

**PROYEK AKHIR  
MATA KULIAH KECERDASAN BUATAN**

**Oleh**

**ARFAN ANDHIKA PRAMUDYA**

**Teknik Informatika**

**Fakultas Teknik Universitas Lampung**



**UNIVERSITAS LAMPUNG  
BANDAR LAMPUNG**

**2025**

## DAFTAR ISI

<b>PEMBAHASAN</b>	<b>3</b>
Bagian 1	3
1. Judul: “PENERAPAN ALGORITMA A* SEARCH DALAM KASUS PENCARIAN RUTE WISATA OPTIMAL MENGGUNAKAN DATASET WORLDWIDE TRAVEL CITIES”	3
2. Teknik AI: Searching.	3
3. Algoritma	3
4. Program	4
5. Link video presentasi/penjelasan: <a href="https://youtu.be/IyPgTL-TUgc">https://youtu.be/IyPgTL-TUgc</a>	10
Bagian 2	11
1. Judul: ”KLASIFIKASI CUSTOMER CHURN PADA PELANGGAN TELEKOMUNIKASI MENGGUNAKAN ALGORITMA DECISION TREE”	11
2. Teknik AI: Learning.	11
3. Algoritma Supervised Learning; Decision Tree (berdasarkan prinsip C4.5/CART).	11
4. Program.	12
5. Link video presentasi/penjelasan: <a href="https://youtu.be/w7irUxNveXk">https://youtu.be/w7irUxNveXk</a>	19

# PEMBAHASAN

## Bagian 1

1. **Judul:** “PENERAPAN ALGORITMA A\* SEARCH DALAM KASUS PENCARIAN RUTE WISATA OPTIMAL MENGGUNAKAN DATASET WORLDWIDE TRAVEL CITIES”

2. **Teknik AI:** Searching.

### 3. Algoritma

A\* (A-Star) merupakan algoritma pencarian jalur optimal atau efektif yang menggabungkan kelebihan dari algoritma Dijkstra dan Greedy Best-First Search. Algoritma ini menggunakan fungsi evaluasi  $f(n) = g(n) + h(n)$  dimana

- $g(n)$  sebagai Jarak aktual dari titik awal ke node  $n$
- $h(n)$  sebagai Heuristic function - estimasi jarak dari node  $n$  ke tujuan
- $f(n)$  sebagai Total estimasi biaya jalur terpendek melalui node  $n$

Adapun Komponen Utama dalam Program yang telah dibuat adalah

1. Heuristic Function yaitu menggunakan formula Haversine untuk menghitung jarak geodesik antara dua titik koordinat di permukaan bumi
2. Open Set yaitu priority queue (heap) untuk menyimpan node yang akan dieksplorasi atau dijelajahi
3. Closed Set yaitu set untuk menyimpan node yang sudah dieksplorasi
4. Path Reconstruction yaitu melacak kembali jalur optimal dari tujuan ke awal

Dalam program pencarian rute optimal antar kota, algoritma A\* berfungsi sebagai

1. Optimasi Jarak untuk menemukan rute dengan total jarak terpendek antar kota
2. Efisiensi Pencarian yang menggunakan fungsi heuristic untuk mengarahkan pencarian ke arah yang lebih menjanjikan
3. Garanteed Optimal dengan heuristic yang admissible (tidak overestimate), A\* menjamin solusi optimal
4. Skalabilitas yaitu dapat menangani dataset kota dalam skala besar dengan performa yang baik

