RAPPORT
ARE-SCIENCES DES DONNÉES

en collaboration avec Xavier Juvigny

et rédigé par Lina Bourenane, Yuva Djebra et Victor Ji

Sommaire

Introduction

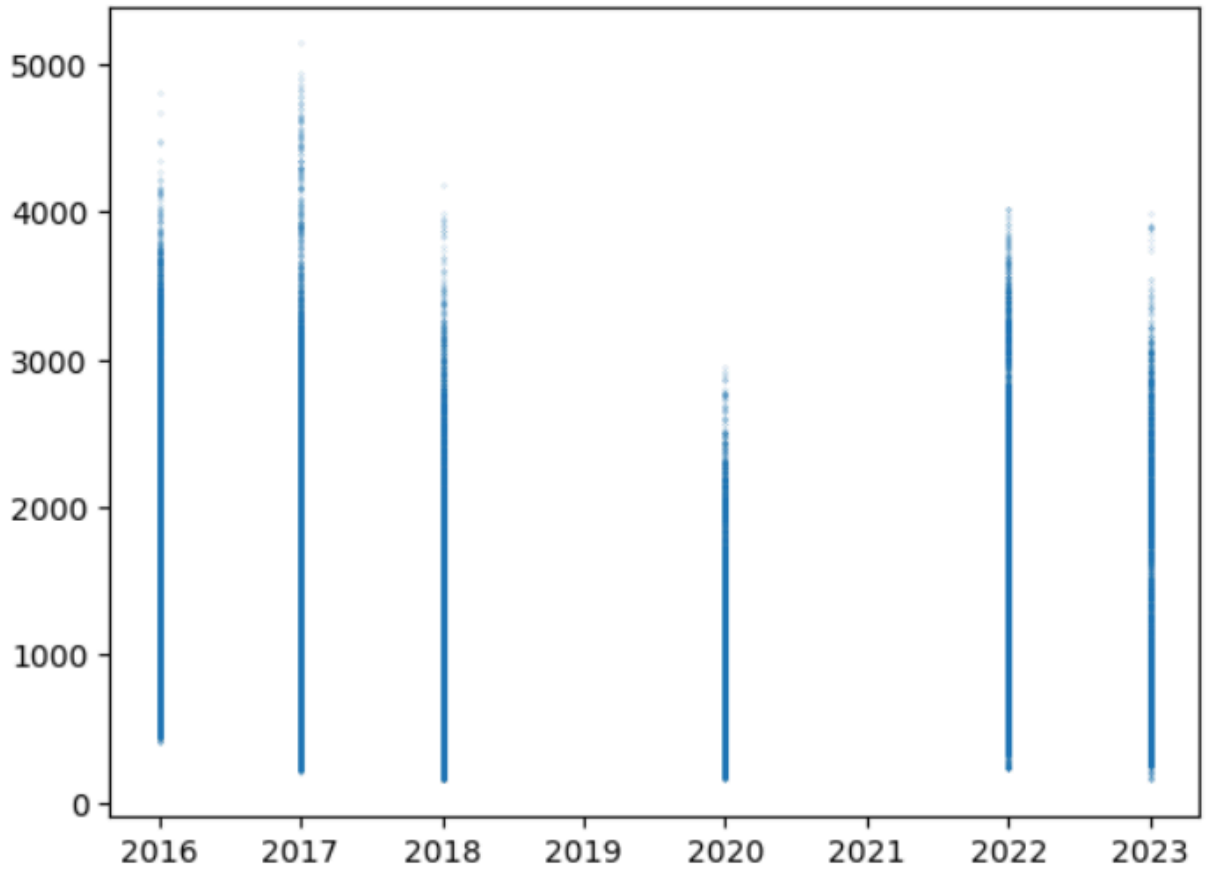
Présentation du Problème :

L'objectif principal de ce projet est de créer un modèle d'apprentissage automatique capable de prédire la « qualité carbone » des mix de production électrique, en tenant compte des variables météorologiques et calendaires. Le jeu de données est open source et fourni par le Réseau Électrique Français (RTE) et Météor France. Ceux-ci ont été modifiés et nettoyés pour offrir un atelier de recherche encadré. La variable attendue est la variable “MixProdElec”, qui mesure la qualité carbone du mix électrique au pas de temps de 30 minutes et peut prendre les valeurs suivantes : Carboné, Normal et Décarboné. La période d’évaluation couvre 2019 et 2021.

