CHAHBAOUI Mohammed
HITCHON Thomas
KARATAS Musab

Rapport Projet algorithme science de données et apprentissage

I. Introduction

Contexte général

Dans le contexte de l'agriculture moderne, l'exploitation des données satellites constitue une opportunité essentielle pour surveiller, analyser et optimiser les pratiques agricoles. La classification des images satellitaires, en particulier, permet d'identifier les types de cultures sur des parcelles agricoles à grande échelle. Cette approche offre des applications variées, comme l'estimation des rendements, la gestion des ressources ou encore le suivi de l'impact des conditions climatiques.

Le problème traité dans ce rapport repose sur l'analyse d'images satellites multispectrales acquises sur une période donnée, afin de classifier chaque parcelle agricole selon son type de culture.

Description du problème

Les données utilisées proviennent d'une série d'images satellites capturées mensuellement sur des champs agricoles, couvrant une période de 10 mois (de février à novembre). Chaque entrée du jeu de donnée de composé de:

- 10 images : une par mois de Février à Novembre
- avec 3 canaux spectraux : B08 proche infrared, B04 rouge, B03 vert
- dimension des images : 32x32 pixels
- la classe associé à la série d'image (le type de culture)

L'objectif est de classifier ces données en fonction du type de culture parmi 20 classes distinctes, telles que le blé tendre de printemps ou d'hiver, le maïs, ou encore la luzerne.

Problématiques

Comment concevoir, entraîner et évaluer des modèles de classification robustes capables de prédire les types de cultures agricoles à partir de données d'images satellites temporelles? Comment prendre en compte le déséquilibre des classes?

Quels sont les meilleurs modèles? Plus un modèle est complexe, mieux est-il pour prédire et classer les données?

Quels paramètres influencent sur les performances des modèles et pourquoi?

II. Description de notre proposition pour aborder la problématique

Afin de résoudre ce problème nous allons utiliser plusieurs modèles de classifications avec différentes combinaisons d'hyperparamètres, puis nous allons les comparer afin de voir lesquels performent le mieux et sur quels points.
Les modèles que nous avons choisis sont les suivants:

Régression logistique :

Modèle simple, interprétable, et efficace pour des tâches de classification multiclasse avec des données tabulaires. C'est un bon modèle baseline pour commencer.

Random Tree Forest :

Ce modèle présente certains avantages. Il est résilient au surapprentissage, gère bien les déséquilibres des classes (important pour nous) et permet d'ajuster le poids en fonction de leur représentation dans le jeu de données. Le modèle est donc adapté pour de la prédiction multiclasses.

Cependant il est moins performant avec des données qui sont hautement dimensionnelles.

CNN (Convolutional Neural Networks) :

Pour finir, nous choisissons ce modèle car il est puissant pour les données d'images, capable d'extraire des caractéristiques visuelles complexes.

Cependant, le modèle nécessite beaucoup de données, que nous possédons, et un bon prétraitement.

On a choisi ces 3 méthodes car on voulait partir d'un modèle le plus simple possible, la régression logistique. On fait ensuite un random tree forest qui est plus compliqué et on finit par le CNN qui est le modèle le plus complexe qu'on utilise.
Nous utilisons ces 3 modèles qui ont des niveaux de complexités différentes pour voir si plus un modèle est compliqué, mieux les résultats seront concernant notre base de données.

Pendant le test d'un modèle, nous voulons pousser au maximum les capacités du modèle en question. Nous allons modifier les hyperparamètres des modèles et voir l'impact qu'ils ont sur les performances lors des phases de test.

Nous avons aussi pensé à une autre approche: entraîner et tester un modèle sur quelques mois spécifiques. Nous pouvons par exemple couper les mois d'été puis tester le modèle sur

les mois d'hiver et inversement, ou bien carrément prendre 2 mois par 2 mois pour voir quels sont les mois qui permettent au modèles de mieux prédire le type d'agriculture. Si par exemple les mois de Juillet et Août permettent une meilleure prédiction que les mois d'Octobre et Novembre, nous pouvons leur donner un plus gros poids lorsqu'on va utiliser nos modèles. Nous donnons donc plus de poids aux mois contenant le plus d'informations discriminantes

Cette théorie est posée de manière générale pour tous les types d'agricultures et pour tous les mois mais nous pouvons encore plus creuser. Certaines agricultures pourraient être plus reconnaissables certains mois tandis que d'autres pourraient être plus reconnaissables d'autres mois.

