LAPORAN UJIAN AKHIR SEMESTER

MATA KULIAH MACHINE LEARNING

TK-45-GAB-G04

Disusun untuk memenuhi nilai UAS mata kuliah

Machine Learning di Program Studi S1 Teknik Komputer

Disusun oleh:

Raihana Fawaz

1103210102

FAKULTAS TEKNIK ELEKTRO

UNIVERSITAS TELKOM BANDUNG

2024

Chapter 5 The Datasets Library

1. What's my dataset isn't on the hub?
    - Notebook memulai dengan menginstal pustaka penting seperti datasets, evaluate, dan transformers menggunakan perintah pip. Ketiga pustaka ini membantu dalam mengelola dataset, evaluasi model, dan menggunakan model NLP dari Transformers Library.
    - Dataset yang digunakan adalah "SQuAD_it" (SQuAD versi Italia). File ini diunduh langsung dari repositori GitHub menggunakan wget. Format file adalah .json.gz, sehingga perlu diekstraksi menggunakan gzip.
    - Dataset dimuat menggunakan pustaka datasets dari Hugging Face. File SQuAD_it-train.json dimuat sebagai contoh pertama, kemudian contoh-contoh lain memuat dataset lengkap dengan struktur pelatihan dan pengujian (train dan test) dari berbagai lokasi (file lokal atau URL langsung).
    - Setelah dimuat, beberapa kode digunakan untuk memeriksa isi dataset, termasuk menampilkan entri pertama dalam dataset dan memverifikasi struktur datanya.
2. Time to slice and dice

    - Notebook dimulai dengan mengunduh dan mengekstrak dataset ulasan obat, yang kemudian dimuat ke dalam pustaka Hugging Face datasets menggunakan load_dataset. Data ini terdiri dari informasi pasien, ulasan, kondisi medis, dan peringkat. Dataset dibersihkan dengan menghapus entri yang tidak memiliki kondisi medis dan mengubah semua nama kondisi menjadi huruf kecil menggunakan fungsi map. Panjang ulasan dihitung dan ditambahkan sebagai kolom baru, dan ulasan pendek (dengan kurang dari 30 kata) difilter keluar untuk mempertahankan data yang lebih informatif.
    - Tokenisasi dilakukan dengan menggunakan tokenizer BERT dari pustaka transformers. Dataset di-tokenisasi secara batched dengan opsi untuk memotong ulasan yang panjang menjadi segmen pendek menggunakan max_length dan overflow_to_sample_mapping. Hasil tokenisasi melibatkan pembuatan dataset baru dengan segmen token yang cocok untuk pelatihan model NLP.
    - Dataset diproses lebih lanjut untuk eksplorasi data, seperti menghitung frekuensi kondisi medis dalam ulasan, mengonversi dataset ke format Pandas DataFrame, dan membuat dataset baru dari frekuensi kondisi. Dataset akhir dibagi menjadi subset pelatihan, validasi, dan pengujian menggunakan train_test_split dengan persentase tertentu. Data kemudian disimpan ke disk dalam berbagai format, termasuk JSONL dan format Hugging Face, untuk digunakan kembali.

3. Big data?

    - Notebook dimulai dengan memuat dataset besar dari PubMed dalam format JSONL terkompresi (zst) menggunakan pustaka datasets. Dataset ini dianalisis untuk memeriksa ukuran cache dan penggunaan memori, menunjukkan efisiensi pustaka dalam menangani data besar. Untuk iterasi batch pada dataset, kode menggunakan ukuran batch tertentu

untuk membaca dan memproses data secara efisien, mengukur kecepatan pemrosesan dalam gigabyte per detik.

- Pemrosesan data dilakukan dengan teknik streaming, di mana data dimuat secara bertahap daripada semuanya sekaligus, mengurangi beban memori. Dataset di-tokenisasi menggunakan tokenizer DistilBERT, dan operasi seperti pengacakan data (shuffle), pengambilan subset (take), dan pembagian dataset ke dalam set pelatihan dan validasi dilakukan secara langsung pada data yang di-streaming.
- Notebook ini juga mendemonstrasikan kombinasi beberapa dataset streaming menggunakan fungsi interleave_datasets, seperti menggabungkan dataset PubMed dan FreeLaw. Dataset tambahan diambil dari proyek AI Pile, yang memuat data dalam beberapa partisi untuk pelatihan, validasi, dan pengujian. Teknik ini menunjukkan skalabilitas pustaka Hugging Face untuk menangani berbagai jenis dataset besar secara efisien dengan memori terbatas.

4. Creating your own dataset

    - Notebook dimulai dengan mempersiapkan lingkungan, termasuk konfigurasi Git dan login ke Hugging Face Hub untuk memungkinkan penyimpanan dataset yang dihasilkan. Pengambilan data dilakukan dari repositori GitHub dengan menggunakan API GitHub, yang memerlukan token otentikasi untuk akses. Dataset terdiri dari informasi tentang isu yang diambil dari repositori Hugging Face datasets, termasuk metadata seperti nomor isu, URL, status (dibuka atau ditutup), dan pull request terkait.
    - Fungsi fetch_issues digunakan untuk mengunduh sejumlah isu tertentu dari GitHub dalam batch, menyimpannya ke file JSONL. Dataset dimuat ke pustaka Hugging Face untuk eksplorasi dan pemrosesan lebih lanjut. Data diperkaya dengan atribut tambahan seperti apakah suatu isu merupakan pull request atau tidak, menggunakan pemetaan fungsi pada dataset. Selain itu, komentar pada setiap isu diambil dari API GitHub dan ditambahkan ke dataset. Setelah dataset selesai diproses, dataset tersebut didorong (*push*) ke Hugging Face Hub menggunakan fungsi push_to_hub, sehingga dapat diakses publik. Dataset yang disimpan ini dapat dimuat ulang dari Hub untuk penggunaan lebih lanjut.

