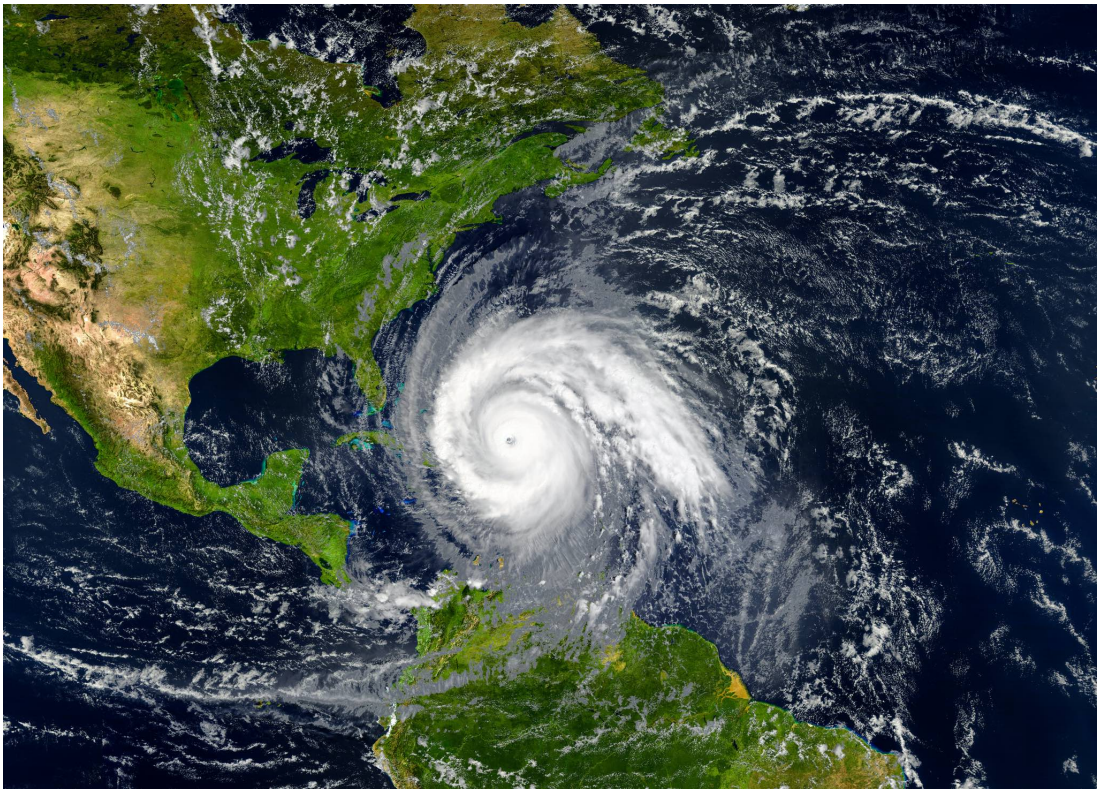
Université de Montréal

Classification of extreme weather events

Kaggle I competition



PREVOT ALEXIA
Pseudo Kaggle : Alex prvt
Matricule : 20278091

IFT6390
Université de Montréal
Fall 2023

9 novembre 2023

Table des matières

Introduction

J'ai eu le plaisir de participer à un concours Kaggle visant à détecter des événements météorologiques extrêmes à partir de données atmosphériques. L'objectif était de concevoir un algorithme d'apprentissage automatique capable de classer automatiquement les variables climatiques en trois catégories : conditions standards, cyclones tropicaux ou rivières atmosphériques.

J'ai commencé par me familiariser avec les données. Puis, après avoir fait quelques étapes de pré-traitement sur les données et les caractéristiques à prendre en compte, les choses sérieuses ont commencé : créer des modèles de classifications. J'ai d'abord fait une régression logistique en codant tout à partir de zéro. Puis, j'ai exploré d'autres modèles, notamment celui de forêts aléatoires que j'ai mis en oeuvre.

