PROYEK AKHIR
MATA KULIAH KECERDASAN BUATAN

Oleh

M. TEGAR BAYU AL-FASYA

2215061023

Teknik Informatika

Fakultas Teknik Universitas Lampung

UNIVERSITAS LAMPUNG

BANDAR LAMPUNG

2025

DAFTAR ISI

(buat daftar isi sesuaikan dengan konten yang dibuat di pembahasan)

PEMBAHASAN

Bagian 1

Judul

PENERAPAN ALGORITMA BREADTH-FIRST SEARCH DALAM KASUS PENCARIAN BERKAS PADA STRUKTUR DIREKTORI KOMPUTER

Teknik AI

Searching

Algoritma yang dibahas

Uninformed Search Algorithms; Breadth-First Search (BFS)

Penjelasan Program

Algoritma Breadth-First Search (BFS) adalah salah satu metode pencarian yang tidak terinformasi (Uninformed Search Algorithms). BFS berfungsi dengan mengeksplorasi node-node dalam grafik secara bertahap berdasarkan tingkat kedalaman, yaitu "level demi level". Ini berarti bahwa algoritma ini akan mengunjungi semua node pada kedalaman tertentu terlebih dahulu sebelum melanjutkan ke node-node di kedalaman berikutnya. BFS memastikan bahwa jika terdapat solusi, maka solusi dengan jalur terpendek (dalam jumlah langkah yang diambil) akan ditemukan terlebih dahulu.

Dalam kasus mencari file pada folder komputer, fungsi BFS adalah sebagai berikut:

1. Representasi Keadaan: Setiap kondisi (titik) digambarkan sebagai sebuah folder atau file di sistem penyimpanan komputer.
2. Transisi Keadaan: Pindah dari satu folder ke subfolder atau file di dalamnya dianggap sebagai sebuah perpindahan.
3. Langkah Pencarian: Dimulai dari folder awal yang telah ditentukan, BFS akan memeriksa semua file dan subfolder yang ada langsung di folder itu. Setelah semua item di tingkat itu diperiksa, algoritma kemudian akan masuk ke subfolder-subfolder tersebut (dalam urutan) dan mengulangi proses pemeriksaan dari tingkat ke tingkat.
4. Kondisi Berhenti: Algoritma akan berhenti saat file yang dicari sesuai namanya ditemukan, atau saat seluruh struktur folder yang dapat diakses dari titik awal sudah diperiksa dan file tersebut tidak ditemukan.

Adapun penjelasan source code adalah sebagai berikut :

```
import os
from collections import deque

def bfs_file_search(start_path, target_filename):
    queue = deque()
    visited = set()
```

Bagian awal program ini bertanggung jawab untuk mempersiapkan alat-alat dasar yang dibutuhkan untuk melakukan pencarian. Dua modul penting diimpor:, yaitu `os` dan

`collections`. Modul `os` memungkinkan program berinteraksi dengan sistem operasi untuk tugas-tugas seperti memeriksa file atau direktori dan menavigasi jalur. Dari modul `collections`, `deque` diimpor, yang merupakan implementasi antrian yang efisien, sebuah struktur data krusial untuk algoritma BFS. Selanjutnya, fungsi `bfs_file_search` didefinisikan, yang akan menjadi inti dari algoritma pencarian. Fungsi ini menerima dua argumen: `start_path` untuk menentukan direktori awal pencarian, dan `target_filename` untuk nama file yang ingin dicari. Di dalam fungsi ini, sebuah antrian (`queue`) dan sebuah set (`visited`) diinisialisasi. `queue` akan digunakan untuk menyimpan jalur-jalur yang perlu dieksplorasi oleh algoritma, sementara `visited` akan melacak jalur mana saja yang sudah dikunjungi untuk menghindari pengulangan dan *loop* tak terbatas.

```
    if not os.path.isdir(start_path):
        print(f"Error: Jalur awal '{start_path}' bukan direktori yang valid
atau tidak ada.")
        return None

    queue.append(start_path)
    visited.add(start_path)

    while queue:
        current_path = queue.popleft()

        if os.path.isfile(current_path) and os.path.basename(current_path) ==
target_filename:
            return current_path

        if os.path.isdir(current_path):
            try:
                for item_name in os.listdir(current_path):
                    item_path = os.path.join(current_path, item_name)
                    if item_path not in visited:
                        visited.add(item_path)
                        queue.append(item_path)
            except PermissionError:
                print(f"Izin ditolak untuk mengakses: {current_path}
(Lewati)")
            except Exception as e:
                print(f"Terjadi kesalahan saat membaca {current_path}: {e}
(Lewati)")
    return None
```

