(teks)

                # Panjang pengaduan dihitung berdasarkan jumlah kata.
                mentah[i, KOLOM_PANJANG] = len(teks.split())
                kata = [
                    k
                    for k in self.penganalisis_(teks)
                    if k in self.indeks_
                ]
                if not kata:
                    mentah[i, :KOLOM_PANJANG] = netral
                    continue
                posisi = [self.indeks_[k] for k in kata]
                # Tiga indikator kelas menggunakan log-odds tertinggi.
                for j in range(len(KELAS_INDIKATOR)):
                    mentah[i, j] = self.log_odds_[j][posisi].max()
                # Indikator gabungan menggunakan rata-rata tingkat prioritas.
                mentah[i, KOLOM_PRIORITAS] = self.tingkat_kata_[posisi].mean()
            return mentah
        def _netral(self):
            return np.array(
                [float(lo.min()) for lo in self.log_odds_] + [self.umum_],
                dtype=float,
            )
        def fit(self, X, y):
            X = list(X)
            y = np.asarray(y)
            self.pencacah_ = CountVectorizer(
                binary=True,

```

```

        min_df=self.min_df,
    )
    D = self.pencacah_.fit_transform(X)
        self.kosakata_ =
self.pencacah_.get_feature_names_out()
    self.indeks_ = {
        kata: i
            for i, kata in
enumerate(self.kosakata_)
    }
        self.penganalisis_ =
self.pencacah_.build_analyzer()
    n_dok = D.shape[0]
        df =
np.asarray(D.sum(axis=0)).ravel().astype(f
loat)
    # Menghitung log-odds setiap kata
terhadap masing-masing kelas.
    self.log_odds_ = []
    for kelas in KELAS_INDİKATOR:
        di_kelas = y == kelas
        n_kelas = float(di_kelas.sum())
        df_kelas = (
            np.asarray(D[di_kelas].sum(axis
=0))
                .ravel()
                .astype(float)
        )
        df_luar = df - df_kelas
        p_dalam = (
            (df_kelas + self.alpha)
            / (n_kelas + 2 * self.alpha)
        )
        p_luar = (
            (df_luar + self.alpha)
                / ((n_dok - n_kelas) + 2 *
self.alpha)
        )
        self.log_odds_.append(
            np.log(p_dalam / p_luar)
        )
    # Menghitung tingkat prioritas rata-rata
setiap kata.
    tingkat_dok = np.array(
        [TINGKAT[str(v)] for v in y],
        dtype=float,
    )
    jumlah = np.asarray(
        D.T.dot(tingkat_dok)
    ).ravel()
        self.umum_ =
float(tingkat_dok.mean())
    self.tingkat_kata_ = (
        jumlah + self.m_smooth *
self.umum_
    ) / (df + self.m_smooth)
    # Menentukan pembagian skor
menjadi 1-5 berdasarkan kuantil data latih.
    mentah = self._nilai_mentah(X)
    self.pembagi_ = []
    for j in range(mentah.shape[1]):
        kolom = mentah[:, [j]]
        if len(np.unique(kolom)) <
NILAI_MAKS:
            self.pembagi_.append(None)
            continue
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
            pembagi = KBinsDiscretizer(
                n_bins=NILAI_MAKS,
                encode="ordinal",

```

```

        strategy="quantile",
        subsample=None,
    )
    pembagi.fit(kolom)
    self.pembagi_.append(pembagi)
    return self

def transform(self, X):
    mentah = self._nilai_mentah(list(X))
    hasil = np.full(
        mentah.shape,
        NILAI_MIN,
        dtype=float,
    )

    for j, pembagi in
enumerate(self.pembagi_):
        if pembagi is None:
            continue
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
            hasil[:, j] = (
                pembagi
                .transform(mentah[:, [j]])
                .ravel()
                + NILAI_MIN
            )
    return hasil