5. Semantic search with FAISS

    Kode dalam file ini tampaknya adalah notebook Jupyter untuk implementasi pencarian semantik menggunakan FAISS (Facebook AI Similarity Search) dan PyTorch. Berikut adalah ringkasan isi dari sebagian kode yang dapat diidentifikasi:

    1. Instalasi Library: Kode dimulai dengan instalasi library penting, seperti datasets, evaluate, transformers[sentencepiece], dan faiss-gpu. Ini menunjukkan bahwa notebook menggunakan pustaka tersebut untuk memproses data, evaluasi, transformasi model, dan pencarian cepat.
    2. Pemuatan Dataset: Menggunakan pustaka datasets, dataset dengan nama lewtun/github-issues diimpor. Dataset ini tampaknya memuat informasi seperti judul, isi, URL, dan komentar terkait isu GitHub.

3. Penyaringan Data: Dataset difilter untuk hanya menyertakan entri yang bukan pull request dan memiliki komentar.
4. Pemilihan Kolom: Kolom dataset yang relevan dipilih (misalnya, title, body, html_url, comments), sedangkan yang tidak relevan dihapus.
5. Konversi Format: Dataset diubah ke format Pandas DataFrame untuk manipulasi lebih lanjut, seperti eksplorasi kolom komentar.
6. Pemrosesan Dataset Komentar: Dataset diperluas untuk memisahkan komentar individu dan panjang setiap komentar dihitung.

Chapter 6 The Tokenizers Library

1. Training a new tokenizer from an old one
   Notebook ini digunakan untuk melatih tokenizer baru berdasarkan tokenizer lama.
   Prosesnya meliputi:
   - Loading Base Tokenizer: Mengimpor tokenizer yang sudah ada, seperti dari transformers.
   - Custom Vocabulary Training: Melatih tokenizer menggunakan dataset baru untuk menyesuaikan kosa kata.
   - Saving & Testing: Menyimpan tokenizer baru dan mengujinya pada teks baru

2. Fast tokenizer's special powers
   Notebook ini menjelaskan penggunaan fast tokenizer dari pustaka transformers untuk tugas *token classification* dalam bahasa Python menggunakan PyTorch Notebook ini dimulai dengan penginstalan pustaka datasets, evaluate, dan transformers. Model tokenizer BERT (bert-base-cased) diinisialisasi untuk memproses teks dan menghasilkan encoding yang mencakup token, ID kata, dan rentang karakter. Fitur fast tokenizer dimanfaatkan untuk secara efisien memetakan token ke kata asli dan menyediakan metode seperti tokens () dan word_ids() untuk eksplorasi data. Selanjutnya, notebook menggunakan pipeline token-classification untuk mengidentifikasi entitas bernama (*Named Entity Recognition*) pada teks, baik secara langsung maupun dengan strategi agregasi.
   - Model khusus, seperti dbmdz/bert-large-cased-finetuned-conll03-english, dimuat untuk demonstrasi lebih lanjut. Model ini digunakan untuk membuat prediksi token-level, di mana hasil logit model diolah dengan fungsi softmax untuk mendapatkan probabilitas. Prediksi dikonversi menjadi label entitas menggunakan konfigurasi model (id2label). Fitur fast tokenizer digunakan untuk menentukan rentang teks yang sesuai dengan prediksi model, sehingga entitas dapat dianalisis dengan informasi skor, kata, dan rentang karakter. Notebook ini juga menjelaskan bagaimana mengelompokkan token dengan label yang sama untuk menghasilkan grup entitas berdasarkan skor rata-rata, rentang, dan teks yang sesuai.

3. Fast tokenizer in the QA pipeline

```
[ ] for candidate, offset in zip(candidates, offsets):
        start_token, end_token, score = candidate
        start_char, _ = offset[start_token]
        _, end_char = offset[end_token]
        answer = long_context[start_char:end_char]
        result = {"answer": answer, "start": start_char, "end": end_char, "score": score}
        print(result)

{'answer': '\n🤗 Transformers: State of the Art NLP', 'start': 0, 'end': 37, 'score': 0.33867}
{'answer': 'Jax, PyTorch and TensorFlow', 'start': 1892, 'end': 1919, 'score': 0.97149}
```

Notebook ini dimulai dengan inisialisasi pipeline QA dari transformers, yang memungkinkan ekstraksi jawaban dari konteks berdasarkan pertanyaan yang diberikan. Menggunakan model bawaan seperti distilbert-base-cased-distilled-squad, notebook menunjukkan cara memproses teks konteks dan pertanyaan menjadi representasi tensor untuk model. Model ini kemudian menghasilkan logit (nilai probabilitas) untuk menentukan posisi awal dan akhir jawaban dalam teks konteks. Proses ini memanfaatkan kemampuan tokenizer untuk menangani teks panjang dengan strategi pemangkasan (truncation) dan jendela bergeser (*sliding window*).

- Kemudian, notebook memanfaatkan offset mapping, fitur dari fast tokenizer, untuk memetakan indeks token ke rentang karakter asli dalam teks. Ini digunakan untuk mengidentifikasi jawaban secara presisi dalam konteks. Selain itu, notebook menunjukkan bagaimana menangani konteks yang sangat panjang dengan membagi input menjadi beberapa bagian menggunakan strategi pemangkasan spesifik. Skor probabilitas jawaban dihitung berdasarkan kombinasi probabilitas token awal dan akhir, dan kandidat jawaban terbaik dievaluasi untuk ditampilkan.
- Fitur lain yang didemonstrasikan adalah kemampuan tokenizer untuk menangani teks berlebih (*overflowing tokens*), pemetaan ke sampel asli, dan pengelolaan padding untuk menghasilkan input tensor dengan panjang seragam. Ini memungkinkan penanganan konteks panjang atau batch teks dengan efisien.

