{

"nbformat": 4,

"nbformat\_minor": 0,

"metadata": {

"colab": {

"provenance": [],

"gpuType": "T4"

},

"kernelspec": {

"name": "python3",

"display\_name": "Python 3"

},

"language\_info": {

"name": "python"

},

"accelerator": "GPU",

,

"cells": [

{

"cell\_type": "markdown",

"source": [

"Mount Google Drive"

],

"metadata": {

"id": "P1DK-GrkI3pp"

}

},

{

"cell\_type": "code",

"execution\_count": 1,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "GRR6a7XQFutZ",

"outputId": "84306a4a-7c64-451b-ed34-3a8cba1dc1b0"

},

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Mounted at /content/drive\n"

]

}

],

"source": [

"from google.colab import drive\n",

"drive.mount('/content/drive')\n"

]

},

{

"cell\_type": "markdown",

"source": [

"Unzip the Dataset"

],

"metadata": {

"id": "itwsJ09UMqDF"

}

},

{

"cell\_type": "code",

"execution\_count": 2,

"metadata": {

"id": "NV62eBgKzfZb"

},

"outputs": [],

"source": [

"import zipfile\n",

"import os\n",

"\n",

"zip\_path = \"/content/drive/MyDrive/LLM assignment 2/IMDB Dataset.csv.zip\"\n",

"extract\_path = \"/content/drive/MyDrive/LLM assignment 2\"\n",

"\n",

"os.makedirs(extract\_path, exist\_ok=True)\n",

"\n",

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

" zip\_ref.extractall(extract\_path)\n"

]

},

{

"cell\_type": "markdown",

"source": [

"Install Required Libraries\n",

"\n",

"\n"

],

"metadata": {

"id": "SkaPy-7BNIFt"

}

},

{

"cell\_type": "code",

"execution\_count": 3,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "ax-PPnvp1RD9",

"outputId": "6c400917-7143-49a9-fe02-eb9eb0f5ddd3"

},

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.53.1)\n",

"Requirement already satisfied: datasets in /usr/local/lib/python3.11/dist-packages (2.14.4)\n",

"Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)\n",

"Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)\n",

"Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)\n",

"Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (4.67.1)\n",

"Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)\n",

"Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.33.2)\n",

"Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)\n",

"Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)\n",

"Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)\n",

"Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)\n",

"Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)\n",

"Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.2)\n",

"Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)\n",

"Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)\n",

"Requirement already satisfied: dill<0.3.8,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.3.7)\n",

"Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-packages (from datasets) (3.5.0)\n",

"Requirement already satisfied: multiprocess in /usr/local/lib/python3.11/dist-packages (from datasets) (0.70.15)\n",

"Requirement already satisfied: fsspec>=2021.11.1 in /usr/local/lib/python3.11/dist-packages (from fsspec[http]>=2021.11.1->datasets) (2025.3.2)\n",

"Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from datasets) (3.11.15)\n",

"Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.3)\n",

"Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)\n",

"Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)\n",

"Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)\n",

"Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)\n",

"Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)\n",

"Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)\n",

"Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)\n",

"Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.58.5)\n",

"Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)\n",

"Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)\n",

"Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)\n",

"Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (2.6.1)\n",

"Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.4.0)\n",

"Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (25.3.0)\n",

"Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.7.0)\n",

"Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (6.6.3)\n",

"Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (0.3.2)\n",

"Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.20.1)\n",

"Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->transformers) (4.14.1)\n",

"Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->transformers) (1.1.5)\n",

"Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)\n",

"Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.2)\n",

"Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)\n",

"Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.4.0)\n",

"Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.6.15)\n"

]

}

],

"source": [

"!pip install transformers datasets scikit-learn pandas matplotlib tqdm\n"

]

},

{

"cell\_type": "markdown",

"source": [

"Import All Required Libraries"

],

"metadata": {

"id": "5eC6BqVkPGEI"

}

},

{

"cell\_type": "code",

"execution\_count": 4,

"metadata": {

"id": "yvXhJzQh1V8H"

},

"outputs": [],

"source": [

"import pandas as pd\n",

"import numpy as np\n",

"import torch\n",

"from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments\n",

"from sklearn.model\_selection import train\_test\_split\n",

"from sklearn.metrics import accuracy\_score, f1\_score\n",

"from datasets import Dataset\n",

"import matplotlib.pyplot as plt\n"