Kemudian, bagian program ini memulai proses pencarian yang sebenarnya. Pertama, ia memvalidasi start_path yang diberikan oleh pengguna untuk memastikan bahwa itu adalah direktori yang valid dan dapat diakses. Jika tidak, pesan error akan dicetak dan fungsi akan berhenti. Jika valid, start_path ditambahkan ke antrian (queue) dan ditandai sebagai sudah dikunjungi (visited). Loop utama algoritma BFS dimulai dengan while queue:, yang akan terus berjalan selama masih ada jalur yang perlu dieksplorasi dalam antrian.

Di setiap iterasi, program mengambil jalur paling kiri (elemen pertama) dari antrian menggunakan queue.popleft(), menjadikannya current_path yang akan diproses. Kemudian,

program memeriksa apakah current_path adalah file yang dicari. Ini dilakukan dengan dua langkah: pertama, os.path.isfile(current_path) memverifikasi bahwa jalur tersebut mengarah ke sebuah file, dan kedua, os.path.basename(current_path) == target_filename membandingkan nama file tersebut dengan nama target. Jika kedua kondisi terpenuhi, berkas ditemukan, dan program akan mengembalikan jalur lengkapnya.

Jika current_path bukan berkas target tetapi merupakan direktori, program akan menjelajahi isinya. Ini dilakukan dalam blok try-except untuk menangani potensi masalah izin (PermissionError) atau kesalahan lain (Exception) yang mungkin terjadi saat mencoba mengakses direktori. Di dalam blok ini, os.listdir(current_path) digunakan untuk mendapatkan daftar semua item (file atau subdirektori) di dalam current_path. Untuk setiap item_name yang ditemukan, os.path.join() digunakan untuk membuat jalur lengkap (item_path). Jalur ini kemudian diperiksa apakah sudah dikunjungi sebelumnya (if item_path not in visited:). Jika belum, jalur tersebut ditambahkan ke set visited dan dimasukkan ke dalam queue untuk diproses di iterasi berikutnya. Proses ini memastikan bahwa semua item di tingkat kedalaman saat ini dieksplorasi sebelum program beralih ke subdirektori yang baru ditambahkan. Jika loop while queue selesai dan berkas target tidak ditemukan setelah menjelajahi semua jalur yang dapat diakses, fungsi akan mengembalikan None.

```
if __name__ == "__main__":
    starting_directory = r"D:\Kuliah"

    print("\n--- Pencarian Berkas BFS ---")
    print(f"Pencarian akan dimulai dari direktori: {starting_directory}")

    base_file_name = input("Masukkan nama file yang ingin dicari (tanpa
ekstensi, cth: 'laporan_akhir'): ")

    file_extension = input("Masukkan tipe/ekstensi file (cth: 'pdf', 'docx',
'txt'): ")

    if not file_extension.startswith('.'):
        full_target_filename = f"{base_file_name}.{file_extension}"
    else:
        full_target_filename = f"{base_file_name}{file_extension}"

    print(f"\nMencari '{full_target_filename}' di dalam
'{starting_directory}'...")

    found_path = bfs_file_search(starting_directory, full_target_filename)

    if found_path:
        print(f"\nBerkas ditemukan di: {found_path}")
    else:
        print(f"\nBerkas '{full_target_filename}' tidak ditemukan di
'{starting_directory}' atau subdirektorinya.")

    print("\n--- Pencarian Selesai ---")
```

