POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

DUPLAT CLÉMENCE 2407959
DURAND ADRIEN 2261799

Projet final: Prédiction de la qualité de l'air

MTH6312 - MÉTHODES STATISTIQUES D'APPRENTISSAGE

October 11, 2024

1 Introduction

Actuellement, l'écologie est au centre de beaucoup de débats. La qualité de l'air est un enjeu environnemental majeur, qui affecte à la fois la santé humaine et le climat. Afin d'anticiper des épisodes polluants, il serait donc judicieux de pouvoir prédire si la qualité de l'air est bonne ou mauvaise à un instant donné.
Pour évaluer cet aspect environnemental, l'Indice de Qualité de l'Air (*Air Quality Index* - AQI) est couramment utilisé. l'AQI permet de mesurer la pollution atmosphérique sur base de la concentration des différents composants polluants.

On se demande donc comment les modèles statiques peuvent permettre de prédiction fiable de la qualité de l'air en exploitant les données passées ?

Pour cela, nous allons utiliser une bibliothèque "Air Quality" afin d'entraîner différents modèles statistiques. Les données seront mises en forme afin de pouvoir les adapter à un modèle de prédiction temporelle, qui permettra de faire des prédictions dans le futur. Chaque modèle sera adapté à deux types de jeux de données, l'un utilisera les quantités instantanées, l'autres essayera d'exploiter les variations temporelles.
Nous comparerons donc trois différentes méthodes statistiques d'apprentissage, qui sont la régression logistique, la classification dite *KNN* et enfin les méthodes d'analyses discriminantes linéaires et quadratiques. Nous avons choisi ces méthodes, car elles sont toutes adaptées à la classification, mais utilise des hypothèse et philosophie différentes.

2 Préparation des données

Pour ce projet, nous avons accès à un Dataset qui contient les mesures horaires de la qualité de l'air en Italie.
Les colonnes du Dataframes sont les suivantes:

```
Date       Time   CO(GT)   PT08.S1(CO)   NMHC(GT)   C6H6(GT) PT08.S2(NMHC)   NOx(GT)   PT08.S3(NOx)
NO2(GT)   PT08.S4(NO2)   PT08.S5(O3)   T     RH       AH
```

Nous allons rajouter une colonne "AQI", qui pourra prendre la valeur 1, si la qualité de l'air est bonne, ou 0, si la qualité de l'air est mauvaise. On va classifier l'air comme bon pour lorsque AQI est entre "très bon" et "moyen" ou de mauvais lorsque AQI est entre "médiocre" et "très mauvais". Se basant sur le tableau donné sur `https://en.wikipedia.org/wiki/Air_quality_index` on va considérer que si $NO_2 < 134\mu g/m^3$ et que $C_6H_6 < 34\mu g/m^3$, la qualité de l'air est bonne. Sinon, elle est mauvaise.

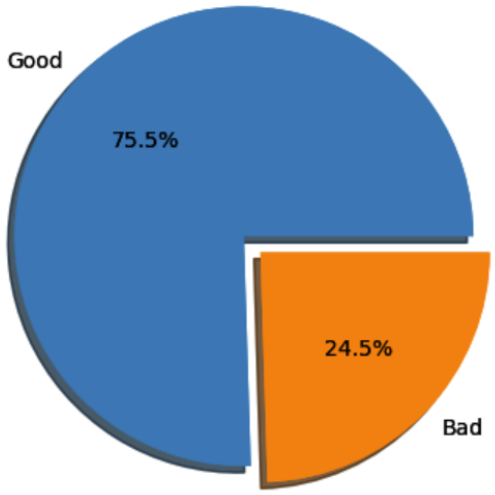


Figure 1: Nombre de données classées comme bonnes (1) ou mauvaises (0)

On voit que chaque donnée est associée à une date et une heure précise, ce qui va nous permettre une analyse temporelle détaillée. On va les combiner ensemble afin d'avoir une valeure combinée pour la date et l'heure.

On a donc les données suivante

$$\mathscr{D}_0 = \{(x_t, y_t) : y_t \in \{0;1\}, x_t \in \mathbb{R}^p\} \text{ avec } p = 13$$

- $t \in \mathbb{N}$ étant une heure d'observation.
Grâce à cela, nous pouvons créer une nouvelle donnée "AQI_lendemain", qui représentera la qualité de l'air du lendemain à la même heure.

Pour ce projet, nous allons analyser deux sortes de DataSet: le **statique** et le **dynamique**. Le Dataset statique correspond à une représentation instantanée des variables à un moment donné. Une observation n'a donc pas de lien avec les observations précédentes. Au contraire, le dataset dynamique inclut des variables décalées dans le temps. Il permet de capturer les tendances temporelles.
Le dataset **statique** va donc nous permettre de prédire la qualité de l'air pour le lendemain de l'observation en ne considérant que les concentrations fixe de polluant.
Tandis que le dataset **dynamique** va nous permettre de calculer la qualité de l'air le lendemain, mais en considérant l'évolution des concentrations sur les heures précédente.
Les données **statique**, telles que nous les avons formulées, sont donc

$$\mathscr{D}_{\text{stat}} = \{(x_{t-24}, y_t) : y_t \in \{0;1\}, x_t \in \mathbb{R}^p, t \geq 24\}$$

On perd donc 24h d'observation donc $|\mathscr{D}_{\text{stat}}| = |\mathscr{D}_0| - 24$.
Et données **dynamique**, telles que nous les avons formulées, sont donc :

$$\mathscr{D}_{\text{dyn}} = \left\{(\mathbf{X}_t, y_t) : y_t \in \{0;1\}, x_t \in \mathbb{R}^{3p}, t \geq 30\right\} \text{ avec } \mathbf{X}_t = \begin{pmatrix} x_{t-30} \\ x_{t-26} \\ x_{t-24} \end{pmatrix}$$

De même, on perd 30h d'observation donc $|\mathscr{D}_{\text{dyn}}| = |\mathscr{D}_0| - 30$. Aussi, on constate que l'espace des variables du jeu de données dynamiques est bien 3 fois plus grand que celui statique ($\mathbb{R}^{3p}$ contre $\mathbb{R}^p$) car il comprendra des observations sur 3 périodes de temps.

Enfin, y_t correspond à la colonne "AQI" du jeu de donnée. On le détermine grâce à la fonction décrite plus haut :

$$y_t = f(x_t) = \begin{cases} 1 \text{ , si } (x_t)_7 = NO_2 < 134\mu g/m^3 \text{ et } (x_t)_4 = C_6H_6 < 34\mu g/m^3 \\ 0 \text{ , sinon} \end{cases}$$

2.1 Construction d'un modèle réduit selon la contribution des variables

On va tester, avec chacun des modèles de prédictions, étudié dans ce travail l'hypothèse que chaque variable ne contribue pas au modèle.
Pour cela, nous posons les hypothèses suivantes :

- H_0 : La variable n'a pas d'effet significatif sur la qualité de l'air (elle peut être ignorée dans le modèle).
- H_1 : La variable a un effet significatif sur la qualité de l'air (elle doit être incluse dans le modèle).

