



Compétition Kaggle - Classer le texte

Partie Rapport

Par Lucky Khounvongsa et Hao Yuan Zhang

Matricules : 20172476 et 20208605

Travail présenté à Ionnis Mitliagkas

dans le cadre du cours de

IFT 3395 – Fondements de l'apprentissage machine

Remis le 13 novembre 2024



1. Introduction

L'objectif dans cette compétition est de développer un modèle d'apprentissage automatique qui classe automatiquement les documents textuels dans un ensemble de catégories binaires. Les documents originaux ne sont pas accessibles cependant, ils sont représentés par des vecteurs de comptage de mots par document. Nous devons optimiser les prédictions sur un ensemble de tests. L'évaluation de la pertinence des prédictions se fait à l'aide du macro F1 score. Plus le score est élevé, plus les prédictions soumises sous forme de fichier .csv sont cohérentes.

Pour la première étape, nous devons battre le classificateur de régression logistique de référence sur le classement public. Nous étions limités à l'utilisation de la librairie *NumPy* et d'autres fonctionnalités de base de Python. Les étapes principales pour le fichier `classification.py` sont les chargements de données, le prétraitement des données, l'entraînement du modèle et l'évaluation. Nous avons pu dépasser le seuil proposé.

	submission.csv	0.69973	0.73236	
Complete · Hao Yuan Zhang · 22d ago				

Pour la deuxième étape, il s'agit de développer un modèle plus performant que le premier en utilisant les méthodes de notre choix. Nous sommes maintenant libres d'utiliser la librairie *sklearn*. Par rapport au premier modèle, dans le fichier `classification_improved.py` nous avons fait une légère amélioration. Les étapes suivies sont les chargements des données, la réduction de dimensions, le prétraitement des données, l'entraînement du modèle et l'ajustement du seuil. (Note : Nous avons modifié le nom du fichier `test_predictions.csv` pour `submission_improved.csv`).

	test_predictions.csv	0.70201	0.73744	
Complete · Hao Yuan Zhang · 2d ago				

À la fin de la compétition, en classement privé, nous sommes classés 117^{ème} sur 151 équipes tandis que pour le classement public, nous nous retrouvons en 40^{ème} place.

117	▼ 77	Hao Yuan Zhang	 	0.70201	14	2d
-----	------	----------------	---	---------	----	----

Notre classement privé.

40	Hao Yuan Zhang	 	0.73744	14	2d
----	----------------	---	---------	----	----

Notre classement public.

2. Description détaillée des codes

2.1. Fichier *classement.py*

La méthode d'extraction utilisée dans ce code est *load_data* pour l'extraction des vecteurs de comptage de termes des fichiers *NumPy*.

Les méthodes de prétraitement implémentées sont *normalize_data* pour la normalisation des données, *split_data* pour la division des données et *oversample_minority_class* pour le suréchantillonnage. Pour la normalisation, nous avons ajusté les vecteurs de comptage de mots en calculant la moyenne et l'écart-type sur chaque colonne. Elle améliore la convergence et la stabilité du modèle en évitant que les caractéristiques ayant des valeurs plus élevées affectent l'apprentissage. Dans la classe *Data*, les données d'entraînement sont

divisées en un ensemble d'entraînement et un ensemble de validation avec une partition de 80/20. Cette division est cruciale pour l'évaluation de la performance du modèle sur les données non vues et pour détecter le surapprentissage. Le suréchantillonnage permet de dupliquer les exemples de la classe minoritaire (Les documents ayant pour étiquette 1) dans un ensemble d'entraînement. Dans le cas où les classes sont déséquilibrées, le modèle va être en faveur de la classe majoritaire. Le suréchantillonnage est responsable de la balance du nombre d'exemples dans chaque classe, ce qui aide le modèle à mieux apprendre et à repérer la classe minoritaire.

Les algorithmes d'apprentissage sont la régression logistique avec la régularisation L2 et la pondération des classes. La régression logistique utilise une fonction sigmoïde pour prédire la probabilité d'appartenance de chaque document à l'une des catégories binaires. La régularisation L2 limite la complexité du modèle ce qui permet d'éviter le surapprentissage. La pondération des classes gère les déséquilibres dans les données.

Le taux d'apprentissage (*learning_rate*) contrôle l'amplitude des ajustements de poids pour chaque itération. Le nombre d'itérations (*iterations*) permet d'assurer une bonne convergence. Le seuil de classification (*threshold*) est appliqué pour transformer les probabilités en classes binaires. Les hyperparamètres tels que le taux d'apprentissage le nombre d'itérations, le paramètre de régularisation (*lambda_l2*) et le seuil de classification sont respectivement fixé à 0.001, 1000, 0.01 et 0.645. Nous avons trouvé ces valeurs en faisant des essais et erreurs pour maximiser le F1 score.

2.2. Fichier *classement_improved.py*

La méthode d'extraction utilisé dans ce code est encore *load_data*. La méthode *load_data* fonctionne de la même manière que le premier modèle.