Par exemple, si une agriculture est plus reconnaissable les mois d'été et pas en hiver, on fait en sorte que le poids des mois d'été est plus important si on pense prédire cette agriculture. Mais si une agriculture est plus reconnaissable les mois d'hiver et pas en été, on fait l'inverse. Mais cela s'annonce très compliqué.

III. Description du protocole expérimental

1. Régression logistique

    a) Préparations de données

Nous avons commencé par aplatir les données afin de passer à une structure bidimensionnelle requise pour utiliser la régression logistique, puis nous avons effectué une standardisation afin d'essayer t'atteindre une convergence plus rapide du modèle (et éviter tout problèmes d'influence disproportionnée ou de pénalisation inégales avec la régularisation).
Afin de prendre en compte les déséquilibres de classes nous utilisons l'attribut class_weights (sur tous les modèles) qui donne un poids à chaque classe en fonction de leur fréquence dans le jeu d'entraînement.

    b) Modélisation

Nous avons réalisé deux modèles de régression logistique, le premier sans régularisation.
Après avoir testé le premier modèle, nous avons utilisé GridSearchCV avec les paramètres 'C' : [0.01, 0.1, 1] et 'penalty': ['l2', 'none'] sur une fraction du jeu de données afin d'estimer si une régularisation l2 serait positif et pour quelle valeur de force C.
Les résultats données par GridSearchCV étant {'C': 0.01, 'penalty': 'l2'}, nous avons implémenté un deuxième modèle avec cette force de régularisation l2.

    c) Entraînement et Évaluation

Les deux modèles ont étés entraînés de la même manière avec lbfgs comme algorithme d'optimisation car étant le plus efficace sur cette taille de données.

Après l'entraînement nous avons effectué des prédictions et mesurer les performances des modèles avec différentes métriques (F1-score, la précision, le recall, l'accuracy, le support) et affiché la matrice de confusion afin de comparer leurs performances.

2. Forêts aléatoires

    a) Préparations de données

Comme pour les régressions logistiques, nous avons aplati les données mais il n'était pas nécessaire de les normaliser car l'échelle des données n'est pas prise en compte par les forêts aléatoires. Nous avons également utilisé class_weight pour équilibrer le jeu de données.

    b) Modélisation

Nous avons exploré plusieurs paramètres afin de trouver les meilleurs grâce à RandomizedSearchCV. Ce qui nous à permis de retenir deux modèles de forêt aléatoire.

Le premier n'est pas limité en profondeur maximale, sa valeur de nombre minimum d'échantillons requis dans un nœud pour qu'il puisse être divisé (min_samples_split) est de 2 et son nombre minimum d'échantillons requis dans une feuille (min_samples_leaf) est de 1. Ces valeurs sont faibles afin de permettre des arbres plus complexes pour capturer plus de complexité des données mais le modèle est donc à risque de surapprentissage.

L'autre modèle est limité à 10 en profondeur maximale et à des valeurs supérieures pour les deux autres paramètres respectivement 5 (min_samples_split) et 4 (min_samples_leaf) cela devrait nous donner un modèle plus simple mais pouvant être à risque de sous-apprentissage.

    c) Entraînement et Évaluation

Tout comme pour les régressions, l'entraînement et les évaluations des modèles ont été réalisés de façon similaire.

3. Réseau de convolution

    a) Préparations de données

Dans le cas des CNN il a d'abord fallu transposer les données afin de correspondre au format des modèles Conv3D de Tensorflow. Puis nous avons effectué une normalisation min max des données afin de les mettre sur une échelle commune pour prévenir des biais liés aux grandes valeurs.

    b) Modélisation

Modèle 1 : Architecture simple avec 2 couches convolutionnelles

Le premier modèle est une architecture de réseau convolutif relativement simple, composée de deux couches convolutionnelles, suivies de couches de pooling, et se terminant par des couches entièrement connectées. La structure de ce modèle est la suivante :

- Première couche convolutionnelle (Conv3D) : 32 filtres de taille 3×3×3, fonction d'activation Relu. Elle est suivie d'une couche de MaxPooling3D avec une taille de pool 2×2×2 pour réduire les dimensions.

