(بسمه تعالی)

گزارش پروژه نهایی درس پردازش زبان های طبیعی(NLP)

عنوان پروژه:  
پروژه اول دستیار هوشمند صوتی

نام و نام خانوادگی:  
فروزان محمدزاده وطنچی

شماره دانشجویی:  
403150287

دانشگاه:  
دانشگاه آزاد اسلامی، واحد علوم و تحقیقات

نام استاد محترم:  
جناب آقای دکتر کوچاری

فهرست مطالب:

1.چکیده------------------------------------------------------- 3

2.مقدمه 3,4----------------------------------------------------

3.تعاریف برخی مفاهیم پایه4-----------------------------------

4. **شرح مراحل اجرای پروژ**ه5-------------------------------------

5.تصاویر رابط گرافیکی و کارکرد دستیار هوشمند با داده های خارجی-----

----------------------------------------------------------------------50

6.نتیجه‌گیری نهایی52--------------------------------------------

**1)چکیده:**

در این پروژه، طراحی و پیاده‌سازی یک بخش ابتدایی از سامانه‌ی دستیار صوتی هوشمند انجام شده است که قادر به درک نیت کاربر و استخراج اطلاعات کلیدی ((NER از جملات ورودی است. دو وظیفه‌ی اصلی در این پروژه شامل **تشخیص نیت کاربر (Intent Detection)** و **شناسایی موجودیت‌های نامدار (Named Entity Recognition)** می‌باشند. برای این منظور، از مدل **ParsBERT** به‌عنوان مدل پایه استفاده شده است و روی داده‌های فارسی آموزش مجدد صورت گرفته است. پس از انجام مراحل پیش‌پردازش، داده‌ها به فرمت مناسب برای مدل تبدیل شده و مدل‌ها به کمک کتابخانه Transformers آموزش داده شدند. نتایج ارزیابی نشان‌دهنده‌ی عملکرد مطلوب مدل‌ها در تشخیص دقیق نیت و برچسب‌گذاری موجودیت‌ها بودند، که نشان می‌دهد استفاده از مدل‌های از پیش آموزش‌دیده فارسی برای کاربردهای عمومی بسیار اثربخش است.

**2)مقدمه:**

با پیشرفت‌های چشم‌گیر در حوزه‌ی **پردازش زبان طبیعی** و ظهور مدل‌های مبتنی بر یادگیری عمیق، توسعه‌ی سامانه‌های هوشمند که بتوانند زبان انسان را به‌درستی تفسیر کنند، به یکی از اولویت‌های اصلی در فناوری‌های تعاملی تبدیل شده است. **دستیارهای صوتی هوشمند** نظیر Siri، Google Assistant و Alexa از جمله موفق‌ترین نمونه‌ها در این زمینه هستند که با استفاده از تکنیک‌های پیشرفته‌ی یادگیری ماشین، می‌توانند دستورات کاربران را تحلیل و اجرا کنند.

دو وظیفه‌ی کلیدی در این سیستم ها وجود دارد:

1. **تشخیص نیت (Intent Detection)**: تعیین هدف کلی جمله‌ی کاربر، مانند "درخواست تاکسی"، "سفارش غذا"، "پخش موسیقی" و...
2. **شناسایی موجودیت‌های نامدار (NER)**: استخراج اطلاعات خاص از جمله مانند نام مکان، زمان، اشخاص، غذاها و ... .

مطالعات متعددی نشان داده‌اند که این دو مؤلفه نقش حیاتی در موفقیت سیستم‌های مکالمه‌محور دارند.

با این حال، اغلب پژوهش‌ها و پیاده‌سازی‌های موجود مبتنی بر زبان انگلیسی هستند و زبان فارسی با چالش‌های خاص خود مانند پیچیدگی ساختاری، کمبود منابع، و نبود ابزارهای بومی‌سازی‌شده مواجه است. در سال‌های اخیر، مدل‌های زبانی فارسی مانند **ParsBERT** توسط [Hooshvare Lab, 2020] توسعه یافته‌اند که امکان استفاده از قدرت مدل‌های ترنسفورمری را برای زبان فارسی فراهم می‌کنند.

در این پروژه سعی شده است که با بهره گیری از مدل ParsBERT و داده های برچسب خورده فارسی یک زیرسیستم پایه برای دستیار صوتی فارسی طراحی شود که شامل دو بخش تشخیص intent , NER است .

**3. تعاریف برخی مفاهیم پایه:**

### پردازش زبان طبیعی (NLP)

پردازش زبان طبیعی حوزه‌ای از هوش مصنوعی است که هدف آن درک و تولید زبان انسان توسط سیستم‌های کامپیوتری است. کاربردهای آن شامل استخراج اطلاعات، تحلیل احساس، ترجمه ماشین، خلاصه‌سازی متن، برچسب‌گذاری گفتار به متن، شناسایی نیت و موجودیت‌ است

### مدل ترنسفورمر (Transformer)

مدل‌های ترنسفورمر به‌عنوان یکی از معماری‌های مؤثر در NLP معرفی شده‌اند. این مدل‌ها مبتنی بر مکانیسم attention هستند که ارتباط بین کلمات را به‌خوبی درک می‌کنند

### بخش‌های مهم ترنسفورمر

این مدل‌ها معمولاً بخش‌های زیر را شامل می‌شوند:

* توکنایزر و embedding
* Positional encoding برای درک ترتیب کلمات
* لایه‌های attention چندسر (multi-head)
* encoder و/یا decoder بسته به کاربرد (مثلاً BERT فقط encoder، GPT فقط decoder)

### نقش ترنسفورمرها در NLP

مطالعه نشان می‌دهد که ترنسفورمرها:

* در وظایف فهمِ زبان مانند دسته‌بندی، NER، پرسش‌وپاسخ بسیار مؤثرند.
* در تولید متن (مانند پاسخگویی گفت‌وگویی یا تولید محتوا) نیز عملکرد قوی دارند.
* معماری‌های دوطرفه (BERT) برای فهم متن مناسب‌تر هستند، در حالی که معماری‌های تک‌جهتی (GPT) برای تولید متن کاربرد بیشتری دارند

**مدل های از پیش آموزش دیده شده**:

مدل‌های از پیش آموزش‌دیده مانند BERT و ParsBERT بر روی حجم زیادی از داده‌های متنی آموزش دیده‌اند و سپس می‌توان آن‌ها را برای وظایف خاصی مانند NER یا Intent Detection تنظیم مجدد (Fine-tune) کرد. در این پروژه از مدل ParsBERT استفاده شده که به طور خاص برای زبان فارسی آموزش دیده است.

**4) شرح مراحل اجرای پروژه**

سلول 1: نصب کتابخانه‌ها و آماده‌سازی محیط

#سلول 1:

!pip install -q transformers seqeval scikit-learn pandas numpy torch

در این بخش، کتابخانه‌های مورد نیاز برای اجرای پروژه را نصب میکنیم:

* : transformers برای استفاده از مدل‌های پیش‌آماده‌ی زبان مانند BERT از کتابخانه Hugging Face.
* Seqeval : کتابخانه‌ای برای ارزیابی مدل‌های NER که قابلیت محاسبه‌ی معیارهایی مانند precision، recall و F1-score را دارد.
* scikit-learn : جهت استفاده از توابع ارزیابی مانند classification\_report برای مدل نیت.
* pandas, numpy: برای پردازش داده‌ها و ساختارهای آرایه‌ای.
* Torch : کتابخانه اصلی PyTorch برای پیاده‌سازی و آموزش مدل‌های یادگیری عمیق

سلول 2: بارگذاری و استخراج داده‌ها از Google Drive

#سلول2:

from google.colab import drive

import zipfile

drive.mount('/content/drive')

zip\_path = '/content/drive/MyDrive/archive.zip'

extract\_path = '/content/dataset'

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

    zip\_ref.extractall(extract\_path)

توضیحات سلول 2:

Google Drive به محیط Colab متصل می‌شود تا به فایل‌های ذخیره‌شده دسترسی داشته باشیم. و فایل فشرده‌ی archive.zip که شامل داده‌های اصلی پروژه (برای آموزش مدل‌ها) است از مسیر مشخص‌شده در درایو بارگذاری می‌شود.با استفاده از zipfile.ZipFile فایل zip استخراج شده و داده‌ها در مسیر محلی /content/dataset قرار می‌گیرند.

به طور کلی در این بخش داده ها را بارگذاری میکنیم و از گوگل درایور استخراح میکنیم.

سلول 3 :

#سلول 3:

from transformers import AutoTokenizer, AutoModelForMaskedLM

import torch

import random

import pandas as pd

from tqdm import tqdm

tokenizer = AutoTokenizer.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

model = AutoModelForMaskedLM.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

model.eval()

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model.to(device)

# بارگذاری داده اصلی

train\_df = pd.read\_csv('/content/dataset/train.csv')

train\_df['words'] = train\_df['words'].apply(lambda x: x.strip().split())

train\_df['words\_label'] = train\_df['words\_label'].apply(lambda x: x.strip().split())

def mask\_and\_predict(words, labels):

    new\_words = []

    new\_labels = labels.copy()

    for i, (word, label) in enumerate(zip(words, labels)):

        if label == "O" and word.isalpha() and random.random() < 0.3:

            temp\_words = words.copy()

            temp\_words[i] = tokenizer.mask\_token

            text = " ".join(temp\_words)

            inputs = tokenizer.encode(text, return\_tensors="pt").to(device)

            mask\_index = (inputs == tokenizer.mask\_token\_id).nonzero(as\_tuple=True)[1]

            if len(mask\_index) == 0:

                continue

            with torch.no\_grad():

                outputs = model(inputs)

            logits = outputs.logits

            predicted\_token\_id = torch.argmax(logits[0, mask\_index.item()]).item()

            predicted\_token = tokenizer.decode([predicted\_token\_id]).strip()

            if predicted\_token and predicted\_token != tokenizer.mask\_token:

                new\_words.append(predicted\_token)

            else:

                new\_words.append(word)

        else:

            new\_words.append(word)

    return new\_words, new\_labels

# تولید 1000 داده جدید

augmented\_data = []

for \_ in tqdm(range(1000), desc="Generating data"):

    row = train\_df.sample(n=1).iloc[0]

    words = row['words']

    labels = row['words\_label']

    intent = row['intent\_label']

    new\_words, new\_labels = mask\_and\_predict(words, labels)

    augmented\_data.append({

        "words": " ".join(new\_words),

        "words\_label": " ".join(new\_labels),

        "intent\_label": intent

    })

aug\_df = pd.DataFrame(augmented\_data)

aug\_df.to\_csv('/content/augmented\_train.csv', index=False)

print("داده‌های افزوده‌شده با ParsBERT با موفقیت ذخیره شدند.")

