Projet Knowledge Extraction from Unstructured Text

Partie A : Preprocessing et Représentation Textuelle

Rapport Technique

Jacques Gastebois

Master 2 VMI - Université Paris Cité

IFLCE085 - Recherche et extraction sémantique à partir de texte

Prof. Salima Benbernou

6 Décembre 2025

Résumé

Ce rapport présente la méthodologie complète de preprocessing appliquée à un dataset NER (Named Entity Recognition) de 2221 phrases annotées. Le pipeline implémenté comprend le nettoyage, la lemmatization et la vectorisation TF-IDF. Les données sont divisées en ensembles Train/Dev/Test (70/15/15) et exportées dans des formats exploitables pour la Partie B du projet.

Table des matières

1 Introduction

1.1 Contexte du Projet

Ce projet s'inscrit dans le cadre du cours IFLCE085 et vise à développer un système complet d'extraction de connaissances à partir de textes. La Partie A se concentre sur le preprocessing et la préparation des données.

1.2 Objectifs

— Charger et analyser le dataset NER (2221 phrases)
— Diviser les données en ensembles Train/Dev/Test
— Appliquer un pipeline NLP complet (nettoyage, lemmatization)
— Générer des représentations vectorielles TF-IDF
— Exporter les résultats dans des formats réutilisables pour la Partie B

1.3 Dataset

NER Dataset (Named Entity Recognition) : Dataset de 2221 phrases annotées pour la reconnaissance d'entités nommées.

Structure du fichier source (`data.csv`) :

— `id` : Identifiant unique de la phrase
— `words` : Liste des mots tokenisés (format array)
— `ner_tags` : Tags NER (0=O, 1=B-LOC, 2=B-PER, 4=B-ORG)
— `text` : Texte brut de la phrase

Split des données :

— **Train** : 1554 phrases (70%)
— **Dev** : 333 phrases (15%)
— **Test** : 334 phrases (15%)

2 Méthodologie

2.1 Pipeline de Preprocessing

Le pipeline suit une approche séquentielle en 3 étapes principales :

Étape 1 : Nettoyage et Normalisation

— Conversion en lowercase
— Suppression des caractères spéciaux (garde lettres, chiffres, espaces)
— Normalisation des espaces multiples

Étape 2 : Lemmatization

— Outil : spaCy `en_core_web_sm`
— Réduction des mots à leur forme canonique
— **Appliqué sur TOUTES les phrases (Train, Dev, Test)**

Étape 3 : Vectorisation TF-IDF

- Outil : scikit-learn `TfidfVectorizer`
- **Apprentissage (fit)** sur le Train set uniquement
- **Transformation** appliquée sur Train, Dev et Test
- Paramètres : `max_features=3000`, `ngram_range=(1,2)`

2.2 Justification des Choix Techniques

2.2.1 Split Train/Dev/Test

La division 70/15/15 est un standard en machine learning qui garantit :

- Suffisamment de données d'entraînement (1554 phrases)
- Des ensembles de validation et test représentatifs (333-334 phrases)
- Une évaluation fiable des performances

2.2.2 Lemmatization

La lemmatization a été préférée au stemming car elle produit des formes linguistiquement correctes (ex : "better" → "good" au lieu de "bett").

2.2.3 TF-IDF sur Textes Lemmatisés

Le TF-IDF est calculé sur les textes lemmatisés pour garantir :

- Une réduction du vocabulaire (formes canoniques)
- Une meilleure capture des concepts sémantiques
- Une cohérence avec le pipeline NLP complet

2.2.4 Nombre de Features

3000 features (au lieu de 5000 pour SciREX) car :

- Dataset plus petit (2221 vs 438 documents)
- Phrases courtes (vs articles scientifiques complets)
- Meilleur ratio features/documents

3 Implémentation

3.1 Technologies Utilisées

3.2 Exemple de Code

3.2.1 Chargement et Split

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Chargement
df = pd.read_csv('data.csv')
```

Librairie	Version/Utilisation
Python	3.12
pandas	2.2.2 (manipulation de données)
numpy	2.0.2 (calculs numériques)
spaCy	3.8.11 (lemmatization)
scikit-learn	1.6.1 (TF-IDF, train_test_split)
scipy	1.16.3 (matrices sparse)

TABLE 1 – Stack technologique

