Nomor 1

Penjelasan Pada Fungsi hitung_akurasi

Algoritma ini bertujuan untuk menghitung akurasi tokenisasi dengan membandingkan token hasil tokenisasi dari suatu tokenizer dengan gold standard, yang merupakan tokenisasi referensi yang benar. Berikut adalah penjelasan cara kerja fungsi:

1. **Variabel Awal**:

```
# Total token yang benar
total_true_token = 0

# Total token di gold standard
total_gold_token = 0
```

total_true_token bertujuan untuk menghitung jumlah token yang benar (token yang sesuai di tokenizer dan gold standard).

total_gold_token bertujuan untuk menghitung jumlah token total dari gold standard, yang akan digunakan sebagai basis perhitungan akurasi.

2. **Iterasi Per Kalimat**

```
# Iterasi tiap kalimat
for tokenizer_tokens, gold_tokens in zip(tokenizer_tokens_list, gold_std_tokens_list):
```

Loop ini bekerja pada daftar kalimat token dari tokenizer (tokenizer_tokens_list) dan gold standard (gold_std_tokens_list).

```
    # String untuk konkatinasi tiap token di tokenizer
    string_tokenizer = ""

    # String untuk konkatinasi tiap token di gold token
    string_gold = ""
```

string_tokenizer dan string_gold adalah variabel yang digunakan untuk menggabungkan token-token dari tokenizer dan gold standard agar dapat dibandingkan secara bertahap.

```
# index token pada suatu kalimat di tokenizer
tokenizer_index = 0

# index token pada suatu kalimat di gold token
gold_index = 0
```

tokenizer_index dan gold_index adalah Indeks untuk melacak posisi token saat ini di tokenizer dan gold standard.

3. **Iterasi tiap Kata / Token**

```
# Ketika semua token belum di dibandingkan
while tokenizer_index < len(tokenizer_tokens) and gold_index < len(gold_tokens):
```

Tiap kalimat dari gold_tokens dan tokenizer_tokens kita ambil tiap elemennya (token-tokennya) untuk dibandingkan.

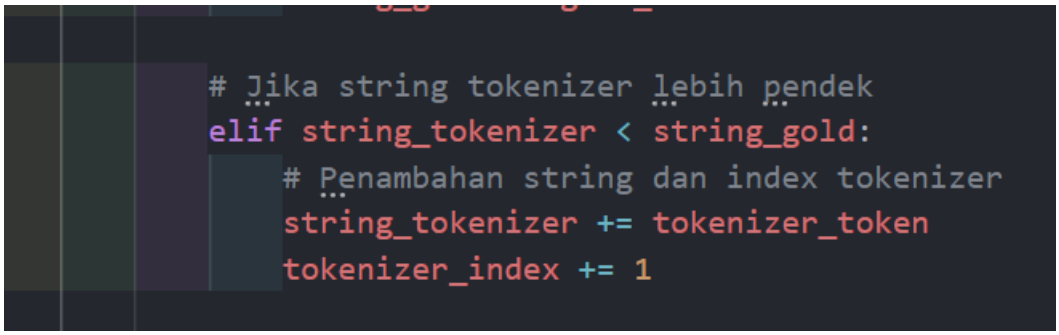
```
    if tokenizer_token == '[UNK]':
        tokenizer_token = '_'
```