Le test repose sur la mesure de la p-value, qui quantifie la probabilité d'obtenir des résultats aussi extrêmes, voire plus, que ceux observés si l'hypothèse nulle H_0 est vraie. Donc si la p-value associée à une variable est supérieure à un seuil fixé $\alpha = 0.05$ qui restera le même pour tout le projet, nous n'avons pas suffisamment de preuves pour rejeter H_0, et nous considérons que cette variable n'est pas significative dans le modèle.

Dans le cas statique, si une variable à un p-value trop grande, nous pouvons simplement la supprimer du modèle.

Dans le cas dynamique cependant, nous allons nous permettre quelques libertés en plus pour avoir un modèle cohérent. Le modèle dynamique vise à exploiter les variations temporelles de chacun des polluants pour prédire la qualité de l'air dans le futur. Ainsi, si un indicateur ne contribue pas à un instant t il est inutile de le considérer dans le modèle à d'autre instant. Ainsi si une variable $(x_t)_i$ est supprimée du car sa p-value est trop élevée, c'est que le polluant i associé ne contribue pas au modèle, au temps, t il n'y a donc pas de raison que le polluant contribue aussi au temps $t-2$ et $t-4$. On supprimera donc tout le polluant à toute heure si jamais celui-ci a une p-value supérieure à 0.05 à n'importe quelle heure.

3 Régression Logistique

4 Introduction

Nous allons commencer par utiliser un modèle de régression logistique pour prédire la qualité de l'air du lendemain (classification binaire). Cette méthode prédit la probabilité qu'une observation appartienne à une classe ($Y \in \{0,1\}$) en fonction des variables explicative (x). Pour cela, on va utiliser la fonction *logit*:

$$P(Y = 1 \mid X = x) = \frac{\exp(x^\top \beta)}{1 + \exp(x^\top \beta)}$$

avec β_i les coefficients du modèle à ajuster. Cette fonction nous permet d'avoir une probabilité comprise entre 0 et 1. Si cette probabilité dépasse le seuil de 0.5 l'observation est classée comme bonne($P(Y = 1|X = x) > 0.5$), sinon elle est classée comme mauvaise.
Après avoir entraîné le modèle, nous pourrons prédire les probabilités de qualité d'air sur l'ensemble test.

4.1 Jeu de données Statique

Dans le cas statique, le modèle va utiliser uniquement les variables à un instant donné x_{t-24}. La matrice de confusion est donnée sur l'image 2.

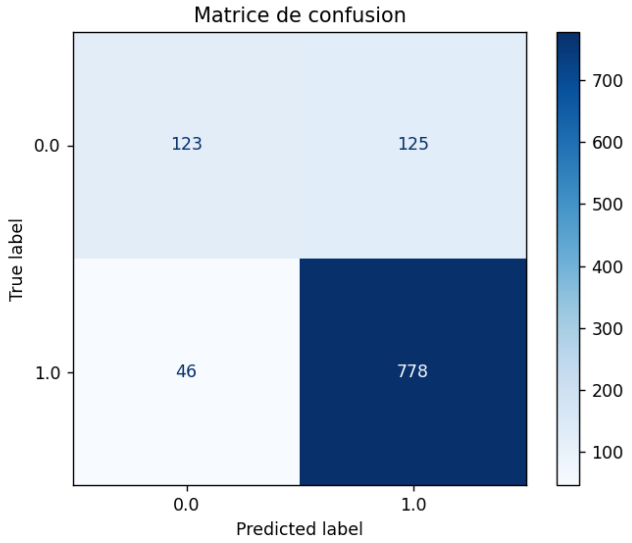


Figure 2: Statique, modèle non réduit : matrice de confusion pour la régression logistique

Les résultats associés plus précis sont les suivants :

```
Le taux d'erreur est : 0.1595

Rapport de classification :
              precision    recall  f1-score   support

         0.0       0.73      0.50      0.59       248
         1.0       0.86      0.94      0.90       824

    accuracy                           0.84      1072
   macro avg       0.79      0.72      0.75      1072
weighted avg       0.83      0.84      0.83      1072
```

On remarque que l'on a une précision relativement élevée, surtout pour la prédiction de la classe "bonne". Cependant, on voit que notre modèle a plus de mal à détecter des périodes de mauvaise qualité d'air.
Comme expliquer dans la section 2.1, on peut maintenant retirer les variables qui ne contribuent pas significativement au modèle afin de voir si cela améliore nos performances. Les p-value associées à chacun des polluants est donné sur l'image 3.

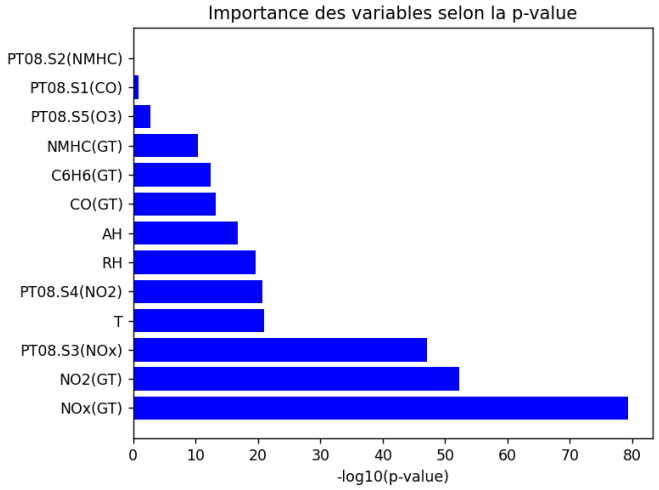


Figure 3: Statique: importance des p-value

Les variables considérées comme inutiles pour le modèle statique sont :

```
Variables considérées comme inutiles (p-value > 0.05) :
['PT08.S1(CO)', 'PT08.S2(NMHC)']
```

La nouvelle matrice de confusion est donnée sur l'image 4.

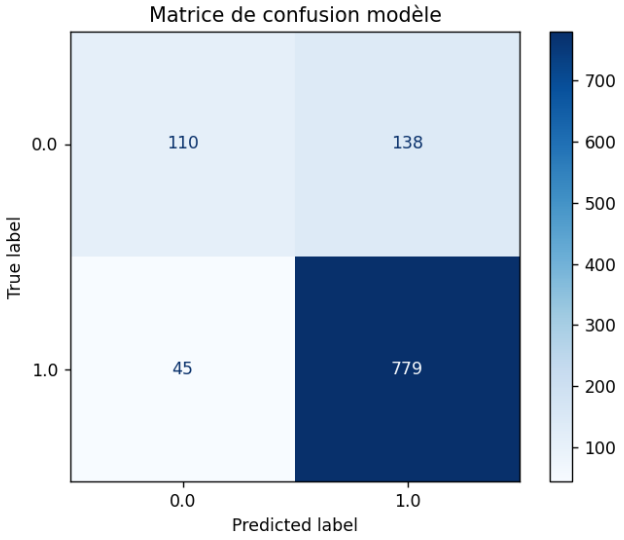


Figure 4: Statique: Matrice de confusion pour la régression logistique, modèle réduit

Les résultats associés plus précis sont les suivant:

```
Le taux d'erreur est : 0.1707
Rapport de classification :
              precision    recall  f1-score   support

         0.0       0.71      0.44      0.55       248
```

```
         1.0       0.85      0.95      0.89       824

    accuracy                           0.83      1072
   macro avg       0.78      0.69      0.72      1072
weighted avg       0.82      0.83      0.81      1072
```

4.2 Jeux de données Dynamique

On va maintenant procéder de la même manière, mais pour le jeu de données Dynamique. La matrice de confusion est donnée sur l'image 5.

