📊 Panduan Benchmark
TF-IDF vs Jaccard Similarity

File: cbf_methods_benchmark_simplified.py

Apa itu Benchmark?
Membandingkan performa dua metode Content-Based Filtering untuk menentukan mana yang lebih baik dalam merekomendasikan destinasi wisata.

1. Dua Metode yang Dibandingkan

TF-IDF ⚔️ Jaccard

🔵 **TF-IDF + Cosine**

- Berbasis **frekuensi kata**
- Mempertimbangkan **keunikan kata**
- Menggunakan **vektor numerik**
- Lebih **sophisticated**

🔴 **Jaccard Similarity**

- Berbasis **overlap kata**
- Hanya melihat **ada/tidak ada**
- Menggunakan **set (himpunan)**
- Lebih **sederhana**

💡 **Analogi:** **TF-IDF** seperti menghitung nilai ujian dengan bobot — soal yang jarang dijawab benar nilainya lebih tinggi.

Jaccard seperti menghitung berapa banyak kata yang sama antara dua dokumen, tanpa peduli seberapa sering kata itu muncul.

2. Alur Benchmark

1️⃣ **LOAD DATA** → Baca file eco_place.csv

↓

2️⃣ **PREPROCESSING** → Lowercase, tokenize, remove stopwords

↓

3️⃣ **BUILD INDEX**

- TF-IDF: vectorizer.fit_transform()
- Jaccard: buat set token tiap item

↓

4️⃣ **CREATE QUERIES** → Query sintetis dari metadata

↓

5️⃣ **SPLIT DATA** → 70% train, 15% val, 15% test

↓

6️⃣ **EVALUATE** → Hitung Recall@10, F1@10, Latency

↓

7️⃣ **COMPARE** → Bandingkan hasil kedua metode

3. Cara Kerja TF-IDF

```
TF-IDF(word, doc) = TF(word, doc) × IDF(word)
```

Komponen	Rumus	Arti
TF (Term Frequency)	log(1 + frekuensi)	Seberapa sering kata muncul

IDF (Inverse Doc Freq)	log(N / df)	Seberapa unik kata tersebut

```
# TF-IDF Vectorizer
vectorizer = TfidfVectorizer(
max_features=5000, # Max 5000 kata
ngram_range=(1, 2), # Unigram + bigram
min_df=1,
max_df=0.95,
sublinear_tf=True, # Pakai log(1+tf)
norm="l2"
)

def score_tfidf(query):
query_vec = vectorizer.transform([query])
scores = cosine_similarity(query_vec, TFIDF_DENSE)[0]
return scores
```

4. Cara Kerja Jaccard Similarity

$$Jaccard(A, B) = |A \cap B| / |A \cup B|$$

Dimana:

- **|A ∩ B|** = Jumlah kata yang sama (intersection)
- **|A ∪ B|** = Jumlah total kata unik (union)

💡 **Analogi:** Bayangkan dua tas berisi buah-buahan:
• Tas A: {apel, jeruk, mangga, pisang}
• Tas B: {apel, jeruk, anggur, semangka}
• Sama (intersection): {apel, jeruk} = 2 buah
• Gabungan (union): {apel, jeruk, mangga, pisang, anggur, semangka} = 6 buah
• Jaccard = 2/6 = 0.33 (33% mirip)

```
def score_jaccard(query):
query_tokens = set(re.findall(r"\w+", query))
scores = np.zeros(len(item_tokens))

for i, item_set in enumerate(item_tokens):
intersection = len(query_tokens & item_set) # ∩
union = len(query_tokens | item_set) # ∪
scores[i] = intersection / union if union > 0 else 0

return scores
```

5. Metrik Evaluasi

🎯 Recall@10

Apakah item yang benar (ground truth) muncul di Top-10 hasil rekomendasi?

$$Recall@10 = 1 \text{ jika GT di Top-10, } 0 \text{ jika tidak}$$

Interpretasi: Recall 0.85 = 85% query berhasil menemukan item yang dicari

📊 F1@10

Harmonic mean dari Precision dan Recall — keseimbangan antara keduanya.

```
F1 = 2 × (Precision × Recall) / (Precision + Recall)
```

⏱️ Latency (ms/query)

Waktu rata-rata yang dibutuhkan untuk memproses satu query.

Interpretasi: Semakin kecil semakin baik (lebih cepat)

6. Contoh Hasil Benchmark

📋 Hasil pada Validation Set

Method	Recall@10	F1@10	Latency (ms)
TF-IDF Cosine WINNER	0.8667 (86.7%)	0.157	~5 ms
Jaccard	0.7200 (72.0%)	0.131	~15 ms

📋 Hasil pada Test Set

Method	Recall@10	F1@10	Latency (ms)
TF-IDF Cosine WINNER	0.8533 (85.3%)	0.155	~5 ms
Jaccard	0.6933 (69.3%)	0.126	~14 ms

Kesimpulan:

• **TF-IDF** menang di semua metrik (Recall lebih tinggi, Latency lebih rendah)

- **Jaccard** lebih sederhana tapi performanya kurang optimal
- Untuk sistem rekomendasi wisata, **TF-IDF adalah pilihan terbaik**

7. Perbandingan Lengkap

Aspek	TF-IDF	Jaccard
Kompleksitas	Medium	Rendah
Akurasi (Recall)	Tinggi (~85%)	Sedang (~70%)
Kecepatan	Cepat (~5ms)	Lambat (~15ms)
Memori	Lebih besar	Lebih kecil
Pertimbangan frekuensi	Ya (dengan bobot)	Tidak (hanya ada/tidak)
Cocok untuk	Teks panjang, beragam	Teks pendek, sederhana

8. Demo Retrieval

Berikut contoh cara menggunakan sistem untuk mendapatkan rekomendasi:

```
def demo_retrieve(query, k=5, method="TF-IDF Cosine"):
score_fn = METHODS[method]
scores = score_fn(preprocess_text(query))
top_indices = np.argsort(-scores)[:k]

print(f"Query: '{query}'")
for rank, idx in enumerate(top_indices, 1):
print(f"{rank}. {df.iloc[idx]['place_name']}")

return top_indices.tolist()

# Contoh penggunaan
demo_retrieve("pantai snorkeling aceh", k=5)
demo_retrieve("wisata alam di bandung", k=5)
```

Contoh Output:

Query: "pantai snorkeling aceh"

1. Pantai Iboih (score: 0.85)
2. Pantai Ujong Blang (score: 0.72)
3. Pulau Weh (score: 0.68)
...

9. Ringkasan

✅ **TF-IDF** lebih unggul untuk rekomendasi destinasi wisata

✅ Recall ~85% artinya 85 dari 100 query berhasil menemukan item yang dicari

✅ Latency ~5ms artinya respons sangat cepat (hampir instan)

✅ Jaccard bisa jadi alternatif jika butuh implementasi yang lebih sederhana

---

📚 Dokumentasi ini dibuat sebagai panduan pembelajaran untuk memahami perbandingan metode Content-Based Filtering

File: cbf_methods_benchmark_simplified.py