Importation des données :

Nous commençons par l’importation des données, que nous divisons en ensemble de tests et de prévision. Après l’importation, nous faisons une première analyse visuelle afin de comprendre et d’avoir une idée générale sur la méthodologie à mettre en place.

Nous traçons une figure de nuage afin de voir le rapport entre le mix carbone et les années entre (2016-2023), grâce à cela nous remarquons que les années 2019 et 2021 ne sont pas représentées, on comprend par cela que nous devons prédire ces deux années (présentent sur le fichier Benchmark).



Loi jointe et Bayes Naïf :

Les Fonctions de Loi Jointe

Les fonctions de “loi_jointe” et “loi_jointe2”, jouent un rôle crucial dans notre analyse statistique du mix électrique français. Elles sont utilisées pour calculer les probabilités conditionnelles entre différentes caractéristiques de nos données, ce qui est essentiel pour comprendre les relations entre ces variables et pour construire des modèles de prédiction précis.

Descriptif de la Loi Jointe

Cette fonction est conçue pour calculer la loi jointe entre deux variables, a et b, dans notre jeu de données. La méthode utilisée consiste à parcourir toutes les combinaisons possibles des valeurs uniques de ces variables et à estimer la probabilité que ces valeurs se produisent conjointement. Plus précisément, nous itérons sur toutes les valeurs uniques de a et b, puis calculons la fréquence à laquelle chaque paire de valeurs se produit ensemble. Cette fréquence est ensuite stockée dans une matrice pxy, où les lignes représentent les valeurs uniques de a et les colonnes représentent les valeurs uniques de b. Cette approche nous permet d'obtenir une estimation pratique de la

probabilité jointe entre les deux variables, ce qui est crucial pour de nombreuses analyses statistiques ultérieures.

Risques d'Erreur et de Surapprentissage

L'une des principales sources d'erreur dans cette approche est la taille limitée de notre ensemble de données. Si notre jeu de données est petit, nos estimations des probabilités jointes peuvent être biaisées ou peu fiables, ce qui peut conduire à des résultats incorrects dans nos analyses ultérieures. De plus, il existe un risque de surapprentissage si notre modèle est trop complexe par rapport à la taille de nos données. Cela peut se produire si nous estimons un grand nombre de paramètres à partir de données limitées, ce qui peut conduire à une mauvaise généralisation à de nouvelles données et à une faible performance de prédiction.

Descriptif de la Loi Jointe 2

Cette fonction généralise la fonction “loi_jointe” pour calculer la loi jointe entre trois variables, a, b et c, dans notre jeu de données. La méthode utilisée est similaire à celle de la fonction “loi_jointe”, mais elle itère maintenant sur toutes les combinaisons possibles des valeurs uniques de a, b et c. Nous calculons ensuite la fréquence à laquelle chaque triplet de valeurs se produit ensemble et stockons ces fréquences dans une matrice à trois dimensions, pxyz. Cette fonction est utile pour modéliser des relations plus complexes entre plusieurs variables, ce qui peut être nécessaire dans certaines analyses statistiques avancées.

Classification Bayésienne

La fonction class_bayes est conçue pour implémenter la classification bayésienne. Elle prend en compte trois arguments : le tableau de données tab, la variable cible y et les caractéristiques prédictives x et x_test. La fonction commence par calculer la loi jointe entre la variable cible et la caractéristique prédictive, puis utilise cette loi pour prédire la classe des données de test en fonction de leurs caractéristiques.