4. Normalization and pre tokenization

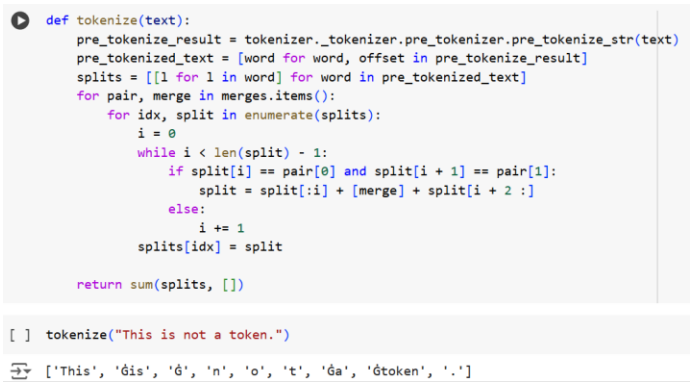
```
[ ] tokenizer = AutoTokenizer.from_pretrained("t5-small")
    tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("Hello, how are  you?")

tokenizer_config.json: 100% 2.32k/2.32k [00:00<00:00, 84.5kB/s]
spiece.model: 100% 792k/792k [00:00<00:00, 11.1MB/s]
tokenizer.json: 100% 1.39M/1.39M [00:00<00:00, 21.4MB/s]
[('_Hello,', (0, 6)),
 ('_how', (7, 10)),
 ('_are', (11, 14)),
 ('_you?', (16, 20))]
```

Notebook ini dimulai dengan penginstalan pustaka datasets, evaluate, dan transformers, yang dibutuhkan untuk manipulasi data teks dan pengolahan model NLP. Model tokenizer seperti bert-base-uncased, gpt2, dan t5-small digunakan untuk menunjukkan bagaimana teks diubah melalui proses normalisasi dan pra-tokenisasi. Pada bagian awal, normalisasi

dilakukan menggunakan normalizer bawaan dari tokenizer, yang mengonversi teks dengan karakter khusus (misalnya, "Héllò hôw are ü?") menjadi versi yang lebih standar. Proses pra-tokenisasi, yang bertujuan untuk memisahkan teks menjadi unit lebih kecil sebelum tokenisasi lengkap, dilakukan oleh pre_tokenizer. Proses ini menyoroti perbedaan antara tokenizer dari model seperti BERT, GPT-2, dan T5, termasuk bagaimana teks seperti "Hello, how are you?" diproses menjadi token dengan offset (posisi).

5. Byte Pair_Encoding tokenization



```
def tokenize(text):
    pre_tokenize_result = tokenizer._tokenizer.pre_tokenizer.pre_tokenize_str(text)
    pre_tokenized_text = [word for word, offset in pre_tokenize_result]
    splits = [[l for l in word] for word in pre_tokenized_text]
    for pair, merge in merges.items():
        for idx, split in enumerate(splits):
            i = 0
            while i < len(split) - 1:
                if split[i] == pair[0] and split[i + 1] == pair[1]:
                    split = split[:i] + [merge] + split[i + 2 :]
                else:
                    i += 1
            splits[idx] = split

    return sum(splits, [])
```

```
[ ] tokenize("This is not a token.")

['This', 'Ġis', 'Ġ', 'n', 'o', 't', 'Ġa', 'Ġtoken', '.']
```

Notebook dimulai dengan mendefinisikan korpus teks yang akan digunakan sebagai data pelatihan. Algoritma BPE bekerja dengan memecah kata menjadi unit terkecil (karakter), menghitung frekuensi pasangan karakter yang paling umum, dan menggabungkannya menjadi token baru. Proses dimulai dengan mengekstraksi semua karakter unik dari korpus untuk membentuk alfabet awal. Setiap kata dalam korpus dipecah menjadi urutan karakter, dan pasangan karakter yang paling sering muncul dihitung menggunakan fungsi compute_pair_freqs. Pasangan yang paling umum kemudian digabungkan menjadi token baru, dan proses ini diulang hingga ukuran vokabulari mencapai jumlah yang diinginkan (misalnya, 50 token).

Notebook juga menunjukkan bagaimana algoritma BPE diterapkan pada teks baru. Fungsi tokenize memanfaatkan hasil penggabungan token dari langkah sebelumnya untuk membagi teks menjadi token sesuai dengan vokabulari yang telah dilatih. Proses ini menunjukkan efisiensi BPE dalam menangani kata-kata baru yang tidak ada dalam korpus pelatihan dengan memecahnya menjadi token-token yang sudah dikenal. Notebook ini mengilustrasikan prinsip dasar di balik algoritma BPE yang digunakan dalam model seperti GPT dan T5 untuk tokenisasi teks.

6. Wordpiece tokenization

```
tokenize("This is the Hugging Face course!")
```

```
['Th',
 '##i',
 '##s',
 'is',
 'th',
 '##e',
 'Hugg',
 '##i',
 '##n',
 '##g',
 'Fac',
 '##e',
 'c',
 '##o',
 '##u',
 '##r',
 '##s',
 '##e',
 '[UNK]']
```

Notebook ini menjelaskan proses tokenisasi menggunakan algoritma **WordPiece**, yang banyak digunakan dalam model seperti BERT. Algoritma ini berfokus pada membagi teks menjadi subword yang paling sering muncul dalam korpus pelatihan untuk membentuk vokabulari yang optimal.

Notebook dimulai dengan mendefinisikan korpus teks sederhana untuk pelatihan. Setiap kata di korpus dipecah menjadi unit terkecil (karakter) dan ditambahkan ke alfabet awal, dengan awalan khusus ## untuk karakter selain karakter pertama dalam kata. Proses ini membentuk daftar vokabulari awal yang mencakup token dasar seperti [PAD], [UNK], [CLS], [SEP], dan [MASK]. Selanjutnya, algoritma menghitung skor untuk setiap pasangan token berdasarkan frekuensi pasangan tersebut dan distribusi token individual. Pasangan dengan skor tertinggi digabung menjadi token baru, dan proses ini diulang hingga ukuran vokabulari mencapai jumlah yang diinginkan.

Untuk teks baru, tokenisasi dilakukan dengan memecah setiap kata menjadi token yang cocok dengan vokabulari yang telah dilatih. Jika kata tidak dapat ditemukan dalam vokabulari, token [UNK] digunakan sebagai fallback. Algoritma ini memungkinkan model menangani kata-kata baru dengan memecahnya menjadi subword yang sudah dikenal, meningkatkan fleksibilitas dan efisiensi representasi teks. Proses ini diakhiri dengan implementasi fungsi tokenize untuk mengaplikasikan WordPiece tokenization pada teks baru.