]

},

{

"cell\_type": "markdown",

"source": [

"Load and Prepare the IMDb Dataset(test,train )"

],

"metadata": {

"id": "hKTxx5DYPOzX"

}

},

{

"cell\_type": "code",

"source": [

"import pandas as pd\n",

"from sklearn.model\_selection import train\_test\_split\n",

"\n",

"df = pd.read\_csv(\"/content/drive/MyDrive/LLM assignment 2/IMDB Dataset.csv\") # Load original data\n",

"train\_df, test\_df = train\_test\_split(df, test\_size=0.2, stratify=df['sentiment'], random\_state=42) # Split data\n",

"train\_df.to\_csv(\"/content/drive/MyDrive/LLM assignment 2/train\_data.csv\", index=False)\n",

"test\_df.to\_csv(\"/content/drive/MyDrive/LLM assignment 2/test\_data.csv\", index=False)\n",

"train\_df, test\_df = pd.read\_csv(\"/content/drive/MyDrive/LLM assignment 2/train\_data.csv\"), pd.read\_csv(\"/content/drive/MyDrive/LLM assignment 2/test\_data.csv\")\n",

"print(f\"Train: {len(train\_df)}, Test: {len(test\_df)}\\n\\nTrain dist:\\n{train\_df['sentiment'].value\_counts()}\\n\\nTest dist:\\n{test\_df['sentiment'].value\_counts()}\\n\\nSamples:\\n{train\_df[['review','sentiment']].head(3)}\")\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "WlxZcOInYB\_r",

"outputId": "9327a675-d65c-4079-8c14-f06efa483168"

},

"execution\_count": 5,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Train: 40000, Test: 10000\n",

"\n",

"Train dist:\n",

"sentiment\n",

"positive 20000\n",

"negative 20000\n",

"Name: count, dtype: int64\n",

"\n",

"Test dist:\n",

"sentiment\n",

"negative 5000\n",

"positive 5000\n",

"Name: count, dtype: int64\n",

"\n",

"Samples:\n",

" review sentiment\n",

"0 I caught this little gem totally by accident b... positive\n",

"1 I can't believe that I let myself into this mo... negative\n",

"2 \*spoiler alert!\* it just gets to me the nerve ... negative\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"class"

],

"metadata": {

"id": "jWbboPZxbLKl"

}

},

{

"cell\_type": "code",

"source": [

"# Print unique class labels\n",

"print(\"Class labels:\", train\_df['sentiment'].unique())\n",

"\n",

"# Print label distribution in train and test sets\n",

"print(\"\\nTraining set label counts:\\n\", train\_df['sentiment'].value\_counts())\n",

"print(\"\\nTest set label counts:\\n\", test\_df['sentiment'].value\_counts())\n",

"class\_labels = sorted(train\_df['sentiment'].unique())\n",

"print(\"Class label list:\", class\_labels)\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "HQKyvmOtZsz5",

"outputId": "41ef2987-e201-4f2b-ccfb-fe0c6cdf0113"

},

"execution\_count": 6,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Class labels: ['positive' 'negative']\n",

"\n",

"Training set label counts:\n",

" sentiment\n",

"positive 20000\n",

"negative 20000\n",

"Name: count, dtype: int64\n",

"\n",

"Test set label counts:\n",

" sentiment\n",

"negative 5000\n",

"positive 5000\n",

"Name: count, dtype: int64\n",

"Class label list: ['negative', 'positive']\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"EDA\n"

],

"metadata": {

"id": "-QfIHKptawFH"