## 4. Program

```
1 # Bagian 1: Import Library yang Dibutuhkan
2 import pandas as pd
3 import numpy as np
4 import heapq
5 import math
6
7 # Bagian 2: Fungsi Kalkulasi Jarak (Haversine)
8 def haversine(lat1, lon1, lat2, lon2):
9     """Menghitung jarak antara dua titik koordinat menggunakan formula Haversine"""
10    R = 6371 # Radius bumi dalam kilometer
11    lat1_rad, lon1_rad, lat2_rad, lon2_rad = map(np.radians, [lat1, lon1, lat2, lon2])
12    dlon = lon2_rad - lon1_rad
13    dlat = lat2_rad - lat1_rad
14    a = np.sin(dlat / 2)**2 + np.cos(lat1_rad) * np.cos(lat2_rad) * np.sin(dlon / 2)**2
15    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
16    return R * c
17
18 # Bagian 3: Implementasi Algoritma A* Search
19 def a_star_search(city_data, start_city_name, end_city_name):
20     """Implementasi algoritma A* untuk mencari rute optimal antar kota"""
21     lat_col = None
22     lng_col = None
23     lat_variations = ['lat', 'latitude', 'Latitude', 'LAT', 'LATITUDE']
24     lng_variations = ['lng', 'lon', 'longitude', 'Longitude', 'LNG', 'LON', 'LONGITUDE']
25     for col in city_data.columns:
26         if col in lat_variations:
27             lat_col = col
28         if col in lng_variations:
29             lng_col = col
30     if not lat_col or not lng_col:
31         print(f"Error: Tidak dapat menemukan kolom koordinat dalam dataset.")
32         print(f"Kolom yang tersedia: {city_data.columns.tolist()}")
33         return None, None
34     cities = {}
35     for index, row in city_data.iterrows():
36         city_name = str(row[city_name]).strip()
37         lat = float(row[lat_col])
38         lng = float(row[lng_col])
39         cities[city_name] = (lat, lng)
40
41     # Inisialisasi Algoritma A*
42     open_set = []
43     heapq.heappush(open_set, (0, start_city_name))
44     came_from = {}
45     g_score = {city: float('inf') for city in cities}
46     g_score[start_city_name] = 0
47
48     # Hitung heuristic
49     start_coors = cities[start_city_name]
50     end_coors = cities[end_city_name]
51     h_score = haversine(start_coors[0], start_coors[1], end_coors[0], end_coors[1])
52
53     f_score = {city: float('inf') for city in cities}
54     f_score[start_city_name] = h_score
55
56     visited = set()
57
58     while open_set:
59         current_f, current_city_name = heapq.heappop(open_set)
60
61         if current_city_name in visited:
62             continue
63
64         visited.add(current_city_name)
65
66         # Jika sudah sampai tujuan
67         if current_city_name == end_city_name:
68             path = []
69             total_distance = g_score[end_city_name]
70             temp_city = current_city_name
71
72             while temp_city in came_from:
73                 path.append(temp_city)
74                 temp_city = came_from[temp_city]
75             path.append(start_city_name)
76             path.reverse()
77
78             return path, total_distance
79
80     # Explore semua kota lain
81     current_coors = cities[current_city_name]
82
83     for neighbor_city_name, neighbor_coors in cities.items():
84         if neighbor_city_name == current_city_name or neighbor_city_name in visited:
85             continue
86
87         # Hitung jarak ke neighbor
88         distance_to_neighbor = haversine(
89             current_coors[0], current_coors[1],
90             neighbor_coors[0], neighbor_coors[1]
91         )
92
93         tentative_g_score = g_score[current_city_name] + distance_to_neighbor
94
95         if tentative_g_score < g_score[neighbor_city_name]:
96             came_from[neighbor_city_name] = current_city_name
97             g_score[neighbor_city_name] = tentative_g_score
98
99         # Hitung heuristic ke tujuan
100        h_score = haversine(
```

```

101         neighbor_coords[0], neighbor_coords[1],
102         end_coords[0], end_coords[1]
103     )
104
105     f_score[neighbor_city_name] = tentative_g_score + h_score
106     heapq.heappush(open_set, (f_score[neighbor_city_name], neighbor_city_name))
107
108     return None, None
109
110 # Bagian 4: Fungsi Utilitas
111 def load_and_display_data(file_path):
112     """Load data CSV dan tampilkan informasi dataset"""
113     try:
114         df = pd.read_csv(file_path)
115         print(f"Dataset berhasil dimuat!")
116         print(f"Jumlah total baris: {len(df)}")
117         print(f"Kolom yang tersedia: {df.columns.tolist()}")
118         print(f"\nContoh data (5 baris pertama):")
119         print(df.head())
120         return df
121     except FileNotFoundError:
122         print(f"Error: File '{file_path}' tidak ditemukan.")
123         print("Pastikan file CSV berada di folder yang sama dengan script Python ini.")
124         return None
125     except Exception as e:
126         print(f"Error saat membaca file: {e}")
127         return None
128
129 def show_available_cities(city_df, limit=50):
130     """Tampilkan daftar kota yang tersedia"""
131     cities = city_df['city'].unique()
132     print(f"\nDaftar kota yang tersedia ({len(cities)} kota):")
133
134     if len(cities) > limit:
135         print(f"Menampilkan {limit} kota pertama:")
136         for i, city in enumerate(cities[:limit]):
137             print(f"{i+1:2d}. {city}")
138         print(f"... dan {len(cities) - limit} kota lainnya")
139     else:
140         for i, city in enumerate(cities):
141             print(f"{i+1:2d}. {city}")
142
143 def find_similar_cities(city_df, search_term, max_results=5):
144     """Cari kota yang mirip dengan input pengguna"""
145     cities = city_df['city'].str.lower().str.contains(search_term.lower(), na=False)
146     similar_cities = city_df[cities]['city'].tolist()
147     return similar_cities[:max_results]
148
149 def get_city_input(city_df, prompt_text):
150     """Fungsi untuk mendapatkan input kota dari pengguna dengan validasi"""
151     available_cities = city_df['city'].unique()
152
153     while True:
154         city_input = input(prompt_text).strip()
155
156         if city_input in available_cities:
157             return city_input
158         else:
159             print(f"Kota '{city_input}' tidak ditemukan dalam dataset.")
160
161             # Cari kota yang mirip
162             similar_cities = find_similar_cities(city_df, city_input)
163             if similar_cities:
164                 print(f"Mungkin maksud Anda: {' '.join(similar_cities)}")
165
166             print("Ketik 'list' untuk melihat semua kota yang tersedia, atau coba lagi:")
167             choice = input().strip().lower()
168             if choice == 'list':
169                 show_available_cities(city_df)
170
171 # Bagian 5: Blok Eksekusi Utama
172 if __name__ == "__main__":
173     print("="*70)
174     print("PROGRAM PENCARIAN RUTE OPTIMAL MENGGUNAKAN ALGORITMA A*")
175     print("="*70)
176
177     # Load data
178     file_path = 'Worldwide Travel Cities Dataset (Ratings and Climate).csv'
179     full_city_df = load_and_display_data(file_path)
180
181     if full_city_df is None:
182         exit()
183
184     # Cek kolom yang diperlukan
185     required_columns = ['city']
186     missing_columns = [col for col in required_columns if col not in full_city_df.columns]
187
188     if missing_columns:
189         print(f"Error: Kolom yang diperlukan tidak ditemukan: {missing_columns}")
190         exit()
191
192     # Filter data jika diperlukan (opsional)
193     print(f"\nApakah Anda ingin memfilter data berdasarkan negara tertentu? (y/n): ",
194           end='')
195     filter_choice = input().strip().lower()
196
197     if filter_choice == 'y':
198         if 'country' in full_city_df.columns:
199             available_countries = full_city_df['country'].unique()
200             print(f"Negara yang tersedia: {' '.join(available_countries[:10])}...")
201             countries_input = input("Masukkan nama negara (pisahkan dengan koma): ").strip()