    def get_feature_names_out(self,
input_features=None):
        return np.asarray(
            NAMA_INDIKATOR,
            dtype=object,
        )

def skor_rinci(self, teks):
    # Menghasilkan skor indikator dan
kata pemicu untuk penjelasan.
    skor = self.transform([teks])[0]

kata = [
    k
    for k in self.penganalisis_(str(teks))
    if k in self.indeks_
]
rinci = {}

        for j, nama in
enumerate(NAMA_INDIKATOR):
            pemicu = []
            if kata and j != KOLOM_PANJANG:
                nilai = (
                    self.tingkat_kata_
                    if j == KOLOM_PRIORITAS
                    else self.log_odds_[j]
                )
                pemicu = sorted(
                    kata,
                    key=lambda k:
nilai[self.indeks_[k]],
                    reverse=True,
                )[:3]
                rinci[nama] = {
                    "skor": int(skor[j]),
                    "kata_pemicu": pemicu,
                }
            return rinci

def ambil_skor_indikator(estimator):
    # Mencari SkorIndikator yang sudah
terlatih di dalam pipeline.
    if isinstance(estimator, SkorIndikator):
        return estimator
    for atribut in ("steps", "transformer_list"):
        for _, bagian in getattr(estimator,
atribut, []) or []:
            ditemukan =
ambil_skor_indikator(bagian)

```

```

        if ditemukan is not None:
            return ditemukan
    return None

```

C.4. gabungan.py

```

import numpy as np
from sklearn.base import BaseEstimator,
ClassifierMixin, clone
# Perbandingan suara Random Forest
terhadap Naive Bayes.
BOBOT_RF = 3.0
BOBOT_NB = 1.0
class Gabungan(BaseEstimator,
ClassifierMixin):
    def __init__(
        self,
        rf=None,
        nb=None,
        bobot_rf=BOBOT_RF,
        bobot_nb=BOBOT_NB,
    ):
        self.rf = rf
        self.nb = nb
        self.bobot_rf = bobot_rf
        self.bobot_nb = bobot_nb
    def fit(self, X, y):
        X = list(X)
        self.rf_ = clone(self.rf).fit(X, y)
        self.nb_ = clone(self.nb).fit(X, y)
        self.classes_ = self.rf_.classes_
        # Menyamakan urutan kelas kedua
        model.
        self.urutan_nb_ = [
            list(self.nb_.classes_).index(k)
            for k in self.classes_
        ]
    return self

```

```

def predict_proba(self, X):
    X = list(X)
    p_rf = self.rf_.predict_proba(X)
    p_nb = self.nb_.predict_proba(X)[:,
self.urutan_nb_]
    total = self.bobot_rf + self.bobot_nb
    return (
        self.bobot_rf * p_rf
        + self.bobot_nb * p_nb
    ) / total
def predict(self, X):
    return np.asarray(
        self.classes_[
            self.predict_proba(X).argmax(ax
is=1)
        ],
        dtype=object,
    )
@property
def steps(self):
    # Memungkinkan SkorIndikator
    ditemukan di dalam model RF.
    return [
        ("rf", getattr(self, "rf_", self.rf)),
        ("nb", getattr(self, "nb_", self.nb)),
    ]
def peluang_rinci(self, teks):
    # Menampilkan probabilitas dari
    masing-masing model.
    satu = [teks]
    p_rf = self.rf_.predict_proba(satu)[0]
    p_nb = self.nb_.predict_proba(satu)[0][self.urutan
_nb_]
    return {
        "random_forest": {

```

```

        str(k): round(float(v), 3)
        for k, v in zip(self.classes_, p_rf)
    },
    "naive_bayes": {
        str(k): round(float(v), 3)
        for k, v in zip(self.classes_, p_nb)
    },
}

```

C.5. jaring_pengaman.py