7. Unigram tokenization

```
percent_to_remove = 0.1
while len(model) > 100:
    scores = compute_scores(model)
    sorted_scores = sorted(scores.items(), key=lambda x: x[1])
    # Remove percent_to_remove tokens with the lowest scores.
    for i in range(int(len(model) * percent_to_remove)):
        _ = token_freqs.pop(sorted_scores[i][0])

    total_sum = sum([freq for token, freq in token_freqs.items()])
    model = {token: -log(freq / total_sum) for token, freq in token_freqs.items()}
```

+ Code + Text

Add code cell Ctrl+M B

```
[ ] def tokenize(text, model):
        words_with_offsets = tokenizer.backend_tokenizer.pre_tokenizer.e_str(text)
        pre_tokenized_text = [word for word, offset in words_with_offsets]
        encoded_words = [encode_word(word, model)[0] for word in pre_tokenized_text]
        return sum(encoded_words, [])


    tokenize("This is the Hugging Face course.", model)
```

```
['_This', '_is', '_the', '_Hugging', '_Face', '_', 'c', 'ou', 'r', 's', 'e', '.']
```

Notebook ini menjelaskan Unigram Tokenization, yaitu metode tokenisasi yang mendasarkan pembagian teks menjadi unit kecil (subword) dengan probabilitas tertentu. Pendekatan ini banyak digunakan dalam model seperti XLNet.

Notebook dimulai dengan mendefinisikan korpus teks sederhana untuk melatih tokenisasi. Setiap kata dipecah menjadi karakter individu dan subword dengan panjang minimum dua karakter. Frekuensi karakter dan subword dihitung, kemudian subword yang paling sering digunakan diprioritaskan untuk membangun model probabilistik. Model ini menghitung skor negatif log-likelihood untuk setiap token berdasarkan distribusi frekuensinya. Proses tokenisasi dilakukan dengan memilih kombinasi subword yang meminimalkan skor total (loss) dari kata yang diberikan. Algoritma ini mencoba mencari segmen terbaik untuk setiap kata menggunakan pendekatan probabilistik dan memanfaatkan model yang telah dilatih.

Notebook ini juga menunjukkan cara menghitung *loss* model, yang digunakan untuk mengevaluasi kualitas tokenisasi. Subword dengan skor kontribusi terendah dapat dihapus untuk mengurangi ukuran vokabulari. Proses ini diulangi hingga jumlah token mencapai target tertentu. Akhirnya, teks baru dapat di-tokenisasi dengan model yang telah dilatih, memungkinkan pembagian kata menjadi subword berdasarkan kombinasi skor terbaik. Pendekatan ini sangat efisien untuk menangani kata-kata baru yang tidak ada dalam korpus pelatihan dengan memecahnya menjadi unit yang sudah dikenal.

8. Building a tokenizer, block by block

```
from transformers import PreTrainedTokenizerFast

wrapped_tokenizer = PreTrainedTokenizerFast(
    tokenizer_object=tokenizer,
    bos_token="<s>",
    eos_token="</s>",
    unk_token="<unk>",
    pad_token="<pad>",
    cls_token="<cls>",
    sep_token="<sep>",
    mask_token="<mask>",
    padding_side="left",
)
```

```
[51] from transformers import XLNetTokenizerFast

     wrapped_tokenizer = XLNetTokenizerFast(tokenizer_object=tokenizer)
```

Notebook ini menunjukkan langkah-langkah membangun tokenizer dari awal menggunakan pustaka Tokenizers. Proses ini mencakup normalisasi, pra-tokenisasi, pelatihan model, hingga wrapping ke dalam format tokenizer yang kompatibel dengan pustaka Transformers.

Notebook dimulai dengan memuat dataset "wikitext-2" sebagai korpus pelatihan. Proses dimulai dengan menentukan normalisasi teks, seperti mengonversi karakter menjadi huruf kecil, menghapus aksen, atau mengganti simbol tertentu. Selanjutnya, dilakukan pra-tokenisasi untuk membagi teks menjadi unit kecil berdasarkan spasi atau tanda baca. Model tokenizer, seperti WordPiece, BPE (Byte Pair Encoding), dan Unigram, digunakan untuk melatih tokenisasi berdasarkan korpus. Setiap model menggunakan algoritma spesifik untuk membangun vokabulari yang efisien, dengan mempertimbangkan token spesial seperti [UNK], [PAD], dan lainnya. Setelah pelatihan, tokenizer dapat memproses teks baru dengan membaginya menjadi token sesuai vokabulari yang dilatih.

Notebook ini juga mencakup post-processing, seperti menambahkan token [CLS] dan [SEP] untuk format input tertentu, serta konfigurasi decoder untuk merekonstruksi teks asli dari token. Akhirnya, tokenizer disimpan ke dalam file JSON untuk digunakan kembali, atau dibungkus ke dalam objek PreTrainedTokenizerFast untuk kompatibilitas dengan pustaka Transformers. Langkah-langkah ini memungkinkan pembentukan tokenizer yang fleksibel, dapat disesuaikan, dan kompatibel dengan model NLP modern.

Chapter 7 Main NLP Task

1. Token classification (PyTorch)

```
[ ] from transformers import pipeline

    # Replace this with your own checkpoint
    model_checkpoint = "huggingface-course/bert-finetuned-ner"
    token_classifier = pipeline(
        "token-classification", model=model_checkpoint, aggregation_strategy="simple"
    )
    token_classifier("My name is Sylvain and I work at Hugging Face in Brooklyn.")
```

```
[{'entity_group': 'PER', 'score': 0.9988506, 'word': 'Sylvain', 'start': 11, 'end': 18},
 {'entity_group': 'ORG', 'score': 0.9647625, 'word': 'Hugging Face', 'start': 33, 'end': 45},
 {'entity_group': 'LOC', 'score': 0.9986118, 'word': 'Brooklyn', 'start': 49, 'end': 57}]
```

- Notebook dimulai dengan memuat dataset CoNLL-2003, yang digunakan untuk tugas pengenalan entitas bernama (Named Entity Recognition, NER). Dataset ini mencakup

token dan label NER, seperti B-PER untuk nama orang dan B-ORG untuk organisasi. Tokenisasi dilakukan menggunakan tokenizer dari model BERT dengan mengatur opsi is_split_into_words=True untuk menyesuaikan dengan token asli. Notebook juga menjelaskan bagaimana menyelaraskan label dengan token hasil tokenisasi, termasuk penyesuaian label untuk token yang dipecah menjadi subword.