توضیحات سلول 3:

در این سلول، از مدل زبانی **ParsBERT** برای **افزایش داده (Data Augmentation)** استفاده شده است. کلمات غیر برچسب‌خورده (Label "O") در داده‌های متنی به‌صورت تصادفی با توکن [MASK] جایگزین می‌شوند و سپس مدل پیش‌بینی می‌کند که چه کلمه‌ای می‌تواند در آن موقعیت قرار بگیرد. این روش باعث تولید نسخه‌های متفاوت از داده‌های اصلی می‌شود بدون اینکه برچسب‌ها (Labels) تغییر کنند.

در نهایت، 1000 نمونه‌ی جدید تولید شده و در قالب فایل CSV ذخیره می‌شوند.

کتابخانه ها و مدل های استفاده شده:

### transformers (از HuggingFace):

* برای استفاده از مدل‌های پیش‌آموزش‌دیده مانند **ParsBERT** استفاده می‌شود.
* توکنایزر و مدل bert-base-parsbert-uncased از مخزن HooshvareLab بارگذاری می‌شود.

### torch (PyTorch):

* برای اجرای مدل و انجام عملیات تنسوری بر روی داده‌ها.

### pandas:

* برای بارگذاری و ذخیره‌سازی داده‌های جدولی (CSV).

### tqdm:

* برای نمایش نوار پیشرفت در حلقه‌های طولانی (در اینجا، حلقه تولید داده).

بررسی بخش های مهم کد:

### 1. **بارگذاری مدل و توکنایزر ParsBERT:**

tokenizer = AutoTokenizer.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

model = AutoModelForMaskedLM.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

این بخش مدل ParsBERT و توکنایزر مربوط به آن را از مخزن اینترنتی HuggingFace بارگذاری می‌کند. مدل مورد استفاده، مخصوص زبان فارسی است و برای وظایف **Masked Language Modeling** آموزش دیده است.

1. **تشخیص دستگاه (GPU یا CPU):**

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model.to(device)

1. **بارگذاری و آماده‌سازی داده‌ها:**

train\_df = pd.read\_csv('/content/dataset/train.csv')

train\_df['words'] = train\_df['words'].apply(lambda x: x.strip().split())

train\_df['words\_label'] = train\_df['words\_label'].apply(lambda x: x.strip().split())

فایل CSV مربوط به داده‌های آموزش بارگذاری شده و ستون‌های متنی (words) و برچسب‌ها (words\_label) به لیست کلمات تبدیل می‌شوند.

1. **تابع اصلی افزایش داده -** mask\_and\_predict**:**

def mask\_and\_predict(words, labels):

این تابع، لیستی از کلمات و برچسب‌های متن ورودی را دریافت می‌کند و تعدادی از کلمات دارای برچسب "O" را با [MASK] جایگزین می‌کند و از مدل می‌خواهد آن را پیش‌بینی کند.

مراحل کلیدی در این تابع:

* انتخاب تصادفی برخی کلمات برای جایگزینی با ماسک (با احتمال 30%).
* تبدیل لیست کلمات به رشته متنی.
* استفاده از مدل برای پیش‌بینی توکن جایگزین.
* بازسازی لیست جدید از کلمات.

نکته: برچسب‌ها (labels) تغییر نمی‌کنند چون تغییر فقط روی کلمات غیرمهم (برچسب O) انجام می‌شود.

**5. تولید 1000 داده جدید:**

for \_ in tqdm(range(1000), desc="Generating data"):

    row = train\_df.sample(n=1).iloc[0]

    words = row['words']

    labels = row['words\_label']

    intent = row['intent\_label']

    new\_words, new\_labels = mask\_and\_predict(words, labels)

    augmented\_data.append({

        "words": " ".join(new\_words),

        "words\_label": " ".join(new\_labels),

        "intent\_label": intent

    })

در این حلقه، 1000 نمونه جدید تولید می‌شود. در هر تکرار، یک ردیف تصادفی از داده‌های اصلی انتخاب و با استفاده از تابع mask\_and\_predict نسخه جدیدی از آن ساخته می‌شود.

1. **ذخیره‌سازی داده‌های افزوده‌شده:**

aug\_df.to\_csv('/content/augmented\_train.csv', index=False)

داده‌های جدید در فایل CSV ذخیره می‌شوند تا در مراحل بعدی آموزش مدل مورد استفاده قرار گیرند.

## نتیجه گیری :

این سلول با استفاده از روش **Masked Language Modeling** و مدل ParsBERT داده‌های جدیدی تولید می‌کند که ساختار زبانی مشابه داده‌های واقعی دارند. این کار می‌تواند به **افزایش تعمیم‌پذیری مدل‌های NER یا Intent Detection** کمک کند، به‌خصوص زمانی که حجم داده‌های اصلی کم باشد. یکی از نقاط قوت این روش، حفظ یکپارچگی برچسب‌ها و استفاده هوشمندانه از مدل زبانی برای تولید داده‌های واقع‌گرایانه است.

سلول 4: بارگذاری داده‌های افزوده‌شده

#سلول4

aug\_df = pd.read\_csv('/content/augmented\_train.csv')

aug\_df.head(10)

خروجی:

|  | **words** | **words\_label** | **intent\_label** |
| --- | --- | --- | --- |
| **0** | لطفا از رستوران سیب برای هر غذاها یه کباب برگ ... | O O B-Restaurant I-Restaurant O O O B-Number B... | Order\_Food |
| **1** | از یه عکس بگیر تا هتل داریوش | O O O O O B-Destination I-Destination | Catch\_a\_Cab |
| **2** | پرطرفدارترین ترانه یاس رو بذار | O O B-Artist O O | Play\_Music |
| **3** | دمای پنجشنبه آینده در کرمان چقدر است | O B-Date I-Date O B-Location O O | Get\_Weather |
| **4** | ترانه ای تو بد ترین آلبوم کامران و هومن پخش کن | O O O O O O B-Artist I-Artist I-Artist O O | Play\_Music |
| **5** | لطفا از اکبرجوجه برای شام سه تا پیتزای مخصوص س... | O O B-Restaurant O O B-Number I-Number B-Food ... | Order\_Food |
| **6** | شنبه قبل اهواز پیش بینی هوا که چه جوری بود | B-Date I-Date B-Location O O O O O O O | Get\_Weather |
| **7** | از میدان نقش جهان تا مدرسه هزینه چند برابر ؟ | O B-Departure I-Departure I-Departure O B-Dest... | Catch\_a\_Cab |
| **8** | من یه عالمه کتاب پرفروش میخوام | O O O O O O | Play\_Music |
| **9** | یک عدد اسنپ میخوام تا من رو ببره رستوران روسی | O O O O O O O O B-Destination I-Destination | Catch\_a\_Cab |

توضیح سلول 4:

در این سلول، فایل augmented\_train.csv که در **سلول ۳** ایجاد و ذخیره شده بود، بارگذاری می‌شود و ۱۰ سطر اول آن برای بررسی نمایش داده می‌شود.این داده‌ها نسخه‌های بازنویسی‌شده‌ی داده‌های اصلی هستند که با استفاده از مدل **ParsBERT** تولید شده‌اند.

سلول 5: ادغام داده‌ها و ذخیره فایل نهایی

#سلول5:

original\_df = pd.read\_csv('/content/dataset/train.csv')

if 'Unnamed: 0' in original\_df.columns:

    original\_df = original\_df.drop(columns=['Unnamed: 0'])

combined\_df = pd.concat([original\_df, aug\_df], ignore\_index=True)

combined\_df.to\_csv('/content/train\_augmented\_combined.csv', index=False)

combined\_df.sample(10)

خروجی:

|  | **Words** | **words\_label** |
| --- | --- | --- |
| **915** | قیمت آژانس از خونه تا شاهچراغ چند برابر است | O O O B-Departure O B-Destination O O O | Catch\_a\_Cab |
| **603** | این ترانه از محسن چاووشی پخش شد | O O O B-Artist I-Artist O O | Play\_Music |
| **1099** | پیش بینی اسیب شناسی دمشق شنبه قبل چطور بود | O O O O B-Location B-Date I-Date O O | Get\_Weather |
| **883** | بهترین موسیقی مایکل جکسون رو پخش کن | O O B-Artist I-Artist O O O | Play\_Music |
| **431** | آژانس از باشگاه تا تجریش چنده | O O B-Departure O B-Destination O | Catch\_a\_Cab |
| **236** | یک ترانه پرفروش از کیتی پری پخش کن | O O O O B-Artist I-Artist O O | Play\_Music |
| **381** | پس فردا در قاهره پیش بینی آب و هوا چگونه است | B-Date I-Date O B-Location O O O O O O O | Get\_Weather |
| **646** | اب و هوا فردا بارانیه، لطفا چتر یادت نره | O O O B-Date O O O O O | Get\_Weather |
| **332** | من برای ناهار میخوام یک عدد همبرگر بخورم از ... | O O O O B-Number I-Number B-Food O O B-Restaur... | Order\_Food |
| **1267** | برای ناهار از ساندویچ آیدا چهار تا کباب کوبیده... | O O O B-Restaurant I-Restaurant B-Number I-Num... | Order\_Food |

توضیح سلول 5:

در این سلول، داده‌های اصلی و افزوده‌شده به هم **ادغام** می‌شوند تا یک مجموعه داده نهایی بزرگ‌تر به‌نام train\_augmented\_combined.csv ایجاد شود. این فایل شامل **داده‌های واقعی + داده‌های مصنوعی** است و برای آموزش بهتر مدل استفاده می‌شود.

توضیح بخش های مهم کد:

**1. بارگذاری داده اصلی:**

original\_df = pd.read\_csv('/content/dataset/train.csv')

داده اصلی (واقعی) دوباره بارگذاری می‌شود تا با داده‌های تولیدشده ترکیب شود.

2.حذف ستون اضافی:

if 'Unnamed: 0' in original\_df.columns:

    original\_df = original\_df.drop(columns=['Unnamed: 0'])

گاهی در ذخیره‌سازی فایل‌های CSV، یک ستون اضافی با نام 'Unnamed: 0' به‌صورت خودکار ایجاد می‌شود. این شرط بررسی می‌کند که اگر چنین ستونی وجود دارد، آن را حذف کند تا اختلالی در ترکیب داده‌ها به وجود نیاید.

**3. ادغام داده‌ها:**

combined\_df = pd.concat([original\_df, aug\_df], ignore\_index=True)

در این خط، دو دیتافریم اصلی (original\_df) و افزوده‌شده (aug\_df) در کنار هم قرار می‌گیرند (تجمیع سطری) و یک دیتافریم ترکیبی (combined\_df) ساخته می‌شود. پارامتر ignore\_index=True کمک می‌کند تا اندیس‌های دیتافریم جدید به‌صورت متوالی تنظیم شوند.

**4. ذخیره دیتافریم نهایی:**

combined\_df.to\_csv('/content/train\_augmented\_combined.csv', index=False)

مجموعه داده نهایی در یک فایل CSV جدید ذخیره می‌شود. پارامتر index=False از ذخیره‌شدن اندیس‌های عددی در فایل جلوگیری می‌کند.