Terakhir, bagian program ini adalah titik masuk utama eksekusi skrip, ditandai dengan if __name__ == "__main__":. Ini memastikan kode di dalamnya hanya berjalan saat skrip dieksekusi langsung, bukan saat diimpor sebagai modul. Di sini, direktori awal pencarian (starting_directory) ditetapkan secara default ke D:\Kuliah untuk kasus spesifik proyek ini. Program kemudian menampilkan pesan pembuka dan menginformasikan direktori awal pencarian kepada pengguna. Interaksi pengguna dimulai dengan meminta base_file_name (nama file tanpa ekstensi) dan file_extension (tipe file) secara terpisah melalui fungsi input(), yang memungkinkan pengguna memasukkan detail file yang dicari. Logika kondisi if not file_extension.startswith('.'): digunakan untuk memastikan full_target_filename (nama file lengkap yang akan dicari) terbentuk dengan benar, menambahkan tanda titik di depan ekstensi jika belum ada. Setelah itu, program mencetak pesan yang menunjukkan file apa yang dicari dan di mana pencarian akan dilakukan. Fungsi bfs_file_search kemudian dipanggil dengan starting_directory dan full_target_filename sebagai argumen. Terakhir, program menampilkan hasil pencarian: jika found_path tidak None (berarti file ditemukan), jalur lengkapnya akan dicetak; jika tidak, pesan bahwa file tidak ditemukan akan ditampilkan. Program diakhiri dengan pesan "Pencarian Selesai".

Link Github

https://github.com/Tgrfasya/UAS-AI-2025

Link video presentasi/penjelasan

https://drive.google.com/file/d/14aZgq_NgVvhFFQsUrYGCEZyjOlPwWuyJ/view?usp=drive_link

Bagian 2

Judul

PENERAPAN ALGORITMA K-MEANS CLUSTERING UNTUK PENGELOMPOKAN ARTIKEL BERITA BERDASARKAN TOPIC PADA DATASET BBC NEWS

Teknik AI

Learning

Algoritma yang dibahas

Unsupervised Learning; Clustering (k-Means)

Penjelasan Program

Algoritma k-Means merupakan salah satu cara dalam pembelajaran tanpa pengawasan yang digunakan untuk mengelompokkan data. Tidak seperti pembelajaran dengan pengawasan yang memerlukan data yang sudah diberi label untuk melatih model, k-Means beroperasi dengan menemukan pola yang tidak terlihat dalam data dan mengelompokkan titik-titik data ke dalam sejumlah klaster k, di mana k adalah jumlah klaster yang ditetapkan sebelumnya. Sasaran utama dari metode ini adalah untuk memperkecil jarak antara titik data dan pusat (centroid) dari klaster yang di mana titik data tersebut berada, sambil juga meningkatkan jarak antara centroid dari klaster yang berbeda.

Dalam kasus pengelompokan artikel berita BBC, berikut adalah fungsi dari algoritma k-Means:

1. Representasi Data: Artikel berita yang awalnya berupa teks biasa perlu diubah menjadi bentuk angka. Proses ini disebut sebagai vektorisasi teks, di mana setiap artikel dikonversi menjadi vektor angka (contohnya menggunakan metode TF-IDF - Frekuensi Istilah-Inverse Frekuensi Dokumen). Vektor ini menggambarkan "fitur" dari artikel tersebut.
2. Penentuan Jumlah Klaster (k): Untuk dataset BBC News, kita tahu ada 5 kategori berita asli (bisnis, hiburan, politik, olahraga, teknologi). Oleh karena itu, kita akan mengatur k=5 untuk algoritma k-Means. Dalam situasi tanpa pengetahuan awal, nilai k dapat ditentukan menggunakan metode seperti Metode Elbow.

Proses Pengelompokan:

1. Algoritma mulai dengan secara acak menetapkan k centroid (pusat klaster) dalam ruang fitur.
2. Setiap artikel (sebagai vektor) kemudian dihubungkan ke centroid terdekat.
3. Setelah semua artikel terhubung, centroid untuk setiap klaster dihitung kembali berdasarkan rata-rata semua artikel yang terhubung ke klaster tersebut.
4. Langkah menghitung dan menetapkan centroid ini diulang terus menerus sampai posisi centroid tidak banyak berubah lagi atau hingga jumlah iterasi maksimum tercapai.

Algoritma k-Means bertujuan untuk secara otomatis menemukan kelompok artikel berita yang memiliki konten atau topik serupa, tanpa diberi tahu sebelumnya tentang topik-topik tersebut. Setelah klaster terbentuk, kita bisa mengenali "topik" dari setiap klaster dengan menganalisis kata kunci utama dalam klaster tersebut, atau (seperti dalam kasus ini) membandingkannya dengan label kategori asli yang ada di dataset untuk memverifikasi dan memberi nama klaster. Ini memungkinkan pengelompokan artikel berita baru secara otomatis ke dalam klaster terkait berdasarkan kesamaan kontennya.