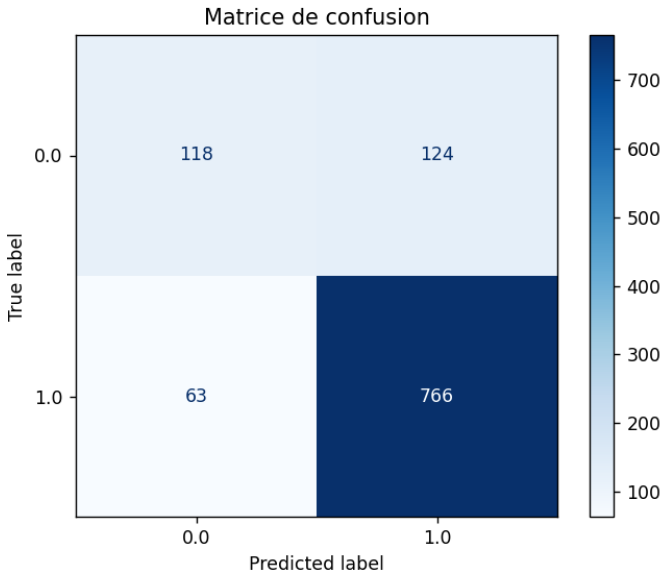


Figure 5: Dynamique: matrice de confusion pour la régression logistique

Les résultats associés plus précis sont les suivants:

```
Le taux d'erreur est : 0.1746

Rapport de classification :
              precision    recall  f1-score   support

         0.0       0.65      0.49      0.56       242
         1.0       0.86      0.92      0.89       829

    accuracy                           0.83      1071
   macro avg       0.76      0.71      0.72      1071
weighted avg       0.81      0.83      0.82      1071
```

Comme expliquer dans la section 2.1, on peut d'abord retirer les variables qui ne contribuent pas significativement au modèle afin de voir si cela améliore nos performances. L'importance des p-value est donnée sur l'image 6.

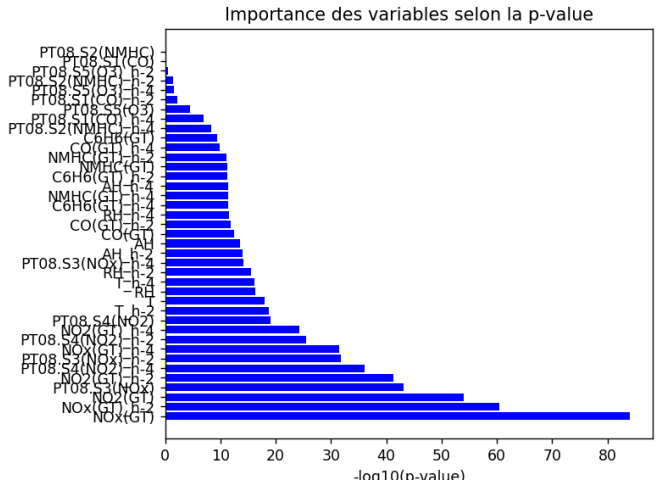


Figure 6: Dynamique: importance des p-value

Les variables considérées comme inutiles pour le modèle dynamique sont :

`['PT08.S5(O3)_h-2', 'PT08.S1(CO)', 'PT08.S2(NMHC)']`

et toutes les heures associées.
La nouvelle matrice de confusion est donnée sur l'image 7.

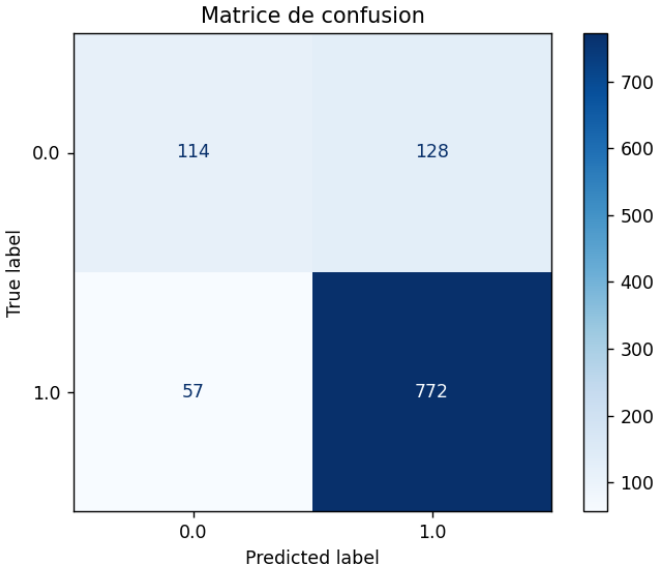


Figure 7: Dynamique: matrice de confusion pour la régression logistique, modèle réduit

Les résultats associés plus précis sont les suivants:

```
Le taux d'erreur est : 0.1774
Rapport de classification :
              precision    recall  f1-score   support

         0.0       0.67      0.42      0.52       242
         1.0       0.85      0.94      0.89       829

    accuracy                           0.82      1071
   macro avg       0.76      0.68      0.70      1071
weighted avg       0.81      0.82      0.81      1071
```

4.3 Choix du modèle

On voit que dans le cas du jeu de données **statique**, on va préférer utiliser le **modèle non réduit**. Bien que la complexité du modèle ne soit alors pas réduite, on voit qu'il est plus performant. Au contraire, dans le cas du jeu de données **dynamique**, on va préférer utiliser le **modèle réduit**. Il va considérablement simplifier la complexité du modèle en retirant les variables peu informatives (et leurs versions décalées dans le temps) et réduit le risque de multicolinéarité, sans pour autant impacter beaucoup la performance du modèle.

5 $K-$ plus proches voisins (KNN)

5.1 Introduction et Choix de la validation croisé

Dans cette section, nous allons étudier le classificateur du type K plus proche voisin. Pour classer un point dans l'espace des données, le modèle observe les K points les plus proches (selon une métrique de distance, souvent la distance euclidienne) et attribue à ce point la classe majoritaire parmi ces K voisins. Ce type de modèle de classification a besoin d'être optimisé, en choisissant le nombre de voisins K qui seront utilisés pour classifier un point de l'espace. Un K trop petit peut entraîner un modèle trop sensible au bruit (sur-apprentissage), tandis qu'un K trop grand peut entraîner un lissage excessif des frontières entre classes (sous-apprentissage).

Pour déterminer la valeur optimale de K, nous utilisons une méthode de **validation croisée** (*k-fold*[1] *cross-validation*). Cette méthode consiste à :

1. Diviser l'ensemble des données en k sous-ensembles ou "blocs".
2. Utiliser successivement $k-1$ blocs pour entraîner le modèle et le bloc restant pour évaluer ses performances.
3. Répéter cette opération k fois, en changeant à chaque itération le bloc utilisé pour la validation.