```

import numpy as np
from sklearn.base import BaseEstimator,
ClassifierMixin, clone
from sklearn.metrics import
accuracy_score, f1_score, recall_score
from sklearn.model_selection import
StratifiedKFold
KELAS_DIJAGA = "Urgent"
# Bobot yang dicoba untuk Urgent dan
Normal.
BOBOT_DICoba = (1.0, 1.15, 1.3, 1.5,
1.8, 2.2)
# Porsi akurasi dan macro-F1 saat memilih
bobot.
CAMPUR = 0.5
# Toleransi untuk menentukan apakah
jaring benar-benar aktif.
TOLERANSI = 1e-9
class JaringPengaman(BaseEstimator,
ClassifierMixin):
    def __init__(
        self,
        dasar=None,
        n_lipatan=5,
        acak=0,
        campur=CAMPUR,
    ):

```

```

        self.dasar = dasar
        self.n_lipatan = n_lipatan
        self.acak = acak
        self.campur = campur
    def _tebak(self, peluang, bobot):
        # Memilih kelas berdasarkan peluang
yang sudah dikalikan bobot.
        return np.asarray(
            self.kelas_[np.argmax(peluang *
bobot, axis=1)],
            dtype=object,
        )
    def _nilai(self, y, p):
        akurasi = accuracy_score(y, p)
        macro = f1_score(
            y,
            p,
            average="macro",
            zero_division=0,
        )
        recall = recall_score(
            y,
            p,
            labels=[KELAS_DIJAGA],
            average="macro",
            zero_division=0,
        )
        nilai = (
            self.campur * akurasi
            + (1 - self.campur) * macro
        )
        return nilai, recall
    def _peluang_luar_lipatan(self, X, y):
        # Menghasilkan peluang dari data
yang tidak digunakan saat melatih model.
        pembagi = StratifiedKFold(

```

```

        self.n_lipatan,
        shuffle=True,
        random_state=self.acak,
    )
    peluang = None
    for latih, uji in pembagi.split(X, y):
        model = clone(self.dasar).fit(
            X[latih],
            y[latih],
        )
        if peluang is None:
            self.kelas_ = np.asarray(
                model.classes_,
                dtype=object,
            )
            peluang = np.zeros(
                (len(y), len(self.kelas_)),
                dtype=float,
            )
            peluang[uji] =
model.predict_proba(X[uji])
        return peluang
    def _pilih_bobot(self, X, y):
        X = np.asarray(X, dtype=object)
        y = np.asarray(y, dtype=object)
        peluang =
self._peluang_luar_lipatan(X, y)
        terbaik = np.ones(len(self.kelas_))
        nilai_terbaik = (-1.0, -1.0)
        for w_urgent in BOBOT_DICоба:
            for w_normal in BOBOT_DICоба:
                bobot = np.array(
                    [
                        {
                            "Urgent": w_urgent,
                            "Normal": w_normal,
                                }.get(str(k), 1.0)
                                for k in self.kelas_
                            ],
                            dtype=float,
                        )
                    nilai = self._nilai(
                        y,
                        self._tebak(peluang, bobot),
                    )
                    if nilai > nilai_terbaik:
                        terbaik = bobot
                        nilai_terbaik = nilai
                return terbaik

    def fit(self, X, y):
        X = list(X)
        y = np.asarray(y, dtype=object)
        self.bobot_ = self._pilih_bobot(X, y)
        self.dasar_ = clone(self.dasar).fit(X, y)
        self.classes_ = self.dasar_.classes_
        self.kelas_ = np.asarray(
            self.classes_,
            dtype=object,
        )
        return self

    def predict(self, X):
        peluang =
self.dasar_.predict_proba(list(X))
        return self._tebak(peluang,
self.bobot_)

    def predict_proba(self, X):
        return
self.dasar_.predict_proba(list(X))