- Deuxième couche convolutionnelle (Conv3D) : Similaire à la première, mais avec 64 filtres pour extraire des caractéristiques plus complexes. Cette couche est également suivie d'une couche de MaxPooling3D.

- Couches entièrement connectées (Dense) : Après l'aplatissement des données, une couche dense de 128 neurones est ajoutée avec une activation ReLU. Enfin, la couche de sortie est une couche dense avec 20 neurones (correspondant aux 20 classes), avec une fonction d'activation softmax permettant de réaliser la classification multiclasses.

Modèle 2 : Architecture plus complexe avec régularisation et BatchNormalization
Le second modèle a été conçu pour tenter de résoudre un problème potentiel de surapprentissage observé dans le premier modèle. Pour cela, plusieurs techniques ont été ajoutées :

- Première couche convolutionnelle (Conv3D) : 32 filtres de taille 3x3x3 et utilise également la régularisation L2. Elle est suivie d'une couche de MaxPooling3D et d'une BatchNormalization pour normaliser les activations et accélérer l'entraînement.

- Deuxième couche convolutionnelle (Conv3D) : Une autre couche de convolution avec 64 filtres et régularisation L2 est ajoutée, suivie d'une couche de MaxPooling3D et de BatchNormalization.

- Couches entièrement connectées (Dense) : Après l'aplatissement des données, une couche dense avec 128 neurones et activation ReLU est ajoutée. La régularisation Dropout est utilisée sur cette couche (avec un taux de 50%) pour réduire encore le risque de surapprentissage. Enfin, la couche de sortie utilise 20 neurones avec activation softmax pour la classification multiclasses.

c) Entraînement et Évaluation

Les modèles sont compilés avec l'optimiseur Adam pour une convergence rapide et efficace, et utilisent sparse_categorical_crossentropy comme fonction de perte, adaptée aux problèmes de classification. L'entraînement se fait sur un nombre d'époques défini (15 pour le premier modèle et 20 pour le second), avec un batch size de 32. L'EarlyStopping est utilisé pour le second modèle pour surveiller l'amélioration de la perte de validation (val_loss) pendant l'entraînement. Si la performance de validation cesse de s'améliorer, l'entraînement est interrompu afin d'éviter le surentraînement.
Au terme de l'entraînement les modèles sont évalués avec les différentes métriques.

4. Modèles avec poids sur les mois
    a) préparation des données, entraînements et tests

Pour tester quels mois contenaient le plus d'informations discriminantes concernant les types des champs, nous les avons testés indépendamment deux par deux.
Pour cela on a enlevé 8 mois sur 10 et on a fait les entraînements et tests sur 2 mois à chaque fois (Février-Mars, Avril-Mai, Juin-Juillet, Août-Septembre, Octobre-Novembre).

Les données passent donc de dimensions (2500, 10, 32, 32, 3) à (2500, 2, 32, 32, 3).

Ensuite on fait une régression logistique pour chaque paire de mois et on regarde quels mois permettent de mieux faire les prédictions lors des tests.

Après avoir fait les tests, nous apercevons que la paire de mois qui a les meilleures statistiques d'accuracy est la paire Avril-Mai.

    b) globaliser l'idée

Maintenant qu'on a repéré les 2 mois qui discriminent le mieux, on veut leur donner plus de poids pendant l'entraînement et les tests de notre modèle.

IV. Présentation des résultats et discussions

1. Régressions logistiques

Nous avions commencé par un modèle entraîné avec l'algorithme d'optimisation saga, mais cela rendait notre modèle beaucoup trop lent (60 min d'exécution) même sur un petit nombre d'itération ce qui le rendait impraticable. Par le suite nous avons utilisé lbfgs.