Jika Kata unknown diubah menjadi “_” agar terhitung tetap 1 karakter

4. **Logika Pembandingan Token**

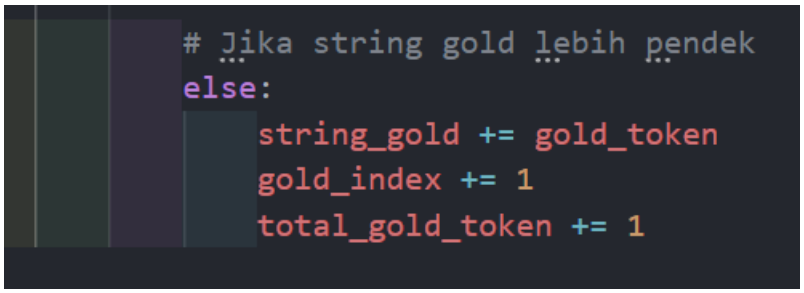
```
    if string_tokenizer == string_gold:
        if tokenizer_token == gold_token:
            total_true_token += 1 # Token benar, tambahkan

        total_gold_token += 1 # Total token di gold bertambah
        tokenizer_index += 1  # Maju ke token berikutnya
        gold_index += 1       # Maju ke token berikutnya

        # Penambahan token diikuti penambahan string
        string_tokenizer += tokenizer_token
        string_gold += gold_token
```

Jika string yang telah digabungkan dari tokenizer (string_tokenizer) dan gold standard (string_gold) memiliki panjang yang sama. Kemudian, jika token di tokenizer sama dengan token di gold, maka token tersebut dianggap benar (sesuai dengan ekspektasi). total_true_token dan total_gold_token akan bertambah.



```
# Jika string tokenizer lebih pendek
elif string_tokenizer < string_gold:
    # Penambahan string dan index tokenizer
    string_tokenizer += tokenizer_token
    tokenizer_index += 1
```

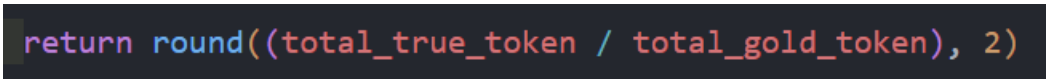
Jika string_tokenizer lebih pendek dari string_gold, maka token selanjutnya dari tokenizer ditambahkan ke string_tokenizer, sehingga fungsi terus membandingkan token sampai string menjadi seimbang.



```
# Jika string gold lebih pendek
else:
    string_gold += gold_token
    gold_index += 1
    total_gold_token += 1
```

Jika string_gold lebih pendek, maka token dari gold standard ditambahkan ke string_gold.

5. **Akurasi**



```
return round((total_true_token / total_gold_token), 2)
```

Setelah semua token diperiksa, akurasi dihitung sebagai rasio antara total token yang benar (total_true_token) dengan total token di gold standard (total_gold_token).

Hasil akhirnya adalah nilai akurasi yang dibulatkan hingga dua desimal.

Fungsi ini menekankan kecocokan urutan token dan jumlah token antara hasil tokenisasi BPE dan gold standard untuk di evaluasi akurasinya.

Penjelasan Kode Pengujian Tokenizer pada Dataset Testing

Kode ini melakukan pengujian tokenizer pada dataset testing (testing_dataset) dengan menggunakan fungsi hitung_akurasi.

```
tokenizer_tokens_list = []
gold_std_tokens_list = []
```

tokenizer_tokens_list akan menyimpan list kalimat yang tiap kalimat mengandung hasil tokenisasi dari tokenizer untuk setiap teks di dataset testing.

gold_std_tokens_list akan menyimpan list kalimat yang tiap kalimat mengandung standard gold token untuk setiap teks di dataset testing.

```
for data in testing_dataset:
    tokenizer_tokens_list.append(tokenizer.encode(data['text']).tokens)
    gold_std_tokens_list.append(data['tokens'])
```

Kode ini melakukan iterasi pada setiap item (data) dalam testing_dataset.

Untuk setiap teks di dataset (data['text']), teks tersebut di-tokenisasi menggunakan tokenizer.encode(data['text']) dan hasil tokenisasi (dalam bentuk list kalimat yang tiap kalimat mengandung token) ditambahkan ke tokenizer_tokens_list.

Token gold standard yang sudah ada dalam data['tokens'] (kalimat yang tiap kalimat mengandung standard gold token) ditambahkan ke gold_std_tokens_list.

```
hitung_akurasi(tokenizer_tokens_list,gold_std_tokens_list)
```

Setelah semua data diproses, kedua list (tokenizer_tokens_list dan gold_std_tokens_list) digunakan sebagai input ke fungsi hitung_akurasi, yang akan menghitung akurasi tokenisasi berdasarkan kecocokan dengan gold standard.