- Dataset ditokenisasi dan labelnya disesuaikan melalui fungsi map, yang menghasilkan dataset siap latih. Notebook ini menggunakan DataCollatorForTokenClassification untuk memproses batch selama pelatihan. Model token classification berbasis BERT diinisialisasi dengan peta id-label (id2label dan label2id). Selanjutnya, argumen pelatihan ditentukan menggunakan TrainingArguments, dan model dilatih menggunakan Trainer. Notebook ini juga mendemonstrasikan penggunaan pustaka Accelerate untuk mempercepat pelatihan. Setelah pelatihan, model diuji dengan pipeline Hugging Face untuk klasifikasi token. Model dapat memprediksi label NER untuk input teks baru, seperti "My name is Sylvain and I work at Hugging Face in Brooklyn."

2. Fine-tuning a masked language model (PyTorch)
   Kode yang ada di dalam notebook ini bertujuan untuk melakukan fine-tuning model DistilBERT dari pustaka Hugging Face pada tugas *masked language modeling* menggunakan dataset IMDB. Berikut adalah rangkuman isi dan maksud dari kodingan dalam beberapa bagian utama:
   1. Persiapan Lingkungan dan Instalasi: Notebook ini memulai dengan menginstal pustaka-pustaka seperti datasets, transformers, accelerate, dan dependensi lainnya. Instalasi ini memungkinkan penggunaan model berbasis Transformer serta pengelolaan dataset dengan efisien. Selanjutnya, notebook mengkonfigurasi git untuk penyimpanan hasil pelatihan di Hugging Face Hub.
   2. Pemanggilan Model dan Tokenizer: Model yang digunakan adalah distilbert-base-uncased, yaitu versi ringan dari BERT yang dirancang untuk efisiensi. Tokenizer digunakan untuk memproses teks input menjadi representasi numerik yang dapat diterima model. Kode juga menghitung jumlah parameter model untuk membandingkan dengan BERT.
   3. Preprocessing dan Tokenisasi Dataset: Dataset IMDB yang berisi ulasan film dimuat dan diacak. Fitur teks tokenized untuk mengubah ulasan menjadi urutan token, dengan pemrosesan lebih lanjut untuk memecahnya menjadi potongan yang sesuai dengan panjang maksimal model. Proses ini juga menciptakan label baru untuk *language modeling*.
   4. Fine-Tuning Model: Kode melakukan pelatihan dengan mendefinisikan *training arguments*, seperti *learning rate*, ukuran *batch*, dan strategi evaluasi. Model kemudian dilatih menggunakan API Trainer, dan hasil evaluasi seperti perplexity dihitung untuk mengukur performa.
   5. Evaluasi dan Penyimpanan Model: Setelah pelatihan selesai, model diunggah ke Hugging Face Hub untuk penggunaan lebih lanjut. Evaluasi menggunakan metrik perplexity

memberikan wawasan tentang seberapa baik model memprediksi kata-kata dalam konteks yang tidak terlihat selama pelatihan.

3. Translation

   Kode dalam notebook ini bertujuan untuk melakukan pelatihan dan evaluasi model translasi berbasis **Seq2Seq** dengan menggunakan **Helsinki-NLP/opus-mt-en-fr** dari pustaka Hugging Face.

   - Notebook ini dimulai dengan persiapan lingkungan, termasuk instalasi pustaka seperti datasets, transformers, dan accelerate. Dataset yang digunakan adalah dataset KDE4**,** yang berisi pasangan terjemahan Inggris-Prancis. Dataset ini diunduh dan dipecah menjadi set pelatihan dan validasi. Untuk preprocessing, teks dalam dataset ditokenisasi menggunakan tokenizer dari model Helsinki-NLP/opus-mt-en-fr**,** dengan mempertimbangkan panjang maksimum dan pemangkasan.
   - Selanjutnya, notebook melibatkan pembuatan data kolator untuk tugas translasi, pelatihan model, dan evaluasi performa dengan metrik BLEU menggunakan pustaka evaluate. Model dilatih dengan argumen seperti *learning rate*, ukuran *batch*, dan strategi penyimpanan. Hasil evaluasi mencakup skor BLEU yang memberikan wawasan tentang akurasi prediksi model terhadap referensi terjemahan.
   - Setelah pelatihan selesai, model diunggah ke Hugging Face Hub untuk penggunaan lebih lanjut. Notebook ini juga mencakup implementasi fungsi *pipeline* untuk pengujian terjemahan secara langsung. Jika diperlukan, notebook menyediakan fitur untuk melatih ulang model dengan skala lebih besar atau menggunakan akselerasi perangkat keras seperti TPU atau GPU.

4. Summarization
   - Notebook ini dimulai dengan menyiapkan lingkungan, termasuk instalasi pustaka-pustaka yang dibutuhkan seperti datasets, transformers, dan evaluate. Dataset yang digunakan adalah Amazon Reviews Multi-lingual, yang berisi ulasan produk dalam bahasa Inggris dan Spanyol. Ulasan dengan kategori *book* dan *digital_ebook_purchase* disaring, kemudian dataset digabungkan untuk mencakup kedua bahasa. Untuk preprocessing, teks ulasan diproses menggunakan tokenizer dari model mT5-small, dengan panjang maksimum untuk teks input dan label disesuaikan sesuai kebutuhan tugas peringkasan.
   - Selanjutnya, notebook melakukan evaluasi awal dengan menggunakan baseline sederhana, seperti mengambil tiga kalimat pertama dari ulasan, dan menghitung skor ROUGE sebagai metrik evaluasi. Model dilatih menggunakan *Seq2SeqTrainer* dengan parameter pelatihan seperti *batch size*, *learning rate*, dan jumlah epoch. Notebook juga mencakup proses evaluasi hasil pelatihan model dengan menghitung metrik ROUGE pada set validasi untuk membandingkan prediksi peringkasan model terhadap referensi.
   - Setelah pelatihan selesai, model yang telah di-finetune diunggah ke Hugging Face Hub untuk penggunaan lebih lanjut. Notebook ini juga menyediakan pipeline *summarization*

untuk menghasilkan ringkasan dari teks ulasan dan membandingkannya dengan judul ulasan. Dengan demikian, notebook ini mencakup seluruh alur pelatihan, evaluasi, dan penggunaan model summarization secara menyeluruh.

