Pratique de la data science

April 4, 2025

1 TP 7 - BERT pour l'analyse de sentiment : Finetuning sur des News financières

Librairies à installer :

```
from datasets import load_dataset, concatenate_datasets, DatasetDict, ClassLabel
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArgument
from sklearn.metrics import classification_report
import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
```

Objectif : Finetuning de modèles basés sur l'architecture BERT (et pré-entrainés sur de larges corpus de données). L'objectif est de ré-entrainer ces modèles sur un dataset de classification de news financières issu d'huggingface. On comparera les performances de deux modèles :

- **BERT** (modèle BERT classique, entrainé sur un corpus général).
- **FinBERT** (modèle BERT entrainé sur un corpus de données financières (en plus du corpus général).

Tout doit être codé sous forme de fonctions. L'objectif est de pouvoir comparer l'utilisation de chaque modèle sur cette tâche.

1.1 Téléchargement des datasets

Pour le finetuning, il est nécessaire d'obtenir des données labellisées. Nous allons utiliser deux datasets disponibles sur huggingface et annotés pour l'analyse de sentiments (Négatif, Neutre, Positif):

- **Twitter Financial News Sentiment** (`zeroshot/twitter-financial-news-sentiment`) : Un jeu de données contenant des tweets financiers.
- **Financial Phrase Bank (nettoyé et reformaté)** (`nickmuchi/financial-classification`): Ce dataset contient des phrases issues de textes économiques.

L'objectif est de concaténér ces datasets pour avoir un corpus de finetuning plus conséquent et général.

1. **Charger ces deux datasets (librairie `datasets`):**

   ```
   ds1 = load_dataset("zeroshot/twitter-financial-news-sentiment")
   ds2 = load_dataset("nickmuchi/financial-classification")
   ```

2. **Concaténer ces deux datasets (sur le train et sur le test) :** On doit obtenir un `DatasetDict()` contenant un ensemble de train et un ensemble de test.

1.2 Finetuning des modèles BERT et FinBERT

Créer la fonction `train_model(model_name, dataset, batch_size, num_epochs)` pour le finetuning de nos modèles sur le dataset créé.

1. **Charger les poids et tokenizer pré-entrainés :**

```
tokenizer = . . .
model = . . .
```

2. **Tokenizer le dataset créé et adapter le format pour l'entrainement :**

```
tokenized_train = . . .
tokenized_test  = . . .

tokenized_train.set_format("torch", columns=["input_ids", "attention_mask", "label"])
tokenized_test.set_format("torch", columns=["input_ids", "attention_mask", "label"])
```

3. **La fonction `Trainer` de `transformers` permet d'entrainer les modèles de NLP.** Pour cela, il faut au préalable définir les arguments d'entrainements :

```
training_args = TrainingArguments(
    output_dir=f"./{model_name}_results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_strategy="epoch",
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=num_epochs,
    weight_decay=0.01,
    report_to="none",
    disable_tqdm=False,
    logging_first_step=True
)
```

4. **Création du Trainer :** `trainer = Trainer(...)`

5. **Entrainement et évaluation du modèle :**

```
trainer.train()
trainer.evaluate()
```

Finetuner les modèles BERT et FinBERT, comparer les performances et sauvegarder les poids pour le TP8.