Dengan demikian, algoritma ini menemukan pola tersembunyi dalam data dan mengelompokkan titik-titik data ke dalam k klaster yang sudah ditentukan sebelumnya. Tujuan utamanya adalah untuk mengurangi jarak antara titik data dan centroid (titik tengah) dari klaster yang sesuai, sembari meningkatkan jarak antar centroid dari klaster yang berbeda.

Adapun penjelasan source code adalah sebagai berikut:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score, homogeneity_score,
completeness_score
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
import re
import joblib
```

Bagian awal dari program ini bertugas untuk mengimpor semua modul dan pustaka Python yang diperlukan untuk menjalankan proyek pengelompokan berita. pandas diimpor sebagai pd untuk memudahkan manipulasi data tabular, khususnya membaca file CSV. TfidfVectorizer dari sklearn.feature_extraction.text sangat penting untuk mengubah teks mentah menjadi

representasi numerik yang dapat diproses oleh algoritma machine learning. KMeans dari sklearn.cluster adalah implementasi dari algoritma clustering k-Means itu sendiri. Berbagai metrik evaluasi seperti adjusted_rand_score, homogeneity_score, dan completeness_score diimpor dari sklearn.metrics untuk menilai kualitas hasil pengelompokan. matplotlib.pyplot sebagai plt dan seaborn sebagai sns digunakan untuk membuat visualisasi data, khususnya heatmap matriks kebingungan. nltk dan nltk.corpus.stopwords diimpor untuk pra-pemrosesan teks, khususnya penghapusan stop words (kata-kata umum yang tidak relevan). re (regular expression) digunakan untuk pembersihan teks dari pola-pola tertentu seperti angka dan tanda baca. Terakhir, joblib diimpor untuk menyimpan dan memuat model yang telah dilatih, sehingga model tidak perlu dilatih ulang setiap kali program dijalankan.

```
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    print("NLTK 'stopwords' belum diunduh. Mengunduh sekarang...")
    nltk.download('stopwords')
    print("NLTK 'stopwords' berhasil diunduh.")
```

Bagian kode ini berfungsi untuk memastikan bahwa sumber daya 'stopwords' dari NLTK tersedia sebelum program melanjutkan. Blok try mencoba mencari data 'stopwords' menggunakan nltk.data.find('corpora/stopwords'). Jika sumber daya tersebut tidak ditemukan, sebuah LookupError akan terjadi. Blok except LookupError akan menangkap kesalahan ini dan kemudian mencetak pesan bahwa 'stopwords' belum diunduh, diikuti dengan perintah nltk.download('stopwords') untuk secara otomatis mengunduhnya. Setelah proses pengunduhan selesai, sebuah pesan konfirmasi akan dicetak. Ini memastikan bahwa program memiliki daftar stop words yang diperlukan untuk pra-pemrosesan teks.

```
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    words = text.split()
    words = [word for word in words if word not in stop_words]
    return " ".join(words)
```

Pada bagian ini, program mempersiapkan daftar kata-kata umum yang tidak memberikan banyak makna (`stop_words`) dengan mengambilnya dari korpus `stopwords` NLTK untuk bahasa Inggris. Selanjutnya, sebuah fungsi bernama `preprocess_text` didefinisikan untuk membersihkan dan menyiapkan teks artikel sebelum dianalisis. Fungsi ini melakukan beberapa langkah: pertama, mengubah seluruh teks menjadi huruf kecil (`text.lower()`). Kedua, menghapus semua angka dari teks (`re.sub(r'\d+', '', text)`). Ketiga, menghilangkan tanda baca dan karakter non-alfanumerik lainnya (`re.sub(r'[^\w\s]', '', text)`). Keempat, memecah teks menjadi daftar kata-kata individu (tokenisasi) menggunakan `text.split()`. Kelima, menyaring daftar kata-kata tersebut dengan menghapus `stop_words` yang telah didefinisikan sebelumnya. Terakhir, kata-kata yang sudah bersih digabungkan kembali menjadi sebuah string tunggal dengan spasi sebagai pemisah (`" ".join(words)`), menghasilkan teks yang siap untuk proses vektorisasi.

