Install camel-tools

```
#!pip install camel-tools
```

We'll download both the model weights and the tokenizer files from Hugging Face.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

model_name = "Ammar-alhaj-ali/arabic-MARBERT-dialect-identification-city"
save_directory = "/content/marbert_dialect_id"

# Download model + tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# Save locally in Colab
tokenizer.save_pretrained(save_directory)
model.save_pretrained(save_directory)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tc
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

tokenizer_config.json: 100% 371/371 [00:00<00:00, 13.3kB/s]

vocab.txt: 100% 1.10M/1.10M [00:00<00:00, 2.80MB/s]

tokenizer.json: 100% 2.69M/2.69M [00:00<00:00, 15.6MB/s]

special_tokens_map.json: 100% 125/125 [00:00<00:00, 3.93kB/s]

config.json: 100% 1.92k/1.92k [00:00<00:00, 79.2kB/s]

pytorch_model.bin: 100% 652M/652M [00:05<00:00, 170MB/s]

Upload the MARDUS CORPUS and unzip the dataset

```
from google.colab import files

# Upload ZIP file manually
uploaded = files.upload()
```

Choose Files MADAR.zip

- **MADAR.zip**(application/x-zip-compressed) - 2873502 bytes, last modified: 3/15/2025 - 100% done

model.safetensors: 100% 651M/651M [00:03<00:00, 223MB/s]

```
Saving MADAR.zip to MADAR.zip
```

```
!mkdir -p /content/MADAR
!unzip MADAR.zip -d /content/MADAR/
!ls /content/MADAR/
```

```
Archive:  MADAR.zip
   creating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/
  inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/The_MADAR_Arabic_Dialect_Corpus_and
  inflating: /content/MADAR/__MACOSX/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/._The_MADAR_Arabic_Dialect
  inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/README.txt
  inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/LICENSE.txt
   creating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/
  inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/Icon
```

```
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Rabat.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.English.i
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Damascus.
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Beirut.ts
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Doha.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Jeddah.ts
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Sfax.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.French.in
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Basra.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Mosul.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Fes.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Baghdad.t
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Algiers.t
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Alexandri
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Jerusalem
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Tripoli.t
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Riyadh.ts
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.MSA.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Sanaa.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Aleppo.ts
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Muscat.ts
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Cairo.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Amman.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Aswan.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Tunis.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Benghazi.
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Salt.tsv
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Khartoum.
    inflating: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/Icon
__MACOSX  MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021
```

```
!head -n 5 /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Muscat.tsv
```

```
sentID.BTEC     split   lang    sent
5       corpus-6-test-corpus-26-train   MUS     اهو هناك، بالضبط جدام امام مكتب المعلومات السياحية.
9       corpus-6-test-corpus-26-train   MUS     انا ماسمعت ابد بالهعنوان.
11      corpus-6-test-corpus-26-train   MUS     روح سيده حتى تشوف الصيدلية.
26      corpus-6-test-corpus-26-train   MUS     بجم سعر الريوق؟
```

```
import pandas as pd

Tripoli_path = "/content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Tripoli.tsv'

df_tripoli = pd.read_csv(Tripoli_path, sep='\t', header=0)

texts = df_tripoli["sent"].tolist()

## DO NOT COMMENT OUT THE LINE BELOW WHEN CHANGING TO TRIPOLI DIALECT ##
sample_texts = texts[:100]

# Make sure the id lable matches the tsv file we use. e.g 18 able is for Tripoli dialect
labels = [18] * len(sample_texts)
```