L'avantage de cette méthode est qu'elle permet d'évaluer la robustesse du modèle sur l'ensemble des données en limitant les biais liés à un découpage arbitraire. Cependant, plus le nombre de blocs k est élevé, plus le processus est long, car le modèle doit être entraîné et évalué autant de fois qu'il y a de blocs.
On choisira dans le cadre de la classification KNN un $\lfloor \frac{n}{10} \rfloor$-fold, avec $n = |\mathscr{D}|$. Cela permet de limiter le temps de calcul en ayant tout de même un nombre conséquent d'ensemble de validation. De plus, comme nous l'avons vu dans la section **Préparation des données,** la taille du jeu de données varie selon que l'on choisit le modèle dit Dynamique ou Statique. En adaptant le nombre bloc $k = \lfloor \frac{n}{10} \rfloor$ on a donc une méthode universelle pour les deux jeux de données.

5.2 Jeu de données Statique

Pour le jeu de donnée statique, on trouve un K optimal $K^* = 3$ qui donne une erreur de 15.34%, l'évolution de l'erreur selon $1/K$ est donnée en Figure 11.

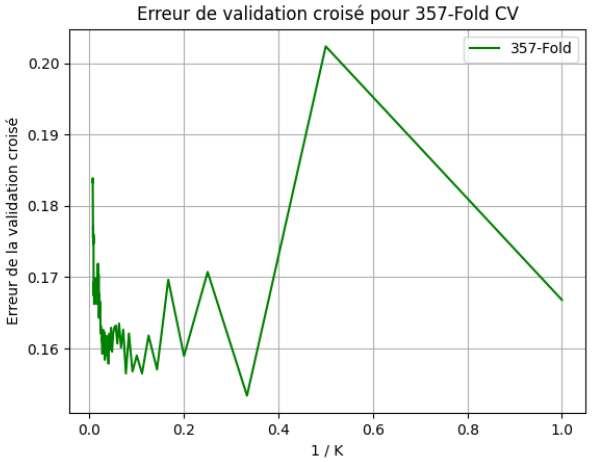
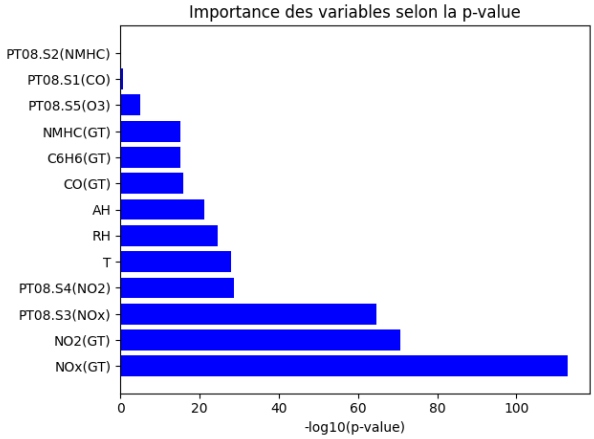


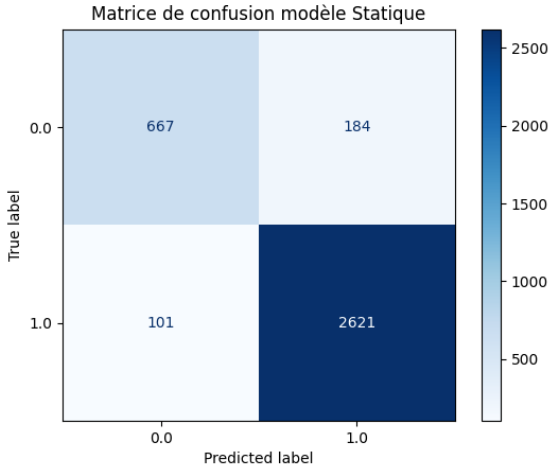
Figure 8: KNN - Statique : Erreur moyenne en fonction de $1/K$

Et voici la matrice de confusion ainsi que les p-value associées à chacun des polluants :

[1] Ici pour éviter toute ambiguïté remarquons que le K en majuscule est utilisé pour parler du nombre de voisins dans la méthode KNN, alors que le k minuscule est utilisé pour parler du nombre de blocs dans la validation croisé $k-$fold



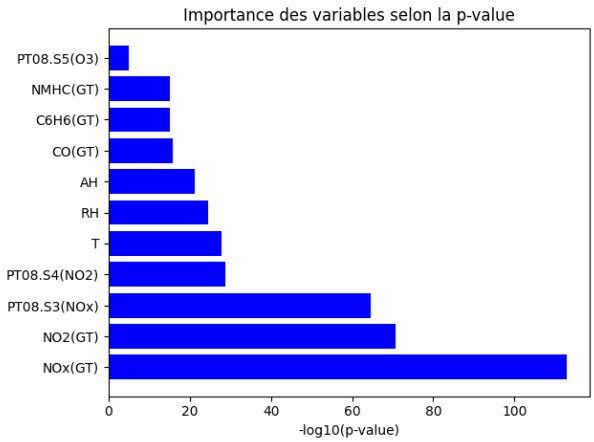
(a) p-value des polluants



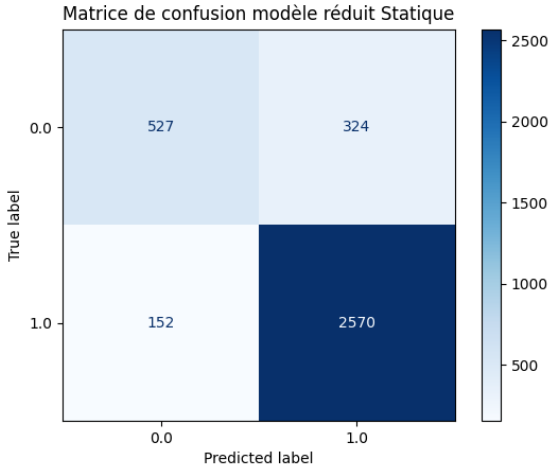
(b) Matrice de confusion

Figure 9: KNN - Statique : p-value des polluants et Matrice de confusion.

On peut donc déterminer le modèle réduit en supprimant des données les polluants qui contribuent le moins au modèle. C'est-à-dire : $PT08.S1(CO)$ et $PT08.S2(NMHC)$.
En ré-entraînant le modèle statistique réduit, on trouve $K^* = 9$ pour une erreur de $15,51\%$
Et voici les résultats sur le modèle réduit :



(a) p-value des polluants



(b) Matrice de confusion

Figure 10: KNN - Statique - Réduit : p-value des polluants et Matrice de confusion.

On trouve que toutes les variables restantes contribuent significativement au modèle (p-value < 0.05)

```
### Comparaison des modèles ###
Modèle non réduit (k = 3):
  - Accuracy moyenne : 0.8044
  - Écart-type : 0.1675

Modèle réduit (k = 9):
  - Accuracy moyenne : 0.8240
  - Écart-type : 0.1569
```

5.3 Jeu de données Dynamique

Pour le jeu de donnée dynamique, on trouve aussi un K optimal $K^* = 3$ avec une erreur de $14,92\%$, l'évolution de l'erreur selon $1/K$ est donnée en Figure 11.