```
# Split 70/15/15
train_df, temp_df = train_test_split(df, test_size=0.3,
   random_state=42)
dev_df, test_df = train_test_split(temp_df, test_size=0.5,
   random_state=42)
```

3.2.2 Fonction de Nettoyage

```
def clean_text(text):
    """Nettoie le texte."""
    if not isinstance(text, str):
        return ""

    # 1. Lowercase
    text = text.lower()

    # 2. Suppression caracteres speciaux
    text = re.sub(r'[^a-z0-9\s]', '␣', text)

    # 3. Normalisation espaces
    text = re.sub(r'\s+', '␣', text).strip()

    return text
```

3.2.3 Lemmatization avec spaCy

```
def lemmatize_text(text):
    """Lemmatise le texte avec spaCy."""
    doc = nlp(text)
    lemmas = [token.lemma_ for token in doc]
    return '␣'.join(lemmas)

# Application
train_df['lemmatized_text'] = train_df['cleaned_text'].apply(
   lemmatize_text)
```

3.2.4 Vectorisation TF-IDF

```
# Preparation des textes LEMMATISES
train_texts = train_df['lemmatized_text'].tolist()
dev_texts = dev_df['lemmatized_text'].tolist()
test_texts = test_df['lemmatized_text'].tolist()

# Vectorisation
tfidf_vectorizer = TfidfVectorizer(
    max_features=3000,
    min_df=2,
    max_df=0.8,
    ngram_range=(1, 2)
)

# Fit sur TRAIN, transform sur tous
tfidf_train = tfidf_vectorizer.fit_transform(train_texts)
tfidf_dev = tfidf_vectorizer.transform(dev_texts)
tfidf_test = tfidf_vectorizer.transform(test_texts)
```

4 Résultats

4.1 Statistiques du Preprocessing

Métrique	Valeur
Total de phrases	2221
Phrases train	1554 (70%)
Phrases dev	333 (15%)
Phrases test	334 (15%)
Features TF-IDF	3000
Taille matrice train	1554 Œ 3000
Taille matrice dev	333 Œ 3000
Taille matrice test	334 Œ 3000
Densité moyenne	∼15-20%

TABLE 2 – Statistiques de preprocessing

4.2 Exemple de Transformation

Texte original :

"When Aeneas later traveled to Hades , he called to her ghost but she neither spoke to nor acknowledged him ."

Texte nettoyé :

"when aeneas later traveled to hades he called to her ghost but she neither spoke to nor acknowledged him"

Texte lemmatisé :

"when aeneas later travel to hade he call to her ghost but she neither speak to nor acknowledge he"

5 Export des Données

5.1 Fichiers Générés

Tous les fichiers sont exportés dans le dossier `preprocessed_data/` :

Fichier	Format	Taille estimée
`train_preprocessed.csv`	CSV	∼2 MB
`dev_preprocessed.csv`	CSV	∼0.5 MB
`test_preprocessed.csv`	CSV	∼0.5 MB
`tfidf_matrix.npz`	NumPy sparse	∼0.5 MB
`tfidf_matrix_dev.npz`	NumPy sparse	∼0.1 MB
`tfidf_matrix_test.npz`	NumPy sparse	∼0.1 MB
`tfidf_vectorizer.pkl`	Pickle	∼0.2 MB
`tfidf_feature_names.npy`	NumPy	∼0.05 MB
`metadata.json`	JSON	∼0.01 MB

TABLE 3 – Fichiers exportés

5.2 Structure des CSV

Les fichiers CSV contiennent 4 colonnes :

— `id` : Identifiant unique de la phrase

— `text` : Texte original

— `cleaned_text` : Texte nettoyé

— `lemmatized_text` : Texte lemmatisé

6 Utilisation pour la Partie B

6.1 Chargement des Données

6.1.1 Textes Prétraités