**5.نمونه گیری از داده های نهایی:**

combined\_df.sample(10)

برای بررسی بصری صحت داده‌ها، ۱۰ ردیف تصادفی از دیتافریم ترکیبی نمایش داده می‌شود

.

سلول 6:

#سلول 6:

class Config:

    def \_\_init\_\_(self):

        self.model\_name = "HooshvareLab/bert-base-parsbert-uncased"

        self.max\_length = 32

        self.batch\_size = 16

        self.learning\_rate = 5e-5

        self.epochs =10

        self.dropout= 0.3

config = Config()

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

torch.manual\_seed(42)

توضیحات سلول 6:

در این بخش به صورت ساختارمند تنظیمات کلی پروژه را در قالب کلاس config انجام میدهیم. تابع init را که یک constructor هست ایجاد کرده ایم .

  self.model\_name = "HooshvareLab/bert-base-parsbert-uncased":

مدل زبانی ParsBERT:

این مدل بر پایه مدل زبانی BERT توسط تیم HooshvareLab ساخته شده است و مدل زبانی عمیق برای زبان فارسی هست .

کاربرد های آن عبارتند از :

1.یادگیری ساختار زبان فارسی و درک کاربرد کلمات در زمینه های مختلف

2.تشخیص نیت (Intent Detection)

3.برچسب‌گذاری کلمات (NER)

4.تحلیل احساسات (Sentiment Analysis)

5.ترجمه و خلاصه‌سازی (با ترکیب با مدل‌های دیگر)

max\_length=32:

همانطور که میدانیم مدل bert جملات با طول یکسان را میپذیرد و اگر طول جمله کم باشد آن را با zero padding جایگزین میکند و یا آن جمله را truncation کوتاه میکند.که در اینجا طول جمله را 32 قرار داده ایم.

self.batch\_size = 16:

یعنی مدل در هربار پردازش 16 تا جمله را مشاهده کند و این کار رو تا جایی ادامه بدهد که جملات تمام شود. به عبارتی دیگر مدل در هر بار از یک epoch 16 جمله را همزمان پردازش میکند و این کار تا پایان اون epoch انجام میشود تا کل دیتاست خوانده شود و این کار باعث آموزش سریع تر میشود چون جملات هم زمان پردازش میشن و مدل با نمونه های متنوع در هر batch یاد میگیرد . و حافظه کم تری هم مصرف میشود

Learning\_rate=5e-5 :

یعنی نرخ یادگیری را 0.00005 در نظر گرفته ایم . که یعنی مدل با این سرعت یادبگیرد و خودش اصلاح کند . ما باید مقدار نرخ یادگیری را جوری تنظیم کنیم که یادگیری خیلی کند یا سریع نباشد که مدل یا خیلی زیاد اصلاح کند یا خیلی آهسته که در این جا مقدار 0.00005 مقدار معقولی میباشد.

:self.epochs =10

یعنی این که چند بار کل دیتاست را مدل بخواند و آموزش ببیند.

self.dropout = 0.3   :

در هر بار آموزش، به صورت تصادفی حدود ۳۰٪ از نورون‌ها غیرفعال می‌شوند تا مدل فقط به برخی ویژگی‌ها تکیه نکند و در نتیجه از **وابستگی زیاد به داده‌های آموزشی (Overfitting)** جلوگیری شود.

config = Config():

در این قسمت از کلاس config یک شی به نام config ایجاد کرده ایم که همه تنظیمات پروژه داخل آن ذخیره شده و در بخش های مختلف کد از آن استفاده خواهیم کرد.

سلول 7:

#سلول 7:

import pandas as pd

def load\_data():

    # خواندن داده‌ها

    train\_df = pd.read\_csv('/content/train\_augmented\_combined.csv')

    test\_df = pd.read\_csv('/content/dataset/test.csv')

    # حذف ردیف‌های ناقص

    train\_df.dropna(subset=['words', 'words\_label'], inplace=True)

    test\_df.dropna(subset=['words', 'words\_label'], inplace=True)

    def split\_row(row):

        words = str(row['words']).split()

        labels = str(row['words\_label']).split()

        return words, labels

    train\_df['tokens'], train\_df['ner\_tags'] = zip(\*train\_df.apply(split\_row, axis=1))

    test\_df['tokens'], test\_df['ner\_tags'] = zip(\*test\_df.apply(split\_row, axis=1))

    unique\_intents = sorted(train\_df['intent\_label'].unique())

    unique\_ner\_tags = sorted(set(tag for tags in train\_df['ner\_tags'] for tag in tags))

    intent2id = {v: k for k, v in enumerate(unique\_intents)}

    intent\_id2label = {k: v for v, k in intent2id.items()}

    ner\_tag2id = {v: k for k, v in enumerate(unique\_ner\_tags)}

    ner\_id2tag = {k: v for v, k in ner\_tag2id.items()}

    return train\_df.reset\_index(drop=True), test\_df.reset\_index(drop=True), intent2id, intent\_id2label, ner\_tag2id, ner\_id2tag

train\_df, test\_df, intent2id, intent\_id2label, ner\_tag2id, ner\_id2tag = load\_data()

توضیح سلول 7:

در این سلول تابعی با نام load\_data() تعریف شده است که:

1. داده‌های ترکیبی (واقعی + افزوده‌شده) و داده‌های آزمون را از فایل CSV بارگذاری می‌کند.
2. داده‌های ناقص را حذف می‌کند.
3. ستون‌های words و words\_label را به لیست توکن‌ها و برچسب‌های NER تبدیل می‌کند.
4. دیکشنری‌های نگاشت برچسب‌ها به شناسه‌های عددی را برای Intent و NER ایجاد می‌کند.
5. در نهایت، این داده‌های پردازش‌شده و نگاشت‌ها را برمی‌گرداند.
6. **خواندن فایل‌های CSV:**

train\_df = pd.read\_csv('/content/train\_augmented\_combined.csv')

test\_df = pd.read\_csv('/content/dataset/test.csv')

فایل آموزش ترکیبی (train\_augmented\_combined.csv) و فایل آزمون (test.csv) بارگذاری می‌شوند.

1. **حذف ردیف‌های ناقص مقدارهای NaN):)**

train\_df.dropna(subset=['words', 'words\_label'], inplace=True)

test\_df.dropna(subset=['words', 'words\_label'], inplace=True)

اگر داده‌ای وجود داشته باشد که words یا words\_label نداشته باشد (مقدار گمشده)، آن ردیف حذف می‌شود.این کار مانع ایجاد خطا در مرحله پردازش می‌شود.

1. **تابع کمکی** split\_row **برای تبدیل رشته‌ها به لیست:**

def split\_row(row):

        words = str(row['words']).split()

        labels = str(row['words\_label']).split()

        return words, labels

* ستون‌های متنی که به‌صورت رشته‌ای ذخیره شده‌اند، به لیست از کلمات و برچسب‌ها تبدیل می‌شوند.
* str() استفاده شده تا مطمئن باشیم حتی اگر به‌صورت تصادفی داده‌ای به‌صورت float باشد، تبدیل به رشته و سپس لیست شود.

**4. تبدیل ستون‌های** words **و** words**\_**label **به** tokens **و** ner\_tags

train\_df['tokens'], train\_df['ner\_tags'] = zip(\*train\_df.apply(split\_row, axis=1))

test\_df['tokens'], test\_df['ner\_tags'] = zip(\*test\_df.apply(split\_row, axis=1))

* ا استفاده از تابع apply بر روی هر ردیف، ستون‌های tokens و ner\_tags ساخته می‌شوند.
* zip(\*) برای جداسازی جفت‌ها در دو ستون جدا استفاده شده است.

**5.استخراج و مرتب‌سازی برچسب‌های منحصر‌به‌فرد:**

 unique\_intents = sorted(train\_df['intent\_label'].unique())

 unique\_ner\_tags = sorted(set(tag for tags in train\_df['ner\_tags'] for tag in tags))

* برچسب‌های Intent و NER به‌صورت یکتا و مرتب استخراج می‌شوند تا آماده نگاشت عددی شوند.
* set برای حذف تکراری‌ها و sorted برای مرتب‌سازی استفاده می‌شود.

1. **ساخت دیکشنری نگاشت برچسب‌ها به اعداد:**

Intent:

intent2id = {v: k for k, v in enumerate(unique\_intents)}

intent\_id2label = {k: v for v, k in intent2id.items()}

NER:

 ner\_tag2id = {v: k for k, v in enumerate(unique\_ner\_tags)}

 ner\_id2tag = {k: v for v, k in ner\_tag2id.items()}

برای استفاده در مدل‌های یادگیری ماشین، لازم است که برچسب‌ها به صورت عددی (integer) باشند.

enumerate برای اختصاص عدد یکتا به هر برچسب استفاده می‌شود.

دو نگاشت ساخته می‌شود: یکی از **برچسب به عدد** و دیگری از **عدد به برچسب** (معکوس).

1. **بازگرداندن خروجی:**

return train\_df.reset\_index(drop=True), test\_df.reset\_index(drop=True), intent2id, intent\_id2label, ner\_tag2id, ner\_id2tag

داده‌های آموزش و آزمون به‌همراه دیکشنری‌های نگاشت برای Intent و NER برگردانده می‌شوند.

reset\_index(drop=True) باعث می‌شود اندیس‌ها مجدد از صفر شروع شوند و ستونی برای اندیس قبلی ایجاد نشود.

نتیجه گیری:

سلول ۷ یکی از مراحل کلیدی در آماده‌سازی داده برای **مدل‌های چندوظیفه‌ای NLP** مانند NER (شناسایی موجودیت‌های نام‌دار) و Intent Classification است. این سلول تضمین می‌کند که داده‌ها به فرمت مناسب برای مدل‌سازی تبدیل شوند، و نگاشت عددی برچسب‌ها نیز آماده باشد.

