UNIVERSITÉ DE MONTRÉAL
Département d'informatique et de recherche opérationnelle // MILA
Devoir de Hakim Simo (20096040)

Automne 2023
Guillaume Rabusseau

IFT 3395-6390A A23

Rapport – Compétition Kaggle 1..

1 Data handling.

La première étape de mon projet fut la préparation des données. J'ai commencé par importer mes données d'entraînement et de test.

Pour un meilleur traitement, j'ai séparé mes données d'entraînements en features et labels. Labels one-hot pour la méthode de régression logistique que j'ai implémenté et labels classiques pour SVM & XGBoost que j'ai importé.

J'ai ensuite procédé au shiffling de mes données, ceci étant une bonne pratique pour éviter que nos modèles n'apprennent des patterns sur le dataset pouvant biaiser le modèle. Après, j'ai procédé au split en utilisant ma variable ratio que j'ai set a 0.8. Il est a noter que durant la compétition j'ai modifié cette variable à plusieurs reprises.

Au final, j'ai fini par diviser mes données en 80% pour le training et 20% pour la validation. Ce qui est plutôt raisonnable.

La prochaine étape fut la normalisation des données pour améliorer la convergence des algorithmes, j'ai appliqué la même normalisation a l'ensemble de test pour éviter d'introduire un biais aussi.

Bref, à la fin de cette étape, j'ai donc toutes les briques pour commencer a entraîner mes modèles

2 Implémentation Régression Logistique

La régression logistique m'a posé quelques soucis, n'ayant pas trop d'intuition mathématique, mais avec l'aide des démos vus en cours, j'ai appris a implémenter d'une manière similaire a l'implémentation binaire faite en démo. En effet, la régression logistique consiste à implémenter la fonction sigmoide pour arriver aux probabilités de classes, mais dans notre cas, on utilise la fonction softmax pour généraliser le problème en un problème multinomial. J'ai appliqué cette fonction aux résultats de la combinaison linéaire de mes entrées, pondérée par des poids et un biais, dans le but de minimiser la fonction de coût en procédent à de la descente de gradiant au fur et à mesure des epochs avec le taux d'apprentissage pour moduler l'amplitude des ajustements apportés aux poids à chaque mise à jour.

Au final je me suis retrouvé avec un modèle assez robuste prenant donc deux hyperparamètres, mon taux d'apprentissage et mes epochs, que j'ai peaufiné en faisant un grid search des meilleurs

paramètres et finissant avec un resultat convaincant sur kaggle battant la baseline. (Finissant avec 0.755 battant largement la baseline de 0.718 pour logreg)

3 Import SVM Linéaire et XGBOOST

Assumant que les données sont linéairement séparables, et ayant beaucoup de features, j'ai commencé par importer SVM linear de Scikit learn, espérant obtenir un meilleur score sur kaggle, n'ayant qu'un seul hyperparamètre, ce n'est pas difficile a traiter comme algorithme et si les résultats étaient prometteurs, j'aurais tenté d'optimiser avec RBF probablement. Le score maximal que j'ai atteint étant de 0.756, j'ai abandonné cette idée et ai procédé a l'implémentation de XGBoost.

XGBoost étant préconisé pour les compétitions kaggle, je l'ai implémenté sans trop y repenser mais la performance de l'algo s'explique surtout par la régularisation L1 et L2 du modèle, ayant eu plusieurs problèmes avec mes modèles précédents qui prédisaient constamment la classe 0 au début a cause de la surreprésentation de la classe, ce problème n'est pas apparu une fois avec XGBoost, m'apprenant l'importance des régularisations. Ce modèle m'a aussi donné mes meilleurs performances, finissant a 0.783, proche du maximum qui était de 0.788. J'ai passé un peu de mon temps à essayer de trouver le moyen d'atteindre ce score maximal mais je pense qu'il me manque quelque chose que je ne manquerais pas de demander aux démonstrateurs au prochain cours.

Aussi, pour XGBoost, en utilisant dart au lieu de gbtree, les résultats sont un peu plus précis mais le code prend beaucoup plus de temps a rouler. mais sinon les paramètres s'expliquent d'eux mêmes :

```
self.params = { # Paramètres de l'algorithme
    'objective': 'multi:softmax', # Car classification multiclasse
    'num_class': num_class, # déduit plus haut
    'booster': 'dart', # gbtree est plus rapide though mais dart prévient le surappren
    'eval_metric': 'merror', # car classification multiclasse
    'eta': eta, # learning rate
    'max_depth': max_depth, # profondeur maximale de l'arbre
}
```

Encore une fois, pour ces deux modèles, j'ai procédé a un grid search exhaustif d'hyperparamètres.

4 Trivia

Dans mon code, vous retrouverez 3 fonctions ; startLR, qui fais l'optimisation d'hyperparamètre et entraine mon modèle pour la régression logistique, startSVM_Linear qui fait de même comme son nom l'indique sur la régression linéaire, et startXGBOOST pour notre dernière implémentation. Veuillez noter l'utilisation du multithreading qui m'a un tant soi peu aidé mais qui n'est pas particulièrement efficace, j'aurais peut-être pu faire une approche d'implémentation GPU ?