Evaluate the model before fine tuning

Evaluation with a test dataset from MADAR

```
import torch
from transformers import AutoTokenizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import torch.nn.functional as F

# Make sure you're on GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Tokenize inputs
```

```
tokenized = tokenizer(
    sample_texts,
    truncation=True,
    padding=True,
    max_length=128,
    return_tensors="pt"
)

# Move tokenized tensors to the same device as the model
input_ids = tokenized["input_ids"].to(device)
attention_mask = tokenized["attention_mask"].to(device)

# Disable grad since this is inference
with torch.no_grad():
    outputs = model(input_ids=input_ids, attention_mask=attention_mask)
    probs = F.softmax(outputs.logits, dim=1)
    preds = torch.argmax(probs, dim=1).cpu().numpy()

# Ground truth
true_labels = labels

# Evaluate
accuracy = accuracy_score(true_labels, preds)
precision = precision_score(true_labels, preds, average='weighted', zero_division=0)
recall = recall_score(true_labels, preds, average='weighted', zero_division=0)
f1 = f1_score(true_labels, preds, average='weighted', zero_division=0)

# Print
print(f"🔍 Accuracy: {accuracy * 100:.2f}%")
print(f"🎯 Precision: {precision * 100:.2f}%")
print(f"📈 Recall: {recall * 100:.2f}%")
print(f"💥 F1 Score: {f1 * 100:.2f}%")
```

```
🔍 Accuracy: 80.00%
🎯 Precision: 100.00%
📈 Recall: 80.00%
💥 F1 Score: 88.89%
```

From the results above we see for a dialect like Sfax we get an accuracy of 80% in which the model predict teh dialect correctly. This is also reflected on the precision of 100% which means it does not misclassify the dialect

Model Improvement

Analyze Class Distribution in MADAR Corpus

```
import os
import pandas as pd
from collections import defaultdict

# Path to your MADAR corpus folder
madar_dir = "/content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus"

# Initialize dictionary to hold sample counts
dialect_counts = defaultdict(int)

# Loop through all .tsv files
for filename in os.listdir(madar_dir):
    if filename.endswith(".tsv") and "MADAR.corpus." in filename:
        # Extract city name between 'MADAR.corpus.' and '.tsv'
        dialect = filename.replace("MADAR.corpus.", "").replace(".tsv", "")
        filepath = os.path.join(madar_dir, filename)

        # Load file and count number of rows (excluding header)
        df = pd.read_csv(filepath, sep="\t")
        dialect_counts[dialect] = len(df)
```

```
# Convert to DataFrame for easy viewing
dialect_distribution = pd.DataFrame.from_dict(dialect_counts, orient='index', columns=['Sample Count'])
dialect_distribution = dialect_distribution.sort_values(by='Sample Count', ascending=False)

# Display result
print(dialect_distribution)
```

```
               Sample Count
Beirut                12000
MSA                   12000
Cairo                 12000
Rabat                 12000
Tunis                 12000
French.index          12000
English.index         12000
Doha                  12000
Khartoum               2000
Amman                  2000
Riyadh                 2000
Aswan                  2000
Sanaa                  2000
Salt                   2000
Alexandria             2000
Jerusalem              2000
Muscat                 2000
Aleppo                 2000
Tripoli                2000
Jeddah                 2000
Algiers                2000
Damascus               2000
Fes                    2000
Mosul                  2000
Basra                  2000
Sfax                   2000
Baghdad                2000
Benghazi               2000
```

Analysis of Dialect Performance in the Model

1 High-Performing Dialects

Dialects with **high performance** in the model include:

- **Cairo**
- **Tunis**
- **Rabat**
- **Beirut**
- **Doha**
- **Modern Standard Arabic (MSA)**

◆ Key Insight

Each of these dialects has **12,000 samples**, which is **6x more data** than lower-performing dialects.

---

2 Low-Performing Dialects

Dialects where the model **underperforms** include:

- **Aleppo**
- **Baghdad**
- **Sanaa**
- **Tripoli**
- **Basra**

◆ Possible Explanation

Each of these dialects has only **2,000 samples**, meaning:

✅ The model **never saw enough examples** during training.

✅ **Generalization** is weak, leading to poor performance on these dialects.