Accuracy : 0.62 0.63

	Modèle 1 sans régularisation l2			Modèle 2 avec régularisation l2		
	Précision	Rappel	F1-score	Précision	Rappel	F1-score
marco avg	0.43	0.44	0.42	0.47	0.46	0.45
weighted avg	0.71	0.62	0.63	0.73	0.63	0.64

Les résultats montrent que, bien que le modèle 1 atteint une précision pondérée décente, ses scores macro (non pondérés) sont faibles. Cela indique qu'il ne traite pas bien les classes minoritaires ou moins représentées.

Le modèle avec pénalisation montre une légère amélioration en termes d'accuracy et de scores pondérés. Les scores macro sont également légèrement meilleurs, indiquant une meilleure prise en compte des classes minoritaires.

Les deux modèles n'arrivent pas à prédire correctement les classes très minoritaires mais le modèle avec pénalisation semble tout de même mieux équilibrer les prédictions pour certaines d'entre elles.

Le modèle avec pénalisation montre une légère amélioration par rapport au modèle sans pénalisation, la pénalisation L2 agit comme un régularisateur, limitant la magnitude des coefficients. On observe une légère amélioration de la gestion des classes minoritaires.

Malgré un nombre d'itération maximal élevé, aucun des deux modèles n'a réussi à converger, ce qui limite leurs performances.

2. Forêts aléatoires

Hypermaratères des deux modèles:

- Modèle 1: nombre d'arbres: 100, max_depth: None, min_samples_split : 2, min_samples_leaf : 1

  Cela devrait permettre une modélisation profonde et flexible qui peut s'adapter aux données complexes, mais risque de surapprentissage.

- Modèle 2: Nombre d'arbres: 50, max_depth: 10, min_samples_split: 5, min_samples_leafmin : 4

  Cette configuration impose des contraintes qui réduisent la complexité et la profondeur du modèle, ce qui devrait améliorer la généralisation et limiter le surapprentissage.

Accuracy :

- Modèle 1 : 0.71
- Modèle 2 : 0.73

Bien que les deux modèles aient des performances similaires en termes d'accuracy, le modèle 2 est légèrement meilleur.

Précision, rappel et F1-score :

Les deux modèles montrent des disparités importantes dans leurs performances par classe. Les classes majoritaires comme Blé tendre d'hiver, Colza d'hiver, et Prairie permanente -

herbe ont de bons scores, ce qui reflète le fait que les données sont déséquilibrées. Les classes minoritaires (par exemple, Avoine d'hiver, Maïs ensilage, et Triticale d'hiver) ont des scores de 0.0, indiquant que les modèles ont du mal à les reconnaître.

Pour le modèle 2 Certaines classes comme Orge d'hiver et Tournesol bénéficient d'une meilleure performance en rappel, ce qui peut être attribué à l'effet des contraintes sur la profondeur (max_depthmax) et le nombre minimal d'échantillons (min_samples_leaf) qui favorisent la généralisation.

	RandomForest Modèle 1			RandomForest Modèle 2		
	Précision	Rappel	F1-score	Précision	Rappel	F1-score
marco avg	0.53	0.45	0.45	0.53	0.47	0.46
weighted avg	0.72	0.71	0.69	0.75	0.73	0.71

Les deux modèles ont des scores moyens similaires lorsqu'on considère chaque classe de manière égale, mais ces scores sont faibles en raison des performances médiocres sur les classes minoritaires.

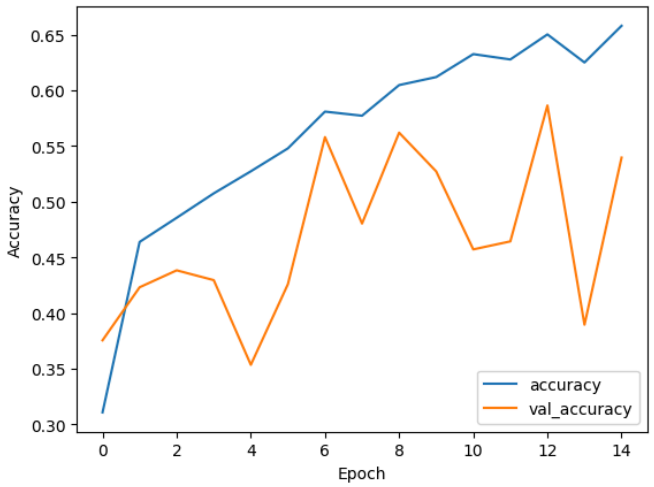
Les scores pondérés, qui tiennent compte de l'importance des classes majoritaires, montrent que le modèle 2 est légèrement meilleur.