Puis 3 autres fonctions, plot_results_SVM_Linear, plot_results_LR et plot_results_XGB qui sont self explanatory et dont les résultats se trouveront en annexe.

Finalement, j'ai 2 helper fonctions, softmax pour le softmax. (obviously) et save_predictions_to_csv pour enregistrer mes résultats.

/ ! Si vous voulez faire rouler le code avec le notebook fourni, juste lancez les 3 fonctions start, et magie :)

5 Annexe - Graphs

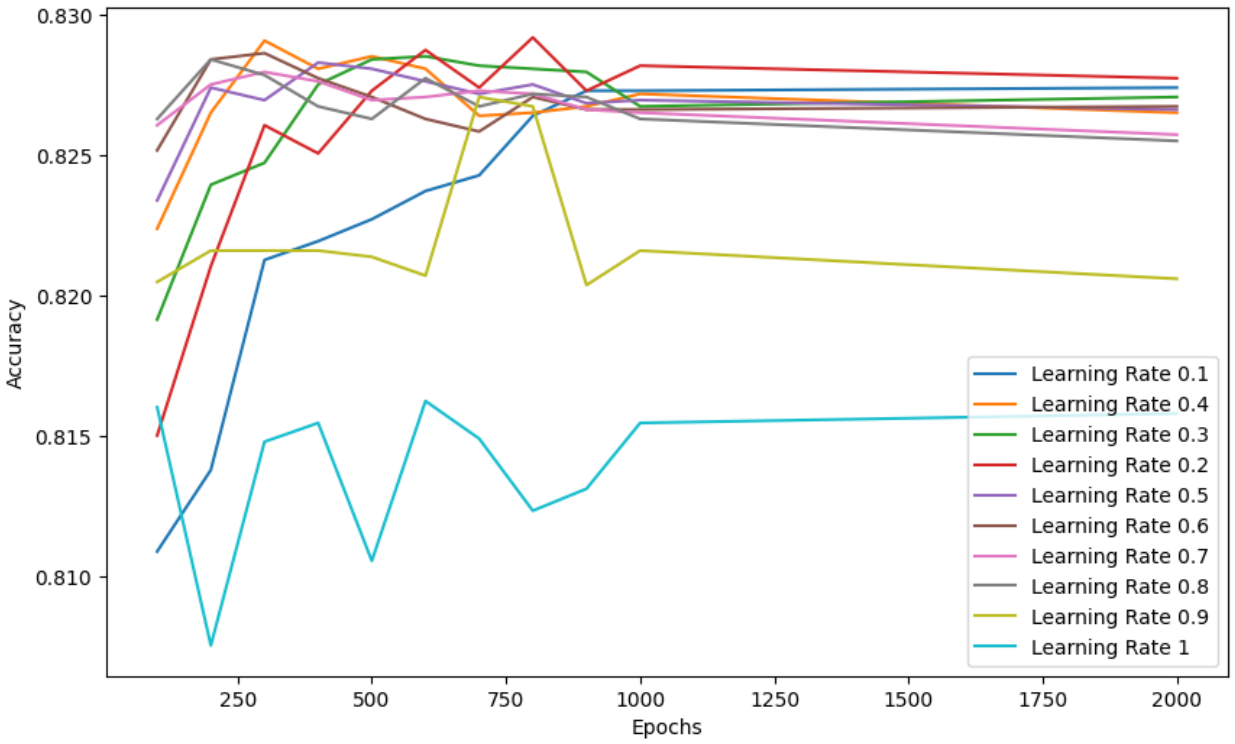

FIGURE 1 – Logistic Regression

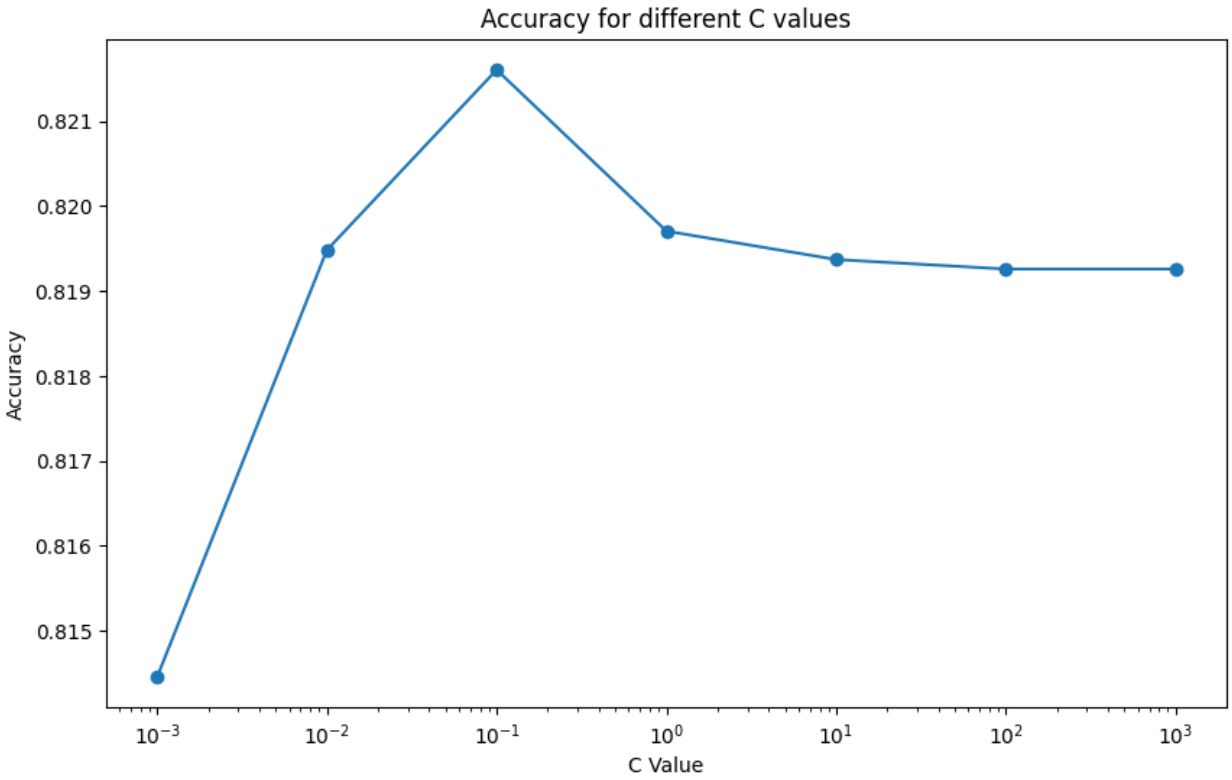

FIGURE 2 – SVM Linear

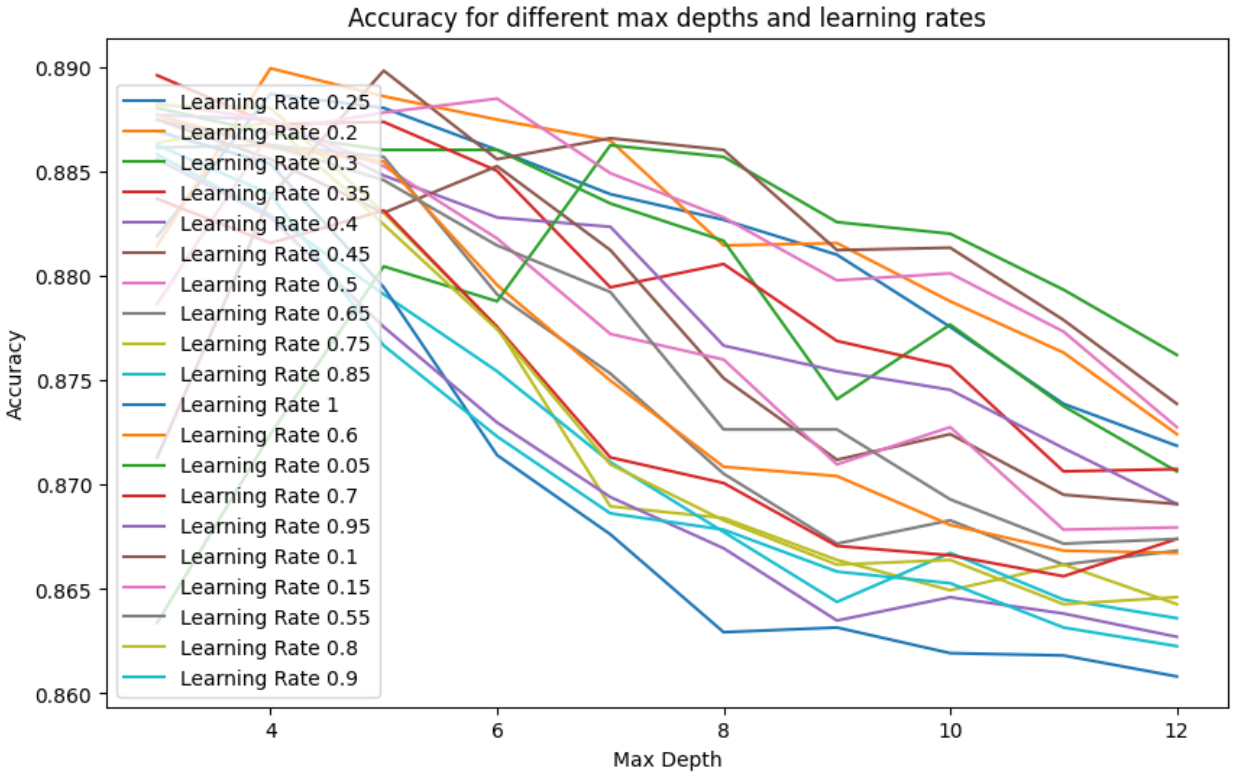

FIGURE 3 – XBGOOST