سلول 8: ساخت Dataset و آماده‌سازی داده‌ها برای مدل BERT

from transformers import AutoTokenizer

from torch.utils.data import Dataset

tokenizer = AutoTokenizer.from\_pretrained(config.model\_name)

class IntentDataset(Dataset):

    def \_\_init\_\_(self, df):

        self.texts = [" ".join(tokens) for tokens in df['tokens']]

        self.labels = [intent2id[i] for i in df['intent\_label']]

    def \_\_len\_\_(self):

        return len(self.texts)

    def \_\_getitem\_\_(self, idx):

        encoding = tokenizer(

            self.texts[idx],

            padding='max\_length',

            truncation=True,

            max\_length=config.max\_length,

            return\_tensors='pt'

        )

        return {

            'input\_ids': encoding['input\_ids'].squeeze(),

            'attention\_mask': encoding['attention\_mask'].squeeze(),

            'labels': torch.tensor(self.labels[idx])

        }

class NERDataset(Dataset):

    def \_\_init\_\_(self, df):

        self.tokens = df['tokens']

        self.labels = df['ner\_tags']

    def \_\_len\_\_(self):

        return len(self.tokens)

    def \_\_getitem\_\_(self, idx):

        tokens = self.tokens[idx]

        ner\_tags = self.labels[idx]

        encoding = tokenizer(

            tokens,

            is\_split\_into\_words=True,

            padding='max\_length',

            truncation=True,

            max\_length=config.max\_length,

            return\_tensors='pt'

        )

        labels = [-100] \* config.max\_length

        word\_ids = encoding.word\_ids()

        for i, word\_idx in enumerate(word\_ids):

            if word\_idx is not None and word\_idx < len(ner\_tags):

                labels[i] = ner\_tag2id[ner\_tags[word\_idx]]

        return {

            'input\_ids': encoding['input\_ids'].squeeze(),

            'attention\_mask': encoding['attention\_mask'].squeeze(),

            'labels': torch.tensor(labels)

        }

توضیحات سلول 8:

در این بخش از پروژه، هدف اصلی ما تعریف دو کلاس مجزا برای آماده‌سازی داده‌ها جهت ورود به مدل‌های یادگیری عمیق است. که دو task مختلف داریم:

1. **Intent Classification (تشخیص نیت جمله):** در این تسک، هر جمله به صورت یک ورودی مستقل وارد مدل شده و خروجی، برچسب نیت جمله خواهد بود (

**مثال:**

متن: تا مهرشهر میخوام تاکسی بگیرم

Intent: Catch\_a\_Cab

1. **Named Entity Recognition (NER) یا برچسب‌گذاری کلمات:** در این تسک، هر کلمه از جمله باید دارای برچسب خاص خود باشد (

مثال: NER Tags:

تا: O

مهرشهر: B-Destination

میخوام: O

تاکسی: O

بگیرم: O

--------------------------------------------------------------------------------------------

1.بارگذاری توکنایزر parsBert

در ابتدا، توکنایزر مربوط به مدل HooshvareLab/bert-base-parsbert-uncased بارگذاری شده است. این توکنایزر وظیفه دارد متن‌های فارسی را به توکن‌هایی تبدیل کند که مدل BERT قادر به درک و پردازش آن‌ها باشد.

توکنایزر نقش مهمی در سازگارسازی ورودی با مدل دارد و به صورت خودکار عملیات padding، truncation و تبدیل متن به شناسه‌های عددی را انجام می‌دهد.

tokenizer = AutoTokenizer.from\_pretrained(config.model\_name)

2.کلاس IntentDataset

class IntentDataset(Dataset):

این کلاس مخصوص آماده‌سازی داده‌ها برای مدل intent classification است. در ادامه توضیح اجزای آن آمده است:

##### - \_\_init\_\_:

در این متد، ستون tokens که شامل لیستی از کلمات هر جمله است، با استفاده از join() به یک رشته متنی تبدیل می‌شود. سپس با نگاشت برچسب‌ها از طریق دیکشنری intent2id، لیستی از برچسب‌های عددی تهیه می‌شود.

self.texts = [" ".join(tokens) for tokens in df['tokens']]

self.labels = [intent2id[i] for i in df['intent\_label']]

\_\_getitem :

در این متد، با استفاده از توکنایزر، جمله متنی به فرمت مورد نیاز مدل BERT (شامل input\_ids و attention\_mask) تبدیل می‌شود. همچنین برچسب عددی نیت به صورت tensor بازگردانده می‌شود.

encoding = tokenizer(

            self.texts[idx],

            padding='max\_length',

            truncation=True,

            max\_length=config.max\_length,

            return\_tensors='pt'

        )

1. کلاس NERDataset (برای برچسب‌گذاری کلمه‌ای)

class NERDataset(Dataset):

این کلاس مخصوص آماده‌سازی داده‌ها برای مدل NER است و پیچیدگی بیشتری نسبت به کلاس قبلی دارد، زیرا باید نگاشت دقیق بین کلمات و برچسب‌هایشان در فرایند توکنایز کردن حفظ شود.

##### - \_\_init\_\_:

در این متد، ستون‌های tokens و ner\_tags از دیتافریم دریافت می‌شوند.

self.tokens = df['tokens']

self.labels = df['ner\_tags']

##### - \_\_getitem\_\_:

در این متد، ابتدا جمله به صورت لیست کلمات وارد توکنایزر می‌شود. گزینه is\_split\_into\_words=True به توکنایزر اطلاع می‌دهد که ورودی به صورت لیستی از کلمات است نه یک رشته متنی.

encoding = tokenizer(

            tokens,

            is\_split\_into\_words=True,

            padding='max\_length',

            truncation=True,

            max\_length=config.max\_length,

            return\_tensors='pt'

        )

سپس با استفاده از تابع word\_ids()، ارتباط بین توکن‌های خروجی و کلمات اصلی مشخص می‌شود. این کار به ما امکان می‌دهد تا بدانیم هر توکن مربوط به کدام کلمه اصلی است و برچسب درست را به آن اختصاص دهیم.

برای مدیریت توکن‌هایی که مربوط به padding هستند یا subwordهایی که نباید برچسب بخورند، لیستی به طول max\_length از مقدار -100 ساخته می‌شود (مقداری که در تابع loss از آن صرف‌نظر خواهد شد).

در ادامه، با پیمایش لیست word\_ids()، هر توکن واقعی برچسب مناسب از ner\_tags دریافت می‌کند:

labels = [-100] \* config.max\_length

        word\_ids = encoding.word\_ids()

        for i, word\_idx in enumerate(word\_ids):

            if word\_idx is not None and word\_idx < len(ner\_tags):

                labels[i] = ner\_tag2id[ner\_tags[word\_idx]]

جمع بندی:

در این سلول، دو کلاس بسیار مهم برای آماده‌سازی داده‌ها برای مدل‌های ترنسفورمر تعریف شده‌اند:

* IntentDataset برای دسته‌بندی نیت جمله
* NERDataset برای برچسب‌گذاری کلمه به کلمه

این ساختار داده‌ها را به فرمت قابل فهم برای مدل BERT تبدیل می‌کند و مراحل بعدی آموزش را ممکن می‌سازد. همچنین با استفاده از توکنایزر قدرتمند ParsBERT، اطمینان حاصل می‌شود که داده‌ها با زبان فارسی سازگاری کامل دارند.

سلول 9: آموزش مدل Intent Classification

from transformers import BertForSequenceClassification, get\_scheduler

from torch.optim import AdamW

from torch.utils.data import DataLoader

intent\_model = BertForSequenceClassification.from\_pretrained(

    config.model\_name,

    num\_labels=len(intent2id)

)

intent\_model.to(device)

train\_loader = DataLoader(IntentDataset(train\_df), batch\_size=config.batch\_size, shuffle=True)

optimizer = AdamW(intent\_model.parameters(), lr=config.learning\_rate, weight\_decay=0.01)

lr\_scheduler = get\_scheduler("linear", optimizer, num\_warmup\_steps=0,

                             num\_training\_steps=config.epochs \* len(train\_loader))

intent\_model.train()

for epoch in range(config.epochs):

    total\_loss = 0

    for batch in train\_loader:

        batch = {k: v.to(device) for k, v in batch.items()}

        outputs = intent\_model(\*\*batch)

        loss = outputs.loss

        loss.backward()

        optimizer.step()

        lr\_scheduler.step()

        optimizer.zero\_grad()

        total\_loss += loss.item()

    print(f"Epoch {epoch+1}, Intent Loss: {total\_loss/len(train\_loader):.4f}")

torch.save(intent\_model.state\_dict(), "intent\_model.pth")

توضیح سلول 9:

در این سلول، فرآیند آموزش مدل یادگیری عمیق برای **تشخیص نیت جمله (Intent Detection)** پیاده‌سازی شده است. این بخش از مهم‌ترین مراحل پروژه است، چرا که مدل اصلی تشخیص عملکرد جملات ورودی (مثل گرفتن تاکسی، پخش موسیقی و ...) در این قسمت آموزش می‌بیند.

1.تعریف مدل پایه براساس parsBERT

در ابتدا، از کلاس آماده BertForSequenceClassification که مخصوص مدل‌های BERT برای دسته‌بندی جملات است، استفاده کرده ایم.

intent\_model = BertForSequenceClassification.from\_pretrained(

    config.model\_name,

    num\_labels=len(intent2id)

)

در اینجا:

* مدل ParsBERT به‌عنوان پایه بارگذاری شده از طریق config.model\_name
* تعداد خروجی‌های مدل (num\_labels) برابر است با تعداد intent های مختلف موجود در دیتاست

در بخش زیر مدل به دستگاه محاسباتی منتقل می‌شود:

intent\_model.to(device)

2.تعریف DataLoader برای آموزش:

داده‌های مربوط به آموزش DataFrame آموزش در قالب کلاس IntentDataset که قبلاً تعریف شده بود، وارد DataLoader می‌شود:

train\_loader = DataLoader(IntentDataset(train\_df), batch\_size=config.batch\_size, shuffle=True)

batch\_size تعداد نمونه‌هایی است که در هر مرحله از آموزش هم‌زمان پردازش می‌شوند که برابر 16 گرفتیم

با shuffle=True ترتیب داده‌ها در هر epoch به صورت تصادفی تغییر می‌کند تا مدل بهتر یاد بگیرد

.4تعریف بهینه ساز و برنامه زمان بندی یادگیری (Learning Rate Scheduler)

برای آموزش مدل، بهینه‌سازی وزن‌ها از طریق الگوریتم **AdamW** انجام می‌شود:

optimizer = AdamW(intent\_model.parameters(), lr=config.learning\_rate, weight\_decay=0.01)

* learning\_rate: نرخ یادگیری (ایجاد تغییر در وزن‌ها)
* weight\_decay: جریمه برای جلوگیری از overfitting

برای تنظیم نرخ یادگیری در طول زمان، از یک Scheduler خطی استفاده شده است:

lr\_scheduler = get\_scheduler("linear", optimizer, num\_warmup\_steps=0,

                             num\_training\_steps=config.epochs \* len(train\_loader))

این Scheduler نرخ یادگیری را به‌تدریج کاهش می‌دهد تا مدل در پایان بهتر همگرا شود.