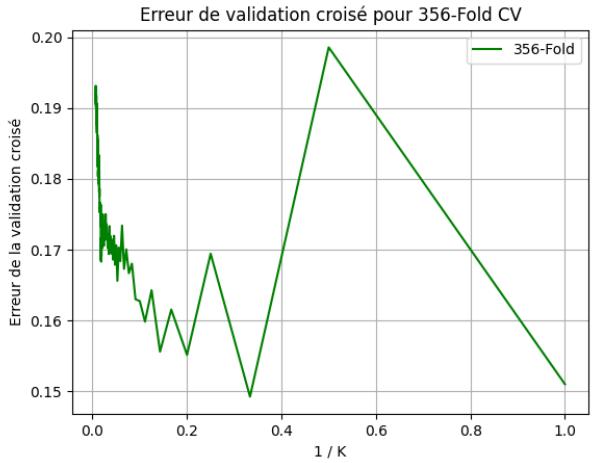
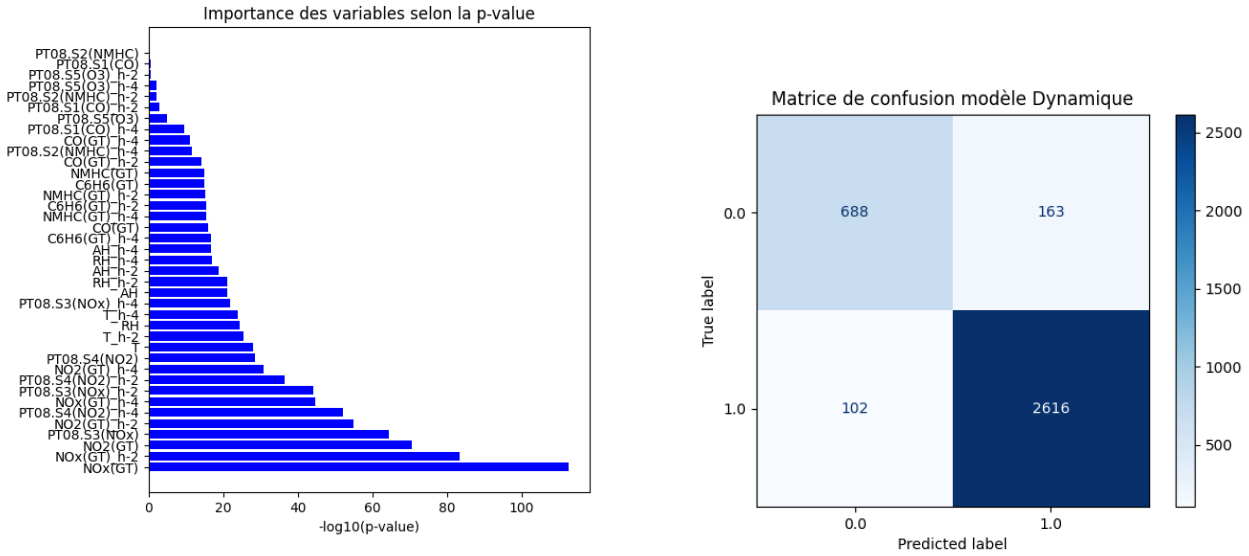


Figure 11: KNN - Dynamique : Erreur moyenne en fonction de $1/K$

Et voici la matrice de confusion ainsi que les p-value associées à chacun des polluants :

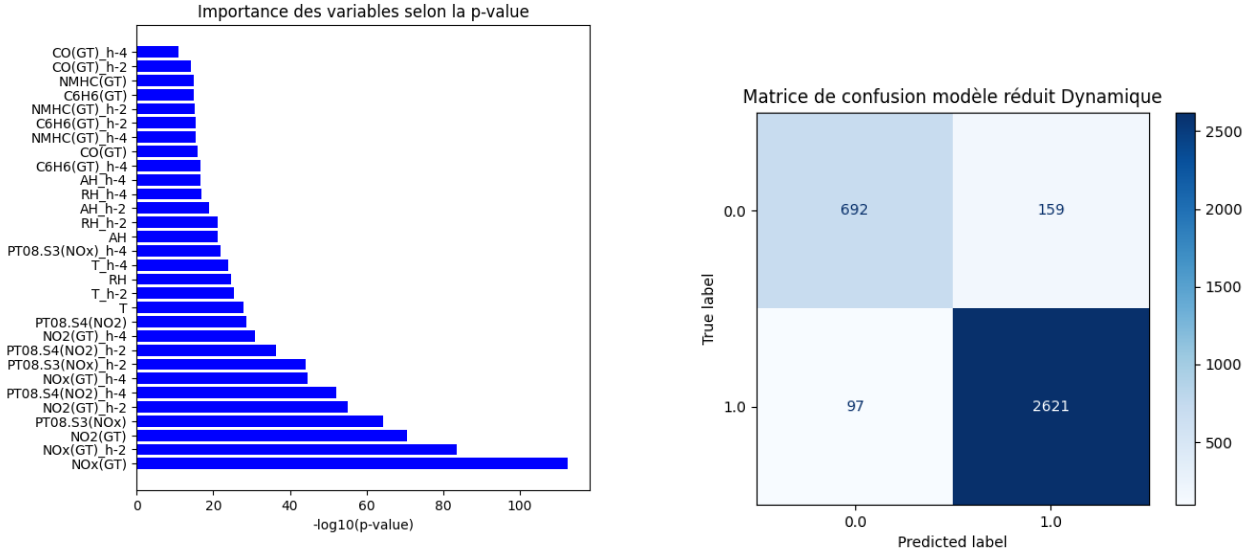


(a) p-value des polluants

(b) Matrice de confusion

Figure 12: KNN - Dynamique : p-value des polluants et Matrice de confusion.

On peut donc déterminer le modèle réduit en supprimant des données les polluants qui contribuent le moins au modèle.
En ré-entraînant le modèle statistique réduit, on trouve $K^* = 3$ pour une erreur de 14.65%. C'est-à-dire : $PT08.S5(O3)_h - 2$, $PT08.S1(CO)$ et $PT08.S2(NMHC)$. On va donc les supprimer pour toutes les heures.
Et voici les résultats pour le modèle dynamique réduit:



(a) p-value des polluants

(b) Matrice de confusion

Figure 13: KNN - Dynamique - Réduit : p-value des polluants et Matrice de confusion.

