Université Bourgogne-Europe
Master 1 “Bases de Données et Intelligence Artificielle”
Module d’Algorithmes fondamentaux de l’IA
Du 14 avril au 1er mai

Mini projet
- Rapport -

Sujet :

Classification de textes

Réalisé par : BELASSEL Meryem, NICOLLE Thomas

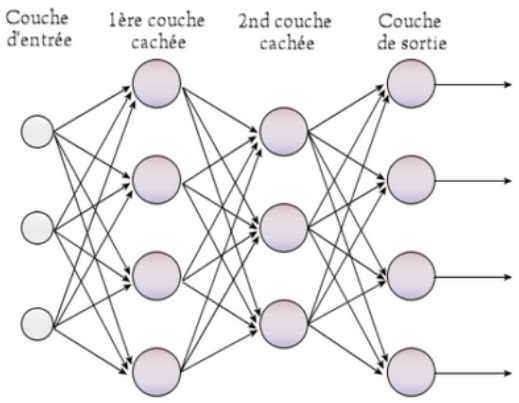
Professeurs : Mr. LALOU Mohammed et Mr. EL VAIGH CHEIKH Brahim

Sommaire

Introduction..... 2
Partie I : MLP et PyTorch..... 3
Partie II : Apprentissage semi-supervisé..... 6
Partie III : Comparaison de plusieurs modèles de classification..... 7
Partie IV : Mise en production..... 8

Introduction

Ce projet traite de la classification de textes en utilisant l'ensemble de données IMDb pour l'analyse des sentiments et un ensemble de données de spam pour la détection de spam. Nous avons utilisé le fichier *mlp.py* fourni pour implémenter un perceptron multicouche (MLP) avec *PyTorch*, l'avons étendu avec des modifications, avons appliqué l'apprentissage semi-supervisé, avons comparé plusieurs modèles, et avons déployé un système de détection de spam en utilisant Flask. Ce rapport synthétise notre méthodologie, nos résultats et notre analyse conformément aux exigences du TP4.



Partie I : MLP et PyTorch

1. **Quelle partie du code charge les données ?**
   Le chargement des données est effectué au début du script mlp.py via les lignes suivantes :

```
df1 = pd.read_csv('imdb_train.csv')
X_train = df1['text']
y_train = df1['label']
df2 = pd.read_csv('imdb_test.csv')
X_test = df2['text']
y_test = df2['label']
```

Ces lignes du code lisent les fichiers CSV contenant les ensembles d'entraînement *imdb_train.csv* et de test *imdb_test.csv* à l'aide de la bibliothèque *pandas*.