🛠 Next Steps for Improvement

1. **Increase Training Data** for low-performing dialects to **match high-performing ones**.
2. **Use Data Augmentation** to synthetically generate more samples.
3. **Implement Transfer Learning** by fine-tuning on underrepresented dialects.
4. **Adjust Model Weights** to prevent the model from favoring overrepresented dialects.

🚀 **By addressing data imbalance, we can improve the model's accuracy across all dialects!**

Model Improvement

```
import pandas as pd
import os
from glob import glob
from sklearn.utils import resample

# Path to the MADAR corpus files
data_dir = "/content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus"

# List all .tsv files
tsv_files = glob(os.path.join(data_dir, "*.tsv"))

# Load all dialects into a list
all_dfs = []
dialect_counts = {}

print("Loading and counting samples per dialect...")

for file_path in tsv_files:
    df = pd.read_csv(file_path, sep='\t')

    if 'lang' not in df.columns or 'sent' not in df.columns:
        print(f"Skipping malformed file: {file_path}")
        continue

    dialect = df['lang'].iloc[0]
    all_dfs.append(df)
    dialect_counts[dialect] = dialect_counts.get(dialect, 0) + len(df)

# Combine everything
full_df = pd.concat(all_dfs, ignore_index=True)
```

```
Loading and counting samples per dialect...
Skipping malformed file: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corp
Skipping malformed file: /content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corp
```

```
# Set target sample count per class
TARGET_SIZE = 12000

# Upsample minority dialects
balanced_dfs = []
print("🛠 Upsampling underrepresented dialects...")

for dialect, group in full_df.groupby('lang'):
    if len(group) < TARGET_SIZE:
        upsampled = resample(group, replace=True, n_samples=TARGET_SIZE, random_state=42)
        print(f"🔼 Upsampled {dialect} from {len(group)} ➡ {len(upsampled)}")
        balanced_dfs.append(upsampled)
    elif len(group) > TARGET_SIZE:
        downsampled = group.sample(n=TARGET_SIZE, random_state=42)
        print(f"🔽 Downsampled {dialect} from {len(group)} ➡ {len(downsampled)}")
        balanced_dfs.append(downsampled)
```

```
    else:
        balanced_dfs.append(group)
```

```
🛠 Upsampling underrepresented dialects...
🔼 Upsampled ALE from 2000 ➡ 12000
🔼 Upsampled ALG from 2000 ➡ 12000
🔼 Upsampled ALX from 2000 ➡ 12000
🔼 Upsampled AMM from 2000 ➡ 12000
🔼 Upsampled ASW from 2000 ➡ 12000
🔼 Upsampled BAG from 2000 ➡ 12000
🔼 Upsampled BAS from 2000 ➡ 12000
🔼 Upsampled BEN from 2000 ➡ 12000
🔼 Upsampled DAM from 2000 ➡ 12000
🔼 Upsampled FES from 2000 ➡ 12000
🔼 Upsampled JED from 2000 ➡ 12000
🔼 Upsampled JER from 2000 ➡ 12000
🔼 Upsampled KHA from 2000 ➡ 12000
🔼 Upsampled MOS from 2000 ➡ 12000
🔼 Upsampled MUS from 2000 ➡ 12000
🔼 Upsampled RIY from 2000 ➡ 12000
🔼 Upsampled SAL from 2000 ➡ 12000
🔼 Upsampled SAN from 2000 ➡ 12000
🔼 Upsampled SFX from 2000 ➡ 12000
🔼 Upsampled TRI from 2000 ➡ 12000
```

We'll balance the dataset by oversampling the dialects with only 2,000 samples so they match the dominant dialects at 12,000.