    @property
    def aktif_(self):

```

```

        # True jika ada bobot yang berbeda
        dari 1.
        return bool(
            np.any(
                np.abs(self.bobot_ - 1.0) >
TOLERANSI
            )
        )
    def ringkasan_bobot(self):
        # Mengembalikan bobot setiap kelas
        dalam bentuk yang mudah dibaca.
        return {
            str(k): round(float(b), 2)
            for k, b in zip(self.kelas_,
self.bobot_)
        }
    def jaring_aktif(self, teks):
        # Memeriksa apakah jaring mengubah
        keputusan untuk teks tertentu.
        peluang =
self.dasar_.predict_proba([teks])[0]
        tanpa = str(
            self.kelas_[int(np.argmax(peluang))]
        )
        dengan = str(
            self.kelas_[
                int(np.argmax(peluang *
self.bobot_))
            ]
        )
        return tanpa != dengan

```

C.6. latih_ulang.py

```

import warnings
import joblib
import numpy as np
import pandas as pd

```

```

from sklearn.base import clone
from sklearn.ensemble import
RandomForestClassifier
from sklearn.feature_extraction.text import
CountVectorizer
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score,
    recall_score,
)
from sklearn.model_selection import (
    RepeatedStratifiedKFold,
    StratifiedKFold,
    cross_val_predict,
)
from sklearn.naive_bayes import
ComplementNB
from sklearn.pipeline import FeatureUnion,
Pipeline
from praproses import bersihkan
from indikator_teks import SkorIndikator,
NAMA_INDIKATOR
from jaring_pengaman import
JaringPengaman, CAMPUR
from gabungan import Gabungan,
BOBOT_RF, BOBOT_NB
warnings.filterwarnings("ignore")
BERKAS_DATASET =
"dataset/dataset_latih_prioritas.xlsx"
BERKAS_HASIL =
"dataset/hasil_pengujian.txt"
BERKAS_MODEL =
"model/model_prioritas.joblib"
KELAS = ["Urgent", "Medium", "Normal"]

```

```

_baris = []
def catat(teks=""):
    print(teks)
    _baris.append(str(teks))
def bangun_model():
    # Random Forest menggunakan bag of
    words + lima indikator teks.
    fitur = FeatureUnion([
        (
            "kata",
            CountVectorizer(
                ngram_range=(1, 2),
                min_df=1,
                binary=True,
            ),
        ),
        (
            "indikator",
            SkorIndikator(),
        ),
    ])
    hutan = Pipeline([
        (
            "fitur",
            fitur,
        ),
        (
            "rf",
            RandomForestClassifier(
                n_estimators=1200,
                class_weight="balanced",
                max_features="sqrt",
                random_state=42,
                n_jobs=-1,
            ),
        ),
    ])
    bayes = Pipeline([
        (
            "kata",
            CountVectorizer(
                ngram_range=(1, 2),
                min_df=1,
                binary=True,
            ),
        ),
        (
            "nb",
            ComplementNB(alpha=0.3),
        ),
    ])
    # Gabungan RF dan NB dengan bobot
    3:1, lalu jaring pengaman.
    return JaringPengaman(
        Gabungan(
            hutan,
            bayes,
        )
    )
def main():
    df =
    pd.read_excel(BERKAS_DATASET)
    X = np.array(
        [bersihkan(t) for t in
         df["teks"].astype(str)],
        dtype=object,
    )
    y = np.array(
        df["prioritas"].values,
        dtype=object,
    )

```

```

)
catat("=" * 70)
    catat("HASIL PELATIHAN DAN
PENGUJIAN MODEL")
    catat("Klasifikasi Prioritas Pengaduan —
Desa Simaninggir")
    catat("=" * 70)
    catat()
    catat("1. DATA")
    catat("-" * 70)
    catat(f"Jumlah data : {len(df)} baris")
    catat("Sebaran kelas :")
    for k in KELAS:
        n = int((y == k).sum())
        catat(
            f" {k:8s} {n:4d} baris ({n / len(y) *
100:4.1f}%)"
        )
    catat()
    if "prioritas_lama" in df.columns:
        berubah = int(
            (df["prioritas"] !=
df["prioritas_lama"]).sum()
        )
        catat(
            "Asal label : instrumen skala
Likert 4 item, dinilai kepala desa,"
        )
        catat(
            " skor total 4-20 dipetakan
ke tiga kelas"
        )
        catat(
            f" {berubah} dari {len(df)}
baris berbeda dari label lama"
        )

```