}

},

{

"cell\_type": "code",

"source": [

"import pandas as pd, seaborn as sns, matplotlib.pyplot as plt\n",

"\n",

"df['length'] = df['review'].str.len()\n",

"\n",

"# Bar plot with counts\n",

"plt.figure(figsize=(6,4))\n",

"ax = sns.countplot(x='sentiment', data=df, palette='Set2')\n",

"plt.title(\"Class Distribution (Full Dataset)\")\n",

"for p in ax.patches:\n",

" ax.annotate(f'{p.get\_height()}', (p.get\_x()+0.3, p.get\_height()+500))\n",

"plt.legend(title=\"Sentiment\", labels=df['sentiment'].unique())\n",

"plt.show()\n",

"\n",

"# Review length histogram\n",

"plt.figure(figsize=(6,4))\n",

"sns.histplot(df['length'], bins=30, kde=False, color='skyblue')\n",

"plt.title(\"Review Lengths (Full Dataset)\")\n",

"plt.xlabel(\"Number of Characters\")\n",

"plt.ylabel(\"Frequency\")\n",

"plt.show()\n",

"\n",

"# Print one example per class\n",

"for label in df['sentiment'].unique():\n",

" print(f\"\\n{label} sample:\\n\", df[df['sentiment']==label]['review'].iloc[0])\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 1000

},

"id": "w4TGdLIHbleX",

"outputId": "d702a573-bdfe-423b-add4-1f38a933debc"

},

"execution\_count": 7,

"outputs": [

{

"output\_type": "stream",

"name": "stderr",

"text": [

"/tmp/ipython-input-7-2603806220.py:7: FutureWarning: \n",

"\n",

"Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.\n",

"\n",

" ax = sns.countplot(x='sentiment', data=df, palette='Set2')\n"

]

},

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 600x400 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

},

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 600x400 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

},

{

"output\_type": "stream",

"name": "stdout",

"text": [

"\n",

"positive sample:\n",

" One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me.<br /><br />The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.<br /><br />It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.<br /><br />I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side.\n",

"\n",

"negative sample:\n",

" Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time.<br /><br />This movie is slower than a soap opera... and suddenly, Jake decides to become Rambo and kill the zombie.<br /><br />OK, first of all when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is watchable. Parents are divorcing & arguing like in real life. And then we have Jake with his closet which totally ruins all the film! I expected to see a BOOGEYMAN similar movie, and instead i watched a drama with some meaningless thriller spots.<br /><br />3 out of 10 just for the well playing parents & descent dialogs. As for the shots with Jake: just ignore them.\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"Preprocessing"

],

"metadata": {

"id": "B1AgViGtbRob"

}

},

{

"cell\_type": "code",

"source": [

"import re\n",

"import nltk\n",

"nltk.download('stopwords')\n",

"from nltk.corpus import stopwords\n",

"\n",

"stop\_words = set(stopwords.words('english'))\n",

"\n",

"def preprocess\_text(text):\n",

" text = text.lower() # Lower case\n",

" text = re.sub(r'<.\*?>', '', text) # Remove HTML tags\n",

" text = re.sub(r'[^a-z\\s]', '', text) # Remove punctuation & numbers\n",

" text = re.sub(r'\\s+', ' ', text).strip() # Remove extra whitespace\n",

" text = ' '.join(word for word in text.split() if word not in stop\_words) # Remove stopwords\n",

" return text\n",

"\n",

"df['clean\_review'] = df['review'].apply(preprocess\_text)\n",

"\n",

"print(df[['review', 'clean\_review']].head(3))\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "Ey61o-lIcGv5",

"outputId": "bce1bed4-da1f-4899-8ade-037cbd493212"

},

"execution\_count": 8,

"outputs": [

{

"output\_type": "stream",

"name": "stderr",

"text": [

"[nltk\_data] Downloading package stopwords to /root/nltk\_data...\n",

"[nltk\_data] Unzipping corpora/stopwords.zip.\n"

]

},

{

"output\_type": "stream",

"name": "stdout",

"text": [

" review \\\n",

"0 One of the other reviewers has mentioned that ... \n",

"1 A wonderful little production. <br /><br />The... \n",

"2 I thought this was a wonderful way to spend ti... \n",

"\n",

" clean\_review \n",

"0 one reviewers mentioned watching oz episode yo... \n",

"1 wonderful little production filming technique ... \n",

"2 thought wonderful way spend time hot summer we... \n"

]

}

]

},

{

"cell\_type": "code",

"source": [

"\n",

"# 1. Preprocess train and test reviews\n",

"train\_df['clean\_review'] = train\_df['review'].apply(preprocess\_text)\n",

"test\_df['clean\_review'] = test\_df['review'].apply(preprocess\_text)\n",

"\n",

"# 2. Extract cleaned texts as lists\n",

"train\_texts = train\_df['clean\_review'].tolist()\n",

"val\_texts = test\_df['clean\_review'].tolist()\n"