Comme pour le modèle statique, dans le modèle dynamique réduit, l'ensemble des variables contribuent significativement au modèle.

```
    ### Comparaison des modèles ###
Modèle non réduit (k = 3):
  - Accuracy moyenne : 0.8047
  - Écart-type : 0.1679

Modèle réduit (k = 3):
  - Accuracy moyenne : 0.8089
  - Écart-type : 0.1646
```

5.4 Influence de la taille de l'ensemble d'entraînement

Dans cette partie, on étudie l'influence de l'ensemble d'entraînement sur l'erreur moyenne commise. Pour cela, on effectue une validation croisée $k-$Fold en faisant varier k ainsi, on entraîne le modèle KNN sur un ensemble plus ou moins grand de données :

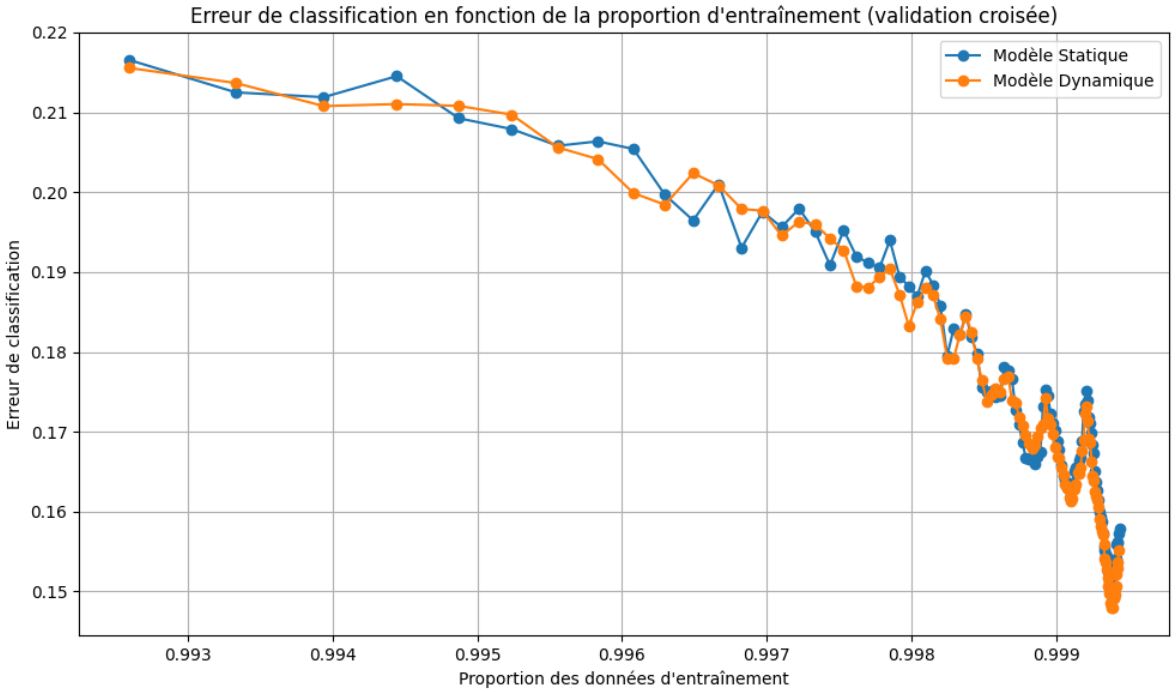


Figure 14: Influence du nombre de données d'entraînement sur l'erreur

L'étude de l'influence de la taille de l'ensemble d'entraînement a mis en évidence une relation claire : plus la taille de l'ensemble d'apprentissage augmente, plus l'erreur moyenne diminue. Cela confirme l'importance d'un volume de données suffisant pour améliorer les performances du modèle KNN, mais c'est le cas pour l'ensemble des classificateurs statistique ou le nombre de données est un enjeu majeur.

5.5 Choix du modèle

Le modèle Dynamique réduit présente une erreur plus faible (14.65%) par rapport au modèle statique réduit (15.51%). Bien que l'amélioration de la **précision moyenne** soit plus marquée dans le modèle statique réduit (82.40%), elle reste très proche de celle du modèle dynamique réduit (80.89%). Cependant, l'écart-type est légèrement plus faible dans le modèle dynamique, traduisant une meilleure stabilité des résultats.
Ainsi, le modèle dynamique réduit peut être considéré comme le meilleur modèle. Bien qu'il n'affiche pas l'accuracy moyenne la plus élevée, il présente une erreur légèrement plus faible et une meilleure stabilité. On peut raisonnablement penser qu'en ayant plus de données et en créant un modèle dynamique avec des informations à $t-1;...;t-n$ on pourrait encore augmenter la précision de ce dernier.

6 Analyses Discriminantes Linéaires et Quadratiques (LDA/QDA)

6.1 Introduction et justifications

Pour cette section nous allons étudier les classificateurs LDA et QDA. LDA et QDA sont toutes les deux des méthodes de classification qui vont reposer sur des hypothèses de distribution des données. L'hypothèse de LDA est que les données de chaque classe suivent une distribution gaussienne (avec une moyenne et variance spécifique) mais toutes les classes ont la même matrice de covariance. Au contraire, QDA va émettre l'hypothèse que chaque classe peut posséder sa propre matrice de covariance. Cette différence d'hypothèse va avoir une influence sur les frontières de décisions entre les différentes classes. La frontière va être linéaire dans le cas de LDA et quadratique pour QDA. En fonction de la manière dont les classes sont séparées, il est judicieux d'utiliser l'une ou l'autre méthode. Notons aussi que LDA est une méthode plus simple et robuste mais moins flexible. On apperçoit rapidement que le classificateur QDA obtient un taux d'erreur élevé (de plus de 40% avec le jeu de données dynamique et 25% avec le statique), tandis que le classificateur LDA obtient un taux d'erreur d'environ 17% pour les 2 dataset. Ici, nous allons donc nous focaliser sur le classificateur LDA.

6.2 Dataset Statique

La matrice de confusion est donnée sur l'image 15.

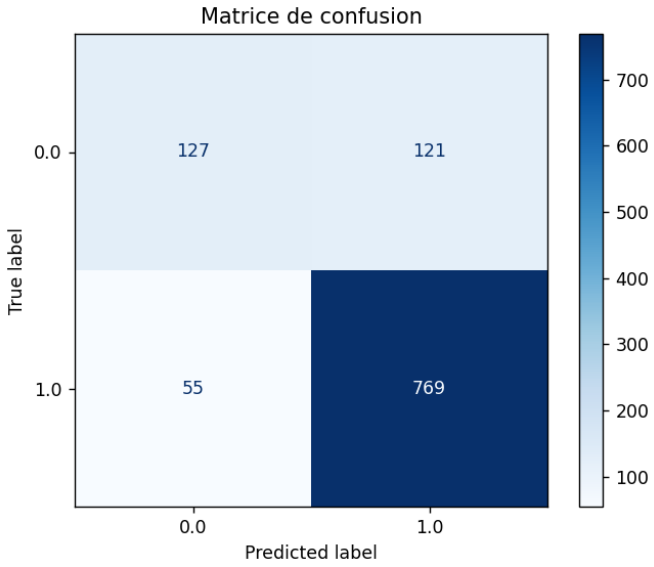


Figure 15: Statique: Matrice de confusion pour LDA

Les résultats associés plus précis sont les suivant:

```
Le taux d'erreur est: 0.1642

Rapport de classification:
              precision    recall  f1-score   support

         0.0       0.70      0.51      0.59       248
         1.0       0.86      0.93      0.90       824

    accuracy                           0.84      1072
   macro avg       0.78      0.72      0.74      1072
weighted avg       0.83      0.84      0.83      1072
```

Comme expliqué dans la section 2.1, on peut maintenant retirer les variables qui ne contribuent pas significativement au modèle:

```
['PT08.S1(CO)', 'PT08.S2(NMHC)']
```

La matrice de confusion est donnée sur l'image 16.