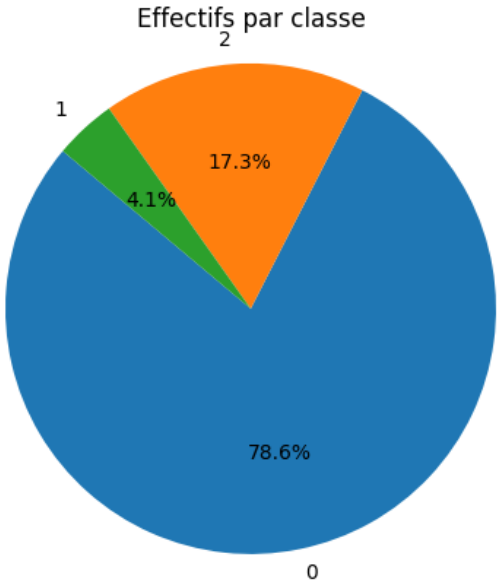
Les performances finales sont évaluées sur l'ensemble de test Kaggle.

1 Feature Design

L'ensemble de données est un sous-ensemble de ClimateNet. Il y a 16 variables atmosphériques telles que la pression, la température et l'humidité.

1.1 Analyse des données brutes

J'ai commencé par par me familiariser avec les données avec la librairie *pandas*. J'ai vérifié s'il n'y avait pas de valeurs manquantes, regardé le noms et le types des colonnes, et visualisé les effectifs par classes des données.



Premier constat : les classes sont très déséquilibré puisqu'il y a une majorité écrasante de la classe 0. Pour le moment j'ai choisi de laisser de côté ce problème et d'y revenir plus tard.

1.2 Standardisation

Par la suite, j'ai standardisé les données en calculant la moyenne et l'écart-type des données d'entraînement (sans prendre "SNo" et "Label") et en appliquant la formule suivante : $z = \frac{x-\mu}{\sigma}$.

Si l'écart type des caractéristiques est différent, leur plage est également différente les unes des autres. Cela réduit l'effet des valeurs aberrantes dans les caractéristiques.

1.3 Données désiquilibrées

Comme vu précédemment, les données sont déséquilibrés alors j'ai pensé qu'il faudrait agir pour rebalancer ces données. La classe la plus minoritaire est la 1 (1825), suivi de la 2 (7756). Il y a alors très peu de données, et si on décidait de sous-échantillonner en enlevant des éléments de la classe 0 (qui contient en 35179) cela enlèverait la majorité des données et on perdrait beaucoup trop d'information. Alors je me suis dis qu'il fallait plutôt sur-échantillonner en copiant des éléments des classes minoritaires pour attendre un jeu balancé. Il y a plusieurs techniques pour le faire et j'ai d'abord tenté d'utiliser le suréchantillonnage SMOTE (avec la librairie *imblearn*).

1.4 Sélection de caractéristiques

Il y a 16 variables dans le jeu de données et afin de savoir à quel point elles sont importantes pour la classification, j'ai réalisé une étude sur ces caractéristiques. Pour se faire j'ai utilisé la bibliothèque *eli*5 que j'avais déjà pu testé dans un autre cours.

D'abord on commence par entraîner un modèle Random Forest, puis on fait des prédictions sur l'ensemble de validation. On calcule ensuite les importances par permutation de ce modèle et enfin on affiche les poids attribués à chaque caractéristique pour ce modèle (on ne prend pas en compte la cible "label" ni le numéro "SNo" car ils n'ont pas d'intérêt pour la prédiction).

Premier constat : la caractéristique "lat" semble être de loin avec "U850" la plus importe avec un grande influence positive sur les résultats. Deuxième constat : certaines caractéristiques comme "PRECT" ou "V850" sont clairement pas très importantes (proches de 0,0001).

J'ai donc fait plusieurs tests en conservant que certaines des meilleures caractéristiques suivantes comme "lat", "U850", "TMQ", "Z200"... Les résultats ne sont pas trop concluant puisque l'accuracy avant la sélection était de 0.884 contre 0.878 après voir moins dépendant des différents tests. Par la suite je n'ai donc pas utilisé cette partie.