*** This avoids losing good data from the strong dialects

*** Ensures the model gets equal exposure to all dialects

```
# Final balanced dataset
balanced_df = pd.concat(balanced_dfs, ignore_index=True)

print("✅ Final balanced dataset created!")
print(balanced_df['lang'].value_counts())

# Save to CSV or TSV for training
balanced_df.to_csv("/content/madar_balanced_dialects.tsv", sep="\t", index=False)
print("Saved to /content/madar_balanced_dialects.tsv")
```

```
✅ Final balanced dataset created!
lang
ALE    12000
ALG    12000
ALX    12000
AMM    12000
ASW    12000
BAG    12000
BAS    12000
BEI    12000
BEN    12000
CAI    12000
DAM    12000
DOH    12000
FES    12000
JED    12000
JER    12000
KHA    12000
MOS    12000
MSA    12000
MUS    12000
RAB    12000
RIY    12000
SAL    12000
SAN    12000
SFX    12000
TRI    12000
TUN    12000
Name: count, dtype: int64
Saved to /content/madar_balanced_dialects.tsv
```

As seen above the classesa are all 12000 in sample count which means we have eliminated the class imbalance issue

Filter out under perfoming weak Dialects only to use for re training the model

```
import pandas as pd

# Load full balanced dataset
balanced_df = pd.read_csv("/content/madar_balanced_dialects.tsv", sep="\t")

# 3-letter codes of strong-performing dialects
high_accuracy_dialect_codes = [
    "CAI",  # Cairo
    "ALG",  # Algiers
    "BEI",  # Beirut
    "DOH",  # Doha
    "MSA",  # Modern Standard Arabic
    "RAB",  # Rabat
    "TUN"   # Tunis
]

# Now filter out those strong dialects
weak_df = balanced_df[~balanced_df['lang'].isin(high_accuracy_dialect_codes)]

# Confirm the filtering worked
print("✅ Final Weak Dialects Dataset:")
print(weak_df['lang'].value_counts())

# Save to new file
weak_df.to_csv("/content/madar_weak_dialects.tsv", sep="\t", index=False)
print("💾 Saved to /content/madar_weak_dialects.tsv")
```

```
✅ Final Weak Dialects Dataset:
lang
ALE    12000
ALX    12000
AMM    12000
ASW    12000
BAG    12000
BAS    12000
BEN    12000
DAM    12000
FES    12000
JED    12000
JER    12000
KHA    12000
MOS    12000
MUS    12000
RIY    12000
SAL    12000
SAN    12000
SFX    12000
TRI    12000
Name: count, dtype: int64
💾 Saved to /content/madar_weak_dialects.tsv
```

Filtering High-Performing Dialects for Model Retraining

✅ High-Performing Dialects (Accuracy > 95%)

We **exclude** these dialects from retraining as they already achieve **high accuracy**:

- **CAI** → Cairo
- **ALG** → Algiers
- **BEI** → Beirut

- **DOH** → Doha
- **MSA** → Modern Standard Arabic
- **RAB** → Rabat
- **TUN** → Tunis

Focusing on Low-Performing Dialects

To **improve model accuracy**, we filter out the **high-performing dialects** and retain only the **underperforming ones**, such as:

- **SFX → Sfax**
- **TRI → Tripoli**

Why This Matters?

By **retraining the model** on these low-performing dialects, we:
✅ Improve generalization for underrepresented dialects.
✅ Balance the dataset, preventing bias toward high-resource dialects.
✅ Enhance overall model performance across **all dialects**.

🚀 **Retraining on these dialects will significantly improve accuracy and robustness!**