5. Training a casual language model from scratch

    - Notebook dimulai dengan persiapan lingkungan, termasuk instalasi pustaka seperti datasets, transformers, dan accelerate. Dataset yang digunakan adalah dataset CodeParrot, yang dirancang untuk pelatihan model bahasa berbasis kode. Dataset difilter untuk hanya menyertakan kode dengan kata kunci tertentu seperti pandas, sklearn, dan matplotlib. Data ini kemudian ditokenisasi menggunakan tokenizer dari model GPT-2, dengan panjang konteks maksimum yang ditentukan.
    - Model GPT-2 dikonfigurasi ulang menggunakan AutoConfig untuk mendukung panjang konteks yang lebih besar dan token BOS/EOS khusus. Pelatihan dilakukan menggunakan Trainer dari Hugging Face dengan parameter seperti ukuran batch, *learning rate*, dan jumlah langkah akumulasi gradien. Model dilatih menggunakan teknik percepatan seperti mixed precision (fp16) dan *gradient accumulation* untuk efisiensi. Evaluasi dilakukan pada set validasi menggunakan metrik perplexity, yang memberikan wawasan tentang kemampuan model untuk memprediksi token berikutnya dalam urutan.

6. Question answering

    - Notebook dimulai dengan memuat dataset SQuAD dan mengidentifikasi elemen penting: konteks, pertanyaan, dan jawaban. Dataset ini kemudian diproses menggunakan tokenizer dari model BERT-base-cased, dengan mengatur panjang maksimum dan mengelola tumpang tindih token untuk menangani konteks panjang. Proses ini menghasilkan peta token ke teks asli, sehingga jawaban dalam bentuk teks dapat diubah menjadi posisi token yang sesuai.
    - Model pelatihan dilakukan menggunakan API Trainer dari Hugging Face, dengan argumen seperti *learning rate*, jumlah epoch, dan strategi penyimpanan. Dataset pelatihan dan validasi diproses lebih lanjut untuk menyertakan label posisi awal dan akhir jawaban. Evaluasi dilakukan dengan menghitung logits awal dan akhir dari model, yang kemudian diproses untuk mengekstrak jawaban terbaik berdasarkan skor logit. Skor F1 dan akurasi dihitung menggunakan metrik bawaan SQuAD.
    - Notebook ini juga menyediakan pipeline question-answering untuk menguji model yang sudah dilatih. Dengan pipeline ini, pengguna dapat memberikan konteks dan pertanyaan untuk mendapatkan jawaban langsung dari model. Pada akhirnya, model yang telah di-finetune diunggah ke Hugging Face Hub untuk digunakan di masa depan.

Chapter 8 How To Ask For Help

1. What to do when you get an error

- Notebook ini dimulai dengan instalasi pustaka-pustaka yang diperlukan seperti datasets, transformers, dan huggingface_hub. Selanjutnya, login ke Hugging Face dilakukan untuk memungkinkan interaksi dengan repositori model. Bagian pertama mencakup proses mengunduh model dari repositori Hugging Face, membuat salinan repositori lokal, dan mengunggah kembali model setelah modifikasi. Proses ini penting untuk memperbaiki atau memodifikasi model yang sudah ada.
- Bagian selanjutnya berfokus pada implementasi pipeline *question-answering* menggunakan model **DistilBERT** yang telah di-finetune pada dataset SQuAD. Kode ini memuat model dan tokenizer, lalu menggunakan pipeline untuk menjawab pertanyaan berbasis konteks. Jika terjadi error dalam pipeline, kode menyediakan cara untuk memeriksa komponen seperti config, tokenizer, atau model untuk memastikan semuanya terhubung dengan benar.

2. Asking for help on the forums

- Notebook ini dimulai dengan instalasi pustaka-pustaka penting, seperti datasets dan transformers, yang sering digunakan untuk memproses dan melatih model NLP. Selanjutnya, model **DistilBERT** yang berukuran ringan dimuat bersama dengan tokenizer-nya menggunakan metode AutoTokenizer dan AutoModel. Model ini dirancang untuk menyederhanakan beban kerja pengguna yang ingin menjalankan tugas berbasis teks seperti klasifikasi atau pemrosesan teks.
- Sebuah teks yang panjang tentang sejarah Transformers digunakan sebagai contoh input untuk menunjukkan bagaimana data teks dapat diproses. Teks tersebut di-tokenisasi menggunakan tokenizer DistilBERT, menghasilkan tensor yang merepresentasikan urutan token. Tensor ini kemudian dimasukkan ke dalam model untuk mendapatkan output logits, yang merupakan skor mentah dari model untuk setiap token input.
- Notebook ini berguna untuk memberikan gambaran kepada pengguna tentang cara mempersiapkan dataset, memproses teks, dan menjalankan model NLP. Dengan contoh yang jelas, pengguna dapat memahami struktur data yang diperlukan sebelum mengajukan pertanyaan di forum, sehingga masalah yang dihadapi dapat lebih mudah dipahami oleh komunitas.