Weight	Feature
0.0392 ± 0.0043	lat
0.0248 ± 0.0034	U850
0.0102 ± 0.0009	TMQ
0.0078 ± 0.0029	Z200
0.0055 ± 0.0014	TS
0.0054 ± 0.0020	time
0.0034 ± 0.0014	lon
0.0033 ± 0.0014	UBOT
0.0028 ± 0.0010	TREFHT
0.0025 ± 0.0011	VBOT
0.0023 ± 0.0010	ZBOT
0.0023 ± 0.0011	T200
0.0022 ± 0.0017	PSL
0.0016 ± 0.0011	PS
0.0012 ± 0.0013	Z1000
0.0011 ± 0.0012	QREFHT
0.0007 ± 0.0018	T500
0.0001 ± 0.0011	V850
-0.0001 ± 0.0001	PRECT

2 Algorithmes

2.1 Régression Logistique

La **régression logistique** est une technique courante d'apprentissage supervisé utilisée pour résoudre des problèmes de classification. Elle fonctionne en modélisant la probabilité qu'une observation appartienne à une classe particulière. L'algorithme de régression logistique est itératif et ajuste les poids du modèle pour minimiser la perte prédéfinie, souvent la perte de log-vraisemblance.

Dans mon cas, j'ai implémenté une régression logistique multinomiale à partir de zéro pour effectuer une classification multiclasse. L'algorithme prend en compte des hyperparamètres tels que le taux d'apprentissage (learning rate), le nombre maximal d'itérations (max_epoch), et un terme de régularisation L2 (reg).

J'ai ajouté une colonne supplémentaire "Biais" remplie de 1 pour toutes les observations, ce qui permet au modèle de capturer un terme de biais. Le terme de biais est essentiel pour ajuster correctement le modèle et garantir qu'il puisse apprendre des données de manière appropriée. Lorsque je n'en avais pas mis au début, les prédictions étaient soient toutes de 0, 1 ou que de 2.

Comme il y a 3 classes, j'ai utilisé un encodage one-hot et la fonction softmax pour calculer les probabilités des classes. Ensuite, à chaque itérations, les poids sont mis à jour à l'aide du gradient de la fonction de coût.

2.2 Random forest

Le modèle **Random Forest** est une technique d'apprentissage supervisé utilisée pour résoudre des problèmes de classification et de régression. Cela permet de combiner la sortie de plusieurs arbres de décision pour aboutir à un résultat unique. Chaque arbre de décision est formé sur un sous-ensemble aléatoire des données et des caractéristiques, ce qui ajoute une composante de variabilité et renforce la robustesse des prédictions.

Dans mon implémentation avec la bibliothèque scikit-learn, j'ai configuré le modèle Random Forest avec certains hyperparamètres. J'ai uniquement spécifié le nombre d'arbres dans l'ensemble (n_estimators) et la profondeur maximale de chaque arbre (max_depth). Ces paramètres sont essentiels pour contrôler la complexité du modèle.

3 Méthodologie

J'ai reflechi à plusieurs possibilités concernant la gestion de l'ensemble d'entraînement et de validation, la régularisation, les techniques d'optimisation, et le choix des hyperparamètres.

3.1 Entraînement et de validation

Tout d'abord, pour garantir une évaluation fiable des modèles, l'ensemble de données d'entraînement a été divisé en deux sous-ensembles : data_train et data_valid. L'entraînement des modèles se sont fait sur data_train et l'évaluation de ceux-ci sur data_valid. A

la fin de l'optimisation des hyperparamètres et des divers choix pour chaque modèle, j'ai ré-entrainé un nouveau modèle sur l'ensemble des données d'entraînement (data_train + data_valid) afin d'avoir un maximum de données. C'est sur ce modèle que je fais la prédiction sur l'ensemble de test avant de faire une submission.

3.2 Régularisation

En ce qui concerne la régularisation, j'ai choisi d'implémenter une **régularisation L2**. Cette régularisation a été appliquée pour limiter les valeurs extrêmes des poids du modèle, afin d'obtenir une meilleure généralisation des modèles possible.

3.3 Optimisation

Il existe diverses techniques d'optimisation pour optimiser les modèles, notamment la descente de gradient stochastique pour la régression logistique. Elle permet d'ajuster itérativement les poids du modèle en utilisant une petite fraction de l'ensemble d'entraînement à chaque itération, ce qui permet d'accélérer la phase apprentissage. Cependant je n'ai pas eu le temps de mettre en oeuvre cette technique.