La méthode de prétraitement utilisé est *preprocess_data* pour la réduction de dimensionnalité et la normalisation des données. La division des données et les partitions sont gérées par la fonction *train_test_split*. La réduction de dimensionnalité, comme son nom l'indique, transforme la matrice de comptage de termes en une matrice de petite dimension pour faciliter la manipulation des données sans perdre de l'information et se fait à l'aide de la méthode *TruncatedSVD*. De plus, elle accélère l'entraînement des données et évite le surapprentissage. Pour la normalisation, elle est très similaire qu'au premier modèle mais on utilise ici *StandardScaler* pour centrer les données et uniformiser l'échelle des caractéristiques.

L'algorithme d'apprentissage est la régression logistique avec l'optimisation des hyperparamètres et du seuil de classification. Pour la recherche du seuil optimal, la fonction *find_best_threshold* cherche différents seuils de probabilité et détermine celui qui franchit un score F1 élevé.

Pour les paramètres de régularisation (*C_values*), nous avons choisi les valeurs suivantes 0.001, 0.01, 0.1 et 1. Pour ajuster le poids des classes, nous utilisons le paramètre *class_weight* qui prend une des deux options: *none* pour indiquer qu'il n'y a aucune pondération ou *'balanced'* qui est la pondération ajustée en fonction de la fréquence des classes. Nous avons fixé le nombre de composantes (*n_components*) à 550 pour réduire la dimension. Enfin, le seuil de classification (*threshold*) est testé pour les valeurs dans l'intervalle 0.45 à 0.8 par pas de 0.005.

3. Résultats

3.1. Fichier *classement.py*

Le meilleur score F1 est obtenu avec un taux d'apprentissage de 0.001, ce qui montre qu'un taux plus faible favorise une meilleure convergence du modèle. De plus, l'utilisation d'une pondération équilibrée des classes améliore systématiquement les performances, atteignant un score F1 de 0.5714 avec ce taux d'apprentissage. Cela souligne l'importance de corriger le déséquilibre des classes pour améliorer la précision du modèle.

La fonction de perte montre une diminution progressive au fil des itérations, avec une réduction plus rapide au début, indiquant une bonne phase d'apprentissage. La perte atteint 0.3532 après 900 itérations (Annexe 5.3), ce qui démontre une convergence stable. Enfin, la régularisation L2 contribue à stabiliser l'entraînement et à éviter le surapprentissage, ce qui aide le modèle à mieux généraliser et à offrir des performances plus robustes.

Taux d'apprentissage	Pondération des classes (Aucune)	Pondération des classes (Équilibrée)
0.001	0.4919	0.5714
0.01	0.4538	0.4878
0.1	0.4324	0.4470
1	0.4274	0.4361

3.2. Fichier *classement_improvement.py*

Les résultats montrent que le meilleur score F1 est obtenu avec un taux d'apprentissage de 0.001 et avec la pondération des classes équilibrée (F1 = 0.5989). Cela suggère qu'un taux d'apprentissage plus faible permet une meilleure convergence et que l'ajustement de la pondération des classes améliore la performance du modèle en équilibrant les classes déséquilibrées. L'absence de pondération des classes donne des résultats légèrement inférieurs, avec un score F1 de 0.4936 pour un taux d'apprentissage de 1, ce qui montre que la pondération est bénéfique dans ce cas.

Concernant la réduction de la dimensionnalité, l'utilisation de la Truncated SVD a permis de réduire le temps de calcul tout en maintenant une performance similaire. Cela est essentiel pour traiter efficacement de grandes quantités de données. De plus, ne pas fixer la graine aléatoire a permis d'obtenir des résultats plus variés, ce qui a favorisé une exploration plus large des comportements du modèle. En résumé, le modèle fonctionne le mieux avec un taux d'apprentissage de 0.001 et la pondération des classes équilibrée, tout en optimisant la réduction de la dimensionnalité avec SVD pour améliorer l'efficacité.

Taux d'apprentissage	Pondération des classes (Aucune)	Pondération des classes (Équilibrée)
0.001	0.2048	0.5989
0.01	0.4559	0.5962
0.1	0.4820	0.5962
1	0.4936	0.5956

3.3. Comparaison avec SVM et Forêt aléatoire

La régression logistique avec pondération équilibrée et un faible taux d'apprentissage (0.001) surpasse les autres méthodes, atteignant un score F1 de 0.5989, démontrant son efficacité à gérer les déséquilibres de classes. Bien que le SVM atteigne une performance modérée avec un taux élevé (0.5637), il reste inférieur à la régression logistique, surtout aux taux d'apprentissage faibles où la pondération des classes est moins influente. La Forêt Aléatoire, même avec plusieurs estimateurs et pondération équilibrée, montre des performances F1 globalement inférieures, ce qui pourrait indiquer que les arbres de décision gèrent moins bien ce déséquilibre. En conclusion, la régression logistique pondérée est la méthode la plus adaptée pour ces données.

4. Discussion

Notre approche avec la régression logistique a plusieurs avantages pour ce projet de classification de texte. Elle est rapide à entraîner, simple à utiliser, et permet de comprendre l'effet des caractéristiques sur les prédictions. En utilisant une réduction de dimensionnalité avec Truncated SVD, nous avons pu réduire le nombre de caractéristiques, rendant le modèle plus efficace tout en conservant les informations importantes. La normalisation des données aide également le modèle à mieux fonctionner.