```
def train_and_save_model(csv_file_path):
    print("\n--- Fase Pelatihan Model (KMeans Clustering) ---")
```

```
    print(f"Memuat dataset '{csv_file_path}'...")
    try:
        df = pd.read_csv(csv_file_path)
    except FileNotFoundError:
        print(f"Error: File '{csv_file_path}' tidak ditemukan. Pastikan file
berada di direktori yang sama.")
        return None, None, None

    documents = df['text']
    true_labels = df['category']

    documents_preprocessed = documents.apply(preprocess_text)

    print("Melakukan vektorisasi TF-IDF...")
    tfidf_vectorizer = TfidfVectorizer(max_features=5000, min_df=5)
    tfidf_matrix = tfidf_vectorizer.fit_transform(documents_preprocessed)

    k = len(true_labels.unique())
    print(f"Jumlah cluster yang akan digunakan (k): {k}")

    print("Melatih model k-Means...")
    kmeans_model = KMeans(n_clusters=k, random_state=42, n_init='auto')
    kmeans_model.fit(tfidf_matrix)

    joblib.dump(tfidf_vectorizer, 'tfidf_vectorizer.pkl')
    joblib.dump(kmeans_model, 'kmeans_model.pkl')
    print("Model TF-IDF Vectorizer dan k-Means berhasil dilatih dan
disimpan.")

    cluster_labels = kmeans_model.labels_
    ari = adjusted_rand_score(true_labels, cluster_labels)
    homogeneity = homogeneity_score(true_labels, cluster_labels)
    completeness = completeness_score(true_labels, cluster_labels)
    print(f"Adjusted Rand Index (ARI): {ari:.4f}")
    print(f"Homogeneity Score: {homogeneity:.4f}")
    print(f"Completeness Score: {completeness:.4f}")

    cluster_category_counts = pd.crosstab(pd.Series(cluster_labels,
name='Cluster Label'), true_labels)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cluster_category_counts, annot=True, fmt='d', cmap='Blues',
linewidths=.5)
    plt.title('Hubungan Cluster vs. Kategori Asli')
    plt.xlabel('Kategori Asli')
    plt.ylabel('Cluster yang Ditemukan')
    plt.show()
    cluster_names = {}
```

```
    print("\nPetakan Cluster ID ke Kategori Asli (berdasarkan observasi
Heatmap):")
    for cluster_id in range(k):
        dominant_category = cluster_category_counts.loc[cluster_id].idxmax()
        cluster_names[cluster_id] = dominant_category
        print(f"Cluster {cluster_id} kemungkinan besar adalah:
{dominant_category}")

    joblib.dump(cluster_names, 'cluster_names.pkl')
    print("Pemetaan nama cluster disimpan.")

    return tfidf_vectorizer, kmeans_model, cluster_names
```

Kemudian bagian fungsi train_and_save_model. Fungsi ini bertanggung jawab penuh atas seluruh proses pelatihan model clustering dan persistensinya. Fungsi ini dimulai dengan mencoba memuat dataset dari jalur file CSV yang diberikan. Jika file tidak ditemukan, ia akan mencetak pesan kesalahan dan mengembalikan tiga nilai None, menunjukkan kegagalan dalam memuat data. Apabila file berhasil dimuat, kolom 'text' dan 'category' dari dataset akan diekstrak masing-masing sebagai documents dan true_labels.

Selanjutnya, semua dokumen akan melalui tahap pra-pemrosesan teks menggunakan fungsi preprocess_text yang telah didefinisikan sebelumnya, memastikan teks dalam format yang bersih. Teks yang sudah diproses ini kemudian diubah menjadi representasi numerik menggunakan TfidfVectorizer. TfidfVectorizer akan menghitung bobot TF-IDF untuk kata-kata, dengan membatasi jumlah fitur unik (max_features) dan mengabaikan kata-kata yang terlalu jarang muncul (min_df), menghasilkan tfidf_matrix yang siap untuk clustering.

Jumlah klaster (k) ditentukan berdasarkan jumlah kategori unik yang ada di true_labels dari dataset asli. Model k-Means kemudian diinisialisasi dengan jumlah klaster yang ditentukan dan dilatih menggunakan tfidf_matrix. Setelah pelatihan selesai, model tfidf_vectorizer dan kmeans_model disimpan ke dalam file .pkl menggunakan joblib.dump(), memungkinkan model untuk dimuat ulang nanti tanpa perlu melatih ulang.

Fungsi ini juga menyertakan bagian evaluasi untuk menilai seberapa baik klaster yang terbentuk cocok dengan kategori asli. Metrik seperti Adjusted Rand Index (ARI), Homogeneity Score, dan Completeness Score dihitung dan dicetak. Sebuah visualisasi heatmap menggunakan seaborn.heatmap dibuat untuk menampilkan Confusion Matrix, yang secara visual menggambarkan hubungan antara cluster ID yang ditemukan dan kategori berita asli.

Terakhir, fungsi ini secara otomatis mengidentifikasi dan memetakan setiap cluster ID ke nama kategori asli yang paling dominan di dalamnya, berdasarkan observasi dari cluster_category_counts. Pemetaan ini disimpan dalam kamus cluster_names dan juga di-dump ke file cluster_names.pkl. Akhirnya, fungsi ini mengembalikan objek tfidf_vectorizer, kmeans_model, dan cluster_names yang sudah terlatih dan terpetakan.