Model Fine tuning

```
!pip install Datasets
```

```
Collecting Datasets
  Downloading datasets-3.4.1-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from Datasets) (3.17.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from Datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from Datasets) (18.1.0)
Collecting dill<0.3.9,>=0.3.0 (from Datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from Datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from Datasets) (2.32.3
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from Datasets) (4.67.1)
Collecting xxhash (from Datasets)
  Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess<0.70.17 (from Datasets)
  Downloading multiprocess-0.70.16-py311-none-any.whl.metadata (7.2 kB)
Requirement already satisfied: fsspec<=2024.12.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-packages (from fsspe
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from Datasets) (3.11.13)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from Datasets)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from Datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from Datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->D
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->Datasets
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->Datasets) (
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->Dataset
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->Datas
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->Datasets
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->Dataset
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from hugging
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->Dat
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->Dat
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->Datasets) (20
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->Datasets) (
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->p
Downloading datasets-3.4.1-py3-none-any.whl (487 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 487.4/487.4 kB 9.1 MB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 7.8 MB/s eta 0:00:00
Downloading multiprocess-0.70.16-py311-none-any.whl (143 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 10.1 MB/s eta 0:00:00
Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 13.5 MB/s eta 0:00:00
Installing collected packages: xxhash, dill, multiprocess, Datasets
Successfully installed Datasets-3.4.1 dill-0.3.8 multiprocess-0.70.16 xxhash-3.5.0
```

We first tokenize the balanced dataset

```
import pandas as pd
from transformers import AutoTokenizer
from sklearn.model_selection import train_test_split
from datasets import Dataset

# Load the balanced dataset
data_path = "/content/madar_weak_dialects.tsv"
df = pd.read_csv(data_path, sep="\t")

# Rename columns to match HuggingFace format
df = df.rename(columns={"sent": "text", "lang": "label"})

# Convert dialect labels to numeric ids
label2id = {label: idx for idx, label in enumerate(sorted(df['label'].unique()))}
id2label = {v: k for k, v in label2id.items()}
df["label"] = df["label"].map(label2id)

# Train/validation split
train_df, val_df = train_test_split(df, test_size=0.1, stratify=df["label"], random_state=42)

# Convert to HuggingFace Datasets
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)

# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained("/content/marbert_dialect_id")

# Tokenization function
def tokenize(batch):
    return tokenizer(batch["text"], padding="max_length", truncation=True, max_length=128)

# Tokenize datasets
train_dataset = train_dataset.map(tokenize, batched=True)
eval_dataset = val_dataset.map(tokenize, batched=True)

# Set format for PyTorch
train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
eval_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
```

```
Map: 100%    205200/205200 [00:47<00:00, 3003.05 examples/s]
Map: 100%    22800/22800 [00:05<00:00, 4768.70 examples/s]
```

Tune hyper parameters

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="/content/marbert_dialect_id_finetuned",
    evaluation_strategy="no",
    save_strategy="no",
    learning_rate=3e-5,
    gradient_accumulation_steps=2,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
    weight_decay=0.01,
    warmup_ratio=0.1,
    logging_dir="./logs",
    label_smoothing_factor=0.1,
    logging_steps=200,
    load_best_model_at_end=True,
```

```
    metric_for_best_model="accuracy",
    save_total_limit=1,
)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` i
  warnings.warn(
```

Re-Train the Model

```
from transformers import BertForSequenceClassification, Trainer, TrainingArguments, BertTokenizerFast
from datasets import Dataset
import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import torch

# Load model and tokenizer
model_path = "/content/marbert_dialect_id"
model = BertForSequenceClassification.from_pretrained(model_path)
tokenizer = BertTokenizerFast.from_pretrained(model_path)



# Define metrics
def compute_metrics(pred):
    labels = pred.label_ids
    preds = np.argmax(pred.predictions, axis=1)
    acc = accuracy_score(labels, preds)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='macro')
    return {"accuracy": acc, "precision": precision, "recall": recall, "f1": f1}


# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=None,
    tokenizer=tokenizer,
    compute_metrics=None,
)
```

```
<ipython-input-18-c2b89c913ecb>:2: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 f
  trainer = Trainer(
```