Cependant, cette approche a des limites. La régression logistique fonctionne mieux quand les données ont une relation linéaire, ce qui n'est pas toujours le cas ici. De plus, même si nous avons ajusté les paramètres et le seuil de décision pour améliorer le score F1, le modèle peut encore avoir du mal avec les classes minoritaires.

Pour améliorer les performances, nous pourrions essayer des modèles plus avancés, comme les modèles d'arbres (ex. Forêt Aléatoire) ou des modèles d'ensemble qui pourraient mieux capturer des relations complexes. Nous pourrions aussi expérimenter d'autres modèles et seulement garder les plus performants.

5. Annexes

5.1. Résultat finale de l'étape 1

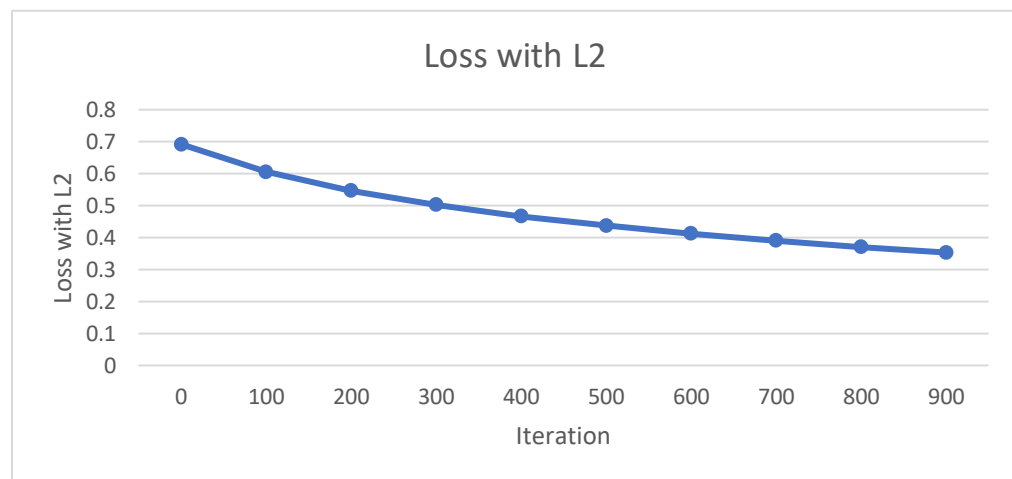
```
Iteration 0, Loss with L2: 0.6915949639277492
Iteration 100, Loss with L2: 0.5780831646423061
Iteration 200, Loss with L2: 0.5125067893796428
Iteration 300, Loss with L2: 0.46877811942563513
Iteration 400, Loss with L2: 0.43654772456236046
Iteration 500, Loss with L2: 0.41112386572705895
Iteration 600, Loss with L2: 0.3901109462047508
Iteration 700, Loss with L2: 0.37216526664836824
Iteration 800, Loss with L2: 0.3564743869819842
Iteration 900, Loss with L2: 0.34251591044618235
Validation Metrics - Accuracy: 0.7517241379310344, Precision: 0.5098039215602973, Recall: 0.6499999999864583, F1 Score: 0.5714285664911646
Submission file 'submission.csv' created.
```

5.2. Résultat finale de l'étape 2

```
C: 0.001, class_weight: None, F1 Score: 0.20475319926873858
C: 0.001, class_weight: balanced, F1 Score: 0.5989492119089317
C: 0.01, class_weight: None, F1 Score: 0.4559023066485753
C: 0.01, class_weight: balanced, F1 Score: 0.5961538461538461
C: 0.1, class_weight: None, F1 Score: 0.48205128205128206
C: 0.1, class_weight: balanced, F1 Score: 0.5961538461538461
C: 1, class_weight: None, F1 Score: 0.49363867684478374
C: 1, class_weight: balanced, F1 Score: 0.5956331877729257

Best parameters : {'C': 0.001, 'class_weight': 'balanced'}
Best F1 score with the best_params : 0.5989492119089317
Best Threshold: 0.49500000000000005 with F1 Score: 0.6661031276415892
Predictions saved into submission_improved.csv
```

5.3. Graphe de la perte L2 pour l'étape 1



5.4. Résultat de la méthode SVM

Taux d'apprentissage	Pondération des classes (Aucune)	Pondération des classes (Équilibrée)
0.001	0	0.3985
0.01	0	0.3985
0.1	0	0.4859
1	0.316	0.5637

5.5. Résultat de la méthode Forêt Aléatoire

N Estimateurs	Pondération des classes (Aucune)	Pondération des classes (Équilibrée)
50	0.0570	0.2061
100	0.0534	0.2134
500	0.0292	0.1764
1000	0.0251	0.1736

6. Références

Train_test_split :

https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.train_test_split.html

LogisticRegression :

https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html

TruncatedSVD :

<https://scikit-learn.org/dev/modules/generated/sklearn.decomposition.TruncatedSVD.html>

StandardScaler :

<https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.StandardScaler.html>