```

catat("Sebaran lama :")
for k in KELAS:
    n = int((df["prioritas_lama"] ==
k).sum())
    catat(
        f" {k:8s} {n:4d} baris ({n / len(df)
* 100:4.1f}%)"
    )
    else:
        catat(
            "Asal label : instrumen skala
Likert "
        )
        catat()
        catat("2. METODE")
        catat("-" * 70)
        catat(
            "Praproses : indoNLP (kata gaul,
perpanjangan huruf, URL/HTML)"
        )
        catat(
            " + Sastrawi (stemming Nazief-
Adriani)"
        )
        catat(
            " + kamus koreksi stemming
(kamus_stem.py)"
        )
        catat(
            "Fitur : bag of words 1-2 kata,
ada/tidak ada "
            "(CountVectorizer binary)"
        )
        catat(
            " + 5 skor indikator 1-5, bobot
dihitung dari data"
        )

```

```

)
catat(
    f"          indikator: {'
.join(NAMA_INDIKATOR)}"
)
catat(
    "Model      : gabungan dua model,
suara dirata-ratakan berbobot"
)
catat(
    f"          Random Forest (1200 pohon,
class_weight balanced) : "
    f"Naive Bayes = {BOBOT_RF:.0f} :
{BOBOT_NB:.0f}"
)
catat(
    "          Random Forest memakai bag
of words + skor indikator;"
)
catat(
    "          Naive Bayes (ComplementNB)
memakai bag of words"
)
catat(
    "Jaring      : bobot keputusan tiap kelas,
dipilih otomatis dari data latih"
)
catat(
    f"          kebijakan pemilihan:
{CAMPUR:.0%} akurasi "
    f"+ {1 - CAMPUR:.0%} macro-F1"
)
catat()
catat("Pengujian : repeated stratified k-
fold cross-validation")

```

```

catat("          5 lipatan x 3 pengulangan
x 4 seed = 60 kali pengujian")
catat(
    "          Seluruh langkah dilatih ulang
di dalam tiap lipatan,"
)
catat(
    "          sehingga data uji tidak ikut
membentuk model."
)
catat()
catat("3. HASIL PENGUJIAN PER
SEED")
catat("-" * 70)
catat(
    f"{'Seed':>6} {'Akurasi':>18} "
    f"{'Macro-F1':>18} {'Recall Urgent':>23}"
)
semua_akurasi = []
semua_macro = []
semua_urgent = []
for seed in (11, 23, 57, 99):
    a = []
    m = []
    u = []
    cv = RepeatedStratifiedKFold(
        n_splits=5,
        n_repeats=3,
        random_state=seed,
    )
    for tr, te in cv.split(X, y):
        model = clone(
            bangun_model()
        ).fit(X[tr], y[tr])
        p = np.asarray(
            model.predict(X[te]),

```