2. **Les données sont-elles chargées en une fois ou en groupes (batch) ?**

Les données sont d'abord vectorisées en une fois, puis regroupées en batchs de taille 32 à l'aide du **DataLoader** :

```
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

3. **Combien de couches cachées sont utilisées dans ce modèle ?**

Le modèle MLP défini dans la classe MLP possède une couche cachée:

- Une couche d'entrée passant de 5000 neurones à 128 neurones *self.fc1.*
- Une couche cachée passant de 128 neurones à 64 neurones *self.fc2.*

La couche finale *self.fc3* est la couche de sortie, passant de 64 neurones à 2 neurones.

4. **Donnez le nombre de neurones d'entrée et de sortie pour les différentes couches.**

- *fc1* : entrée = 5000, car le paramètre *max_features*=5000 dans le TF-IDF, sortie = 128.
- *fc2* : entrée = 128, sortie = 64.
- *fc3* : entrée = 64, sortie = 2

5. **Quelle fonction de perte est utilisée dans ce modèle ?**

La fonction de perte utilisée est *l'entropie croisée* (nn.CrossEntropyLoss), spécifiée dans la fonction train.

6. **Complétez et testez le code.**

MLP :

	precision	recall	f1-score	support
0	0.83	0.88	0.85	12500
1	0.87	0.82	0.84	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

MLP :

	precision	recall	f1-score	support
0	0.84	0.86	0.85	12500
1	0.86	0.84	0.85	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

Le code a été complété et testé, il implémente un modèle MLP pour la classification binaire sur le dataset IMDb. Le code a été exécuté plusieurs fois pour des fins d'analyse et de comparaison. Le tableau affiché fournit des variations dans les métriques (précision, recall, F1-score, accuracy). Les variations des résultats à chaque exécution est dû au fait que les poids des couches sont initialisés aléatoirement, ainsi que le mélange aléatoire des données, via l'application du paramètre **shuffle=true**.

Le modèle MLP atteint une moyenne de 0.855 en accuracy et F1-score, ce qui est correcte pour le dataset IMDb.

7. **Adaptez les hyper-paramètres de votre modèle. Comparez les résultats avec et sans cette adaptation.**

L'adaptation des hypre-paramètres du modèle a permis d'améliorer les résultats des métriques.

Résultats :

MLP :

	precision	recall	f1-score	support
0	0.86	0.86	0.86	12500
1	0.86	0.86	0.86	12500
accuracy			0.86	25000
macro avg	0.86	0.86	0.86	25000
weighted avg	0.86	0.86	0.86	25000

MLP :

	precision	recall	f1-score	support
0	0.84	0.86	0.85	12500
1	0.86	0.84	0.85	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

8. **Ajouter une couche de dropout (couche d'oublie) pour éviter le sur-apprentissage.**

L'ajout d'une couche de dropout (probabilité = 0.2 ou 0.3) au modèle MLP visait à réduire le surapprentissage. Les résultats montrent une accuracy et un F1-score moyens de 0.85, tant avec que sans dropout, sur le dataset IMDb. Les performances sont équilibrées entre les classes (F1-score ≈ 0.85 pour les deux classes) et stables (variation de ±0.01). L'absence de différence notable suggère que le modèle de base ne souffrait pas d'overfitting significatif,

probablement en raison de la simplicité de l'architecture, du nombre limité d'époques (10 à 15), et de la taille du dataset.

Résultats:

MLP :

	precision	recall	f1-score	support
0	0.83	0.88	0.85	12500
1	0.87	0.82	0.85	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

MLP :

	precision	recall	f1-score	support
0	0.85	0.88	0.86	12500
1	0.87	0.85	0.86	12500
accuracy			0.86	25000
macro avg	0.86	0.86	0.86	25000
weighted avg	0.86	0.86	0.86	25000

9. **Changer la fonction objective en la remplaçant par une de votre choix (L'Entropie Croisée Binaire Pondérée).**