```
# Train the model to improve accuracy
trainer.train()
```

Save Fine-Tuned Model

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
tokenizer = AutoTokenizer.from_pretrained("CAMeL-Lab/bert-base-arabic-camelbert-mix-did-madar-corpus26")
model = AutoModelForSequenceClassification.from_pretrained("CAMeL-Lab/bert-base-arabic-camelbert-mix-did-madar-corpus2
save_path = "/content/marbert_dialect_id_finetuned2"
model.save_pretrained(save_path)
tokenizer.save_pretrained(save_path)
```

```
tokenizer_config.json: 100%    86.0/86.0 [00:00<00:00, 2.23kB/s]
config.json: 100%    1.44k/1.44k [00:00<00:00, 67.6kB/s]
vocab.txt: 100%    305k/305k [00:00<00:00, 3.54MB/s]
special_tokens_map.json: 100%    112/112 [00:00<00:00, 5.95kB/s]
pytorch_model.bin: 100%    436M/436M [00:04<00:00, 137MB/s]
('/content/marbert_dialect_id_finetuned2/tokenizer_config.json',
 '/content/marbert_dialect_id_finetuned2/special_tokens_map.json',
 '/content/marbert_dialect_id_finetuned2/vocab.txt',
 '/content/marbert_dialect_id_finetuned2/added_tokens.json',
 '/content/marbert_dialect_id_finetuned2/tokenizer.json')
```

```
## Save the configurations and tokenized files
model_name = "Ammar-alhaj-ali/arabic-MARBERT-dialect-identification-city"
save_directory = "/content/marbert_dialect_id_finetuned"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

tokenizer.save_pretrained(save_directory)
model.save_pretrained(save_directory)
```

```
model.safetensors: 100%    436M/436M [00:10<00:00, 147MB/s]
```

```
from transformers import BertTokenizerFast, BertForSequenceClassification

model_path = "/content/marbert_dialect_id_finetuned"
model = BertForSequenceClassification.from_pretrained(model_path)
tokenizer = BertTokenizerFast.from_pretrained(model_path)
```

Test the fine tuned model for a low perfroming dialect e.g Sfax

Initially the model gave a fair accuracy of 85% for Cairo and 72% Algiers Dialect

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pandas as pd

# ----------------------------
# Helper function for evaluation
# ----------------------------
def evaluate_dialect(file_path, label_id, dialect_name, num_samples=100):
    # Load TSV file
    df = pd.read_csv(file_path, sep='\t', header=0)

    # Extract the dialectal sentence column
    texts = df["sent"].tolist()[:num_samples]
    labels = [label_id] * len(texts)
    labels_tensor = torch.tensor(labels).to(device)

    # Tokenize
    tokenized = tokenizer(
        texts,
        truncation=True,
        padding=True,
        max_length=128,
        return_tensors="pt"
    )

    input_ids = tokenized["input_ids"].to(device)
    attention_mask = tokenized["attention_mask"].to(device)

    # Inference
    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
```

```
        probs = F.softmax(outputs.logits, dim=1)
        preds = torch.argmax(probs, dim=1)

    # Evaluation
    accuracy = accuracy_score(labels_tensor.cpu(), preds.cpu())
    precision = precision_score(labels_tensor.cpu(), preds.cpu(), average='weighted', zero_division=0)
    recall = recall_score(labels_tensor.cpu(), preds.cpu(), average='weighted', zero_division=0)
    f1 = f1_score(labels_tensor.cpu(), preds.cpu(), average='weighted', zero_division=0)

    # Print results
    print(f"\n Fine tuned Model Evaluation for dialect: {dialect_name}")
    print(f" Accuracy: {accuracy * 100:.2f}%")
    print(f" Precision: {precision * 100:.2f}%")
    print(f" Recall: {recall * 100:.2f}%")
    print(f" F1 Score: {f1 * 100:.2f}%")
```

Use the new fine tuned model to evaluate the 2 underperforming dialects from earlier i.e Cairo and Algiers

```
Sfax_path = "/content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Cairo.tsv"
Sanaa_path = "/content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Algiers.tsv"
Tripoli_path = "/content/MADAR/MADAR.Parallel-Corpora-Public-Version1.1-25MAR2021/MADAR_Corpus/MADAR.corpus.Beirut.tsv"

evaluate_dialect(Sfax_path, label_id=12, dialect_name="Cairo")
evaluate_dialect(Sanaa_path, label_id=13, dialect_name="Algiers")
evaluate_dialect(Tripoli_path, label_id=17, dialect_name="Beirut")
```