```
def predict_new_article(article_text, vectorizer, model, cluster_names_map):
    print("\n--- Memprediksi Kategori Berita Baru ---")
    if not vectorizer or not model:
        print("Error: Model belum dilatih atau dimuat.")
        return "Model tidak siap"
```

```
    preprocessed_text = preprocess_text(article_text)
    text_vector = vectorizer.transform([preprocessed_text])
    predicted_cluster_id = model.predict(text_vector)[0]
    predicted_category_name = cluster_names_map.get(predicted_cluster_id,
"Tidak Dikenal")

    print(f"Berita baru dikategorikan ke Cluster ID: {predicted_cluster_id}")
    print(f"Kemungkinan kategori: {predicted_category_name}")
    return predicted_category_name
```

Selanjutnya, program mendefinisikan fungsi `predict_new_article`, yang bertujuan untuk mengategorikan teks berita baru yang diberikan oleh pengguna. Fungsi ini menerima empat argumen: teks artikel yang akan diprediksi (`article_text`), objek vektorizer TF-IDF yang sudah dilatih (`vectorizer`), model k-Means yang juga sudah dilatih (`model`), dan peta nama cluster ke kategori asli (`cluster_names_map`). Sebagai langkah pengaman awal, program akan memeriksa apakah `vectorizer` atau `model` belum dimuat atau dilatih; jika salah satu tidak ada, pesan error akan dicetak dan fungsi akan mengembalikan status "Model tidak siap". Setelah itu, teks artikel yang baru akan melalui tahap pra-pemrosesan yang sama menggunakan fungsi `preprocess_text` yang telah didefinisikan sebelumnya, memastikan konsistensi dalam pembersihan teks. Teks yang sudah diproses ini kemudian diubah menjadi representasi vektor numerik menggunakan `vectorizer.transform()`, yang penting untuk menggunakan kosakata dan bobot yang sama seperti saat model dilatih. Vektor teks ini kemudian diberikan kepada model k-Means (`model.predict()`) untuk memprediksi *cluster ID* mana yang paling cocok dengan teks berita tersebut. Akhirnya, *cluster ID* yang diprediksi ini digunakan untuk mendapatkan nama kategori yang lebih mudah dipahami dari `cluster_names_map`. Hasil *cluster ID* dan nama kategori yang diprediksi akan dicetak ke konsol, dan nama kategori yang diprediksi akan dikembalikan oleh fungsi.

```
if __name__ == "__main__":
    print("Memulai proyek Pengelompokan Berita BBC dengan k-Means.")
    csv_path = 'bbc-text.csv'
    tfidf_vectorizer_loaded = None
    kmeans_model_loaded = None
    cluster_names_map_loaded = None

    try:
        tfidf_vectorizer_loaded = joblib.load('tfidf_vectorizer.pkl')
        kmeans_model_loaded = joblib.load('kmeans_model.pkl')
        cluster_names_map_loaded = joblib.load('cluster_names.pkl')
        print("\nModel TF-IDF Vectorizer, k-Means, dan pemetaan nama cluster
berhasil dimuat.")
    except FileNotFoundError:
        print("\nModel belum ditemukan. Melatih model baru...")
        tfidf_vectorizer_loaded, kmeans_model_loaded, cluster_names_map_loaded
= train_and_save_model(csv_path)
```