```

        dtype=object,
    )
    a.append(
        accuracy_score(y[te], p)
    )
    m.append(
        f1_score(
            y[te],
            p,
            average="macro",
            zero_division=0,
        )
    )
    u.append(
        recall_score(
            y[te],
            p,
            labels=["Urgent"],
            average="macro",
            zero_division=0,
        )
    )
    catat(
        f'{seed:>6} '
        f'{np.mean(a) * 100:6.1f}% ± {np.std(a,
ddof=1) * 100:4.1f}% '
        f'{np.mean(m) * 100:6.1f}% ± {np.std(m,
ddof=1) * 100:4.1f}% '
        f'{np.mean(u) * 100:6.1f}% ± {np.std(u,
ddof=1) * 100:4.1f}% '
    )
    semua_akurasi += a
    semua_macro += m
    semua_urgent += u
    catat("-" * 70)
    catat(
        f'{'RATA-RATA':>6} '
        f'{np.mean(semua_akurasi) *
100:6.1f}% ± {np.std(semua_akurasi,
ddof=1) * 100:4.1f}% '
        f'{np.mean(semua_macro) * 100:6.1f}%
± {np.std(semua_macro, ddof=1) *
100:4.1f}% '
        f'{np.mean(semua_urgent) * 100:6.1f}%
± {np.std(semua_urgent, ddof=1) *
100:4.1f}% '
    )
    catat()
    catat("4. PERFORMA PER KELAS")
    catat("-" * 70)
    pred = cross_val_predict(
        bangun_model(),
        X,
        y,
        cv=StratifiedKFold(
            5,
            shuffle=True,
            random_state=7,
        ),
    )
    pred = np.asarray(pred, dtype=object)
    catat(
        classification_report(
            y,
            pred,
            digits=3,
            zero_division=0,
        )
    )
    catat("5. MATRIKS KEBINGUNGAN
(CONFUSION MATRIX)")
    catat("-" * 70)

```

```

cm = confusion_matrix(
    y,
    pred,
    labels=KELAS,
)
catat(
    f{"sebenarnya / prediksi":>22} '
    + "".join(f"{k:>10}" for k in KELAS)
)
for i, k in enumerate(KELAS):
    catat(
        f"{k:>22} "
        + "".join(f"{v:>10}" for v in cm[i])
    )
catat()
benar = int(np.trace(cm))
total = int(cm.sum())
catat(
    f"Dinilai benar: {benar} dari {total}
pengaduan "
    f"({{benar / total * 100:.1f}}%)
)
catat()
catat("6. MODEL")
catat("-" * 70)
model = bangun_model().fit(X, y)
joblib.dump(model, BERKAS_MODEL)
if not model.aktif_:
    catat(
        "Jaring pengaman : tidak aktif —
pada data ini bobot 1 "
        "untuk semua kelas"
    )
    catat(
        "
        sudah yang terbaik."
    )
else:
    bobot = model.ringkasan_bobot()
    catat(
        "Jaring pengaman : aktif, bobot
keputusan tiap kelas"
    )
    catat(
        "
        "
        + " | ".join(
            f"{k} x{v}"
            for k, v in bobot.items()
        )
    )
    catat(
        "
        (dipilih otomatis dari
data latih)"
    )
    catat()
    catat(
        "Sepuluh indikator paling berpengaruh
menurut Random Forest"
    )
    catat(
        "(bagian Naive Bayes tidak memakai
skor indikator, "
        "jadi tidak tercantum):"
    )
    rf = model.dasar_.rf_.named_steps["rf"]
    nama = model.dasar_.rf_.named_steps[
        "fitur"
    ].get_feature_names_out()
    for i in np.argsort(
        rf.feature_importances_
    )[::-1][:10]:
        catat(

```

```

        f' {str(nama[i]).replace("kata__",
"").replace("indikator__", "[indikator]
"):32s}'
        f' {rf.feature_importances_[i]:.4f}"
    )
    catat()
    catat(
        f"Model dilatih pada seluruh data dan
disimpan ke {BERKAS_MODEL}"
    )
    catat()
    catat("=" * 70)
    with open(
        BERKAS_HASIL,
        "w",
        encoding="utf-8",
        newline="\r\n",
    ) as f:
        f.write(
            "\n".join(_baris) + "\n"
        )
    print(
        f"Hasil pengujian disimpan ke
{BERKAS_HASIL}"
    )
if __name__ == "__main__":
    main()

```

C.7. prediksi.py