آموزش مدل در طول چند Epoch:

در این قسمت، مدل وارد مرحله‌ی یادگیری می‌شود:

intent\_model.train()

سپس با استفاده از یک حلقه تودرتو، در طول config.epochs دوره مدل آموزش می‌بیند:

intent\_model.train()

for epoch in range(config.epochs):

    total\_loss = 0

    for batch in train\_loader:

        batch = {k: v.to(device) for k, v in batch.items()}

        outputs = intent\_model(\*\*batch)

        loss = outputs.loss

        loss.backward()

        optimizer.step()

        lr\_scheduler.step()

        optimizer.zero\_grad()

        total\_loss += loss.item()

print(f"Epoch {epoch+1}, Intent Loss: {total\_loss/len(train\_loader):.4f}")

در هر تکرار:

* یک batch داده ورودی گرفته می‌شود
* ورودی‌ها به دستگاه (GPU/CPU) منتقل می‌شوند
* مدل مقدار **loss** (میزان خطا) را محاسبه می‌کند
* سپس با loss.backward() گرادیان‌ها محاسبه و با optimizer.step() وزن‌ها بروزرسانی می‌شوند
* scheduler.step() نرخ یادگیری را تغییر می‌دهد
* optimizer.zero\_grad() گرادیان‌ها را پاک می‌کند تا برای batch بعدی آماده باشد

پس از پایان هر epoch، میانگین خطا چاپ می‌شود

5. ذخیره‌سازی وزن‌های مدل آموزش‌دیده‌شده

در پایان آموزش، وزن‌های مدل ذخیره می‌شوند:

torch.save(intent\_model.state\_dict(), "intent\_model.pth")

فایل intent\_model.pth شامل وزن‌های آموزش‌دیده‌ی مدل است و می‌تواند در آینده برای ارزیابی یا پیش‌بینی مجدد بارگذاری شود.

### جمع‌بندی:

در این سلول، مراحل زیر به‌صورت کامل اجرا شده‌اند:

* تعریف مدل دسته‌بندی بر پایه ParsBERT
* آماده‌سازی داده‌ها با استفاده از DataLoader
* تعریف optimizer و scheduler برای آموزش دقیق‌تر
* اجرای حلقه آموزش مدل در طول چند epoch
* محاسبه و چاپ میانگین خطا در هر دوره
* ذخیره‌سازی مدل آموزش‌دیده

این سلول مرحله ی اصلی پروژه‌ی "دستیار صوتی هوشمند" محسوب می‌شود، زیرا مدل اکنون قادر خواهد بود بر اساس جمله‌ی ورودی، نیت گوینده را به درستی تشخیص دهد.

تحلیل خروجی ها:

Epoch 1, Intent Loss: 0.1161

Epoch 2, Intent Loss: 0.0031

Epoch 3, Intent Loss: 0.0015

Epoch 4, Intent Loss: 0.0010

Epoch 5, Intent Loss: 0.0008

Epoch 6, Intent Loss: 0.0006

Epoch 7, Intent Loss: 0.0006

Epoch 8, Intent Loss: 0.0005

Epoch 9, Intent Loss: 0.0005

Epoch 10, Intent Loss: 0.0005

تفسیر وتحلیل خروجی ها:

1. **کاهش سریع و پایدار خطا:**

مقدار اولیه خطا در epoch 1 برابر 0.1161 بوده که نسبتاً کم و مناسب برای شروع آموزش است.اما از epoch 2 به بعد، خطا با **شیب بسیار زیادی کاهش یافته** و به مقدار 0.0031 رسیده که نشانهٔ یادگیری سریع مدل است.در epoch 3 تا 5 روند کاهش ادامه پیدا کرده و به حدود 0.0008 رسیده‌ایم.

1. **رسیدن به حالت تقریباً پایدار (convergence):**

از epoch 6 به بعد، خطا تقریباً **به مقدار ثابت و نزدیک به صفر** رسیده و نوسانات آن بسیار کم شده است.این موضوع نشان می‌دهد که مدل **یادگیری نهایی را تثبیت کرده** و دیگر نیازی به ادامهٔ زیاد آموزش نیست.

3.عدم بروز overfiting:

مقدار بسیار پایین loss به‌خودی‌خود نشانهٔ overfitting نیست، اما اگر در کنار آن مدل روی داده تست هم عملکرد عالی داشته باشد (که در این پروژه دارد)، نشان می‌دهد که **مدل به‌درستی آموزش دیده و به داده خاصی وابسته نشده است.**

1. **بهره‌گیری مناسب از داده‌های افزوده‌شده:**

کاهش loss نشان می‌دهد که استفاده از داده‌های افزایش‌یافته (augmented) باعث **تنوع و کمک به تعمیم بهتر مدل** شده است.

نتیجه گیری :

مدل تشخیص نیت توانسته است در مدت زمان کوتاه (فقط چند epoch) یادگیری دقیق و باکیفیتی انجام دهد. رسیدن به loss بسیار پایین و پایدار در اواخر آموزش، و همچنین نتایج ارزیابی ۱۰۰٪ روی داده تست (در بخش ارزیابی)، نشان می‌دهد که مدل به‌خوبی ساختار معنایی جملات را یاد گرفته است و آمادهٔ استفاده در کاربردهای واقعی است.

سلول 10: آموزش مدل برچسب‌گذاری اسامی (Named Entity Recognition - NER)

#سلول 10:

from transformers import BertForTokenClassification

from torch.utils.data import DataLoader

ner\_model = BertForTokenClassification.from\_pretrained(

    config.model\_name,

    num\_labels=len(ner\_tag2id)

)

ner\_model.to(device)

ner\_loader = DataLoader(NERDataset(train\_df), batch\_size=config.batch\_size, shuffle=True)

optimizer = AdamW(ner\_model.parameters(), lr=config.learning\_rate, weight\_decay=0.01)

lr\_scheduler = get\_scheduler("linear", optimizer, num\_warmup\_steps=0,

                             num\_training\_steps=config.epochs \* len(ner\_loader))

ner\_model.train()

for epoch in range(config.epochs):

    total\_loss = 0

    for batch in ner\_loader:

        batch = {k: v.to(device) for k, v in batch.items()}

        outputs = ner\_model(\*\*batch)

        loss = outputs.loss

        loss.backward()

        optimizer.step()

        lr\_scheduler.step()

        optimizer.zero\_grad()

        total\_loss += loss.item()

    print(f"Epoch {epoch+1}, NER Loss: {total\_loss/len(ner\_loader):.4f}")

torch.save(ner\_model.state\_dict(), "ner\_model.pth")

خروجی:

Epoch 1, NER Loss: 0.3203

Epoch 2, NER Loss: 0.0267

Epoch 3, NER Loss: 0.0125

Epoch 4, NER Loss: 0.0077

Epoch 5, NER Loss: 0.0026

Epoch 6, NER Loss: 0.0016

Epoch 7, NER Loss: 0.0010

Epoch 8, NER Loss: 0.0008

Epoch 9, NER Loss: 0.0010

Epoch 10, NER Loss: 0.0007

توضیح کلی:

در این سلول، مدل BertForTokenClassification بر اساس معماری ParsBERT برای تشخیص موجودیت‌های اسمی (NER) آموزش داده می‌شود. داده‌های آموزشی از قبل آماده شده‌اند و مدل روی آن‌ها با استفاده از تابع خطا و به‌روزرسانی وزن‌ها طی چند دوره (Epoch) آموزش می‌بیند. در پایان نیز وزن‌های آموزش‌دیده‌شده مدل ذخیره می‌شوند.

مفاهیم مهم مورد استفاده در این سلول:

BertForTokenClassification:

مدل بر پایه BERT که برای مسائل برچسب‌گذاری ترتیبی مثل NER طراحی شده است. خروجی مدل برای هر توکن در دنباله به‌صورت یک توزیع احتمال برای برچسب‌های ممکن است.

AdamW:

الگوریتم بهینه‌سازی که اغلب در آموزش مدل‌های Transformer استفاده می‌شود؛ نسخه بهبود یافته‌ای از Adam با کنترل بهتر روی weight decay

get\_scheduler:

برای تنظیم نرخ یادگیری در طول آموزش استفاده می‌شود. اینجا از scheduler خطی استفاده شده است.

NERDataset:

کلاس سفارشی‌سازی‌شده برای آماده‌سازی داده‌های NER و تبدیل آن‌ها به قالب مورد نیاز برای مدل

توضیح بخش‌های مهم کد:

1. تعریف مدل NER

ner\_model = BertForTokenClassification.from\_pretrained(

    config.model\_name,

    num\_labels=len(ner\_tag2id)

)

مدل پایه از config.model\_name بارگذاری می‌شود (که باید HooshvareLab/bert-base-parsbert-uncased باشد)

تعداد برچسب‌های خروجی برابر با تعداد تگ‌های NER در داده‌ها است.

2. انتقال مدل به GPU یا CPU

ner\_model.to(device)

اگر GPU در دسترس باشد، مدل روی GPU آموزش می‌بیند که سرعت آموزش را افزایش می‌دهد.

3. تعریف DataLoader برای NER

ner\_loader = DataLoader(NERDataset(train\_df), batch\_size=config.batch\_size, shuffle=True)

داده‌ها به کمک کلاس NERDataset پردازش شده و در قالب Batch به مدل داده می‌شوند.

4. تنظیم بهینه‌ساز (Optimizer) و Scheduler

optimizer = AdamW(ner\_model.parameters(), lr=config.learning\_rate, weight\_decay=0.01)

lr\_scheduler = get\_scheduler("linear", optimizer, num\_warmup\_steps=0,

                             num\_training\_steps=config.epochs \* len(ner\_loader))

از AdamW برای آموزش استفاده شده و با کاهش خطی نرخ یادگیری در طول زمان، از overfitting جلوگیری می‌شود.

5. حلقه آموزش مدل

for epoch in range(config.epochs):

    total\_loss = 0

    for batch in ner\_loader:

        batch = {k: v.to(device) for k, v in batch.items()}

        outputs = ner\_model(\*\*batch)

        loss = outputs.loss

        loss.backward()

        optimizer.step()

        lr\_scheduler.step()

        optimizer.zero\_grad()

        total\_loss += loss.item()

    print(f"Epoch {epoch+1}, NER Loss: {total\_loss/len(ner\_loader):.4f}")

مدل در هر epoch روی تمام داده‌ها آموزش می‌بیند.

در هر Batch:مقدار Loss محاسبه می‌شود.

گرادیان‌ها محاسبه شده و وزن‌ها به‌روزرسانی می‌شوند

نرخ یادگیری توسط Scheduler تنظیم می‌شود

6. ذخیره مدل

torch.save(ner\_model.state\_dict(), "ner\_model.pth")

پس از آموزش کامل، وزن‌های مدل در فایلی به نام ner\_model.pth ذخیره می‌شوند تا برای inference یا تست آینده مورد استفاده قرار گیرد.

جمع‌بندی:

این سلول وظیفه آموزش مدل ParsBERT برای وظیفه NER را بر عهده دارد.

از معماری BertForTokenClassification استفاده شده و مدل برای هر توکن در یک جمله، برچسب NER مناسب را پیش‌بینی می‌کند.

با استفاده از AdamW و get\_scheduler فرایند آموزش بهینه‌سازی شده است.

روند آموزش شامل محاسبه loss، backward کردن گرادیان‌ها، تنظیم پارامترها و در نهایت ذخیره مدل است.