```

```

201     selected_countries = [country.strip() for country in countries_input.split(',')]
202
203     city_df = full_city_df[full_city_df['country'].isin(selected_countries)].copy()
204     city_df = city_df.drop_duplicates(subset='city', keep='first')
205     print(f"Data difilter. Tersedia {len(city_df)} kota dari negara yang dipilih.")
206 else:
207     print("Kolom 'country' tidak tersedia. Menggunakan semua data.")
208     city_df = full_city_df.drop_duplicates(subset='city', keep='first')
209 else:
210     city_df = full_city_df.drop_duplicates(subset='city', keep='first')
211     print(f"Menggunakan semua data. Total: {len(city_df)} kota unik.")
212
213 # Tampilkan beberapa kota sebagai contoh
214 show_available_cities(city_df, limit=20)
215
216 print("\n" + "="*50)
217 print("PENCARIAN RUTE")
218 print("="*50)
219
220 start_city = get_city_input(city_df, "Masukkan nama kota awal: ")
221 end_city = get_city_input(city_df, "Masukkan nama kota tujuan: ")
222
223 if start_city == end_city:
224     print("Kota awal dan tujuan sama!")
225     exit()
226
227 print(f"\nMencari rute optimal dari {start_city} ke {end_city}...")
228 print("Menggunakan algoritma A* Search...")
229
230 # Jalankan algoritma A*
231 path, distance = a_star_search(city_df, start_city, end_city)
232
233 print("\n" + "="*50)
234 print("HASIL PENCARIAN")
235 print("="*50)
236
237 if path and distance:
238     print("✅ RUTE OPTIMAL DITEMUKAN!")
239     print(f"\nRute: {' - '.join(path)}")
240     print(f"Total Jarak: {distance:.2f} km")
241     print(f"Jumlah kota yang dilalui: {len(path)}")
242
243 # Tampilkan detail setiap segmen
244 if len(path) > 1:
245     print(f"\nDetail perjalanan:")
246     total_check = 0
247     for i in range(len(path) - 1):
248         # Cari koordinat kota
249         city1_data = city_df[city_df['city'] == path[i]].iloc[0]
250         city2_data = city_df[city_df['city'] == path[i+1]].iloc[0]
251
252         # Deteksi kolom koordinat
253         lat_col = None
254         lng_col = None
255         lat_variations = ['lat', 'latitude', 'Latitude', 'LAT', 'LATITUDE']
256         lng_variations = ['lng', 'lon', 'longitude', 'Longitude', 'LNG', 'LON', 'LONGITUDE']
257
258         for col in city_df.columns:
259             if col in lat_variations:
260                 lat_col = col
261             if col in lng_variations:
262                 lng_col = col
263
264         if lat_col and lng_col:
265             segment_distance = haversine(
266                 city1_data[lat_col], city1_data[lng_col],
267                 city2_data[lat_col], city2_data[lng_col]
268             )
269             total_check += segment_distance
270             print(f" {i+1}. {path[i]} - {path[i+1]}: {segment_distance:.2f} km")
271
272     print(f"\nVerifikasi total jarak: {total_check:.2f} km")
273 else:
274     print("❌ RUTE TIDAK DITEMUKAN")
275     print("Kemungkinan penyebab:")
276     print("- Tidak ada data koordinat yang valid")
277     print("- Algoritma tidak dapat menemukan jalur optimal")

```

Cara penggunaan program diatas sebagai berikut.

1. Unduh dataset yang diinginkan dalam format .csv dan letakkan di dalam folder yang sama dengan file program Python.
2. Buka terminal atau *command prompt*, arahkan ke folder tempat menyimpan file, lalu jalankan program dengan perintah “python nama\_file\_anda.py”
3. Program akan bertanya apakah User ingin memfilter data berdasarkan negara. Ketik y (lalu Enter) jika Anda ingin mencari rute di negara tertentu. Masukkan

- nama negara yang diinginkan (contoh: Indonesia, Malaysia). Ketik `n` (lalu Enter) jika Anda ingin menggunakan semua data kota yang ada di dalam dataset.
4. Program akan menampilkan daftar kota yang tersedia. Masukkan nama kota awal sesuai dengan yang ada di daftar, lalu tekan Enter.
  5. Masukkan nama kota tujuan, lalu tekan Enter. Jika Anda salah ketik, program akan memberi saran nama kota yang mirip atau pilihan untuk melihat daftar lengkap.
  6. Program akan memproses dan menampilkan rute optimal yang ditemukan, total jaraknya, serta detail jarak antar kota yang dilalui.

### Penjelasan

Bagian 1 Import Library yang diperlukan. Pada baris ke-1 hingga ke-5, program mengimpor beberapa library standar Python yang penting. Baris pertama `import pandas as pd` berfungsi untuk mengimpor library *Pandas* yang digunakan untuk membaca, mengelola, dan memanipulasi data dari file CSV dalam sebuah struktur data yang efisien bernama *DataFrame*. Baris kedua `import numpy as np` mengimpor library *NumPy* untuk operasi numerik, terutama untuk fungsi-fungsi matematika seperti `np.radians` dan `np.sqrt` yang digunakan dalam perhitungan jarak. Baris ketiga `import heapq` mengimpor library *heapq* yang menyediakan implementasi priority queue atau antrian prioritas, ini adalah komponen inti dari algoritma A\* untuk secara efisien mengambil kota dengan biaya estimasi terendah. Terakhir, baris keempat `import math` mengimpor library *math* yang juga berisi fungsi-fungsi matematika dasar.

Bagian 2 Fungsi Kalkulasi Jarak (Haversine). Pada baris ke-10 hingga ke-16 terdapat fungsi `haversine(lat1, lon1, lat2, lon2)`. Fungsi ini bertujuan untuk menghitung jarak geografis (jarak garis lurus di atas permukaan bumi) antara dua titik koordinat. Di dalamnya, baris ke-10 mendefinisikan variabel `R` dengan nilai 6371, yang mewakili radius rata-rata Bumi dalam satuan kilometer. Baris ke-11 menggunakan `map(np.radians, ...)` untuk mengubah semua nilai input latitude dan longitude dari derajat menjadi radian, karena fungsi trigonometri dalam komputasi menggunakan satuan radian. Baris ke-12 dan ke-13 menghitung selisih latitude dan longitude dalam radian. Baris ke-14 adalah penerapan inti dari formula Haversine untuk menghitung kuadrat dari setengah panjang garis antara dua titik. Baris ke-15 menghitung jarak

angular dalam radian dan mengalikannya dengan radius bumi (R) untuk mendapatkan jarak akhir dalam kilometer, yang kemudian dikembalikan sebagai hasil dari fungsi.