],

"metadata": {

"id": "MpSrN4dQd5dc"

},

"execution\_count": 9,

"outputs": []

},

{

"cell\_type": "markdown",

"source": [

"Tokenization with BERT"

],

"metadata": {

"id": "BxNBPRRteRIU"

}

},

{

"cell\_type": "code",

"source": [

"from transformers import BertTokenizer\n",

"\n",

"tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')\n",

"def tokenize\_function(texts):\n",

" return tokenizer(texts, padding=\"max\_length\", truncation=True, max\_length=256)\n",

"\n",

"train\_encodings = tokenize\_function(train\_texts)\n",

"val\_encodings = tokenize\_function(val\_texts)\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 272,

"referenced\_widgets": [

"d5c0880b77804ed9a66f3f9e7be60d94",

"20e8a4d6cb134f409f780b0e2f4e5525",

"66c9ea5ab9e7484db9d9fa91e151694f",

"01e8aeb9d16b452c9d26c471318a15eb",

"91ec85970c8f4e989dda0845ec7b8275",

"592c579da11746318a850768fe0213b8",

"38e9610a0691416a84d9de14a12af7f5",

"a9113df88e7f4500a8d336bd38a5062c",

"071d09ab23a1430a9d01cc42a491f7e8",

"21c091b2eb31493e875e6b2680246ffe",

"20cb5f55100b471ebd4b913ed2d399f4",

"90640224e6104123add42a24e6631ca3",

"4abe3945e7714ad6a5138c4156c1f23a",

"df56bb7bbd424f2cb7afd07565a225aa",

"dfe19ce8497d40f9b5360b1384d0e3b6",

"7faad8b61f0549d6ba365fa8e62deec3",

"36c82465d74e490c9eb4837d55de5ce5",

"621722180acf4129afac57b11cabc519",

"b6f154bb03e14318bb93bf1b7277330c",

"cb4e20bcecf946afa5b344386aadb08a",

"ea3b640fda9e40e4a501373154a983c2",

"9cb0bc0f6acf41bbb3bb78074ffab0d8",

"e734413b98bc4835bc9b6136be0648cc",

"16e900b72d244dbd8e4cc628bbd7f6b5",

"9d3cab148a9245fcb3a0ccd3895d8cbe",

"dd7c450f03444e339f2aeb40250b2daf",

"81eb7ba3f5a645518a0bac0f82bab38f",

"d8147ead64ac40a1b2cc48b29c996f17",

"69d2b6323afb41c5b792419f7875da85",

"df5837a5d7d44e68a62dfbb43fb74a63",

"d222f4dccef04f7c8e025274cedb71d9",

"215d1a89ae4d4904a052b5ea71f072c0",

"90362bc6983d4bb3b5303628bc800e88",

"2068db549da04ce7aeb21174ce5e34a4",

"5a646fb35b334749895c90ac702ab697",

"127d44f53ce7403da9d8ead1b5219608",

"5d3ade9d6b664c6ba144d8928db12797",

"b3c4db81eea44982bcf7ed55d05b610a",

"2c7210afa5d0413c9a534c98387669a9",

"fd676571ce514ac8893c9950dfccd578",

"a75d5c05c75c4eae8bf41136f041622a",

"c5cacf1be5f34ac984ffed981ace5b27",

"d91f7eab1fcf47e48e1e062a9b57785f",

"bedce2c1af714598b86eb87de56bd14e"

]

},

"id": "sGBLdsqCdlqr",

"outputId": "914c78ac-ed0a-4de6-c31d-39c34f88d126"

},

"execution\_count": 10,

"outputs": [

{

"output\_type": "stream",

"name": "stderr",

"text": [

"/usr/local/lib/python3.11/dist-packages/huggingface\_hub/utils/\_auth.py:94: UserWarning: \n",

"The secret `HF\_TOKEN` does not exist in your Colab secrets.\n",

"To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.\n",

"You will be able to reuse this secret in all of your notebooks.\n",

"Please note that authentication is recommended but still optional to access public models or datasets.\n",

" warnings.warn(\n"