تحلیل خروجی:

Epoch 1, NER Loss: 0.3203

Epoch 2, NER Loss: 0.0267

Epoch 3, NER Loss: 0.0125

Epoch 4, NER Loss: 0.0077

Epoch 5, NER Loss: 0.0026

Epoch 6, NER Loss: 0.0016

Epoch 7, NER Loss: 0.0010

Epoch 8, NER Loss: 0.0008

Epoch 9, NER Loss: 0.0010

Epoch 10, NER Loss: 0.0007

تحلیل خروجی:

1. **شروع با خطای قابل توجه:**
   * در epoch 1 مقدار loss نسبتاً بالاست (0.3360)، که کاملاً طبیعی است زیرا مدل هنوز در مراحل ابتدایی یادگیری قرار دارد.
2. **کاهش شدید در مراحل اولیه:**
   * بین epoch 1 تا epoch 2 کاهش loss چشم‌گیر است (از 0.3360 به 0.0223)، که نشان‌دهنده **یادگیری مؤثر اولیه** مدل و قدرت بالای مدل پایه ParsBERT در درک ساختار زبان فارسی است.
3. **رسیدن به مقدار بسیار پایین و پایدار:**
   * از epoch 6 به بعد، مدل به **مرحله‌ی همگرایی (convergence)** رسیده و loss به مقادیر بسیار پایین و تقریباً ثابت (بین 0.0006 تا 0.0009) رسیده است.
4. **عملکرد فوق‌العاده مدل:**
   * مقادیر بسیار پایین خطا و ثبات آن در مراحل پایانی آموزش نشان‌دهنده آن است که مدل توانسته به‌خوبی ساختار معنایی و برچسب‌های NER را یاد بگیرد.

نتیجه گیری:

مدل NER پس از ۱۰ دوره آموزش، به دقت بسیار بالایی در یادگیری موجودیت‌های نامدار دست یافته است. روند کاهش loss بسیار سریع، منظم و بدون نوسان‌های غیرطبیعی بوده که نشانه‌ای از **یادگیری پایدار و مؤثر** مدل است. همچنین، عملکرد عالی مدل روی داده‌های آزمون (ارزیابی با seqeval) این نتیجه‌گیری را تأیید می‌کند.

**سلول 11: بخش ارزیابی مدل‌ها (Evaluation) وگزارش نهایی عملکرد مدل‌ها**

from sklearn.metrics import classification\_report

from seqeval.metrics import classification\_report as seqeval\_report

def evaluate\_intent(model, df):

    model.eval()

    loader = DataLoader(IntentDataset(df), batch\_size=config.batch\_size)

    preds, labels = [], []

    with torch.no\_grad():

        for batch in loader:

            batch = {k: v.to(device) for k, v in batch.items()}

            output = model(\*\*batch)

            pred = torch.argmax(output.logits, dim=1)

            preds.extend(pred.cpu().numpy())

            labels.extend(batch['labels'].cpu().numpy())

    print("\n Intent Classification Report:")

    print(classification\_report(labels, preds, target\_names=intent\_id2label.values()))

def evaluate\_ner(model, df):

    model.eval()

    loader = DataLoader(NERDataset(df), batch\_size=1)

    all\_preds, all\_labels = [], []

    with torch.no\_grad():

        for batch in loader:

            input\_ids = batch['input\_ids'].to(device)

            attention\_mask = batch['attention\_mask'].to(device)

            labels = batch['labels'].to(device)

            outputs = model(input\_ids=input\_ids, attention\_mask=attention\_mask)

            preds = torch.argmax(outputs.logits, dim=2)

            true\_labels, pred\_labels = [], []

            for i in range(config.max\_length):

                if labels[0][i] != -100:

                    true\_labels.append(ner\_id2tag[labels[0][i].item()])

                    pred\_labels.append(ner\_id2tag[preds[0][i].item()])

            all\_labels.append(true\_labels)

            all\_preds.append(pred\_labels)

    print("NER Report:")

    print(seqeval\_report(all\_labels, all\_preds))

evaluate\_intent(intent\_model, test\_df)

evaluate\_ner(ner\_model, test\_df)

**خروجی:**

Intent Classification Report:

precision recall f1-score support

Catch\_a\_Cab 1.00 1.00 1.00 31

Get\_Weather 1.00 1.00 1.00 31

Order\_Food 1.00 1.00 1.00 29

Play\_Music 1.00 1.00 1.00 31

accuracy 1.00 122

macro avg 1.00 1.00 1.00 122

weighted avg 1.00 1.00 1.00 122

/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder\_attention\_mask` is deprecated and will be removed in version 4.55.0 for `BertSdpaSelfAttention.forward`.

return forward\_call(\*args, \*\*kwargs)

NER Report:

precision recall f1-score support

Artist 1.00 1.00 1.00 25

Date 1.00 0.95 0.98 21

Departure 1.00 1.00 1.00 12

Destination 1.00 1.00 1.00 31

Food 1.00 1.00 1.00 29

Location 1.00 1.00 1.00 22

Number 1.00 1.00 1.00 29

Restaurant 1.00 1.00 1.00 30

micro avg 1.00 0.99 1.00 199

macro avg 1.00 0.99 1.00 199

weighted avg 1.00 0.99 1.00 199

**توضیح بخش 11:**

تابع evaluate\_intent **:**

این تابع برای **ارزیابی مدل تشخیص intent** طراحی شده است. در این تابع:

* مدل در حالت eval() قرار می‌گیرد تا از اجرای لایه‌هایی مانند dropout جلوگیری شود.
* داده‌های تست از طریق کلاس IntentDataset و DataLoader بارگذاری می‌شوند.
* سپس در هر batch از داده‌ها، پیش‌بینی نیت انجام می‌شود.
* از تابع argmax برای استخراج نیت پیش‌بینی‌شده از logits مدل استفاده شده است.
* نهایتاً با استفاده از تابع classification\_report از کتابخانه sklearn، یک گزارش دقیق شامل **Precision، Recall، F1-score** و **دقت کلی (Accuracy)** برای هر کلاس نیت چاپ می‌شود.

این گزارش به ما کمک می‌کند تا متوجه شویم مدل، کدام نیت‌ها را خوب یاد گرفته و در کدام دسته‌بندی‌ها ضعف دارد.

تابع evaluate\_ner

این تابع برای **ارزیابی مدل تشخیص موجودیت‌های اسمی (Named Entity Recognition)** طراحی شده است. نحوه عملکرد آن به شرح زیر است:

* مشابه قبل، ابتدا مدل در حالت ارزیابی (eval()) قرار داده می‌شود.
* داده‌ها با استفاده از کلاس NERDataset بارگذاری می‌شوند (در اینجا **batch\_size=1** است چون بررسی دقیق تگ‌ها به تفکیک کلمه نیاز دارد).
* سپس مدل بر روی هر جمله ورودی، برچسب NER را پیش‌بینی می‌کند.
* **توجه ویژه به مقدار** -100 **در برچسب‌ها شده است**. این مقدار نمایانگر کلمات padding است که نباید در ارزیابی لحاظ شوند.
* در نهایت، از تابع classification\_report کتابخانه seqeval استفاده می‌شود که مخصوص ارزیابی مدل‌های توالی (Sequence Labeling) مثل NER است و نتایج را برای هر کلاس (مانند B-LOC، I-PER، O و ...) نشان می‌دهد.

**تحلیل خروجی ها :**

عملکرد مدل در تشخیص تمام موجودیت‌ها تقریباً **بی‌نقص** است.

تنها در موجودیت Date مقدار **recall کمی پایین‌تر (0.95)** ثبت شده که نشان می‌دهد **در تعداد کمی از موارد، مدل نتوانسته همه تاریخ‌ها را تشخیص دهد**.

به طور کلی، **مقدار f1-score بالا (تقریباً 1.00 برای همه برچسب‌ها)** گواهی بر قدرت مدل در تشخیص دقیق، کامل و هماهنگ موجودیت‌ها است.

هر دو مدل **Intent و NER عملکردی بسیار قوی و دقیق** از خود نشان داده‌اند.

کیفیت آموزش مدل‌ها بالا بوده و نشان می‌دهد:

داده‌های باکیفیت و کافی برای آموزش استفاده شده‌اند.

ساختار مدل‌ها (مبتنی بر ParsBERT) برای زبان فارسی بسیار مناسب بوده‌اند.

**دقت نزدیک به 100٪** در هر دو وظیفه، این پروژه را **به‌عنوان یک دستیار هوشمند فارسی با قابلیت اعتماد بالا** مطرح می‌کند.

## **سلول 12:** پیش‌بینی نیت و موجودیت‌های نام‌دار (Intent & NER Prediction)

#سلول 12:

def predict\_intent(text):

    encoding = tokenizer(text, padding='max\_length', truncation=True,

                         max\_length=config.max\_length, return\_tensors='pt')

    input\_ids = encoding['input\_ids'].to(device)

    attention\_mask = encoding['attention\_mask'].to(device)

    intent\_model.eval()

    with torch.no\_grad():

        output = intent\_model(input\_ids=input\_ids, attention\_mask=attention\_mask)

        pred = torch.argmax(output.logits, dim=1).item()

        return intent\_id2label[pred]

def predict\_ner(text):

    tokens = text.split()

    encoding = tokenizer(tokens, is\_split\_into\_words=True,

                         padding='max\_length', truncation=True,

                         max\_length=config.max\_length, return\_tensors='pt')

    input\_ids = encoding['input\_ids'].to(device)

    attention\_mask = encoding['attention\_mask'].to(device)

    word\_ids = encoding.word\_ids()

    ner\_model.eval()

    with torch.no\_grad():

        outputs = ner\_model(input\_ids=input\_ids, attention\_mask=attention\_mask)

        predictions = torch.argmax(outputs.logits, dim=2)[0].cpu().numpy()

    final\_tags = []

    current\_word = None

    for i, word\_idx in enumerate(word\_ids):

        if word\_idx is not None and word\_idx != current\_word:

            word = tokens[word\_idx]

            tag = ner\_id2tag[predictions[i]]

            final\_tags.append(f"{word}: {tag}")

            current\_word = word\_idx

    return final\_tags

# مثال تستی

sample\_texts = [

    "تا مهرشهر میخوام تاکسی بگیرم",

    "فردا هوا در کرج چطوره",

    "از رستوران هرما دو تا پیتزا و یک پاستا میخوام",

    "آهنگ جدید بیلی آیلیش رو پخش کن"

]

for text in sample\_texts:

    intent = predict\_intent(text)

    tags = predict\_ner(text)

    print(f"\nمتن: {text}")

    print(f"Intent: {intent}")

    print("NER Tags:")

    for tag in tags:

        print(" ", tag)

**توضیح کلی سلول 12:**

**در این بخش، عملکرد مدل‌های تشخیص نیت (Intent Classification) و تشخیص موجودیت‌های نام‌دار (NER) روی جملات فارسی واقعی ارزیابی شده است تا آمادگی آن‌ها برای کاربردهای عملی سنجیده شود.**

1. **تابع predict\_intent(text):**جمله ورودی با استفاده از tokenizer به توکن تبدیل می‌شود، سپس داده‌ها به دستگاه محاسباتی (GPU) منتقل شده، مدل در حالت ارزیابی قرار می‌گیرد و خروجی مدل (logits) گرفته می‌شود. نهایتاً با انتخاب برچسب با بالاترین احتمال (argmax)، شناسه پیش‌بینی شده به برچسب فارسی متناظر تبدیل و بازگردانده می‌شود.
2. **تابع predict\_ner(text):**ابتدا جمله به کلمات شکسته می‌شود و سپس هر کلمه با توکنایزر به توکن‌هایی تقسیم می‌شود. مدل پیش‌بینی برچسب هر توکن را انجام می‌دهد، اما تنها برچسب‌های مرتبط با کلمات اصلی (نه بخش‌های داخلی توکن‌ها) استخراج شده و به برچسب‌های قابل فهم انسانی تبدیل می‌شوند. نتیجه، لیستی از جفت‌های (کلمه، برچسب) است که موجودیت‌های نام‌دار را نشان می‌دهد.