Risques et Limitations

La classification bayésienne suppose des probabilités conditionnelles fixes. Si les données ne suivent pas cette hypothèse, les prédictions peuvent être biaisées.
Si une caractéristique particulière a une forte influence sur la variable cible mais n'est pas bien représentée dans les données, cela peut conduire à des prédictions inexactes.
Le calcul de la loi jointe peut être coûteux en termes de temps de calcul, surtout pour de grandes bases de données.

En revanche, la classification bayésienne est généralement simple à implémenter et peut fournir des résultats raisonnables même avec des données complexes.
À noter que sa performance dépend fortement de l'adéquation des données aux hypothèses sous-jacentes du modèle.

Classification Bayésienne Naïve

La fonction “bayes_naif” implémente la classification bayésienne naïve. Elle prend en compte quatre arguments : le tableau de données tab, la variable cible y, et deux caractéristiques prédictives x1 et x2, ainsi que les données de test x_test. Contrairement à la classification bayésienne, cette approche suppose une indépendance conditionnelle entre les caractéristiques prédictives.

Risques et Limitations

L'hypothèse d'indépendance conditionnelle peut être trop simpliste pour certaines situations réelles, ce qui peut entraîner des prédictions inexactes.
Si les caractéristiques prédictives sont fortement corrélées, cela peut enfreindre l'hypothèse d'indépendance conditionnelle et ainsi affecter la qualité des prédictions.

Efficacité

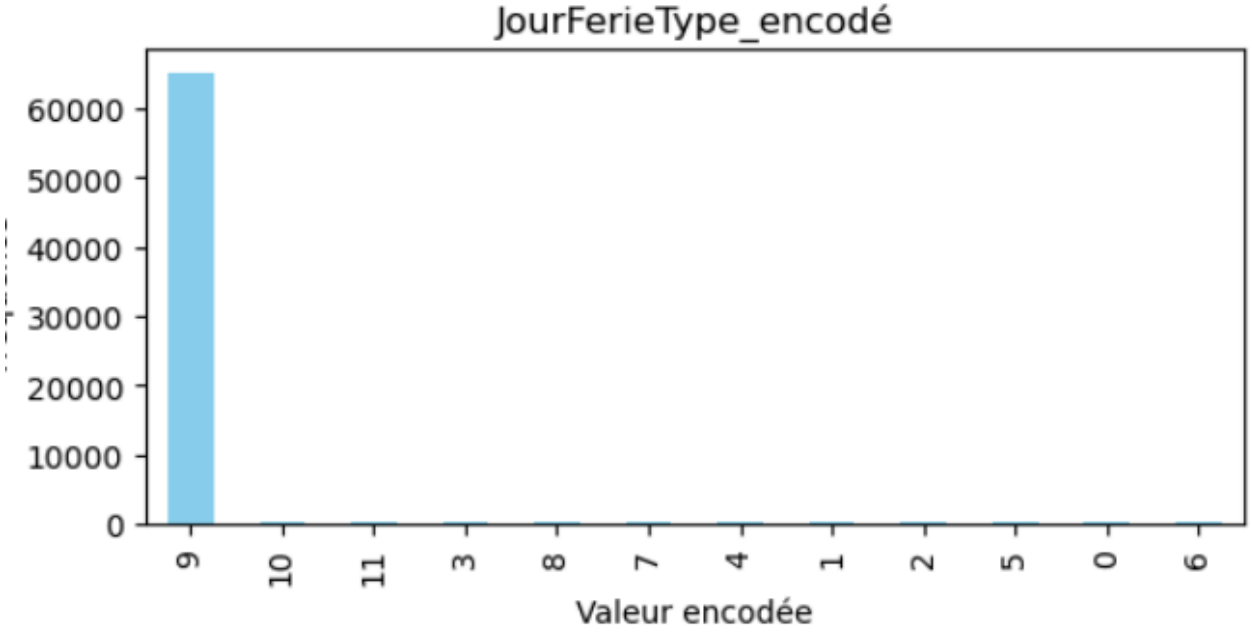
La classification bayésienne naïve est rapide à mettre en œuvre et peut être utilisée avec des ensembles de données de grande taille.
Cependant, elle peut être sensible à la présence de valeurs dites “aberrantes” (très éloignées du reste des données) ou à des distributions non conformes aux hypothèses du modèle.