Bagian 3 Penerapan Algoritma A Search\*. Pada baris ke-20 hingga ke-108 didefinisikan fungsi utama `a_star_search(...)` dimana ini adalah penerapan algoritma A\*. Pada baris ke-21 hingga ke-33, terdapat blok kode pintar yang diset untuk mendeteksi nama kolom koordinat secara dinamis. Ini dilakukan dengan mendefinisikan daftar variasi nama kolom yang mungkin untuk latitude (`lat_variations`) dan longitude (`lng_variations`). Program kemudian melakukan iterasi melalui kolom-kolom yang ada di dataset dan mencocokkannya dengan daftar variasi ini. Jika kolom koordinat tidak ditemukan, maka program akan mencetak pesan error dan berhenti.

Selanjutnya, pada baris ke-34 hingga ke-39, program menginisialisasi dictionary cities untuk memetakan nama setiap kota ke koordinatnya (latitude, longitude) agar pencarian data koordinat menjadi lebih cepat. Pada baris ke-42 hingga ke-46 adalah tahap inisialisasi algoritma A\*. `open_set` dibuat sebagai arraylist kosong yang akan difungsikan sebagai priority queue oleh library `heapq`. `g_score` diinisialisasi sebagai dictionary yang menyimpan jarak aktual dari kota awal ke setiap kota lain dengan nilai awal tak terhingga (`inf`). `f_score` juga diinisialisasi serupa untuk menyimpan nilai estimasi total (`g_score + heuristic`). Nilai `g_score` untuk kota awal diatur menjadi 0 dan `f_score` untuk kota awal diatur menjadi nilai heuristik (jarak Haversine langsung ke tujuan).

Pada baris ke-56, sebuah set bernama `visited` dibuat untuk menyimpan kota-kota yang sudah selesai diproses untuk mencegah pemrosesan ganda dan potensi perulangan tak terbatas. Loop utama algoritma dimulai pada baris ke-58 (`while open_set`), yang akan terus berjalan selama masih ada kota yang perlu dipertimbangkan. Baris ke-59 (`heapq.heappop(open_set)`) mengambil kota dengan nilai `f_score` terendah dari antrian prioritas. Baris ke-67 adalah kondisi pemberhentian: jika kota saat ini adalah kota tujuan, maka rute optimal telah ditemukan. Blok kode di dalamnya (baris ke-68 hingga ke-78) akan merekonstruksi rute dengan menelusuri kembali `came_from` dari tujuan ke awal, lalu mengembalikan rute dan total jaraknya.



Pada baris ke-81 hingga ke-106 adalah inti dari eksplorasi. Program melakukan iterasi ke semua kota lain (`neighbor_city_name`). Di dalam loop ini, program menghitung `tentative_g_score`, yaitu total jarak dari kota awal ke tetangga ini jika melalui kota saat ini. Baris ke-95 adalah kondisi kunci A\* yaitu jika `tentative_g_score` ini lebih baik (lebih kecil) daripada `g_score` yang tercatat sebelumnya untuk tetangga tersebut, maka program telah menemukan rute yang lebih baik. Blok di dalamnya (baris ke-96 hingga ke-106) akan memperbarui `came_from` untuk mencatat rute baru ini, memperbarui `g_score` dan `f_score` untuk tetangga tersebut, dan memasukkan tetangga itu ke dalam `open_set` untuk dievaluasi pada iterasi selanjutnya. Jika loop selesai tanpa menemukan tujuan, fungsi akan mengembalikan `None, None` pada baris ke-108.

Bagian 4 Fungsi Utilitas. Pada baris ke-111 hingga ke-169 terdapat beberapa fungsi pembantu untuk meningkatkan interaktivitas dan pengalaman pengguna. Fungsi `load_and_display_data` (baris 111-127) bertanggung jawab untuk membaca file CSV menggunakan `try-except` untuk menangani error jika file tidak ditemukan, dan kemudian menampilkan informasi dasar tentang dataset. Fungsi `show_available_cities` (baris 129-141) menampilkan daftar kota yang tersedia dari data yang sudah dimuat, dengan batasan jumlah yang ditampilkan agar tidak membanjiri layar. Fungsi `find_similar_cities` (baris 143-147) menyediakan fitur "Mungkin maksud Anda..." dengan mencari nama kota yang mengandung kata kunci yang diketik pengguna jika terjadi salah ketik. Terakhir, fungsi `get_city_input` (baris 149-169) adalah fungsi canggih untuk menerima input dari pengguna, yang di dalamnya melakukan validasi, memberikan saran jika input salah, dan memberikan opsi untuk menampilkan semua kota yang tersedia.

Bagian 5 Blok Main. Pada baris ke-172 hingga akhir, terdapat blok `if __name__ == "__main__":`, yang merupakan titik awal eksekusi program. Baris ke-178 memanggil fungsi `load_and_display_data` untuk memuat data dari file CSV. Blok pada baris ke-193 hingga ke-211 menangani logika untuk memfilter data berdasarkan negara jika pengguna memilih opsi 'y'. Ini membuat program lebih fleksibel dan cepat dengan mengurangi jumlah kota yang perlu diproses. Setelah data siap (baik difilter maupun tidak), program memanggil fungsi `get_city_input` pada baris ke-220 dan ke-221 dua kali untuk mendapatkan kota awal dan tujuan yang valid dari pengguna. Terakhir, pada baris ke-231, fungsi utama `a_star_search` dipanggil. Blok kode setelahnya (baris

ke-233 hingga akhir) bertanggung jawab untuk menampilkan hasil pencarian kepada pengguna secara rapi dan informatif, termasuk menampilkan rute yang ditemukan, total jarak, serta rincian jarak per segmen perjalanan.