تست مدل روی نمونه های مختلف و تحلیل خروجی سلول 12:

چهار جمله‌ی متنوع و واقعی که کاربران ممکن است در تعامل با یک دستیار فارسی‌زبان استفاده کنند، برای تست انتخاب شدند. نتایج مدل به صورت زیر است:

جمله اول: تا مهرشهر میخوام تاکسی بگیرم

* **Intent** :درخواست تاکسی
* **NER :**
  + مهرشهر: B-Destination → مقصد سفر

**تفسیر**: مدل به درستی نیت درخواست تاکسی را تشخیص داده و مکان مقصد (مهرشهر) را نیز به درستی به عنوان مقصد (Destination) تشخیص داده است.

------------------------------------------------------------------------------------------------------------

**جمله دوم**: فردا هوا در کرج چطوره

* **Intent:** وضعیت آب‌وهوا
* **NER Tags :**
  + فردا: B-Date → زمان
  + کرج: B-Location → مکان موردنظر

**تفسیر**: مدل( زمان (فردا) و مکان (کرج را به درستی تشخیص داده و نیت مربوط به آب‌وهوا را هم به دقت شناسایی کرده است.

---------------------------------------------------------------------------------------------------------------------

جمله سوم: از رستوران هرما دو تا پیتزا و یک پاستا میخوام

* **Intent:** سفارش غذا
* **NER Tags** :
  + رستوران هرما: B-Restaurant, I-Restaurant → نام رستوران
  + دو تا, یک: B-Number, I-Number → تعداد غذاها
  + پیتزا, پاستا: B-Food → نوع غذا

**تفسیر**: شناسایی دقیق نام رستوران، تعداد اقلام و نوع غذاها نشان از یادگیری مناسب مدل NER دارد.

----------------------------------------------------------------------------------------

جمله چهارم: آهنگ جدید بیلی آیلیش رو پخش کن

* **Intent**: درخواست پخش آهنگ
* **NER Tags**:بیلی آیلیش: B-Artist, I-Artist → نام خواننده

**تفسیر**: مدل به درستی نام هنرمند را تشخیص داده و intent درخواست پخش موسیقی را نیز کاملاً درست شناسایی کرده است.

جمع بندی:

مدل‌های طراحی‌شده برای تشخیص intentو NER، در این تست‌های واقعی عملکردی بسیار دقیق از خود نشان داده‌اند. هم مدل intent و هم مدل NER موفق شده‌اند اطلاعات کلیدی را در متن فارسی استخراج کرده و آن را با برچسب‌های معنایی صحیح برچسب‌گذاری کنند.

این نشان می‌دهد که این مدل‌ها آمادگی بسیار بالایی برای استفاده در سیستم‌های هوشمند فارسی‌زبان مانند دستیار مجازی، بات‌های گفت‌وگویی، اپلیکیشن‌های تاکسی‌یاب یا سفارش غذا دارند.

سلول 13: گزارش عملکرد کد رابط کاربری دستیار هوشمند فارسی (Gradio Interface)

import gradio as gr

import torch

from transformers import AutoTokenizer

# فرض می‌کنیم این متغیرها در کد اصلی تعریف شده‌اند

# config, device, tokenizer, ner\_model, ner\_id2tag, intent2id, intent\_id2label

def predict\_intent(text):

    encoding = tokenizer(

        text,

        padding='max\_length',

        truncation=True,

        max\_length=config.max\_length,

        return\_tensors='pt'

    ).to(device)

    # این بخش باید با مدل واقعی intent شما جایگزین شود

    # به عنوان مثال فرضی:

    with torch.no\_grad():

        outputs = intent\_model(\*\*encoding)

        pred = torch.argmax(outputs.logits, dim=1).item()

        return intent\_id2label[pred]

def predict\_ner(text):

    intent = predict\_intent(text)

    tokens = text.split()

    SPECIAL\_RULES = {

        "و": "O", "های": "O", "رو": "O", "از": "O", "به": "O", "تا": "O", "را": "O", "ها": "O",

        "Catch\_a\_Cab": {"از": "O", "به": "O", "میخوام": "O", "بگیرم": "O"},

        "Order\_Food": {"و": "O", "سفارش": "O", "بده": "O", "میخوام": "O"},

        "Play\_Music": {"های": "O", "رو": "O", "از": "O", "پخش": "O", "کن": "O"},

        "Get\_Weather": {"هوا": "O", "چطور": "O", "چگونه": "O", "است": "O"}

    }

    encoding = tokenizer(

        tokens,

        is\_split\_into\_words=True,

        padding='max\_length',

        truncation=True,

        max\_length=config.max\_length,

        return\_tensors='pt'

    ).to(device)

    ner\_model.eval()

    with torch.no\_grad():

        outputs = ner\_model(\*\*encoding)

    predictions = torch.argmax(outputs.logits, dim=2)[0].cpu().numpy()

    word\_ids = encoding.word\_ids()

    results = []

    current\_word = None

    for i, word\_idx in enumerate(word\_ids):

        if word\_idx is not None and word\_idx != current\_word:

            word = tokens[word\_idx]

            original\_tag = ner\_id2tag[predictions[i]]

            tag = SPECIAL\_RULES.get(word, SPECIAL\_RULES.get(intent, {}).get(word, original\_tag))

            if intent == "Catch\_a\_Cab" and word in ["میدان", "بلوار", "خیابان"] and word\_idx+1 < len(tokens):

                next\_word = tokens[word\_idx+1]

                results.append(f"{word} {next\_word}: B-Destination")

                current\_word = word\_idx + 1

                continue

            results.append(f"{word}: {tag}")

            current\_word = word\_idx

    return intent, "\n".join(results)

with gr.Blocks(title="دستیار هوشمند فارسی") as demo:

    gr.Markdown("## دستیار هوشمند فارسی")

    with gr.Row():

        with gr.Column():

            input\_text = gr.Textbox(label="درخواست خود را وارد کنید",

                                 placeholder="مثال: آهنگ های ابی را پخش کن")

            with gr.Row():

                analyze\_btn = gr.Button("آنالیز کن", variant="primary")

                clear\_btn = gr.Button("پاک کردن", variant="secondary")

        with gr.Column():

            intent\_output = gr.Textbox(label="نوع درخواست")

            ner\_output = gr.Textbox(label="موجودیت‌های شناسایی شده",

                                 interactive=False)

    examples = gr.Examples(

        examples=[

            ["آهنگ جدید بیلی آیلیش رو پخش کن"],

            ["هوای فردا در تهران چطوره؟"],

            ["از میدان انقلاب به تجریش تاکسی می‌خواهم"]

        ],

        inputs=input\_text

    )

    analyze\_btn.click(

        fn=predict\_ner,

        inputs=input\_text,

        outputs=[intent\_output, ner\_output]

    )

    clear\_btn.click(

        fn=lambda: ["", "", ""],

        outputs=[input\_text, intent\_output, ner\_output],

        queue=False

    )

if \_\_name\_\_ == "\_\_main\_\_":

    demo.launch(debug=True)

توضیح کلی سلول 13:

در این سلول، یک رابط گرافیکی تحت وب با استفاده از کتابخانه Gradio طراحی شده است. این رابط به کاربر اجازه می‌دهد یک جمله به زبان فارسی وارد کند و با کلیک روی دکمه "آنالیز کن"، دو نوع تحلیل روی متن انجام شود:

• تشخیص نیت (Intent Classification): مشخص می‌کند هدف کلی جمله چیست (مثلاً سفارش غذا، گرفتن تاکسی، پخش موسیقی، پرسش درباره آب‌وهوا).

• شناسایی موجودیت‌های نامدار (NER - Named Entity Recognition): عبارات مهمی مانند مکان‌ها یا افراد را در متن مشخص می‌کند.

خروجی‌ها در دو کادر متنی جداگانه به کاربر نمایش داده می‌شود و امکان تست با مثال‌های از پیش تعیین‌شده و دکمه‌ی پاک‌سازی نیز فراهم شده است.

مفاهیم و کتابخانه‌های کلیدی استفاده شده:

:Gradio

کتابخانه‌ای برای طراحی سریع رابط‌های کاربری تحت وب برای مدل‌های یادگیری ماشین.

کلاس Blocks برای طراحی رابط ساختاریافته با بخش‌بندی.

توابع Textbox, Button, Examples و غیره برای ساخت اجزای گرافیکی.

توابع click() برای اتصال توابع پایتونی به دکمه‌ها.

:PyTorch

استفاده از مدل‌های از پیش آموزش‌دیده برای انجام پیش‌بینی.

استفاده از torch.no\_grad() برای غیرفعال کردن گرادیان‌ها در مرحله inference.

توضیح بخش‌های مهم کد:

. تابع predict\_intent(text) :

هدف این تابع پیش‌بینی "نوع درخواست" کاربر (Intent) با استفاده از مدل طبقه‌بندی متنی.

def predict\_intent(text):

    encoding = tokenizer(

        text,

        padding='max\_length',

        truncation=True,

        max\_length=config.max\_length,

        return\_tensors='pt'

    ).to(device)

Tokenization: تبدیل متن ورودی به توکن‌های قابل پردازش برای مدل BERT یا مشابه.

padding و truncation: برای یکسان‌سازی طول ورودی‌ها.

return\_tensors='pt': خروجی به صورت tensor‌های PyTorch.

with torch.no\_grad():

        outputs = intent\_model(\*\*encoding)

        pred = torch.argmax(outputs.logits, dim=1).item()

        return intent\_id2label[pred]

اجرای مدل بدون محاسبه گرادیان (افزایش سرعت و کاهش مصرف حافظه).

استفاده از argmax برای انتخاب کلاس (intent) با بالاترین احتمال.

return intent\_id2label[pred]

تبدیل عدد پیش‌بینی شده به برچسب متنی مانند Play\_Music, Get\_Weather, و غیره.