KNN :

On utilise le code knn pour une prédiction plus efficaces, nous commençons tout d’abord par les étapes suivnates :

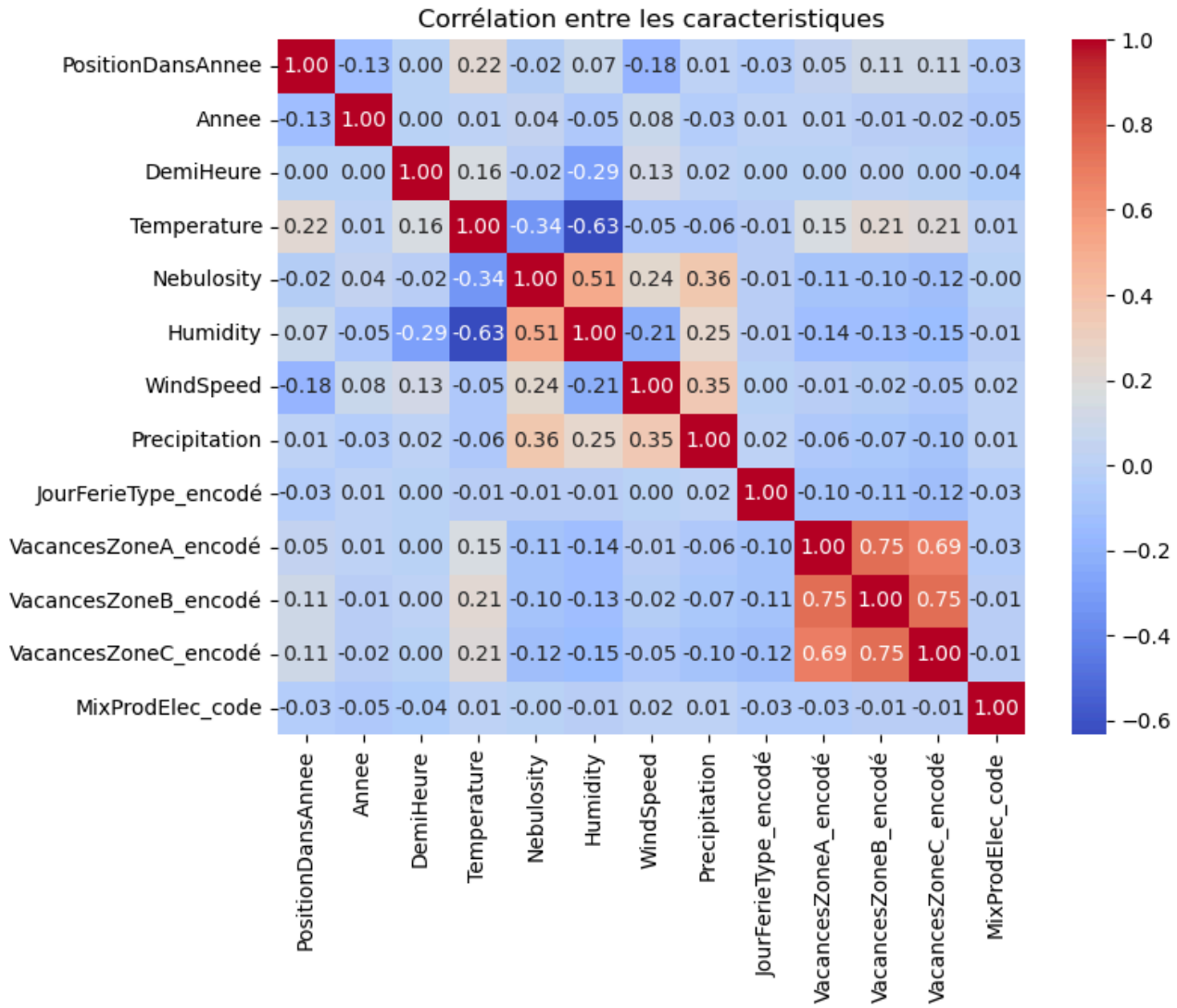
1. **Lecture des données de référence (benchmark)**: Le code commence par lire un fichier CSV contenant les données de référence. Celles-ci seront utilisées pour évaluer les performances de la classification réalisée. Le nom benchmark dans notre code knn représente le fichier «**prev**», et non pas le véritable benchmark.
2. **Encodage des variables catégorielles**: Les variables catégorielles telles que "JourFerie", "JourFerieType", "VacancesZoneA", "VacancesZoneB" et "VacancesZoneC" sont encodées en utilisant la fonction label_encoder, qui attribue à chaque catégorie un identifiant numérique unique, pour pouvoir calculer les distances.