<https://github.com/arfanPramudya/AI-Final-Project-Search-and-Learning>

5. **Link video presentasi/penjelasan:** <https://youtu.be/IyPgTL-TUgc>

## Bagian 2

1. **Judul:** "KLASIFIKASI CUSTOMER CHURN PADA PELANGGAN TELEKOMUNIKASI MENGGUNAKAN ALGORITMA DECISION TREE"
2. **Teknik AI:** Learning.
3. **Algoritma** Supervised Learning; Decision Tree (berdasarkan prinsip C4.5/CART).

Algoritma Decision Tree adalah sebuah metode Supervised Learning yang bertujuan untuk membuat model prediksi dengan mempelajari serangkaian aturan keputusan sederhana yang diinferensikan dari fitur data. Model ini direpresentasikan dalam bentuk struktur pohon yang intuitif dan mudah dipahami.

Untuk kasus Telco Customer Churn, pendekatan yang digunakan mengacu pada prinsip algoritma CART (Classification and Regression Trees). Fungsi utama algoritma dalam penerapan ini adalah menganalisis data historis pelanggan untuk mengidentifikasi fitur-fitur yang paling berpengaruh terhadap keputusan pelanggan untuk berhenti berlangganan (churn). Hasil akhirnya adalah sebuah model klasifikasi yang tidak hanya dapat memprediksi kemungkinan churn untuk pelanggan baru, tetapi juga menyajikan aturan-aturan keputusan tersebut dalam format yang transparan dan dapat ditindaklanjuti oleh pemangku kepentingan bisnis.

## 4. Program.

```
1 # DECISION TREE - CHURN PREDICTION
2 import pandas as pd
3 import numpy as np
4 from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
5 from sklearn.preprocessing import LabelEncoder
6 from sklearn.tree import DecisionTreeClassifier
7 from sklearn.feature_selection import SelectKBest, mutual_info_classif
8 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11 import warnings
12 warnings.filterwarnings('ignore')
13
14 # DATA PIPELINE
15 class ChurnPredictor:
16     """Efficient Churn Prediction Pipeline"""
17
18     def __init__(self, file_path):
19         self.file_path = file_path
20         self.model = None
21         self.selected_features = None
22         self.le = LabelEncoder()
23
24     def load_and_clean_data(self):
25         """Combined loading and cleaning for efficiency"""
26         print(f"📂 Dataset loaded: {df.shape}")
27         print(f"🧹 LOADING & CLEANING DATA")
28         print(f"📊 Churn Distribution: Stay={churn_counts.get(0,0)}")
29         print(f"📊 Churn={churn_counts.get(1,0)}")
30
31         try:
32             df = pd.read_csv(self.file_path)
33             print(f"📂 Dataset loaded: {df.shape}")
34         except FileNotFoundError:
35             print(f"❌ File not found: {self.file_path}")
36             return None, None
37
38         if 'Churn' in df.columns:
39             churn_counts = df['Churn'].value_counts()
40             print(f"📊 Churn Distribution: Stay={churn_counts.get(0,0)}")
41             print(f"📊 Churn={churn_counts.get(1,0)}")
42
43         # Cleaning remove unnecessary columns and data leakage features
44         drop_columns = ['customerID', 'Unnamed: 0', 'PromptInput', 'CustomerFeedback', 'sentiment',
45                         'feedback_length']
46         df_clean = df.drop(columns=[col for col in drop_columns if col in df.columns])
47
48         removed = [col for col in drop_columns if col in df.columns]
49         print(f"🗑️ Removed columns: {removed}")
50
51         X = df_clean.drop('Churn', axis=1)
52         y = df_clean['Churn']
53
54         categorical_cols = X.select_dtypes(include=['object']).columns
55         for col in categorical_cols:
56             X[col] = self.le.fit_transform(X[col])
57
58         print(f"📊 Encoded {len(categorical_cols)} categorical columns")
59         print(f"📊 Final features: {list(X.columns)}")
60
61         return X, y
62
63     def select_features(self, X, y, k=8):
64         """Efficient feature selection"""
65         print(f"🔍 SELECTING TOP {k} FEATURES")
66         print(f"📊 Feature Scores: {feature_scores}")
67
68         selector = SelectKBest(score_func=mutual_info_classif, k=min(k, X.shape[1]))
69         X_selected = selector.fit_transform(X, y)
70
71         self.selected_features = X.columns[selector.get_support()].tolist()
72         feature_scores = selector.scores_[selector.get_support()]
73
74         # Display selected features
75         for i, (feature, score) in enumerate(zip(self.selected_features, feature_scores), 1):
76             print(f"{i:2d}. {feature: <20} ({score: .4f})")
77
78         return pd.DataFrame(X_selected, columns=self.selected_features)
79
80     def optimize_and_train(self, X_train, y_train):
81         """Efficient hyperparameter tuning and training"""
82         print(f"⚙️ OPTIMIZING HYPERPARAMETERS")
83         print(f"📊 Hyperparameter Search Results")
84
85         # Streamlined parameter grid (most effective combinations only)
86         param_grid = {
87             'max_depth': [2, 3],
88             'min_samples_split': [500, 1000],
89             'min_samples_leaf': [200, 300],
90             'criterion': ['gini'] # Use Gini typically faster than entropy
91         }
92
93         # Efficient grid search with reduced CV folds
94         grid_search = GridSearchCV(
95             DecisionTreeClassifier(random_state=42, max_features='sqrt'),
96             param_grid,
97             cv=3,
98             scoring='f1',
99             n_jobs=-1 # Use all CPU cores
100         )
101
102         grid_search.fit(X_train, y_train)
```

```

101
102     self.model = grid_search.best_estimator_
103     print(f"✅ Best params: {grid_search.best_params}")
104     print(f"✅ Best F1 score: {grid_search.best_score_:.4f}")
105
106     return self.model
107
108 def evaluate_model(self, X_train, X_test, y_train, y_test, show_plot=True):
109     """Comprehensive but efficient evaluation"""
110     print(f"\n📊 MODEL EVALUATION")
111     print("-" * 40)
112
113     # Predictions
114     y_train_pred = self.model.predict(X_train)
115     y_test_pred = self.model.predict(X_test)
116     y_test_proba = self.model.predict_proba(X_test)[:, 1]
117
118     # Core metrics
119     train_acc = accuracy_score(y_train, y_train_pred)
120     test_acc = accuracy_score(y_test, y_test_pred)
121     roc_auc = roc_auc_score(y_test, y_test_proba)
122     gap = train_acc - test_acc
123
124     # Display results
125     print(f"📊 Training Accuracy: {train_acc:.4f}")
126     print(f"📊 Testing Accuracy: {test_acc:.4f}")
127     print(f"📊 ROC-AUC Score: {roc_auc:.4f}")
128     print(f"📊 Train-Test Gap: {gap:.4f}")
129
130     # Validation check
131     issues = []
132     if test_acc > 0.90: issues.append("Test accuracy too high")
133     if roc_auc > 0.95: issues.append("ROC-AUC too high")
134     if gap > 0.10: issues.append("Large train-test gap")
135     if train_acc > 0.95: issues.append("Training accuracy too high")
136
137     if issues:
138         print(f"⚠️ Issues detected: {'', '.join(issues)}")
139         is_valid = False
140     else:
141         print(f"✅ Model validation passed!")
142         is_valid = True
143
144     # Classification report (compact format)
145     print(f"\n📊 Classification Report:")
146     print(classification_report(y_test, y_test_pred, target_names=['Stay', 'Churn']))
147
148     # Optional confusion matrix plot
149     if show_plot:
150         cm = confusion_matrix(y_test, y_test_pred)
151         plt.figure(figsize=(6, 4))
152         sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
153                    xticklabels=['Stay', 'Churn'], yticklabels=['Stay', 'Churn'])
154         plt.title('Confusion Matrix')
155         plt.tight_layout()
156         plt.show()
157
158     return train_acc, test_acc, roc_auc, is_valid
159
160 def cross_validate(self, X, y):
161     """Quick cross-validation"""
162     print(f"\n✅ CROSS-VALIDATION")
163     print("-" * 40)
164
165     cv_acc = cross_val_score(self.model, X, y, cv=3, scoring='accuracy')
166     cv_f1 = cross_val_score(self.model, X, y, cv=3, scoring='f1')
167
168     print(f"📊 CV Accuracy: {cv_acc.mean():.4f} ± {cv_acc.std():.4f}")
169     print(f"📊 CV F1 Score: {cv_f1.mean():.4f} ± {cv_f1.std():.4f}")
170
171     return cv_acc.mean() <= 0.90 # Return if acceptable
172
173 def run_full_pipeline(self, test_size=0.2, n_features=8, show_plots=True):
174     """Complete pipeline in one method"""
175     print(f"🚀 STARTING EFFICIENT CHURN PREDICTION PIPELINE")
176     print("-" * 60)
177
178     # Step 1 - Load and clean
179     X, y = self.load_and_clean_data()
180     if X is None:
181         return None
182
183     # Step 2 - Feature selection
184     X_selected = self.select_features(X, y, k=n_features)
185
186     # Step 3 - Train-test split
187     X_train, X_test, y_train, y_test = train_test_split(
188         X_selected, y, test_size=test_size, random_state=42, stratify=y
189     )
190     print(f"\n📊 Data split: Train={X_train.shape}, Test={X_test.shape}")
191
192     # Step 4 - Optimize and train
193     self.optimize_and_train(X_train, y_train)
194
195     # Step 5 - Evaluate
196     train_acc, test_acc, roc_auc, is_valid = self.evaluate_model(
197         X_train, X_test, y_train, y_test, show_plots
198     )
199
200     # Step 6 - Cross-validate