]

},

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"tokenizer\_config.json: 0%| | 0.00/48.0 [00:00<?, ?B/s]"

],

"application/vnd.jupyter.widget-view+json": {

"version\_major": 2,

"version\_minor": 0,

"model\_id": "d5c0880b77804ed9a66f3f9e7be60d94"

}

},

"metadata": {}

},

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"vocab.txt: 0%| | 0.00/232k [00:00<?, ?B/s]"

],

"application/vnd.jupyter.widget-view+json": {

"version\_major": 2,

"version\_minor": 0,

"model\_id": "90640224e6104123add42a24e6631ca3"

}

},

"metadata": {}

},

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"tokenizer.json: 0%| | 0.00/466k [00:00<?, ?B/s]"

],

"application/vnd.jupyter.widget-view+json": {

"version\_major": 2,

"version\_minor": 0,

"model\_id": "e734413b98bc4835bc9b6136be0648cc"

}

},

"metadata": {}

},

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"config.json: 0%| | 0.00/570 [00:00<?, ?B/s]"

],

"application/vnd.jupyter.widget-view+json": {

"version\_major": 2,

"version\_minor": 0,

"model\_id": "2068db549da04ce7aeb21174ce5e34a4"

}

},

"metadata": {}

}

]

},

{

"cell\_type": "markdown",

"source": [

"Encode Labels"

],

"metadata": {

"id": "Z3fJ\_Nj6eclr"

}

},

{

"cell\_type": "code",

"source": [

"label\_map = {'negative': 0, 'positive': 1}\n",

"\n",

"train\_labels = train\_df['sentiment'].map(label\_map).tolist()\n",

"val\_labels = test\_df['sentiment'].map(label\_map).tolist()\n"

],

"metadata": {

"id": "tdWXrXUVec3E"

},

"execution\_count": 11,

"outputs": []

},

{

"cell\_type": "markdown",

"source": [

"Create Hugging Face Dataset objects"

],

"metadata": {

"id": "3UZYT2rGekIO"

}

},

{

"cell\_type": "code",

"source": [

"from datasets import Dataset\n",

"\n",

"train\_dataset = Dataset.from\_dict({\n",

" 'input\_ids': train\_encodings['input\_ids'],\n",

" 'attention\_mask': train\_encodings['attention\_mask'],\n",

" 'labels': train\_labels\n",

"})\n",

"\n",

"val\_dataset = Dataset.from\_dict({\n",

" 'input\_ids': val\_encodings['input\_ids'],\n",

" 'attention\_mask': val\_encodings['attention\_mask'],\n",

" 'labels': val\_labels\n",

"})\n"

],

"metadata": {

"id": "O-ee-VQyek0b"

},

"execution\_count": 12,

"outputs": []

},

{

"cell\_type": "markdown",

"source": [

"Load BERT Model for Fine-Tuning"

],

"metadata": {

"id": "FPqW8EnDhCQe"

}

},

{

"cell\_type": "code",

"source": [

"from transformers import BertForSequenceClassification\n",

"\n",

"# Load BERT base model with a classification head (2 classes: positive, negative)\n",

"model = BertForSequenceClassification.from\_pretrained('bert-base-uncased', num\_labels=2)\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 105,

"referenced\_widgets": [

"3dab77f382a84cd1bfe382a3d1cfac88",

"7f6b2827fede468c884ccbe04ab220e3",

"63adaf995f994837876b940a4a780206",

"393e273f27864d7da5d4a0ce0a84bdaf",

"720e8083a8084c4fa22bf5d999e1f404",

"5af5d43331214484b0c91d96f18b6409",

"8f941a9873064e488c5108156b5f7189",

"08b1f720830449f4905934800edfecab",

"290ee96b7c414063aa731338e1dd160f",

"09eb4ab9535a4f68989a8e6c0bac2b6f",

"6861748777294114948f65493c8edfdb"

]

},

"id": "K69KI3nJhCnh",

"outputId": "004f32dd-9086-4ad4-9d3f-b27d7282c547"

},

"execution\_count": 13,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"model.safetensors: 0%| | 0.00/440M [00:00<?, ?B/s]"