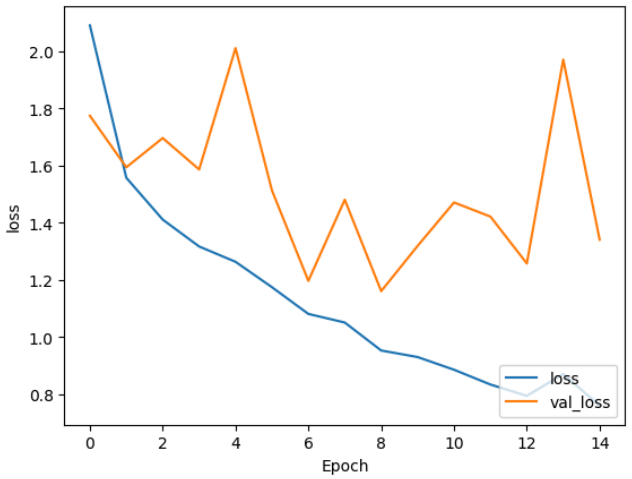
Le modèle 1 à une grande profondeur qui devrait lui permettre de capturer les relations complexes mais semble tendre au surapprentissage, ce qui est visible dans les classes majoritaires où le modèle peut se spécialiser au détriment de la généralisation.

La capacité réduite du modèle 2 peut lui empêcher de capturer des relations très complexes, ce qui se reflète dans les scores légèrement inférieurs pour certaines classes comme Luzerne déshydratée mais favorise la généralisation, en particulier sur des classes où le rappel est meilleur.

3. Réseaux de convolution

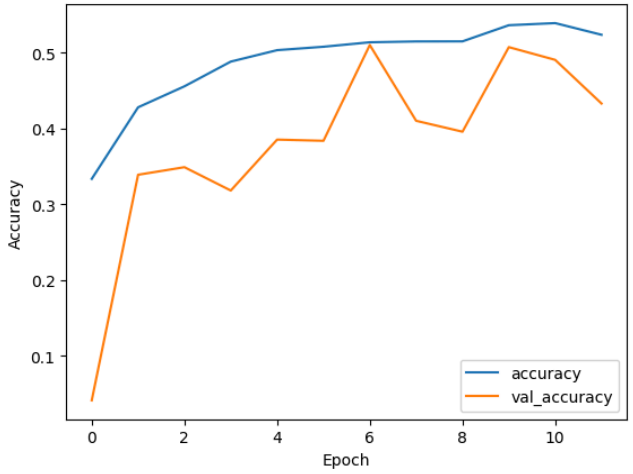
Accuracy et loss du modèle 1.

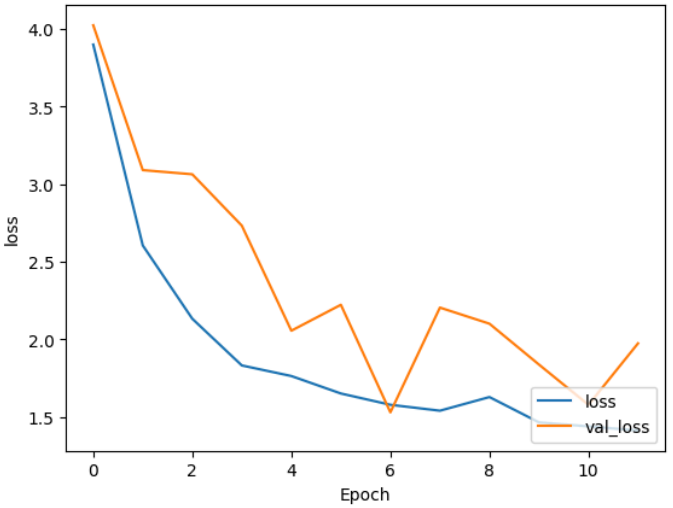




On peut observer l'évolution de l'accuracy et de la loss du premier modèle, on voit ce qui ressemble à un cas de surapprentissage avec la l'erreur de validation (val_loss) qui oscille alors que l'erreur d'entraînement décroit de façon stable.