```

```

201     cv_valid = self.cross_validate(X_selected, y)
202
203     # Final summary
204     self.print_final_summary(train_acc, test_acc, roc_auc, is_valid and cv_valid)
205
206     return {
207         'model': self.model,
208         'features': self.selected_features,
209         'performance': {
210             'train_acc': train_acc,
211             'test_acc': test_acc,
212             'roc_auc': roc_auc,
213             'is_production_ready': is_valid and cv_valid
214         }
215     }
216
217 def print_final_summary(self, train_acc, test_acc, roc_auc, is_production_ready):
218     """Compact final summary"""
219     print(f"\n" + "=" * 60)
220     print("🔴 FINAL RESULTS")
221     print("=" * 60)
222
223     print(f"🔧 Model Config: Depth={self.model.max_depth}, "
224           f"MinSplit={self.model.min_samples_split}, "
225           f"MinLeaf={self.model.min_samples_leaf}")
226
227     print(f"📊 Performance: Accuracy={test_acc:.3f}, "
228           f"ROC-AUC={roc_auc:.3f}, "
229           f"Gap={train_acc-test_acc:.3f}")
230
231     status = "✅ PRODUCTION READY" if is_production_ready else "❌ NEEDS IMPROVEMENT"
232     print(f"🔴 Status: {status}")
233     print("=" * 60)
234
235 # EXECUTION
236 if __name__ == "__main__":
237     # File path
238     file_path = "C:/Users/lenovo/Documents/Semester 4/Kecerdasan Buatan/Project AI UAS/telco_prep.csv"
239
240     # Create and run pipeline
241     predictor = ChurnPredictor(file_path)
242
243     # Run complete pipeline with efficient settings
244     results = predictor.run_full_pipeline(
245         test_size=0.2,
246         n_features=8,
247         show_plots=True
248     )
249
250     """
251     if results and results['performance']['is_production_ready']:
252         print("\n🔴 Creating balanced version for better churn detection...")
253
254         balanced_model = DecisionTreeClassifier(
255             **predictor.model.get_params(),
256             class_weight='balanced'
257         )
258
259         # Quick refit and test
260         balanced_model.fit(X_train, y_train)
261         balanced_pred = balanced_model.predict(X_test)
262
263         print("📊 Balanced Model Results:")
264         print(classification_report(y_test, balanced_pred, target_names=['Stay', 'Churn']))
265     """
266
267     print("\n🔴 EFFICIENT PIPELINE COMPLETE!")