```
    if tfidf_vectorizer_loaded is None or kmeans_model_loaded is None:
        print("Program tidak dapat melanjutkan karena model tidak tersedia.
Pastikan 'bbc-text.csv' ada.")
        exit()

    while True:
        print("\n-------------------------------------")
        print("Pilih Opsi:")
        print("1. Masukkan teks berita baru untuk dikategorikan")
        print("2. Keluar")
        choice = input("Pilihan Anda: ")

        if choice == '1':
            print("\nSilakan tempel atau ketik teks berita Anda. Ketik
'END_NEWS' di baris baru untuk mengakhiri.")
            lines = []
            while True:
                line = input()
                if line.strip().upper() == 'END_NEWS':
                    break
                lines.append(line)
            new_article_text = "\n".join(lines)

            if not new_article_text.strip():
                print("Tidak ada teks berita yang dimasukkan.")
                continue

            predicted_cat = predict_new_article(new_article_text,
tfidf_vectorizer_loaded, kmeans_model_loaded, cluster_names_map_loaded)
            print(f"Hasil Prediksi: Berita ini kemungkinan masuk kategori
**{predicted_cat}**.")
            print("Catatan: Kategori ini didasarkan pada cluster terdekat,
bukan label asli yang dilatih secara supervised.")

        elif choice == '2':
            print("Terima kasih. Program selesai.")
            break
        else:
            print("Pilihan tidak valid. Silakan coba lagi.")
```

Bagian terakhir program ini merupakan titik eksekusi utama, diawali dengan if __name__ == "__main__":, memastikan kode di dalamnya hanya berjalan saat skrip dieksekusi secara langsung. Program dimulai dengan mencetak pesan pembuka dan menginisialisasi variabel csv_path ke nama file dataset, serta menetapkan variabel untuk model yang akan dimuat (tfidf_vectorizer_loaded, kmeans_model_loaded, cluster_names_map_loaded) ke None.

Kemudian, program mencoba memuat model yang sudah tersimpan (.pkl) dari disk menggunakan joblib.load() dalam blok try. Jika file-file model berhasil dimuat, pesan konfirmasi akan dicetak. Apabila terjadi FileNotFoundError (yang berarti model belum dilatih atau file .pkl tidak ada), program akan mencetak pesan bahwa model tidak ditemukan dan

memanggil fungsi train_and_save_model() untuk melatih model baru dari dataset CSV dan menyimpannya.

Setelah itu, ada pemeriksaan kondisi untuk memastikan bahwa tfidf_vectorizer_loaded dan kmeans_model_loaded tidak None. Jika keduanya masih None (misalnya, karena bbc-text.csv sendiri tidak ditemukan saat pelatihan awal), program akan mencetak pesan kesalahan dan keluar, karena tidak dapat melanjutkan tanpa model.

Jika model berhasil dimuat atau dilatih, program masuk ke dalam sebuah loop while True yang tidak akan berakhir sampai pengguna memilih untuk keluar. Di setiap iterasi loop, program akan menampilkan pilihan kepada pengguna: memasukkan teks berita baru untuk dikategorikan (opsi 1) atau keluar dari program (opsi 2).

Jika pengguna memilih opsi '1', program akan meminta mereka untuk menempel atau mengetik teks berita. Input teks multi-baris ini akan dikumpulkan baris per baris hingga pengguna mengetik 'END_NEWS' (tidak case-sensitive dan toleran terhadap spasi). Teks yang terkumpul kemudian digabungkan menjadi satu string. Jika teks berita yang dimasukkan kosong, program akan mencetak pesan dan kembali ke menu pilihan. Jika ada teks, fungsi predict_new_article() akan dipanggil dengan teks berita baru dan model-model yang sudah dimuat. Hasil prediksi kategori akan dicetak, disertai catatan bahwa kategorisasi didasarkan pada clustering (pembelajaran tak terawasi) dan bukan pada label asli yang dilatih secara supervised.

Jika pengguna memilih opsi '2', program akan mencetak pesan terima kasih dan keluar dari loop, mengakhiri eksekusi. Untuk pilihan lain yang tidak valid, program akan mencetak pesan kesalahan dan kembali ke menu pilihan.

Link Github

https://github.com/Tgrfasya/UAS-AI-2025

Link video presentasi/penjelasan

https://drive.google.com/file/d/1hcse5NQMNuI1-Qurjm9SoIJbUUk0BYIE/view?usp=drive_link