3. **Visualisation des données encodées**: Les variables encodées sont ensuite visualisées sous forme de diagrammes à barres pour examiner leur distribution dans l'ensemble des données et pouvoir choisir les plus efficaces.



On remarque ici dans ce graphique que la proportion de la valeur encodée “9” domine sur le reste des données. En fonction de la corrélation de ce paramètre cela pourrait nuire à notre performance car il y a un risque de surapprentissage éventuel d’un paramètre non pertinent.

Pour éviter cela, on regarde sa corrélation en affichant une matrice de corrélation entre les différents paramètres à l’aide de la bibliothèque “seaborn” :



On voit que sa corrélation avec les autres paramètres est très faible (>=0) donc on décide de l'enlever des paramètres.

4. **Préparation des données d'entraînement et de test**: Les caractéristiques pertinentes sont sélectionnées pour l'entraînement du modèle KNN. Certaines caractéristiques telles que "MixProdElec_code", "EmissionCO2", "JourFerie_encodé" et "JourFerieType_encodé" sont exclues car elles ne sont pas pertinentes pour la prédiction.
5. **Normalisation des données**: Les données d'entraînement et de test sont normalisées en soustrayant la moyenne et en divisant par l'écart-type. Cela permet de mettre toutes les caractéristiques à la même échelle, ce qui est essentiel pour optimiser l'efficacité de l'algorithme KNN.

6. **Classification à l'aide de l'algorithme KNN**: L'algorithme KNN est appliqué aux données d'entraînement normalisées pour prédire les classes des données de test. Nous avons effectué le calcul des distances selon deux méthodes (distances euclidienne et distance de Newton), pour voir quelle distance est plus efficace. Les prédictions sont ensuite évaluées en utilisant la fonction ARE, qui calcule le pourcentage de prédictions correctes par rapport aux valeurs réelles.

Risques et Limitations

Surapprentissage: En utilisant un k trop petit dans l'algorithme KNN, nous pourrions risquer de “surapprendre” les données d'entraînement, ce qui conduirait à une faible généralisation sur de nouvelles données. Il est important de sélectionner judicieusement la valeur de k pour éviter ce problème.

Choix des variables: Le processus de sélection des variables est crucial pour la qualité du modèle. Si des caractéristiques pertinentes sont omises ou si des caractéristiques inutiles sont incluses, cela peut entraîner des performances médiocres du modèle.

Normalisation: La normalisation des données est une étape essentielle pour l'algorithme KNN. Cependant, si les données sont mal normalisées ou si des valeurs aberrantes sont présentes, cela peut fausser les résultats de la classification.

La validation croisée :

Nous avons utilisé la validation croisée avec l'algorithme des K plus proches voisins (KNN). Nous avons utilisé la bibliothèque scikit-learn pour mettre en œuvre cette technique de validation.