L'utilisation de L'entropie croisée binaire pondérée permet de gérer les déséquilibres de classes ainsi qu'améliorer les performances du modèle sur une dataset donnée où les classes ne sont pas représentées. Dans notre cas, ça pourrait être traduit du fait que les critiques positives sont plus importantes (nombreuses) que les critiques négatives (ou vice versa). Sans cette fonction, le traitement des erreurs de classification se fait de manière égale. Grâce à **nn.BCEWithLogitsLoss**, la classe minoritaire est mieux gérée, améliorant ainsi les métriques comme le rappel et le F1-score pour cette dernière. Dans le cas de ce dataset, la pondération a été moins marquée.

Résultat:

MLP avec nn.BCEWithLogitsLoss pondéré:

	precision	recall	f1-score	support
0.0	0.85	0.88	0.87	12500
1.0	0.88	0.85	0.86	12500
accuracy			0.86	25000
macro avg	0.86	0.86	0.86	25000
weighted avg	0.86	0.86	0.86	25000

10. **Comparer les résultats avec et sans ce changement.**

Après avoir remplacé la fonction de perte par une Entropie Croisée Binaire Pondérée, avec une pondération (**pos_weight**=1.5) pour gérer la discordance potentiel des classes au sein du dataset IMDb pour la classification des sentiments, les performances du modèle MLP ont été légèrement amélioré, avec une augmentation de l'*exactitude* globale de 0.85 à 0.86, ainsi qu'une amélioration du rappel pour la classe 1 de 0.82 à 0.85, montrant l'efficacité de la pondération qui a aidé le modèle à mieux identifier les exemples positifs, puis une augmentation du F1-score pour les deux classes, de 0.85 à 0.87 pour la classe 0, et de 0.84 à 0.86 pour la classe 1, indiquant ainsi un meilleur équilibre entre la précision et le rappel. Enfin, les métriques globales passent toutes de 0.85 à 0.86.

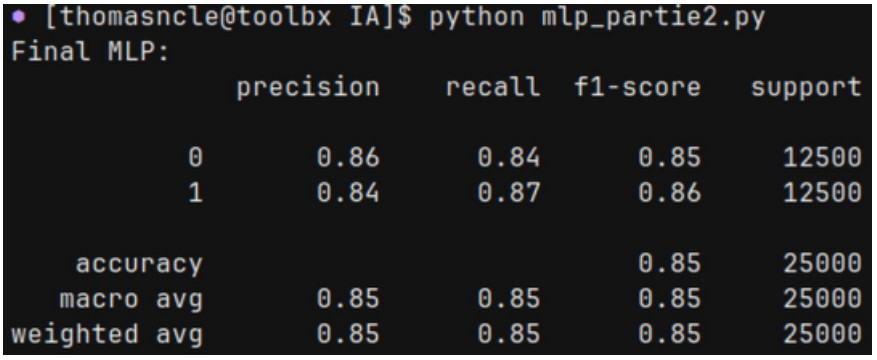
Néanmoins, comme le dataset est peu déséquilibré, l'impact de la pondération est relativement faible. Dans un autre contexte, avec une dataset plus déséquilibré, la modification des hyper-paramètres ainsi que l'ajout de la couche drop out et l'utilisation de

l’entropie croisée binaire pondérée auront eu un effet plus important, en améliorant encore plus les métriques de la classe minoritaire.

Partie II : Apprentissage semi-supervisé

Ce script Python implémente un modèle de réseau de neurones (MLP) pour la classification de textes en utilisant le dataset IMDB. Les principales étapes incluent :

- **Modèle MLP** : Trois couches linéaires avec activation ReLU.
- **Préparation des Données** : Chargement du dataset IMDB, vectorisation des textes avec TF-IDF, et conversion en tenseurs PyTorch.
- **Entraînement Initial** : Utilisation de *DataLoader* pour l'entraînement par lots sur 10 époques.
- **Pseudo-étiquetage** : Prédiction des étiquettes pour les données non supervisées et sélection des prédictions de haute confiance.
- **Réentraînement** : Ajout des échantillons de haute confiance à l'ensemble d'entraînement et réentraînement du modèle.
- **Évaluation** : Évaluation du modèle sur l'ensemble de test et affichage des métriques de performance.



Le résultat affiché dans l'image montre les métriques de performance du modèle MLP final :

- **Précision, Rappel, F1-score** : Les scores pour chaque classe (0 et 1) sont autour de 0.84 à 0.87, indiquant une bonne performance du modèle.
- **Accuracy** : La précision globale est de 0.85, ce qui signifie que le modèle prédit correctement 85% des étiquettes.
- **Support** : Chaque classe a 12,500 échantillons, montrant un équilibre dans le dataset de test.

Les métriques montrent que le modèle est efficace pour la classification des textes en sentiments positifs et négatifs, avec une performance équilibrée entre les deux classes.

Partie III : Comparaison de plusieurs modèles de classification