```

### Cara Penggunaan Program

1. Pastikan file dataset telco\_prep.csv sudah tersedia di folder project yang sama dengan file proyek\_churn.py. File ini harus berisi 26 kolom dengan kolom target bernama 'Churn'.
2. Install Library yang Diperlukan.
3. Eksekusi Program/
4. Program akan otomatis membaca file CSV dan menampilkan informasi dasar seperti ukuran dataset, jumlah missing values, dan distribusi target variable antara Stay dan Churn.

5. Sistem akan menghapus kolom yang tidak diperlukan dalam proses learning seperti customerID dan Unnamed: 0, serta menghapus fitur data leakage seperti PromptInput, CustomerFeedback, sentiment, dan feedback\_length untuk mencegah overfitting.
6. Program melakukan transformasi otomatis untuk mengubah semua kolom kategorikal menjadi format numerik menggunakan LabelEncoder agar dapat diproses oleh algoritma machine learning.
7. Sistem akan memilih 8 fitur terbaik dari dataset berdasarkan mutual information score dan menampilkan daftar fitur yang dipilih beserta nilai scorenya.
8. Testing Data akan dibagi secara otomatis menjadi 80% untuk training dan 20% untuk testing dengan stratified sampling untuk menjaga proporsi kelas churn dan stay.
9. Program menjalankan grid search untuk mencari kombinasi parameter terbaik seperti maxdepth, minsamplesplit, dan minsamples\_leaf menggunakan 3-fold cross validation.
10. Model Decision Tree akan dilatih menggunakan parameter optimal hasil grid search dan menampilkan konfigurasi final model yang akan digunakan.
11. Sistem menghitung dan menampilkan metrik evaluasi lengkap termasuk training accuracy, testing accuracy, ROC-AUC score, classification report, dan confusion matrix visualization.
12. Program melakukan validasi tambahan dengan 3-fold cross validation untuk memastikan konsistensi performa model dan stabilitas prediksi.
13. Sistem akan menganalisis perbedaan performa antara data training dan testing untuk memastikan model tidak mengalami overfitting dan dapat generalize dengan baik.
14. Final Assessment dan Status Program memberikan penilaian akhir apakah model sudah siap untuk production berdasarkan kriteria akurasi, stabilitas, dan kemampuan generalisasi.
15. Review hasil akhir dimana akurasi 72-78% menunjukkan performa yang realistis, ROC-AUC ~75% menunjukkan kemampuan diskriminasi yang baik, dan gap <5% menunjukkan model tidak overfitting.
16. Gunakan model hasil training untuk memprediksi pelanggan yang berisiko churn dan fokuskan strategi retention pada pelanggan dengan probability churn tinggi untuk mengoptimalkan ROI retention campaign.

Penjelasan baris, fungsi, class pada program

Program dimulai dengan mengimpor semua library yang diperlukan untuk machine learning dan visualisasi data. Pada baris 4-6, pandas digunakan untuk manipulasi data dalam format DataFrame, numpy untuk operasi numerik, sementara sklearn menyediakan algoritma machine learning dan tools evaluasi. Baris 9-10 mengimpor matplotlib dan seaborn untuk membuat visualisasi yang menarik, sedangkan baris 12 `warnings.filterwarnings('ignore')` ditambahkan untuk menyembunyikan peringatan teknis yang tidak perlu. Pemilihan library ini sudah menjadi standar industri dan terbukti stabil untuk project machine learning skala production.

Program menggunakan pendekatan Object-Oriented Programming (OOP) dengan membuat class `ChurnPredictor` pada baris ke-15 yang mengemas semua fungsi prediksi churn dalam satu tempat. Konstruktor `init` pada baris ke-18 menerima parameter `filepath` sebagai lokasi dataset, lalu menginisialisasi atribut `self.model=None` untuk menyimpan model yang akan ditraining, `self.selectedfeatures=None` untuk fitur yang dipilih, dan `self.le=LabelEncoder()` untuk encoding data kategorikal. Pendekatan class ini memudahkan maintenance kode dan memungkinkan reusability untuk dataset berbeda tanpa perlu menulis ulang seluruh pipeline yang ada.

Method `loadandcleandata()` pada baris ke-24 menggabungkan proses loading dan cleaning data dalam satu fungsi untuk efisiensi. Baris ke-30 hingga ke-35 mencoba membaca CSV file menggunakan `pd.readcsv()` dengan exception handling yang proper - jika file tidak ditemukan, program akan memberikan error message yang jelas tanpa crash. Setelah berhasil load, baris ke-37 hingga ke-40 menampilkan distribusi churn menggunakan `valuecounts()` untuk memberikan insight awal tentang class imbalance. Bagian cleaning pada baris ke-43 dan ke-44 melakukan drop columns untuk menghapus fitur yang tidak relevan seperti `customerID` dan fitur data leakage seperti `PromptInput`, `CustomerFeedback`, `sentiment`, dan `feedbacklength` yang bisa menyebabkan overfitting karena mengandung informasi yang seharusnya menjadi bahan uji bagi model.



Proses encoding pada baris ke-52 hingga ke-54 menggunakan `X.select_dtypes(include=['object'])` untuk otomatis mendeteksi kolom kategorikal, kemudian menerapkan `LabelEncoder` untuk mengubah text menjadi angka. Pendekatan ini lebih efisien dibanding manual encoding karena program secara otomatis mengidentifikasi kolom yang perlu di-encode. `LabelEncoder` dipilih karena sederhana dan cukup untuk `Decision Tree` yang bisa handle ordinal encoding dengan baik, meskipun untuk algoritma lain mungkin perlu `OneHotEncoder`. Baris 56-57 menampilkan konfirmasi proses encoding dan daftar fitur final yang tersedia.

