M2SOL034 Corpus, ressources et linguistique outillée

TD1 : Introduction aux Corpus et à l'IA

Wiem TAKROUNI

Sorbonne Université

Master « Langue et Informatique »

UFR Sociologie et Informatique pour les Sciences Humaines

Semestre 2, 2024-2025, le 04 avril 2025

Objectifs du TD

L'objectif de cet exercice est de **tester et comparer différents modèles de plongements de mots** sur une tâche de classification de textes. Vous devrez :

- **Charger et visualiser** les embeddings d'un modèle pré-entraîné.
- **Utiliser ces embeddings** pour entraîner un modèle de classification.
- **Fine-tuner BERT** sur un dataset spécifique pour comparer les performances.

Données :

Utilisez le dataset **IMDB Reviews** (classification de critiques de films en "positif" ou "négatif"). Il est disponible dans torchtext ou datasets de Hugging Face.

Partie 1 : Chargement et Exploration des Embeddings

Étape 1 : Charger Word2Vec et visualiser les mots proches

Implémentez un script Python permettant de charger un modèle pré-entraîné Word2Vec et de trouver les mots les plus proches d'un mot donné.

Instructions :

- Chargez GoogleNews-vectors-negative300.bin ou fasttext-wiki-news-subwords-

300.

- Affichez les 10 mots les plus proches de *"film"*, *"acteur"*, *"histoire"*.

```python
from gensim.models import KeyedVectors

# Charger le modèle Word2Vec pré-entraîné
model = KeyedVectors.load_word2vec_format("GoogleNews-vectors-negative300.bin", binary=True)

# Trouver les mots les plus proches
print(model.most_similar("film"))
```

1. Question 1 : Interprétez les résultats obtenus. Pourquoi ces mots sont-ils proches ?
2. Question 2 : Modifiez la requête en testant d'autres mots et analysez les résultats.

Partie 2 : Classification avec Word2Vec

Étape 2 : Extraire des représentations de textes avec Word2Vec

Instructions :

- Chargez les critiques de films du dataset IMDB.
- Représentez chaque critique comme la moyenne des embeddings Word2Vec des mots qui la composent.
- Entraînez un classificateur SVM ou Random Forest pour prédire si un avis est positif ou négatif.

```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from datasets import load_dataset

# Charger les données IMDB
dataset = load_dataset("imdb")
train_texts, test_texts = dataset['train']['text'], dataset['test']['text']
train_labels, test_labels = dataset['train']['label'], dataset['test']['label']

# Fonction pour obtenir la moyenne des embeddings Word2Vec d'un texte
def get_embedding(text, model):
    words = text.split()
    word_vectors = [model[word] for word in words if word in model]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)

# Transformer les textes en vecteurs
X_train = np.array([get_embedding(text, model) for text in train_texts])
X_test = np.array([get_embedding(text, model) for text in test_texts])

# Entraîner un classificateur SVM
clf = SVC(kernel='linear')
clf.fit(X_train, train_labels)

# Prédire et évaluer
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(test_labels, y_pred))
```

3. **Question 3 :** Quelle est l'**accuracy** obtenue ? Word2Vec permet-il une bonne classification ?
4. **Question 4 :** Comparez avec FastText. Ce modèle est-il plus performant ?

Partie 3 : Fine-Tuning de BERT pour la Classification

Étape 3 : Fine-tuning de BERT sur IMDB

Instructions :

- Utilisez **Hugging Face Transformers** pour **fine-tuner BERT** sur le dataset IMDB.
- Comparez la performance avec celle de Word2Vec.

5. **Question 5 :** Quelle est l'accuracy obtenue avec BERT ? Est-elle meilleure que Word2Vec ?
6. **Question 6 :** Pourquoi BERT est-il plus performant que Word2Vec ?
7. **Question 7 :** Testez le modèle sur des phrases de votre choix et analysez les résultats.

```
from transformers import BertTokenizer, BertForSequenceClassification, Trainer,
TrainingArguments
from datasets import load_dataset

# Charger le dataset IMDB
dataset = load_dataset("imdb")

# Charger BERT et tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForSequenceClassification.from_pretrained("bert-base-uncased")

# Tokenization
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Définir les arguments d'entraînement
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
)

# Entraînement
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["test"],
)

trainer.train()
```