Nomor 2

Vocab_size: 500

Akurasi: 0.35

```python
# Membuat tokenizer BPE dan trainer-nya
tokenizer = Tokenizer(BPE(unk_token="[UNK]"))
trainer = BpeTrainer(
    vocab_size=500,
    special_tokens=["[UNK]", "[CLS]", "[SEP]", "[PAD]", "[MASK]"],
)
```

✓ 0.0s

```python
# TODO: Uji tokenizer pada `testing_dataset` dengan fungsi `hitung_akurasi` sesuai ketentuan soal
tokenizer_tokens_list = []
gold_std_tokens_list = []
for data in testing_dataset:
    tokenizer_tokens_list.append(tokenizer.encode(data['text']).tokens)
    gold_std_tokens_list.append(data['tokens'])
hitung_akurasi(tokenizer_tokens_list,gold_std_tokens_list)
```

✓ 0.0s

```
0.35
```

Vocab_size: 1000

Akurasi: 0.46

```python
# Membuat tokenizer BPE dan trainer-nya
tokenizer = Tokenizer(BPE(unk_token="[UNK]"))
trainer = BpeTrainer(
    vocab_size=1000,
    special_tokens=["[UNK]", "[CLS]", "[SEP]", "[PAD]", "[MASK]"],
)
```

✓ 0.0s

```python
# TODO: Uji tokenizer pada `testing_dataset` dengan fungsi `hitung_akurasi` sesuai ketentuan soal
tokenizer_tokens_list = []
gold_std_tokens_list = []
for data in testing_dataset:
    tokenizer_tokens_list.append(tokenizer.encode(data['text']).tokens)
    gold_std_tokens_list.append(data['tokens'])
hitung_akurasi(tokenizer_tokens_list,gold_std_tokens_list)
```

✓ 0.0s

```
0.46
```

Vocab_size: 2000

Akurasi: 0.57

```python
# Membuat tokenizer BPE dan trainer-nya
tokenizer = Tokenizer(BPE(unk_token="[UNK]"))
trainer = BpeTrainer(
    vocab_size=2000,
    special_tokens=["[UNK]", "[CLS]", "[SEP]", "[PAD]", "[MASK]"],
)
```