Accuracy et loss du second modèle.





L'effet d'oscillation de la fonction de perte est moins important grâce à la régularisation et à l'early stopping mais le modèle ne semble pas être plus performant pour autant.

Malgré cela, les performances ne sont pas meilleures que le premier modèle:

	CNN Modèle 1			CNN Modèle 2		
	Précision	Rappel	F1-score	Précision	Rappel	F1-score
marco avg	0.45	0.48	0.39	0.43	0.39	0.34
weighted avg	0.72	0.54	0.57	0.73	0.43	0.48

4. Poids sur les mois

Malheureusement les résultats ne sont pas à la hauteur, on ne remarque pas de grandes différences entre la régression logistique qu'on avait de base et celle avec des poids différents sur les mois d'Avril-Mai.

Nous avons ensuite fait la même chose mais avec le CNN et la seule différence qu'on a aperçu est que l'entraînement du modèle s'arrête plus tôt. Le modèle ne fait que 5 epochs sur 15. En effet nous avons un early stopping pour éviter l'overfitting, cela pourrait dire que le modèle apprend trop avec cette méthode. Nous avons testé sans early stopping et on a des résultats moins bons que nos 2 CNN précédents que ce soit pour l'accuracy ou le F1-Score, sûrement dû à l'overfitting.

Cette approche est donc un échec que ce soit pour la régression logistique ou le CNN.

Nous avons 2 théories sur cet échec:
-Les modèles font de l'overfitting en utilisant cette méthode, d'où les résultats insatisfaisants. Donc l'idée est mauvaise.
-l'erreur vient de nous, nous n'avons pas réussi à bien implémenter ce concept avec nos modèles. Donc le problème viendrait de nous et du code et l'idée ne serait peut-être pas si mauvaise que ça.

(Nous avons tout de même rendu 2 deuxième notebook spécialement pour cette méthode)

V. Conclusion sur le travail réalisé

Au début nous avions commencé avec les données brutes, sans mettre de poids sur les données. Nous avions des résultats décevants, et lors des tests la plupart des entrées étaient classées dans les classes qui avaient beaucoup d'instances. Nous avons donc pondéré les chaque classe et plus en ajoutant du poids et de l'importance aux petites classes.

Le meilleur modèle est le modèle Random Forest 2, il a la meilleure performance globale (weighted avg f1-score et accuracy les plus élevés).
Les modèles les plus robustes pour classes déséquilibrées sont la régression logistique avec régularisation et les deux Random Forest car ils ont la meilleure prise en compte des classes déséquilibrées grâce à une macro avg f1-score plus élevée (0.45-0.46).
Les modèles de CNN sont clairement en retrait par rapport aux forêts aléatoires et à la régression logistique, en termes de performance et de robustesse.

Cependant dans leur globalité ils ont tous des performances plutôt médiocre sur ces données. Le déséquilibre entre les classes et le décalage entre distribution des données d'entraînement et de validation contribue sûrement à ces performances.

Nous aurions pu employer des méthodes de suréchantillonnage sur les petites classes et sous échantillonnage sur les grandes classes pour créer un jeu de données plus équilibré. Nous avons cependant préféré garder le jeu de données intact et essayer plutôt de traiter ces problèmes par l'attribution des poids plus importants aux plus petites classes afin de les rendre plus impactantes tout au long du processus d'entraînement.

D'autres perspectives peuvent être envisagées pour améliorer les résultats comme la combinaison de plusieurs architectures ou des modèles hybrides pouvant travailler chacun à un niveau des données.

Membres de l'équipe et leur contribution:

Nous avons principalement travaillé ensemble et nous nous sommes mutuellement entraidé sur la majorité du projet. Cependant nous avions aussi des rôles spécifiques instaurés au début:
HITCHON Thomas:
CNN / rapport
CHAHBAOUI Mohammed:
Régression logistique / rapport
KARATAS Musab
Random Tree Forest / Poids sur les mois / Rapport