],

"application/vnd.jupyter.widget-view+json": {

"version\_major": 2,

"version\_minor": 0,

"model\_id": "3dab77f382a84cd1bfe382a3d1cfac88"

}

},

"metadata": {}

},

{

"output\_type": "stream",

"name": "stderr",

"text": [

"Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']\n",

"You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"Training Configuration"

],

"metadata": {

"id": "oNEv3vGZhgqP"

}

},

{

"cell\_type": "code",

"source": [

"from transformers import Trainer, TrainingArguments\n",

"from sklearn.metrics import accuracy\_score, f1\_score\n",

"import numpy as np\n",

"\n",

"def compute\_metrics(eval\_pred):\n",

" logits, labels = eval\_pred\n",

" predictions = np.argmax(logits, axis=-1)\n",

" acc = accuracy\_score(labels, predictions)\n",

" f1 = f1\_score(labels, predictions)\n",

" return {'accuracy': acc, 'f1': f1}\n",

"\n",

"training\_args = TrainingArguments(\n",

" output\_dir='./results',\n",

" num\_train\_epochs=3,\n",

" per\_device\_train\_batch\_size=16,\n",

" per\_device\_eval\_batch\_size=32,\n",

" eval\_strategy='epoch',\n",

" save\_strategy='epoch',\n",

" logging\_dir='./logs',\n",

" logging\_steps=100,\n",

" load\_best\_model\_at\_end=True,\n",

" metric\_for\_best\_model='f1',\n",

" greater\_is\_better=True,\n",

" seed=42,\n",

" fp16=True,\n",

" report\_to=[]\n",

")\n",

"\n",

"trainer = Trainer(\n",

" model=model,\n",

" args=training\_args,\n",

" train\_dataset=train\_dataset,\n",

" eval\_dataset=val\_dataset,\n",

" compute\_metrics=compute\_metrics,\n",

")\n",

"\n",

"trainer.train()\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 241

},

"id": "1GQVBFvliLrm",

"outputId": "db993751-a999-4bcf-cf68-96ec8f22f460"

},

"execution\_count": 16,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<IPython.core.display.HTML object>"

],

"text/html": [

"\n",

" <div>\n",

" \n",

" <progress value='7500' max='7500' style='width:300px; height:20px; vertical-align: middle;'></progress>\n",

" [7500/7500 28:46, Epoch 3/3]\n",

" </div>\n",

" <table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: left;\">\n",

" <th>Epoch</th>\n",

" <th>Training Loss</th>\n",

" <th>Validation Loss</th>\n",

" <th>Accuracy</th>\n",

" <th>F1</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <td>1</td>\n",

" <td>0.286100</td>\n",

" <td>0.261122</td>\n",

" <td>0.894500</td>\n",

" <td>0.900519</td>\n",

" </tr>\n",

" <tr>\n",

" <td>2</td>\n",

" <td>0.184000</td>\n",

" <td>0.303050</td>\n",

" <td>0.916500</td>\n",

" <td>0.915682</td>\n",

" </tr>\n",

" <tr>\n",

" <td>3</td>\n",

" <td>0.090600</td>\n",

" <td>0.396338</td>\n",

" <td>0.915900</td>\n",

" <td>0.915638</td>\n",

" </tr>\n",

" </tbody>\n",

"</table><p>"

]

},

"metadata": {}

},

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"TrainOutput(global\_step=7500, training\_loss=0.18953113848368328, metrics={'train\_runtime': 1728.5391, 'train\_samples\_per\_second': 69.423, 'train\_steps\_per\_second': 4.339, 'total\_flos': 1.57866633216e+16, 'train\_loss': 0.18953113848368328, 'epoch': 3.0})"

]

},

"metadata": {},

"execution\_count": 16

}

]

},

{

"cell\_type": "markdown",

"source": [],

"metadata": {

"id": "Zml3IipWpiOF"

}

},

{

"cell\_type": "code",

"source": [

"eval\_results = trainer.evaluate()\n",

"print(\"Evaluation results:\", eval\_results)\n",

"\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 75

},

"id": "Gus2lW5hpif0",

"outputId": "bb7f8489-06aa-47f1-d543-15a2e09b40b3"

},

"execution\_count": 17,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<IPython.core.display.HTML object>"