Method `selectfeatures()` pada baris 61 menggunakan `SelectKBest` dengan `mutualinfoclassif` sebagai scoring function untuk memilih fitur terbaik. Parameter `k=8` pada baris 66 dipilih berdasarkan eksperimen untuk mendapatkan balance antara informasi yang cukup dan kompleksitas model yang terkontrol. Baris 66-67 menerapkan selector dengan `mutualinfo_classif` yang dipilih karena bisa menangkap hubungan non-linear antara fitur dan target, lebih menjanjikan dibanding correlation biasa. Baris 73-74 menampilkan score setiap fitur yang dipilih untuk transparency dan debugging purpose.

Grid search dalam `optimizeandtrain()` pada baris 78 menggunakan parameter yang sangat konservatif untuk mencegah overfitting. Baris 84-88 mendefinisikan paramgrid dengan `maxdepth=[2,3]` yang membatasi kedalaman pohon untuk menjaga interpretability dan mencegah overfitting - pohon yang terlalu dalam cenderung menghafal training data. `minsamplesplit=[500,1000]` memastikan setiap split harus memiliki minimal 500-1000 sampel, sementara `minsamplesleaf=[200,300]` memastikan setiap leaf node memiliki minimal 200-300 sampel. Angka-angka ini dipilih berdasarkan ukuran dataset 7000 rows, dimana rule of thumb adalah `minsamplesplit` sekitar 5-10% dari total data dan `minsamplesleaf` sekitar 2-5%.

Baris 92-97 mendefinisikan `GridSearchCV` dengan parameter `cv=3` untuk 3-fold cross validation yang dipilih sebagai manjanjukan antara validasi yang reliable dan efisiensi computational. Untuk dataset berukuran 7000+ rows, 3-fold sudah cukup memberikan estimasi performa yang valid. `scoring='f1'` dipilih karena dataset memiliki class imbalance (73% Stay vs 27% Churn), dimana F1-score lebih informatif dibanding

accuracy untuk imbalanced dataset. `n_jobs=-1` menggunakan semua CPU cores available untuk mempercepat grid search yang computationally intensive.

Method `evaluate_model()` pada baris 108 menghitung berbagai metrik evaluasi untuk assessment yang komprehensif. Baris 114-116 melakukan prediksi untuk training dan testing set serta menghitung probability untuk ROC-AUC. Baris 119-122 menghitung core metrics termasuk training accuracy, testing accuracy, ROC-AUC score, dan gap antara training-testing. Program menggunakan threshold validasi yang ketat pada baris 125-128: `testacc > 0.90` dianggap terlalu tinggi dan mencurigakan untuk churn prediction, `rocauc > 0.95` juga terlalu tinggi, `gap > 0.10` menandakan overfitting, dan `trainacc > 0.95` suspicious. Threshold ini berdasarkan industry benchmark untuk churn prediction dimana akurasi realistis biasanya 70-85%.

Visualization pada baris 149-156 menggunakan `plt.figure(figsize=(6,4))` dengan ukuran yang compact tapi readable. `sns.heatmap()` dengan `annot=True` menampilkan angka di setiap cell, `fmt='d'` untuk format integer, dan `cmap='Blues'` untuk color scheme yang professional. Parameter `show_plot=True` memungkinkan user disable visualisasi untuk eksekusi yang lebih cepat dalam production environment. `xticklabels` dan `yticklabels` pada baris 153 memberikan label yang jelas untuk matrix.

Method `crossvalidate()` pada baris 161 menggunakan `cv=3` lagi untuk konsistensi, dengan scoring 'accuracy' dan 'f1' untuk memberikan sudut pandang yang berbeda tentang performa model. Baris 165-166 menjalankan cross validation untuk kedua metrik, sedangkan baris 168-169 menampilkan hasil dengan mean dan standard deviation. Return value `cvacc.mean() <= 0.90` pada baris 156 adalah validation check - jika CV accuracy lebih dari 90%, dianggap overfitting dan function return False.

Method `run_full_pipeline()` pada baris 173 mengorkestrasi seluruh proses dengan parameter default yang sudah dioptimasi. `testsize=0.2` pada baris 173 memberikan 80-20 split yang standar industri production model, `nfeatures=8` hasil tuning untuk balance information vs complexity, dan baris 187-188 menggunakan `stratify=y` yang memastikan proporsi churn tetap sama di training dan testing set. `random_state=42` pada baris 188 digunakan untuk reproducibility - angka 42 adalah convention yang

sering dipakai dalam komunitas Mechine Learning untuk memastikan hasil yang konsisten.

Method `printfinalsummary()` pada baris 217 memberikan ringkasan dengan format yang clean dan professional. Baris 223-225 menampilkan konfigurasi model yang digunakan termasuk `depth`, `minsamplessplit`, dan `minsamplesleaf`. Baris 227-229 menampilkan performance summary dengan `accuracy`, `ROC-AUC`, dan `gap`. Status `PRODUCTION READY` pada baris 231 hanya diberikan jika model pass semua validation check - tidak `overfitting`, akurasi dalam rentang realistis, dan `CV` stable. Ini memastikan hanya model yang benar-benar reliable yang akan dideploy ke production environment.

Bagian eksekusi pada baris 236-264 menjalankan seluruh pipeline dengan konfigurasi yang sudah dioptimasi. Baris 238 mendefinisikan file path dataset, baris 241 membuat instance `ChurnPredictor`, dan baris 244-247 menjalankan full pipeline dengan parameter default. Ada juga optional code pada baris 250-265 yang di-comment untuk membuat balanced model jika diperlukan - ini bisa diaktifkan untuk meningkatkan detection rate churn dengan mengorbankan sedikit precision. Section ini memungkinkan program dijalankan langsung tanpa perlu input tambahan dari user.

<https://github.com/arfanPramudya/AI-Final-Project-Search-and-Learning>

5. Link video presentasi/penjelasan: <https://youtu.be/w7irUxNveXk>