3. Debugging the training pipeline
    - Notebook ini dimulai dengan memuat dataset **GLUE (General Language Understanding Evaluation)**, khususnya subset MNLI (Multi-Genre Natural Language Inference), yang bertujuan untuk mengklasifikasikan hubungan antara pasangan premis dan hipotesis (entailment, neutral, atau contradiction). Dataset diproses menggunakan tokenizer **DistilBERT** dengan fungsi preprocessing untuk mempersiapkan input model, termasuk pemangkasan teks panjang.
    - Model **DistilBERT** untuk klasifikasi sekuens dimuat, dan konfigurasi pelatihan diatur menggunakan TrainingArguments, seperti *learning rate*, jumlah epoch, dan strategi evaluasi. Notebook ini mencakup berbagai langkah debugging, termasuk memeriksa data tokenized, panjang tensor, nilai label, dan bagaimana data dikompilasi oleh *data collator*.

Evaluasi model dilakukan dengan menghitung metrik seperti akurasi menggunakan pustaka evaluate.

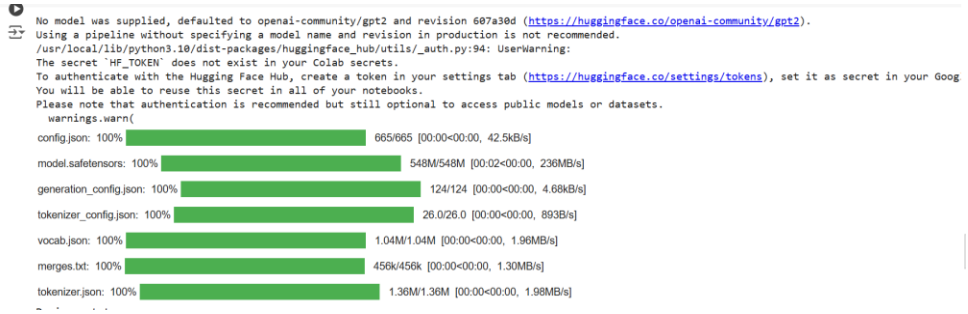
- Pada bagian debugging pelatihan, notebook memeriksa output logit dari model, memverifikasi bahwa panjang tensor *attention mask* sesuai dengan input ID, dan memeriksa apakah model menghasilkan output dengan dimensi yang benar untuk prediksi. Selain itu, notebook memuat fungsi loss backward untuk memantau optimisasi dan memastikan gradien dihitung dengan benar.

4. How to write a good issue

   Notebook dimulai dengan menginstal pustaka-pustaka yang relevan seperti datasets, evaluate, dan transformers[sentencepiece]. Instalasi ini bertujuan untuk memastikan bahwa lingkungan memiliki dependensi yang diperlukan untuk menjalankan eksperimen atau menganalisis permasalahan yang mungkin dihadapi pengguna.

Chapter 9 Building and Sharing Demos

1. Building your first demo



   Notebook ini menjelaskan cara membuat aplikasi demo interaktif menggunakan **Gradio**, sebuah pustaka Python untuk membangun antarmuka pengguna (UI) yang sederhana, terutama untuk model pembelajaran mesin.

   Notebook ini dimulai dengan menginstal pustaka yang dibutuhkan, termasuk datasets, transformers, dan gradio. Demo pertama yang dibuat adalah fungsi sederhana bernama greet yang menerima nama sebagai input teks dan mengembalikan pesan sapaan. Dengan menggunakan gr.Interface, demo ini menghasilkan antarmuka pengguna berupa kotak teks untuk input dan output, yang diluncurkan dengan fungsi launch(). Selanjutnya, antarmuka yang lebih kustom diperkenalkan dengan penggunaan widget seperti Textbox, memungkinkan label khusus dan teks placeholder.

   Selain fungsi sederhana, notebook juga menunjukkan cara mengintegrasikan model pembelajaran mesin dari pustaka transformers. Contohnya, pipeline untuk *text generation* digunakan untuk membuat fungsi prediksi yang menerima prompt teks sebagai input dan menghasilkan teks yang dihasilkan oleh model. Fungsi ini juga diintegrasikan ke dalam antarmuka Gradio untuk membuat demo interaktif di mana pengguna dapat memasukkan prompt dan melihat hasil keluaran model secara real-time.

Notebook ini memberikan dasar untuk membangun aplikasi berbasis AI dengan antarmuka pengguna yang mudah digunakan.

2. Understanding interface class

   Notebook ini mendemonstrasikan penggunaan kelas Interface dari pustaka Gradio untuk membangun antarmuka pengguna interaktif yang memproses data audio. Notebook ini dimulai dengan instalasi pustaka datasets, transformers, dan gradio untuk mempersiapkan lingkungan pengembangan. Contoh pertama menunjukkan cara membuat antarmuka untuk memproses audio yang direkam melalui mikrofon. Fungsi reverse_audio menerima data audio sebagai input, membalik urutan data audio menggunakan numpy.flipud, dan mengembalikan hasilnya. Antarmuka Gradio dibuat menggunakan gr.Audio untuk menangkap input audio dari mikrofon, dengan hasil berupa file audio terbalik yang dapat didengarkan kembali.

- Contoh kedua membangun antarmuka untuk menghasilkan nada (*tone generation*). Pengguna dapat memilih nada dari dropdown, menentukan oktaf melalui slider, dan durasi melalui kotak teks. Fungsi generate_tone menghitung frekuensi berdasarkan nada dan oktaf yang dipilih, lalu menghasilkan gelombang sinusoidal menggunakan numpy untuk membuat nada sesuai dengan parameter pengguna. Hasilnya berupa file audio yang dapat didengarkan.
- Contoh ketiga mengintegrasikan model *automatic speech recognition* (ASR) dari pustaka transformers. Fungsi transcribe_audio menerima input berupa audio dari mikrofon atau file yang diunggah pengguna, lalu memprosesnya dengan pipeline ASR untuk menghasilkan transkripsi teks. Antarmuka Gradio memungkinkan pengguna untuk memilih input audio dari mikrofon atau file, dan hasilnya ditampilkan sebagai teks transkripsi.

  Notebook ini menunjukkan fleksibilitas Gradio dalam membuat antarmuka untuk berbagai aplikasi berbasis audio, mulai dari manipulasi data, generasi audio, hingga pengenalan suara otomatis.

