Report for milestone2: AI System Reproducibility

Introduction:

The purpose of this milestone is to make sure that our chosen model is reproducible through ensuring these following requirements:

Requirements	**Tool recommendation**
Setting the project structure	Cookiecutter data science
Code and data versioning	Git with GitHub Flow or DVC
Experiment tracking	MLFlow, TensorFlow Extended (TFX)
Setting up a meta store for metadata	Depending on your experiment tracking choice, either MLFlow tracking or TFX metadata store

In this report I will be showcasing my progress on each of these requirements with proof of them working.

Setting up the project structure with Cookiecutter:

.dvc	2/28/2024 9:22 PM	File folder	
data	2/28/2024 9:22 PM	File folder	
docs	2/27/2024 11:02 AM	File folder	
experiments	3/4/2024 2:41 PM	File folder	
models	3/2/2024 1:38 PM	File folder	
notebooks	2/27/2024 11:02 AM	File folder	
references	2/27/2024 1:51 PM	File folder	
reports	2/27/2024 11:02 AM	File folder	
src	2/27/2024 11:02 AM	File folder	
venv	2/28/2024 4:38 PM	File folder	
.dvcignore	2/27/2024 12:21 PM	DVCIGNORE File	1 KB
.env	2/27/2024 11:02 AM	ENV File	1 KB
.gitattributes	2/28/2024 9:56 PM	Fichier source Git ...	1 KB
.gitignore	3/7/2024 2:38 PM	Fichier source Git I...	2 KB
LICENSE	2/27/2024 11:02 AM	File	1 KB
Makefile	2/27/2024 11:02 AM	File	5 KB
README	2/27/2024 11:02 AM	Fichier source Mar...	3 KB
requirements	2/27/2024 11:02 AM	Text Document	1 KB
setup	2/27/2024 11:02 AM	Fichier source Pyth...	1 KB

This is the project structure proposed by the cookiecutter datascience to organize the project.

Code and Data Versioning:

For code versioning I opted for GitHub as it is the most used platform for code versioning.

Regarding data versioning on the other hand, I opted for DVC and chose as a remote storage for my datasets' versions google drive.

This is the config file of my ".dvc" folder, it contains the path to the remotes storage (My personal google drive storage).

```
[core]
    remote = mygdrive
['remote "mygdrive"']
    url = https://drive.google.com/drive/folders/1c_QwvUk4kq5tFTifWWvIB1vceFl6f7q6?usp=sharing
```

Experiment tracking using MLFlow:

The goal for experiment tracking is making sure that our chosen model is the one with better performance among our available models, and that is done according to 4 metrics which are accuracy, precision, F1-score, and recall.

The models taken into consideration are 3:

-distilbert/distilbert-base-uncased-finetuned-sst-2-english

-cardiffnlp/twitter-roberta-base-sentiment-latest(Our most performant one)

-lxyuan/distilbert-base-multilingual-cased-sentiments-student

Since the 3 models were tested using the same experiments, I will be only showcasing the one that we are interested in which is "cardiffnlp/twitter-roberta-base-sentiment-latest".

The load data function:

This function reads lines from 2 .txt files, the one containing the tweets and the other the labels and returns them concatenated.

```python
def load_data(texts_file_path, labels_file_path):
    with open(texts_file_path, 'r', encoding='utf-8') as texts_file, \
         open(labels_file_path, 'r', encoding='utf-8') as labels_file:
        texts = texts_file.readlines()
        labels = labels_file.readlines()
        labels = [int(label.strip()) for label in labels]
        data = [{"text": text.strip(), "label": label} for text, label in zip(texts, labels)]
    return data
```

The evaluation function:

This function evaluates the performance of the model and extracts the 4 metrics discussed earlier for each model.

```
def evaluate_model(model, tokenizer, data):
    model.eval()
    predictions, true_labels = [], []

    for item in data:
        inputs = tokenizer(item['text'], padding=True, truncation=True, return_tensors="pt", max_length=512)
        with torch.no_grad():
            outputs = model(**inputs)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=-1).numpy()
        predictions.extend(preds)
        true_labels.append(item['label'])

    accuracy = accuracy_score(true_labels, predictions)
    precision = precision_score(true_labels, predictions, average='weighted', zero_division=0)
    recall = recall_score(true_labels, predictions, average='weighted', zero_division=0)
    f1 = f1_score(true_labels, predictions, average='weighted', zero_division=0)
    return accuracy, precision, recall, f1
```

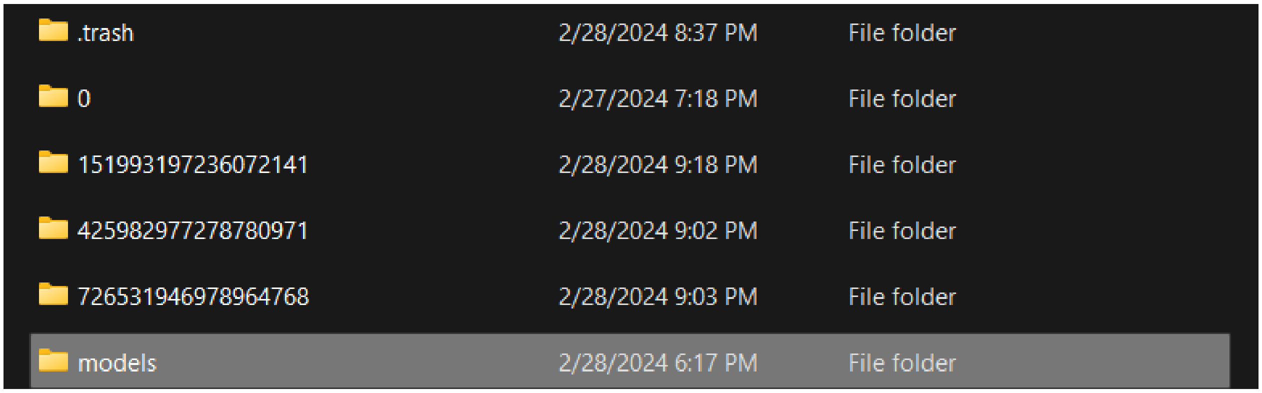
The experiment tracking using MLFlow:

This is the part that takes the model from Huggingface and applies the evaluation function to it and finally logs the output to MLflow:

```
mlflow.set_experiment("experiment_twitter-roberta-base")

with mlflow.start_run(run_name="cardiffnlp/twitter-roberta-base-sentiment-latest"):
    model_name = "cardiffnlp/twitter-roberta-base-sentiment-latest"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSequenceClassification.from_pretrained(model_name)

    mlflow.log_param("model_name", model_name)

    accuracy, precision, recall, f1 = evaluate_model(model, tokenizer, data)

    mlflow.log_metric("accuracy", accuracy)
    mlflow.log_metric("precision", precision)
    mlflow.log_metric("recall", recall)
    mlflow.log_metric("f1_score", f1)
```

The experiment:

The experiment is done using 3 different models and 2 different datasets with 2000 samples each, resulting in this comparison:



Conclusion:

Following this comparison, it is safe to say that our chosen model is our best performing one.

Metadata store for our metadata:

MLFlow provides a built in metadata store that automatically stores the metadata for our data and model samples.

artifacts	2/28/2024 8:41 PM	File folder	
metrics	2/28/2024 8:43 PM	File folder	
params	2/28/2024 8:41 PM	File folder	
tags	2/28/2024 8:41 PM	File folder	
meta	2/28/2024 8:43 PM	Fichier source Yaml	1 KB