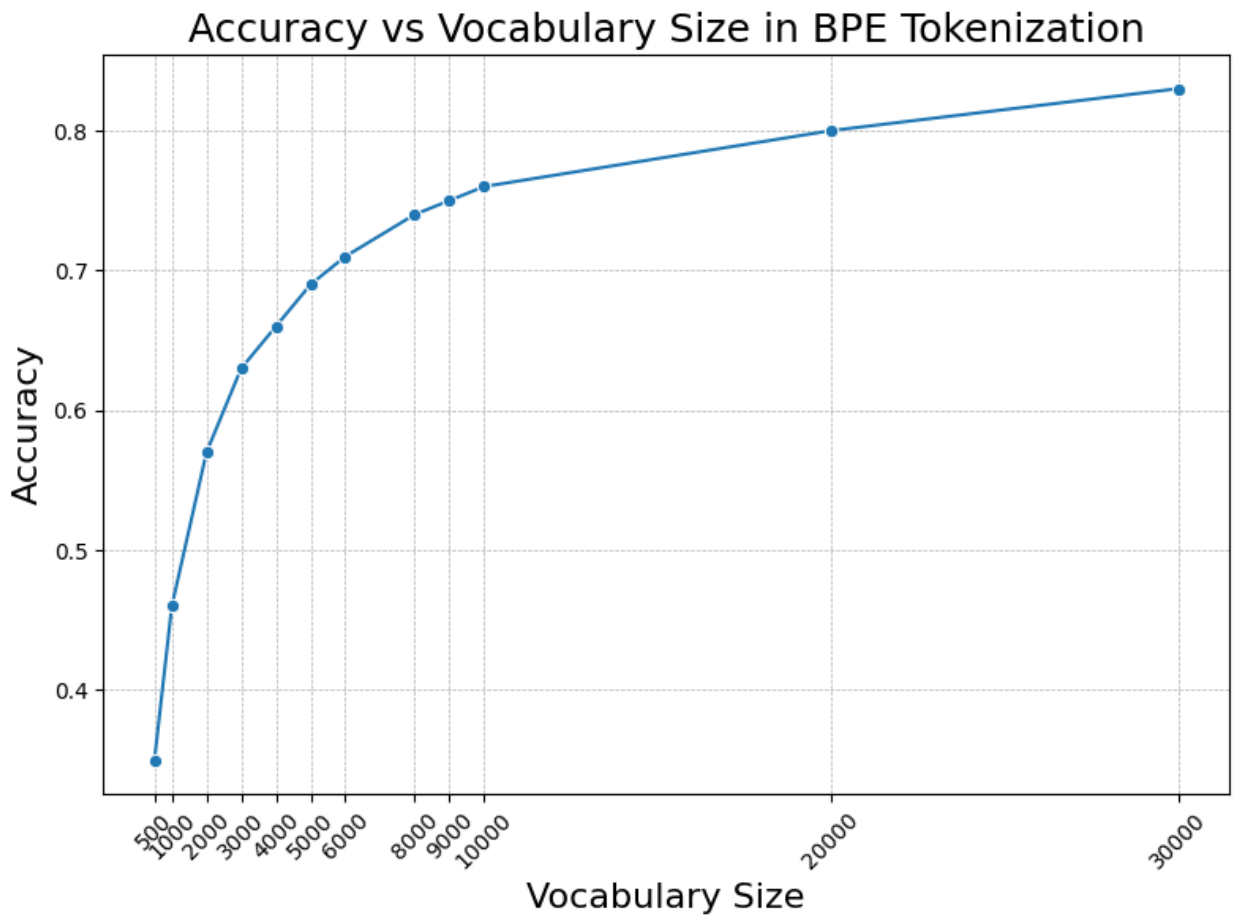
✓ 0.0s

```python
tokenizer_tokens_list = []
gold_std_tokens_list = []
for data in testing_dataset:
    tokenizer_tokens_list.append(tokenizer.encode(data['text']).tokens)
    gold_std_tokens_list.append(data['tokens'])
hitung_akurasi(tokenizer_tokens_list,gold_std_tokens_list)
```

✓ 0.0s

```
0.57
```

Nomor 3



Grafik ini menunjukkan hubungan antara ukuran kosa kata (vocabulary size) dan akurasi dalam tokenisasi BPE (Byte Pair Encoding).

1. **Peningkatan Akurasi Seiring Dengan Bertambahnya Vocabulary Size**

   Pada awal grafik, terlihat peningkatan akurasi yang cukup signifikan ketika ukuran kosa kata meningkat dari 500 hingga 10.000. Ini menunjukkan bahwa vocab size yang lebih besar memungkinkan tokenisasi yang lebih presisi, karena token-token lebih terperinci dan sesuai dengan gold standard.

2. **Diminishing Returns**

   Setelah mencapai kosa kata sekitar 5.000, peningkatan akurasi mulai melambat, meskipun vocabulary size terus bertambah hingga 30.000. Ini menandakan bahwa menambah ukuran kosa kata lebih dari titik tertentu tidak memberikan peningkatan akurasi yang signifikan. Pada titik ini, sebagian besar token yang penting mungkin sudah tercover dengan baik.

3. **Trade-off Antara Vocabulary Size dan Performance**

Peningkatan vocabulary size akan membutuhkan lebih banyak memori dan daya komputasi. Oleh karena itu, meskipun akurasi terus naik, trade-off antara resource yang diperlukan dan peningkatan akurasi menjadi pertimbangan penting, terutama setelah ukuran kosa kata melewati angka tertentu, seperti 10.000 dalam kasus ini.

Secara keseluruhan, ukuran kosa kata yang terlalu kecil menyebabkan akurasi rendah, tetapi setelah mencapai ukuran 5.000, peningkatan kosa kata tidak lagi memberikan lonjakan akurasi yang besar.