-------------------------------------------------------------------------------------------------

2. تابع predict\_ner(text): هدف در این جا استخراج موجودیت‌های نامدار (مانند مکان، زمان، شخص) از جمله ورودی.

مرحله اول – تشخیص نیت و مرحله دوم – پیش‌پردازش متن:

def predict\_ner(text):

    intent = predict\_intent(text)

    tokens = text.split()

در مرحله اول از تابع قبلی استفاده می‌شود تا نیت جمله مشخص شود. این نیت در ادامه برای اعمال قوانین خاص (SPECIAL\_RULES) استفاده خواهد شد.

در مرحله دوم تقسیم متن به کلمات با استفاده از فاصله. البته این کار ساده‌سازی شده و ممکن است در متون پیچیده‌تر جایگزین بهتری لازم باشد.

قانون‌های خاص:

   SPECIAL\_RULES = {

        "و": "O", "های": "O", "رو": "O", "از": "O", "به": "O", "تا": "O", "را": "O", "ها": "O",

        "Catch\_a\_Cab": {"از": "O", "به": "O", "میخوام": "O", "بگیرم": "O"},

        "Order\_Food": {"و": "O", "سفارش": "O", "بده": "O", "میخوام": "O"},

        "Play\_Music": {"های": "O", "رو": "O", "از": "O", "پخش": "O", "کن": "O"},

        "Get\_Weather": {"هوا": "O", "چطور": "O", "چگونه": "O", "است": "O"}

    }

کلمات پرتکرار و فاقد معنای NER خاص، به صورت دستی برچسب O می‌گیرند (یعنی Outside).برای هر Intent، کلمات خاص نادیده گرفته می‌شوند. این قوانین به کاهش نویز در خروجی مدل کمک می‌کنند.

توکنایز و پیش‌بینی مدل NER:

    encoding = tokenizer(

        tokens,

        is\_split\_into\_words=True,

        padding='max\_length',

        truncation=True,

        max\_length=config.max\_length,

        return\_tensors='pt'

    ).to(device)

    ner\_model.eval()

    with torch.no\_grad():

        outputs = ner\_model(\*\*encoding)

    predictions = torch.argmax(outputs.logits, dim=2)[0].cpu().numpy()

مدل NER روی متن اجرا می‌شود.

خروجی مدل، توالی برچسب‌هاست (مثل B-Location, I-Location, B-Person و غیره).

با argmax، برچسب نهایی هر توکن انتخاب می‌شود.

نگاشت توکن‌ها به کلمات اصلی:

word\_ids = encoding.word\_ids()

چون مدل ممکن است یک کلمه را به چند توکن بشکند، این تابع کمک می‌کند توکن‌ها را به کلمات اصلی نگاشت کنیم.

مرحله نهایی – اعمال قوانین و برچسب‌گذاری:

    for i, word\_idx in enumerate(word\_ids):

        if word\_idx is not None and word\_idx != current\_word:

            word = tokens[word\_idx]

            original\_tag = ner\_id2tag[predictions[i]]

            tag = SPECIAL\_RULES.get(word, SPECIAL\_RULES.get(intent, {}).get(word, original\_tag))

برچسب واقعی را از خروجی مدل می‌گیریم، اما اگر کلمه در SPECIAL\_RULES وجود داشت، آن برچسب استفاده می‌شود.

مثال: اگر کلمه "از" باشد، صرف‌نظر از پیش‌بینی مدل، با استفاده از قانون، برچسب "O" دریافت می‌کند.

قانون ترکیبی خاص برای مکان‌ها:

  if intent == "Catch\_a\_Cab" and word in ["میدان", "بلوار", "خیابان"] and word\_idx+1 < len(tokens):

                next\_word = tokens[word\_idx+1]

                results.append(f"{word} {next\_word}: B-Destination")

اگر هدف جمله گرفتن تاکسی باشد و واژه‌ای مانند "خیابان" در جمله باشد، کلمه‌ی بعدی آن به عنوان مقصد در نظر گرفته می‌شود.

رابط گرافیکی با Gradio:

طراحی با gr.Blocks:

with gr.Blocks(title="دستیار هوشمند فارسی") as demo:

رابط گرافیکی کلی را با ساختار بلوکی شروع می‌کنیم. Gradio از ساختار مبتنی بر بلاک برای نظم‌دهی به اجزا استفاده می‌کند.

عناصر اصلی رابط:

  input\_text = gr.Textbox(label="درخواست خود را وارد کنید",

                                 placeholder="مثال: آهنگ های ابی را پخش کن")

کادر ورودی برای دریافت جمله از کاربر.

دکمه‌ها:

analyze\_btn = gr.Button("آنالیز کن", variant="primary")

                clear\_btn = gr.Button("پاک کردن", variant="secondary")

دکمه‌ی آنالیز برای فراخوانی مدل.

دکمه‌ی پاک کردن برای ریست کردن فیلدها

خروجی:

intent\_output = gr.Textbox(label="نوع درخواست")

            ner\_output = gr.Textbox(label="موجودیت‌های شناسایی شده",

                                 interactive=False)

نمایش خروجی Intent و لیست موجودیت‌ها.

مثال‌ها:

 examples = gr.Examples(

        examples=[

            ["آهنگ جدید بیلی آیلیش رو پخش کن"],

            ["هوای فردا در تهران چطوره؟"],

            ["از میدان انقلاب به تجریش تاکسی می‌خواهم"]

        ],

        inputs=input\_text

    )

دکمه‌هایی برای پر کردن سریع کادر ورودی با مثال‌های آماده.

اتصال عملکرد دکمه‌ها:

analyze\_btn.click(

        fn=predict\_ner,

        inputs=input\_text,

        outputs=[intent\_output, ner\_output]

    )

اجرای تابع predict\_ner هنگام کلیک روی دکمه تحلیل.

  clear\_btn.click(

        fn=lambda: ["", "", ""],

        outputs=[input\_text, intent\_output, ner\_output],

        queue=False

    )

پاک کردن تمام کادرها.

اجرای برنامه:

if \_\_name\_\_ == "\_\_main\_\_":

    demo.launch(inline=True)

جمع‌بندی:

این کد، یک رابط کامل و هوشمند فارسی را با استفاده از تکنولوژی‌های پردازش زبان طبیعی و یادگیری عمیق ارائه می‌دهد. ترکیب تحلیل نیت (Intent) و شناسایی موجودیت (NER) به همراه قوانین دستی، دقت و کاربردپذیری سیستم را در محیط‌های فارسی‌زبان افزایش می‌دهد.

**5.تصاویر رابط گرافیکی و کارکرد دستیار هوشمند با داده های خارجی:**













**5)نتیجه گیری نهایی پروژه:**

در این پروژه، هدف طراحی و پیاده‌سازی یک دستیار هوشمند فارسی بود که بتواند به‌طور هم‌زمان دو وظیفه مهم پردازش زبان طبیعی را انجام دهد:

**1)تشخیص نوع درخواست کاربر (Intent Classification)**

**2)شناسایی موجودیت‌های نام‌دار در متن (Named Entity Recognition - NER)**

خلاصه‌ای از دستاوردها:

* پیاده‌سازی دو مدل مبتنی بر معماری BERT برای انجام دو وظیفه اصلی Intent و NER
* آموزش مدل‌ها با استفاده از داده‌های برچسب‌خورده و سفارشی‌سازی آنها برای زبان فارسی
* ایجاد رابط کاربری گرافیکی ساده و کارآمد با استفاده از Gradio برای تست و تعامل با مدل‌ها
* استفاده از قوانین تکمیلی (Rule-Based Rules) برای بهبود دقت سیستم در موقعیت‌هایی که مدل به تنهایی دچار خطا می‌شود
* طراحی ساختاری منعطف و قابل گسترش که امکان ارتقا در آینده را دارد

ارزیابی عملکرد و نمونه‌های تست:

با استفاده از رابط کاربری Gradio، جملات مختلفی به سیستم داده شد. مدل توانست با دقت قابل قبولی:

نوع درخواست کاربر را از بین چند دسته‌ (مثل پخش موسیقی، گرفتن تاکسی، سفارش غذا و دریافت وضعیت آب‌وهوا) تشخیص دهد.

موجودیت‌هایی مانند نام مکان‌ها، رستوران‌ها، هنرمندان، زمان، و آهنگ‌ها را در متن شناسایی و برچسب‌گذاری کند.

در کنار عملکرد موفق، برخی نقاط ضعف در تشخیص بعضی موجودیت‌ها (مثل "رستوران مهدی یار" یا "آهنگ جدید") شناسایی شد که با افزودن قوانین خاص متنی به تابع NER (در سلول ۱۳) رفع گردید. این روش ترکیبی (Hybrid) باعث شد که سیستم نسبت به ساختار جملات فارسی حساس‌تر شود.

مزیت های کلیدی پروژه:

پشتیبانی از زبان فارسی:

اکثر پروژه‌های مشابه برای زبان انگلیسی طراحی شده‌اند، اما این پروژه بومی‌سازی شده است.

مدل ترکیبی (Hybrid):

استفاده از ترکیب یادگیری عمیق و قواعد دستی برای پوشش بیشتر موارد خاص.

رابط تعاملی:

کاربر می‌تواند بدون دانش فنی، به راحتی با مدل ارتباط برقرار کند.

قابل توسعه بودن:

مدل‌ها و کدها طوری طراحی شده‌اند که بتوان به سادگی آن‌ها را آموزش مجدد داد یا به زبان‌های دیگر تعمیم داد.

پیشنهادات برای توسعه‌های آینده:

* آموزش مدل بر روی دیتاست‌های گسترده‌تر و غنی‌تر فارسی برای افزایش دقت
* پیاده‌سازی سامانه تعاملی گفتاری (با تبدیل گفتار به متن)
* افزودن قابلیت تشخیص چندزبانه یا پاسخ‌دهی هوشمند به کاربر
* استفاده از مدل‌های جدیدتر مانند ParsBERT, mBERT, XLM-RoBERTa برای دقت بیشتر در زبان فارسی
* پیاده‌سازی سیستم در محیط وب یا موبایل با استفاده از API

جمع‌بندی:

این پروژه توانست یک گام کاربردی در جهت توسعه ابزارهای هوشمند پردازش زبان طبیعی برای زبان فارسی بردارد. تلفیق مدل‌های یادگیری عمیق با قواعد نگارشی زبان باعث شد که سیستم علاوه بر دقت بالا، در مواجهه با داده‌های واقعی نیز عملکرد رضایت‌بخشی داشته باشد. به‌طور کلی، این پروژه نشان داد که با ساختار مناسب، طراحی ماژولار و بهره‌گیری از منابع موجود، می‌توان یک سیستم فارسی‌زبان با قابلیت‌های کاربردی برای تعامل انسان و ماشین طراحی کرد.

پایان...