Détails de la Méthodologie

Nous avons utilisé la classe KFold de la bibliothèque scikit-learn pour diviser notre ensemble de données en 6 plis (splits). Cette valeur a été choisie de manière arbitraire pour s’assurer qu’on ait une répartition adéquate des données tout en minimisant le biais introduit par une taille de pli trop petite ou trop grande.

Risques d'Erreur et de Surapprentissage

L'un des principaux risques lors de l'utilisation de la validation croisée est le risque d'erreur d'évaluation. Ce risque peut survenir si les plis ne sont pas sélectionnés de manière aléatoire ou si le nombre de plis est trop petit pour fournir une estimation fiable de la performance du modèle.

De plus, le surapprentissage (overfitting) peut également être un problème potentiel lors de l'utilisation de la validation croisée. Cela peut se produire si le modèle est trop

complexe ou si les données d'entraînement ne sont pas représentatives de la population sous-jacente. Pour atténuer ce risque, il est important de choisir judicieusement les hyperparamètres du modèle et de s'assurer que le modèle ne sur-apprend pas les données d'entraînement et pour cela nous avons sélectionné les paramètres efficaces lors de la réalisation du KNN

Exécution de la Validation Croisée

Une fois les plis créés, nous avons exécuté un processus itératif à l'intérieur d'une boucle for pour chaque pli. À chaque itération de la boucle, nous avons divisé nos données en ensembles de données d'entraînement et de test, normalisé les données d'entraînement en soustrayant la moyenne et en divisant par l'écart-type, et enfin effectué des prédictions à l'aide de l'algorithme KNN pour différentes valeurs de k.

Nous avons choisi de limiter le nombre de prédictions à 1000 pour chaque pli afin de réduire le temps de calcul et d'assurer une exécution rapide du processus de validation croisée. Cependant, cette approche peut introduire un certain biais dans notre estimation de la performance du modèle, en particulier si les données de test ne sont pas représentatives de la population sous-jacente.

Évaluation de la Performance

À chaque itération de la boucle, nous avons calculé la précision des prédictions en comparant les valeurs prédites aux valeurs réelles de la variable cible. Nous avons ensuite stocké ces scores dans une liste pour calculer la moyenne de la précision sur l'ensemble des plis à la fin du processus de validation croisée.

Conclusion :

En conclusion, ce rapport illustre notre démarche pour modéliser l'impact carbone du mix électrique français. À travers une combinaison d'analyses descriptives, de construction de nouvelles variables et de modélisation statistique, nous avons pu explorer différentes approches pour prédire cette variable d'intérêt. Les résultats obtenus, notamment à travers l'algorithme des K plus proches voisins, montrent des performances encourageantes. Cependant, des améliorations potentielles peuvent être envisagées, notamment en explorant d'autres algorithmes de modélisation (comme les arbres ou bien les réseaux neuronaux…) et en affinant nos stratégies de validation.

Bibliographie :

Ouvrage

Mueller, J. P., & Massaron, L. (2023). *Python for Data Science For Dummies*. John Wiley & Sons.

Mckinney, W. (2021). *Analyse de données avec Python - Optimiser la préparation des données avec Pandas, Numpy, Jupyter et IPython-collection O'Reilly*. First Interactive.
Coelho, L. P., & Richert, W. (2015). *Building Machine Learning Systems with Python - Second Edition*. Packt Publishing Ltd.
John Shovic & Alan Simpson (2021) *Python tout-en-un pour les nuls*. "Livre trouvé à la BU des L1"

Site internet qui nous ont énormément aidés :

Pour le knn : https://www.elastic.co/fr/what-is/knn

Pour la validation croisé : https://www.jedha.co/formation-ia/cross-validation#:~:text=La%20cross%20validation%20ou%20validation,fonctionner%20sur%20de%20nouvelles%20donn%C3%A9es.

Pour le Bayes /Bayes Naives : https://www.geeksforgeeks.org/naive-bayes-classifiers/