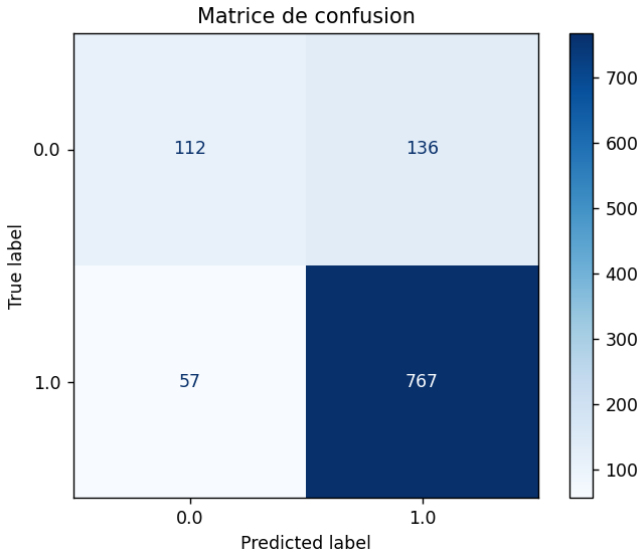


Figure 16: Statique: Matrice de confusion pour LDA, modèle réduit

Les résultats sont les suivants:

```
Le taux d'erreur est: 0.1800

Rapport de classification :
              precision    recall  f1-score   support

         0.0       0.66      0.45      0.54       248
         1.0       0.85      0.93      0.89       824

    accuracy                           0.82      1072
   macro avg       0.76      0.69      0.71      1072
weighted avg       0.81      0.82      0.81      1072
```

On constate que le nombre de faux-positif (prédire 1 alors que le vrai label est 0) est plus élevé que le nombre de vrai-négatif (prédire 0 quand le vrai label est 0), notre classificateur a donc tendance à classer trop souvent dans la classe 1, qui correspond à une bonne qualité de l'air. Comme vu dans la Section 2 75% du temps, la qualité de l'air est bonne, le classificateur a donc tout intérêt à classer plus souvent dans la classe 1 pour avoir au moins 75% de bonnes réponses. Essayons de corriger ce problème en passant au modèle dynamique.

6.3 Jeu de données Dynamique

On va maintenant procéder de la même manière pour le jeu de données Dynamique. La matrice de confusion est donnée sur l'image 17.

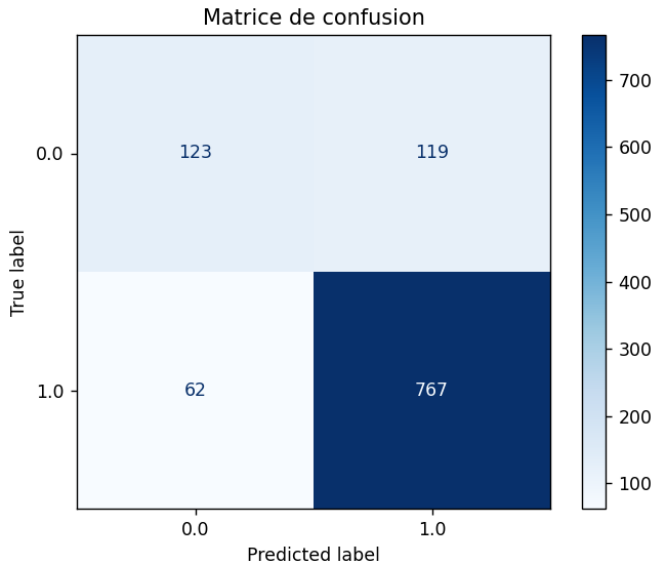


Figure 17: Dynamique: Matrice de confusion pour LDA

Les résultats associés plus précis sont les suivants:

```
Taux d'erreur est: 0.1690
Rapport de classification :
              precision    recall  f1-score   support

         0.0       0.66      0.51      0.58       242
         1.0       0.87      0.93      0.89       829

    accuracy                           0.83      1071
   macro avg       0.77      0.72      0.74      1071
weighted avg       0.82      0.83      0.82      1071
```

Comme expliqué dans la section 2.1, on peut maintenant retirer les variables qui ne contribuent pas significativement au modèle.

La matrice de confusion est donnée sur l'image 18

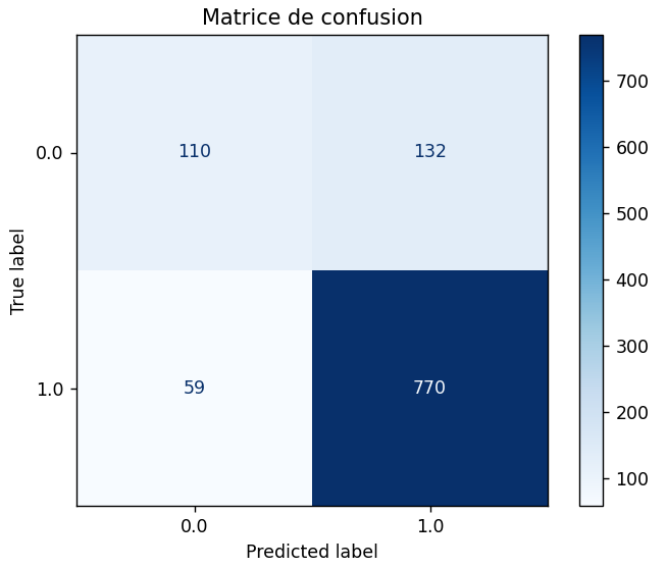


Figure 18: Dynamique: Matrice de confusion pour LDA, modèle réduit

Les résultats associés plus précis sont les suivants:

```
Taux d'erreur est: 0.1783

Rapport de classification :
              precision    recall  f1-score   support

         0.0       0.65      0.45      0.54       242
         1.0       0.85      0.93      0.89       829

    accuracy                           0.82      1071
   macro avg       0.75      0.69      0.71      1071
weighted avg       0.81      0.82      0.81      1071
```

6.4 Choix du modèle

On voit que les erreurs pour les 2 jeux de donnée sont proches l'une de l'autre.
Les classificateurs LDA non réduits obtiennent des meilleurs résultats (17%) par rapport aux modèles réduits. En revanche, on voit que le taux d'erreur n'est pas beaucoup plus élevé. On va donc préférer simplifier notre modèle au coût d'une légère hausse de ce taux d'erreur.
Si on compare les 2 types de jeux de données, on va préférer utiliser le jeu de données dynamique. Il va effectivement permettre de capturer des relations temporelles importantes et donc améliorer la robustesse des prédictions dans le temps.

Notre choix optimal se portera donc sur le modèle réduit avec données dynamiques.

7 Conclusion

Dans ce projet, nous avons comparé plusieurs approches statistiques d'apprentissage pour la prédiction de la qualité de l'air en utilisant deux types de jeux de données : statique et dynamique.
En comparant les modèles, plusieurs conclusions se dégagent :
Le modèle dynamique réduit avec KNN présente l'erreur la plus faible (14.65%) et une stabilité améliorée. Bien que son accuracy moyenne (80.89%) soit légèrement inférieure à celle du modèle statique (82.4%), la prise en compte des variables temporelles offre une prédiction plus robuste et réaliste. Les modèles statiques, bien que performants en termes d'accuracy, perdent en précision lorsqu'ils ignorent les relations temporelles, essentielles pour une problématique comme la qualité de l'air.
Les résultats obtenus dans ce travail, montrent l'importance de sélectionner judicieusement les variables explicatives. De plus nous avons démontré l'intérêt de prendre en compte les interactions temporelles dans les phénomènes de pollution atmosphérique pour améliorer les prédictions et, par conséquent, anticiper efficacement les épisodes de pollution.
L'état de la qualité de l'air à un instant t ne dépend que de deux polluants NO_2 et C_6H_6 . Cependant dans les différents modèle de prédiction que nous avons construit, nous remarquons que 11 des 13 variables contribuent de façon significative aux modèles. On peut donc supposer qu'il existe des relations et réactions entre les polluants et les conditions atmosphériques qui permet de décrire la cinétique de certain polluant, clés dans la bonne qualité de l'air, à l'avance.