1. Implémenter et comparer plusieurs modèles de machine learning sur IMDb (voir TP3) et sur le dataset spam (les données du fichier spam.csv).

Dans cette partie, nous allons implémenter et comparer plusieurs modèles sur les deux datasets. Nous allons implémenter plusieurs modèles de classification (Régression logistique, SVM et Random Forest) sur la dataset IMDb et la dataset spam, tout en utilisant les métriques pour comparer.

Résultats:

Résultats pour Régression Logistique sur IMDb :

	precision	recall	f1-score	support
0	0.88	0.88	0.88	12500
1	0.88	0.88	0.88	12500
accuracy			0.88	25000
macro avg	0.88	0.88	0.88	25000
weighted avg	0.88	0.88	0.88	25000

Résultats pour SVM sur IMDb :

	precision	recall	f1-score	support
0	0.88	0.88	0.88	12500
1	0.88	0.88	0.88	12500
accuracy			0.88	25000
macro avg	0.88	0.88	0.88	25000
weighted avg	0.88	0.88	0.88	25000

Résultats pour Random Forest sur IMDb :

	precision	recall	f1-score	support
0	0.83	0.85	0.84	12500
1	0.85	0.82	0.83	12500
accuracy			0.84	25000
macro avg	0.84	0.84	0.84	25000
weighted avg	0.84	0.84	0.84	25000

Résultats pour Régression Logistique sur Spam :

	precision	recall	f1-score	support
0	0.97	1.00	0.99	966
1	1.00	0.81	0.90	149
accuracy			0.97	1115
macro avg	0.99	0.91	0.94	1115
weighted avg	0.98	0.97	0.97	1115

Résultats pour SVM sur Spam :

	precision	recall	f1-score	support
0	0.98	1.00	0.99	966
1	0.99	0.89	0.94	149
accuracy			0.98	1115
macro avg	0.98	0.95	0.96	1115
weighted avg	0.98	0.98	0.98	1115

Résultats pour Random Forest sur Spam :

	precision	recall	f1-score	support
0	0.97	1.00	0.99	966
1	1.00	0.81	0.89	149
accuracy			0.97	1115
macro avg	0.99	0.90	0.94	1115
weighted avg	0.97	0.97	0.97	1115

Tableau comparatif pour IMDb :

	Modèle	Accuracy	F1-score (Classe 0)	F1-score (Classe 1)
0	Régression Logistique	0.88264	0.882443	0.882837
1	SVM	0.87896	0.878815	0.879105
2	Random Forest	0.83732	0.839598	0.834977
3	MLP (Partie 1)	0.86000	0.870000	0.860000

Tableau comparatif pour Spam :

	Modèle	Accuracy	F1-score (Classe 0)	F1-score (Classe 1)
0	Régression Logistique	0.974888	0.985714	0.896296
1	SVM	0.983857	0.990750	0.936620
2	Random Forest	0.973991	0.985212	0.892193

PS C:\Users\belas\Documents\Fac\Master\M1\S2\Algo_IA\Projet> ^C

2. Comment sont découpées les données en train et test pour le datasets spam ?

Les données du dataset spam sont découpées en utilisant la fonction *train_set_split* de la bibliothèque **scikit-learn**, avec comme paramètres : la proportion (80% pour l'entraînement et 20% pour le test), la stratification, avec *stratify=y_spam*, qui permet de préserver dans les ensembles d'entraînement et de test, la répartition des classes (ham et spam), puis enfin la reproductibilité, avec *random_state=42*, qui garantit que la division reste la même à chaque exécution.

3. Quel modèle donne les meilleurs résultats pour chaque dataset ?

- **IMDb:** La régression logistique est le meilleur modèle, car elle atteint la plus haute *accuracy* (0.88264) et des *F1-score* très équilibrés pour la classe 0 et la classe 1.
- **Spam:** La **SVM** est le meilleur modèle pour le dataset, car il atteint la plus haute *accuracy* (0.983857) et le meilleur *F1-score* pour la classe spam (0.936620), ce qui est significatif dans le cas où la classe **spam** est minoritaire.

4. Est-ce que votre évaluation est significative ? (le score change sur plusieurs évaluations ?)

Les données IMDb sont chargées à partir de 2 fichiers qui définissent des ensembles d'entraînement et de test fixes. Pour répondre à cette question, nous avons décidé d'évaluer le dataset **spam** en modifiant le paramètre *random_state* et donc avec des divisions train et test différentes:

- **Random_state = 80 :**