3. Sharing demos with others
   - Notebook dimulai dengan menginstal pustaka yang diperlukan seperti gradio, transformers, dan torch. Contoh pertama menunjukkan bagaimana membuat demo dengan menambahkan **judul**, **deskripsi**, **artikel tambahan**, serta **contoh input**. Demo ini adalah chatbot bertema *Rick and Morty* yang menjawab pertanyaan berdasarkan dialog dari serial tersebut. Elemen-elemen seperti gambar dan tautan eksternal dapat ditambahkan untuk memperkaya antarmuka.
   - Selanjutnya, antarmuka untuk klasifikasi gambar dibuat dengan model PyTorch. Model ini menerima gambar input, memprosesnya, dan menghasilkan label dengan probabilitas tertinggi. Notebook juga menampilkan cara membuat demo berbasis sketchpad, di mana pengguna dapat menggambar objek dan algoritma akan mencoba

mengenali gambar tersebut secara real-time. Elemen seperti tema, deskripsi interaktif, dan opsi berbagi ditambahkan untuk meningkatkan pengalaman pengguna. Fitur **share=True** digunakan untuk membuat URL yang dapat diakses publik, memungkinkan pengguna lain mencoba demo yang telah dibuat.

4. Integrations with hugging face hub

   Notebook dimulai dengan instalasi pustaka yang relevan seperti gradio dan transformers. Contoh pertama membuat antarmuka Gradio untuk model GPT-J-6B**,** sebuah model bahasa besar dengan 6 miliar parameter. Antarmuka ini memuat deskripsi model, tautan ke dokumentasi, dan contoh input untuk memudahkan pengguna memulai. Fungsi gr.Interface.load digunakan untuk memuat model langsung dari Hugging Face Hub, dengan input berupa teks dan output berupa teks yang dihasilkan oleh model. Opsi seperti title**,** description**,** dan article memberikan konteks tambahan tentang model, sementara examples menyediakan input contoh untuk eksplorasi cepat. Selanjutnya, notebook menunjukkan bagaimana memuat demo dari ruang Gradio di Hugging Face Spaces, seperti "remove-bg," yang digunakan untuk menghapus latar belakang gambar. Fungsi ini juga dapat disesuaikan untuk menerima input dari webcam, memberikan fleksibilitas kepada pengguna dalam memilih sumber gambar. Proses ini memperlihatkan bagaimana Gradio memanfaatkan model yang dihosting di Hugging Face Hub untuk membangun aplikasi interaktif tanpa memerlukan pengaturan model tambahan. Dengan integrasi ini, pengguna dapat dengan mudah menggunakan dan membagikan aplikasi AI berbasis model yang dihosting di Hugging Face Hub, mempercepat proses pengembangan dan penyebaran aplikasi pembelajaran mesin.

5. Advanced interface features

   - Contoh pertama memperlihatkan pembuatan *chatbot* interaktif yang dapat menanggapi pesan berdasarkan pola tertentu. Fungsi chat menerima pesan pengguna dan riwayat percakapan sebelumnya. Respons dihasilkan secara acak berdasarkan kata awal pesan (misalnya, "How many" menghasilkan angka acak). Riwayat percakapan diperbarui dan dikembalikan untuk ditampilkan di antarmuka chatbot. Gradio memanfaatkan komponen state untuk menyimpan dan mengelola riwayat percakapan. Fitur seperti *disabling screenshots* dan *flagging* digunakan untuk membatasi opsi pengguna.
   - Contoh kedua adalah demo klasifikasi gambar menggunakan model MobileNetV2 dari TensorFlow. Notebook ini memuat model dan label kelas dari dataset ImageNet. Fungsi classify_image memproses gambar masukan dengan pra-pemrosesan sesuai kebutuhan model, menghasilkan prediksi dalam bentuk probabilitas untuk setiap kelas. Komponen Gradio seperti Image digunakan untuk menerima input gambar dengan ukuran tertentu, dan Label menampilkan probabilitas untuk tiga kelas teratas. Fitur interpretasi bawaan Gradio diaktifkan untuk membantu memahami bagaimana model menghasilkan prediksi.

6. Introduction to Gradio Blocks

   Notebook dimulai dengan contoh sederhana untuk membalik teks yang dimasukkan pengguna menggunakan fungsi flip_text. Dengan menggunakan gr.Blocks(), antarmuka dibuat dengan elemen seperti Markdown untuk memberikan instruksi, serta Textbox untuk input dan output. Ketika pengguna mengetik, fungsi ini dipanggil secara otomatis melalui metode .change() untuk memperbarui hasil.

   Contoh kedua memperluas konsep ini dengan menambahkan tab untuk berbagai fungsi, seperti membalik teks atau gambar. Tab "Flip Text" menyediakan antarmuka berbasis teks, sementara "Flip Image" memungkinkan pengguna memuat gambar untuk diproses. Elemen seperti Button digunakan untuk memicu pemrosesan dengan fungsi tertentu.

   Notebook ini juga mendemonstrasikan integrasi dengan model dari Hugging Face Hub. Sebagai contoh, antarmuka memuat model GPT-J-6B untuk melengkapi teks pengguna. Interaksi ini diatur dengan elemen Button untuk memulai proses prediksi dengan model.

   Selanjutnya, integrasi lebih lanjut dengan model NLP seperti *automatic speech recognition* (ASR) dan klasifikasi teks dilakukan. Input berupa file audio diproses untuk menghasilkan teks menggunakan pipeline ASR, yang kemudian diklasifikasikan untuk analisis sentimen. Gradio Blocks mengatur alur kerja dengan komponen seperti tombol Button untuk menjalankan setiap langkah secara berurutan.

   Akhirnya, contoh tentang perubahan dinamis antarmuka diberikan. Sebuah elemen Radio memungkinkan pengguna memilih jenis teks yang ingin ditulis, seperti "short" atau "long". Pilihan ini secara otomatis memperbarui properti Textbox, menunjukkan fleksibilitas Gradio dalam menciptakan antarmuka yang interaktif dan responsif.

   Notebook ini menunjukkan bagaimana Gradio Blocks dapat digunakan untuk membangun aplikasi interaktif dengan berbagai fungsi, termasuk teks, gambar, audio, dan integrasi model pembelajaran mesin.