],

"text/html": [

"\n",

" <div>\n",

" \n",

" <progress value='313' max='313' style='width:300px; height:20px; vertical-align: middle;'></progress>\n",

" [313/313 00:37]\n",

" </div>\n",

" "

]

},

"metadata": {}

},

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Evaluation results: {'eval\_loss': 0.30304983258247375, 'eval\_accuracy': 0.9165, 'eval\_f1': 0.9156821165303444, 'eval\_runtime': 38.0592, 'eval\_samples\_per\_second': 262.749, 'eval\_steps\_per\_second': 8.224, 'epoch': 3.0}\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"prediction for test datasets"

],

"metadata": {

"id": "IqoedQqesITZ"

}

},

{

"cell\_type": "code",

"source": [

"test\_texts = test\_df['clean\_review'].tolist()\n",

"test\_encodings = tokenizer(test\_texts, padding=\"max\_length\", truncation=True, max\_length=256)\n",

"test\_labels = test\_df['sentiment'].map(label\_map).tolist()\n",

"from datasets import Dataset\n",

"\n",

"test\_dataset = Dataset.from\_dict({\n",

" 'input\_ids': test\_encodings['input\_ids'],\n",

" 'attention\_mask': test\_encodings['attention\_mask'],\n",

" 'labels': test\_labels\n",

"})\n",

"predictions\_output = trainer.predict(test\_dataset)\n",

"pred\_logits = predictions\_output.predictions\n",

"pred\_labels = np.argmax(pred\_logits, axis=1)\n",

"\n",

"# Optionally, print accuracy and F1 on test set\n",

"from sklearn.metrics import accuracy\_score, f1\_score\n",

"\n",

"acc = accuracy\_score(test\_labels, pred\_labels)\n",

"f1 = f1\_score(test\_labels, pred\_labels)\n",

"print(f\"Test Accuracy: {acc:.4f}\")\n",

"print(f\"Test F1 Score: {f1:.4f}\")\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 53

},

"id": "hx9afT67r35a",

"outputId": "834522b3-cc81-45dd-de87-18f488f342b3"

},

"execution\_count": 25,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<IPython.core.display.HTML object>"

],

"text/html": []

},

"metadata": {}

},

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Test Accuracy: 0.9165\n",

"Test F1 Score: 0.9157\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"prediction for validation\n"

],

"metadata": {

"id": "ouewVhUMsClw"

}

},

{

"cell\_type": "code",

"source": [

"\n",

"# Predict\n",

"predictions\_output = trainer.predict(val\_dataset)\n",

"logits = predictions\_output.predictions\n",

"pred\_labels = np.argmax(logits, axis=-1)\n",

"true\_labels = predictions\_output.label\_ids\n",

"\n",

"\n",

"print(classification\_report(true\_labels, pred\_labels, target\_names=['negative', 'positive']))\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 178

},

"id": "CZldMlRtqXGG",

"outputId": "79ee16d3-f47f-4bfb-a79f-38af86793845"

},

"execution\_count": 21,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<IPython.core.display.HTML object>"

],

"text/html": []

},

"metadata": {}

},

{

"output\_type": "stream",

"name": "stdout",

"text": [

" precision recall f1-score support\n",

"\n",

" negative 0.91 0.93 0.92 5000\n",

" positive 0.92 0.91 0.92 5000\n",

"\n",

" accuracy 0.92 10000\n",

" macro avg 0.92 0.92 0.92 10000\n",

"weighted avg 0.92 0.92 0.92 10000\n",

"\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"metrics visuals for test\n"

],

"metadata": {

"id": "Ehj1u0LirWvV"

}

},

{

"cell\_type": "code",

"source": [

"# 15. Visualize results\n",

"metrics = {'Accuracy': acc, 'F1 Score': f1}\n",

"plt.bar(metrics.keys(), metrics.values(), color=['skyblue', 'lightgreen'])\n",

"plt.title('Test Set Performance Metrics')\n",

"plt.ylim(0, 1)\n",

"plt.show()"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 452

},

"id": "afbpsmIasuPF",

"outputId": "49936de0-e728-4ddf-ad2e-75e20d1be3ab"

},

"execution\_count": 26,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

}

]

}