3.4 Hyperparamètres

3.4.1 Suivis des entraînements

Dans le cas de la régression logistique, j'ai fait en sorte de pouvoir visualiser l'évolution de la loss et de l'accuracy pendant l'entraînement. Cela permet d'éviter du sur/sous-apprentissage et de choisir un bon nombre d'époques.

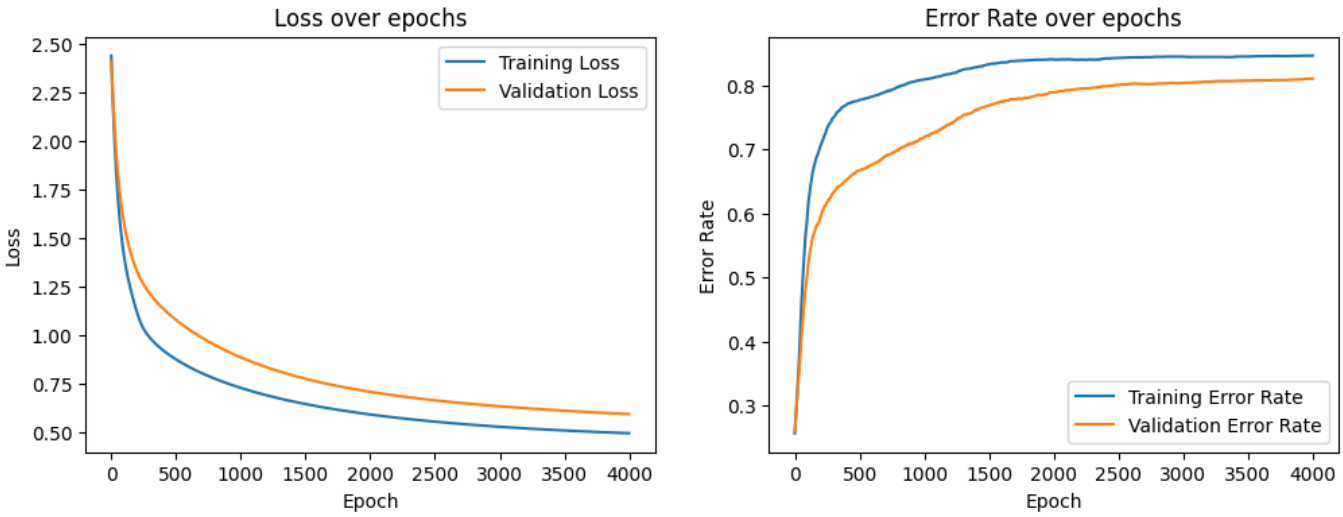


FIGURE 1 – nombre d'époques = 4000, lr = 0.01 et reg = 0.01

Ici, je trouve qu'à partir de 2500/3000 époques, le modèle a suffisamment appris alors pour les prochains modèles de régression logistique, le nombre d'époques sera de 3000.

3.4.2 Meilleurs hyperparamètres

Pour le choix des hyperparamètres, j'ai essayé d'entreprendre une démarche rigoureuse pour trouver les valeurs optimales. Pour le modèle de régression logistique, j'ai réalisé une **validation croisée** avec k-fold (k=5) pour déterminer les meilleurs hyperparamètres (learning rate, max_epoch et le terme de régularisation reg). De même, pour le modèle de Random Forest, j'ai effectué une **recherche sur grille** (grid search) pour trouver les combinaisons optimales d'hyperparamètres, notamment le nombre d'estimateurs (n_estimators) et la profondeur maximale (max_depth). Je me suis basée sur un ancien projet où j'avais eu recours aussi à la validation croisée et une recherche sur grille.

Pour le Random Forest, il en sort que les meilleurs hyperparamètres sont les suivants : $n\_estimators = 300$ et $max\_depth = 10$.

Quant à la régression logistique, il s'agit de $learning_rate = 0.01$ et $reg = 0.1$ (cela a été fait avec 3000 époques d'après la section "Suivis des entraînements").

4 Résultats

Remarque : avant de chercher à optimiser les modèles ou traiter les données, j'ai d'abord tester sur les données brutes avec des paramètres quelconques le modèle de regression logistique et celui de Random Forest.