```

import os
import re
import sys
import joblib
from praproses import bersihkan
from indikator_teks import
ambil_skor_indikator
_MODEL = None

```

```

_PATH = os.path.join(
    os.path.dirname(os.path.abspath(__file
__)),
    "model",
    "model_prioritas.joblib",
)
def _muat():
    # Memuat model hanya sekali saat
pertama digunakan.
    global _MODEL
    if _MODEL is None:
        _MODEL = joblib.load(_PATH)
    return _MODEL
def _tanpa_huruf(teks):
    # Memeriksa apakah teks tidak memiliki
huruf.
    return not re.search(
        r"[a-zA-Z]",
        str(teks),
    )
def prediksi_prioritas(teks):
    # Menghasilkan prioritas untuk satu teks
atau daftar teks.
    satuan = isinstance(teks, str)
    daftar = [teks] if satuan else list(teks)
    hasil = [
        str(x)
        for x in _muat().predict(
            [bersihkan(t) for t in daftar]
        )
    ]
    hasil = [
        "Normal" if _tanpa_huruf(asli) else
pred
        for pred, asli in zip(hasil, daftar)
    ]

```

```

        return hasil[0] if satuan else hasil
def prediksi_rinci(teks):
    # Menghasilkan prediksi lengkap beserta
    peluang dan skor indikator.
    model = _muat()
    bersih = bersihkan(teks)
    prob = model.predict_proba([bersih])[0]
    peluang = {
        str(k): round(float(v), 3)
        for k, v in zip(model.classes_, prob)
    }
    indikator =
ambil_skor_indikator(model.dasar_)
    skor_indikator = (
        {
            nama: info["skor"]
            for nama, info in
indikator.skor_rinci(bersih).items()
        }
        if indikator is not None
        else {}
    )
    if _tanpa_huruf(teks):
        return {
            "prioritas": "Normal",
            "sumber":
"teks_tidak_dapat_dinilai",
            "peluang_model": peluang,
            "skor_indikator": skor_indikator,
        }
    prioritas = str(
        model.predict([bersih])[0]
    )
    sumber = (
        "jaring_pengaman"
        if model.jaring_aktif(bersih)

```

```

        else "model"
    )
    return {
        "prioritas": prioritas,
        "sumber": sumber,
        "peluang_model": peluang,
        "skor_indikator": skor_indikator,
    }
if __name__ == "__main__":
    if len(sys.argv) > 1:
        for teks in sys.argv[1:]:
            hasil = prediksi_rinci(teks)
            print(
                f"{hasil['prioritas']:7s} "
                f"{[hasil['sumber']:15s]} "
                f"{hasil['peluang_model']}"
                f" indikator={hasil['skor_indikator']}"
            )
            f" <- {teks}"
        )
    else:
        print("Gunakan:")
        print(" python prediksi.py \"teks
pengaduan\"")

```

C.8. api.py

```

from flask import Flask, jsonify, request
from prediksi import prediksi_rinci
app = Flask(__name__)
@app.route("/prediksi",
methods=["POST"])
def prediksi():
    data = request.get_json(silent=True) or
{}
    teks = data.get("teks", "")
    if not isinstance(teks, str):
        return jsonify({

```

```

        "error": "teks harus berupa string"
    }, 400
    teks = teks.strip()
    if not teks:
        return jsonify({
            "error": "teks pengaduan tidak boleh
kosong"
        }, 400
    try:
        hasil = prediksi_rinci(teks)
        return jsonify({
            "prioritas": hasil["prioritas"],
            "sumber": hasil["sumber"],
            "peluang_model":
hasil["peluang_model"],
            "skor_indikator":
hasil["skor_indikator"],
        })
    except Exception as e:
        return jsonify({
            "error": "Gagal melakukan prediksi",
            "detail": str(e),
        }, 500
if __name__ == "__main__":
    app.run(
        host="127.0.0.1",
        port=5000,
        debug=False,
    )

```