```
 Fine tuned Model Evaluation for dialect: Cairo
 Accuracy: 95.00%
 Precision: 100.00%
 Recall: 95.00%
 F1 Score: 97.44%

 Fine tuned Model Evaluation for dialect: Algiers
 Accuracy: 94.00%
 Precision: 100.00%
 Recall: 94.00%
 F1 Score: 96.91%

 Fine tuned Model Evaluation for dialect: Beirut
 Accuracy: 94.00%
 Precision: 100.00%
 Recall: 94.00%
 F1 Score: 96.91%
```

Web User Interface

```
!pip install gradio --quiet
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.2/46.2 MB 17.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.2/322.2 kB 19.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.9/94.9 kB 5.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.3/11.3 MB 110.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 4.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.3/62.3 kB 4.0 MB/s eta 0:00:00
```

```
import matplotlib.pyplot as plt
import gradio as gr
import torch
import torch.nn.functional as F
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import re
import io
from PIL import Image
```

```
# Load model and tokenizer
model_path = "/content/marbert_dialect_id_finetuned"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForSequenceClassification.from_pretrained(model_path)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Label mapping
id2label = {
    0: "Jerusalem", 1: "Tunis", 2: "Mosul", 3: "Sfax", 4: "Rabat",
    5: "Basra", 6: "Muscat", 7: "Khartoum", 8: "Amman", 9: "Sanaa",
    10: "Baghdad", 11: "Jeddah", 12: "Cairo", 13: "Algiers", 14: "Alexandria",
    15: "Aleppo", 16: "Fes", 17: "Beirut", 18: "Tripoli", 19: "Doha",
    20: "MSA", 21: "Riyadh", 22: "Salt", 23: "Damascus", 24: "Aswan", 25: "Benghazi"
}

# Arabic validation
def is_arabic(text):
    return re.fullmatch(r'[\u0600-\u06FF\s]+', text) is not None

# Prediction + convert plot to image
def predict_dialect_with_plot(arabic_text):
    if not is_arabic(arabic_text):
        return None, "❌ Please enter text in Arabic only."

    encoded = tokenizer(
        arabic_text,
        return_tensors="pt",
        truncation=True,
        padding=True,
        max_length=128
    )

    input_ids = encoded["input_ids"].to(device)
    attention_mask = encoded["attention_mask"].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        probs = F.softmax(outputs.logits, dim=1)
        top_probs, top_ids = torch.topk(probs, k=5)

    labels = [id2label[top_ids[0][i].item()] for i in range(5)]
    scores = [top_probs[0][i].item() * 100 for i in range(5)]

    # Text summary
    text_result = "\n".join([f"🔹 {labels[i]}: {scores[i]:.2f}%" for i in range(5)])

    # Create plot
    fig, ax = plt.subplots()
    ax.barh(labels[::-1], scores[::-1], color='skyblue')
    ax.set_xlim(0, 100)
    ax.set_title("Top 5 Predicted Dialects")
    ax.set_xlabel("Confidence (%)")
    ax.set_ylabel("Dialect")
    plt.tight_layout()

    # Convert plot to image
    buf = io.BytesIO()
    plt.savefig(buf, format='png')
    plt.close(fig)
    buf.seek(0)
    img = Image.open(buf)

    return img, text_result



iface = gr.Interface(
    fn=predict_dialect_with_plot,
    inputs=gr.Textbox(lines=2, placeholder="📝 أدخل نصًا بالعربية هنا..."),
    outputs=["image","text"],
    title="معرف اللهجة العربية 🌍",
```

```
    description="🚀 أدخل النص العربي هنا........"
)

iface.launch()

## Cick the ink that appears to access the web interface
```