Les premiers résultats Kaggle m'ont montré que les résultats de Random Forest étaient meilleurs. ALors certaines des expériences ont été réalisé uniquement sur Random Forest (comme l'over-sampling).

Les données de performances dans le tableaux ci-dessous ont été obtenues en divisant le jeu d'entraînement en un jeu d'entraînement et un de validation. Cela permet de calculer l'accuracy.

TABLE 1 – Accuracy du modèle Random Forest selon les données

	Accuracy
Données brutes	0.884
Données standardisées	0.789 [1]
Données balancées partiellement	0.897
Données balancées totalement	0.908

Cependant, il faut noter que les scores ne sont pas ceux que l'on obtient sur Kaggle et par exemple, le modèle Random Forest sur les données balancées semble meilleur ici mais ce n'était pas le cas sur les données de test.
Comme le balancement des données n'a rien apporté de concluant, je n'ai pas essayé sur le modèle de régression linéaire (enfin j'ai fait des tests mais non concluant et non présentables).

Malgré tous ces résultats, c'est le modèle random forest sur les données brutes qui a obtenu le meilleur score pour la compétition Kaggle.

1. les résultats sont clairement mauvais car on obtient que des 0

5 Discussions

J'ai plusieurs choses à redire quant à mes différentes approches et méthodologies.

Tout d'abord, je regrette de ne pas avoir le temps de tester plus de combinaisons lorsque je cherchais les meilleurs hyperparamètres de mes deux modèles. En effet, j'ai obtenu mes "meilleurs hyperparamètres" mais c'était selon une présélection faite avant et qui pourrait de pas être bien faite.

Par ailleurs, je ne suis pas satisfaite des résultats d'over-sampling. Je pense que cela aurait pu aider les modèles à mieux généraliser puisqu'ils avaient appris sur un jeu ayant une très grosse majorités d'éléments de la classe 0. J'ai testé d'équilibrer entièrement le jeu mais cela n'a pas été efficace ; sans doute car il y avait beaucoup trop de données factices et trop ressemblantes les unes des autres pour les classes minoritaires. Puis, j'ai testé d'équilibrer partiellement mais je n'ai pas eu le temps de faire plusieurs combinaisons (j'ai testé seulement avec 10000 éléments pour chaque classes minoritaires.

Aussi, la sélection des caractéristiques n'a pas non plus était concluante comme je l'ai expliqué plus haut. Je pense qu'en s'y penchant plus j'aurai pu trouver d'autres méthodes pour la sélection des caractéristiques et essayé de faire un tri plus fin de celles-ci afin d'améliorer mes résultats.

J'ai encore de nombreux points à améliorer comme par exemple optimiser le modèle de régression logistique en appliquant la méthode de descente de gradient stochastique (je n'ai pas eu le temps et ai préféré me consacrer au modèle random forest). Ou encore j'aurai aimé explorer encore d'autres modèles sur ces données (comme Machine à vecteurs de support, Naïve Bayes, Forêts d'arbres décisionnels...).

Enfin, je retiens surtout des leçons que j'essaierai d'appliquer pour la prochaine compétition Kaggle et notamment le fait de faire dès le début des codes propres et ordonnés afin de na pas perdre énormément de temps à tout remettre en ordre et surtout faire un suivi des résultats afin de ne rien perdre. En effet, j'ai perdu trop de temps à retrouver des résultats et j'en ai même perdu sur des tests qui n'étaient pas concluant (même si les résultats sont mauvais j'aurai du laissé des traces). Je retiens toutes mes erreurs et je vais essayer de m'améliorer pour la prochaine compétition Kaggle !

6 Contributions

Je déclare par la présente que tous les travaux présentés dans ce rapport sont ceux de Alexia Prevot

Références

[1] Benoit Cayla. Algorithme du gradient stochastique. https://www.kaggle.com/code/vitorgamalemos/multinomial-logistic-regression-from-scratch, 2019.

[2] Kaggle. Régression logistique. https://www.kaggle.com/code/vitorgamalemos/multinomial-logistic-regression-from-scratch, 2021.

[3] scikit learn. sklearn.ensemble.randomforestclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.