```
import pandas as pd

# Charger les CSV
df_train = pd.read_csv('preprocessed_data/train_preprocessed.csv'
    )
df_dev = pd.read_csv('preprocessed_data/dev_preprocessed.csv')
df_test = pd.read_csv('preprocessed_data/test_preprocessed.csv')

# Acceder aux differentes versions
```

```
lemmatized_texts = df_train['lemmatized_text']
```

6.1.2 Matrices TF-IDF

```
from scipy.sparse import load_npz
import pickle
import numpy as np

# Charger les matrices TF-IDF
tfidf_train = load_npz('preprocessed_data/tfidf_matrix.npz')
tfidf_dev = load_npz('preprocessed_data/tfidf_matrix_dev.npz')
tfidf_test = load_npz('preprocessed_data/tfidf_matrix_test.npz')

# Charger le vectoriseur (pour transformer de nouveaux textes)
with open('preprocessed_data/tfidf_vectorizer.pkl', 'rb') as f:
    vectorizer = pickle.load(f)

# Charger les noms de features
feature_names = np.load('preprocessed_data/tfidf_feature_names.
   npy')

# Transformer un nouveau texte
new_text = ["your␣new␣text␣here"]
new_vector = vectorizer.transform(new_text)
```

6.1.3 Métadonnées

```
import json

# Charger les metadonnees
with open('preprocessed_data/metadata.json', 'r') as f:
    metadata = json.load(f)

# Acceder aux informations
n_train = metadata['train_size']
n_features = metadata['tfidf_features']
```

6.2 Recommandations pour la Partie B

1. **Utiliser les textes lemmatisés** pour toute analyse sémantique
2. **Réutiliser le vectoriseur TF-IDF** pour transformer de nouveaux documents
3. **Exploiter les matrices sparse** pour économiser la mémoire
4. **Respecter le split Train/Dev/Test** pour l'évaluation

7 Accès aux Ressources

7.1 Liens Importants

— **GitHub Repository :**
https://github.com/635jack/Projet_Semantique_AB_BEM_FD_JG

— **Google Colab Notebook :**
https://colab.research.google.com/github/635jack/Projet_Semantique_AB_BEM_FD_JG/blob/master/PartieA_Preprocessing.ipynb

7.2 Instructions d'Exécution

7.2.1 Sur Google Colab

1. Ouvrir le lien Colab ci-dessus
2. Uploader le fichier `data.csv`
3. Exécuter toutes les cellules : `Runtime` → `Run all`
4. Télécharger les fichiers exportés depuis `preprocessed_data/`

7.2.2 En Local

1. Cloner le repository :
   `git clone https://github.com/635jack/Projet_Semantique_AB_BEM_FD_JG`
2. Installer Jupyter :
   `pip install jupyter`
3. Exécuter le notebook :
   `jupyter notebook PartieA_Preprocessing.ipynb`

8 Conclusion

8.1 Résumé des Contributions

Ce travail a permis de :

— Implémenter un pipeline NLP complet et efficace
— Traiter l'intégralité du dataset NER (2221 phrases)
— Créer un split Train/Dev/Test reproductible
— Générer des représentations vectorielles de haute qualité
— Exporter les résultats dans des formats standardisés

8.2 Points Clés

1. **Pipeline simplifié** : Nettoyage → Lemmatization → TF-IDF
2. **Split stratégique** : 70/15/15 avec `random_state=42`
3. **TF-IDF sur textes lemmatisés** : Garantit la cohérence du preprocessing
4. **3000 features** : Adapté à la taille du dataset
5. **Formats réutilisables** : CSV, NPZ, Pickle, JSON

8.3 Perspectives pour la Partie B

Les données préparées sont prêtes pour :

— Extraction d'entités nommées (NER)

— Classification de textes

— Clustering sémantique

— Analyse de relations

— Modèles d'embeddings avancés

Annexes

A. Commandes Utiles

Vérifier la taille des fichiers :

```
ls -lh preprocessed_data/
```

Compter les lignes des CSV :

```
wc -l preprocessed_data/*.csv
```

B. Dépendances

Les dépendances sont installées automatiquement dans le notebook :

— pandas, numpy, nltk, spacy, scikit-learn, scipy

— Modèle spaCy : `en_core_web_sm`