```
Tableau comparatif pour Spam :
                 Modèle  Accuracy  F1-score (Classe 0)  F1-score (Classe 1)
0  Régression Logistique  0.974888             0.985714             0.896296
1                    SVM  0.988341             0.993302             0.955017
2          Random Forest  0.980269             0.988741             0.920290
PS C:\Users\belas\Documents\Fac\Master\M1\S2\Algo_IA\Projet>
```

- **Randome_state = 100:**

```
Tableau comparatif pour Spam :
                 Modèle  Accuracy  F1-score (Classe 0)  F1-score (Classe 1)
0  Régression Logistique  0.972197             0.984143             0.887273
1                    SVM  0.988341             0.993289             0.955631
2          Random Forest  0.983857             0.990750             0.936620
PS C:\Users\belas\Documents\Fac\Master\M1\S2\Algo_IA\Projet>
```

L'évaluation initiale (avec *random_state=42*) n'était pas la meilleure, car elle ne permettait pas de mesurer la différence des scores. En réalisant deux divisions différentes, les scores changent légèrement, confirmant que la performance des modèles dépend en partie de la répartition des données. Cependant, l'évaluation est à moitié significative, car les scores changent sur plusieurs évaluations, mais la variabilité reste faible. La classe 1 (spam) montre un changement plus notable, ce qui est estimé en raison du déséquilibre des classes.

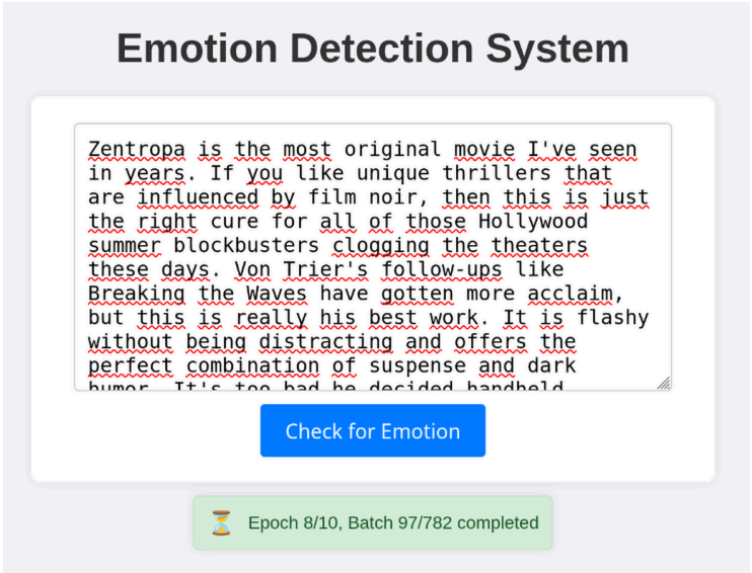
Partie IV : Mise en production

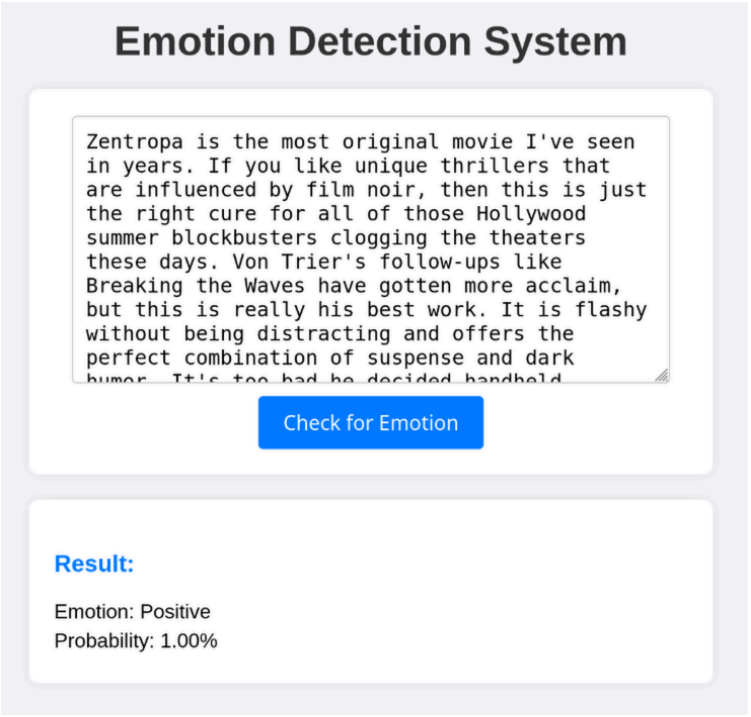
Ce programme est une application web utilisant Flask et Flask-SocketIO pour classifier des textes en temps réel. Il intègre un modèle de réseau de neurones (MLP) entraîné sur le dataset IMDB. Les principales fonctionnalités incluent :

- **Modèle MLP** : Trois couches linéaires avec activation ReLU.
- **Préparation des Données** : Chargement et vectorisation des textes avec TF-IDF.
- **Entraînement** : Utilisation de *DataLoader* pour l'entraînement par lots sur 10 époques.

- **Routes Flask** : Soumission de texte via formulaire, prédiction de l'étiquette (positive/négative) et réentraînement du modèle avec les nouvelles données de haute confiance.
- **SocketIO** : Notifications en temps réel des étapes de traitement.

L'application permet une classification interactive et continue des textes avec mise à jour du modèle.





Conclusion

Ce projet a permis d'explorer plusieurs approches de classification de textes, notamment via un perceptron multicouche (MLP) et des modèles classiques comme la régression logistique, SVM et Random Forest. Les expériences réalisées ont démontré l'importance de la sélection des modèles et des hyperparamètres pour améliorer les performances. L'application de l'apprentissage semi-supervisé et la mise en œuvre avec Flask ont révélé des usages concrets dans la détection des spams et l'analyse des sentiments. Ces conclusions mettent en évidence la valeur de l'expérimentation pour améliorer les performances dans le domaine du traitement automatique du langage.