8 Appendix: code

```
if __name__ == "__main__":
    # model_type = "Statique"
    model_type = "Dynamique"
    df = dataset_formatting(model_type)
    X = df.drop(columns=['AQI', 'Date', 'Time', 'AQI_lendemain', 'Datetime'])
    y = df['AQI_lendemain']
    seuil_p_value = 0.05
    #comme au devoir 3 en utilisant directement la fonction LogisticRegression accessible, dans le dynamique parfois un probl me avec le max_iter (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
    #------PART 1-----#
    #mod le non r duit
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
    log_model = LogisticRegression(max_iter=10000)
    log_model.fit(X_train, y_train)
    #ISLP website: the predict_proba() method estimates the probability that each observation belongs to a particular class.
    pred_log = log_model.predict_proba(X_test)
    proba_0_1 = np.where(pred_log[:, 1] > 0.5, 1, 0) #changer pour avoir 1 ou 0
    taux_erreur = np.mean(y_test != proba_0_1)
    print(f"Le taux d'erreur est : {taux_erreur:.4f}\n")

    print("\nMatrice de confusion :")
    conf_matrix = confusion_matrix(y_test, proba_0_1)
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(y))
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Matrice de confusion")
    plt.show()
    print("\nRapport de classification :")
    print(classification_report(y_test, proba_0_1))

    #p-value, si dynamique-> on retire les vraiables d cal es dans le temps
    variables_inutiles = variable_inut(X_train, y_train, seuil_p_value)
    if model_type== "Dynamique":
        X_reduit, y= suppr_diff_time_var(df,variables_inutiles)
    else:
        X_reduit = X.drop(columns=variables_inutiles)
    #------PART 2-----#
    # mod le r duit
    X_train, X_test, y_train, y_test = train_test_split(X_reduit, y, test_size=0.3, random_state=42)
    log_model.fit(X_train, y_train)
    pred_log = log_model.predict_proba(X_test)
    proba_0_1 = np.where(pred_log[:, 1] > 0.5, 1, 0) #changer pour avoir 1 ou 0
    taux_erreur = np.mean(y_test != proba_0_1)
    print(f"Le taux d'erreur est: {taux_erreur:.4f}\n")
    print("\nMatrice de confusion :")
    conf_matrix = confusion_matrix(y_test, proba_0_1)
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(y))
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Matrice de confusion")
    plt.show()
    print("\nRapport de classification :")
    print(classification_report(y_test, proba_0_1))
```

Listing 1: Code utilisé pour classification Log

```
if __name__ == "__main__" :
    model_type = "Dynamique"

    k_best = optimize_KNN(model_type=model_type)
    plt.show()

    variables_inutiles = analyse_KNN(k_best,model_type=model_type,seuil_p_value=0.05)

    k_best2 = optimize_KNN_reduit(model_type=model_type,variables_inutiles=variables_inutiles)
    plt.show()

    autres_variables_inutiles = analyse_KNN_reduit(k_best2, model_type=model_type, seuil_p_value=0.05, variables_inutiles=variables_inutiles)
```

```
13
14      compare_models(k_best, k_best2, model_type, variables_inutiles)
```

Listing 2: Code utilisé pour classification KNN

```
1  if __name__ == "__main__":
2      model_type = "Statique"
3      #model_type = "Dynamique"
4      seuil_p_value = 0.05
5      df = dataset_formatting(model_type)
6      X = df.drop(columns=['AQI', 'Date', 'Time', 'AQI_lendemain', 'Datetime'])
7      y = df['AQI_lendemain']
8      #d'abord je v rifie si LDA ou QDA est mieux
9      #section 4.7.3 ISLP_website et devoir 3
10     # loo = LeaveOneOut()
11     # kf = KFold(n_splits=5, shuffle=True, random_state=42)
12     # qda_model = QDA(store_covariance=True)
13     lda_model = LDA(store_covariance=True)
14     # lda_loocv_error = mean_error(lda_model, X_train, y_train, loo)
15     # lda_kfold_error = mean_error(lda_model, X_train, y_train, kf)
16
17     # qda_loocv_error = mean_error(qda_model, X_train, y_train, loo)
18     # qda_kfold_error = mean_error(qda_model, X_train, y_train, kf)
19
20     # print("LDA LOOCV ", lda_loocv_error) #LDA-> bcp mieux
21     # print("LDA 5-Fold ", lda_kfold_error)
22     # print("QDA LOOCV ", qda_loocv_error)
23     # print("QDA 5-Fold ", qda_kfold_error)
24     #------PART 1-----#
25     #mod le non r duit
26     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
27     lda_model.fit(X_train, y_train)
28     y_pred = lda_model.predict(X_test)
29     taux_erreur = np.mean(y_test != y_pred)
30     print(f"Taux d'erreur est: {taux_erreur:.4f}\n")
31
32     #Matrice de confusion et classifiaction rapport
33     conf_matrix = confusion_matrix(y_test, y_pred)
34     disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(y))
35     disp.plot(cmap=plt.cm.Blues)
36     plt.title("Matrice de confusion")
37     plt.show()
38     print("\nRapport de classification:")
39     print(classification_report(y_test, y_pred))
40
41     #p-value, si dynamique-> on retire les vraiables d cal es dans le temps
42     variables_inutiles = variable_inut(X_train, y_train, seuil_p_value)
43     if model_type == "Dynamique":
44         X_reduit, y = suppr_diff_time_var(df, variables_inutiles)
45     else:
46         X_reduit = X.drop(columns=variables_inutiles)
47
48     #------PART 1-----#
49     # mod le r duit
50     X_train, X_test, y_train, y_test = train_test_split(X_reduit, y, test_size=0.3, random_state
       =42)
51     lda_model.fit(X_train, y_train)
52     y_pred = lda_model.predict(X_test)
53     taux_erreur = np.mean(y_test != y_pred)
54     print(f"Taux d'erreur est: {taux_erreur:.4f}\n")
55     #Matrice de confusion et classifiaction rapport
56     conf_matrix = confusion_matrix(y_test, y_pred)
57     disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(y))
58     disp.plot(cmap=plt.cm.Blues)
59     plt.title("Matrice de confusion")
60     plt.show()
61     print("\nRapport de classification:")
62     print(classification_report(y_test, y_pred))
```

Listing 3: Code utilisé pour classification LDA

```
1
```

```
def dataset_formatting(type_dataset = "Statique"):
    air_quality = fetch_ucirepo(id=360)
    # Cr ation du DataFrame
    df = air_quality.data.features.copy()
    # Conversion des colonnes 'Date' et 'Time' en un format datetime combin
    df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], dayfirst=True, errors='coerce')
    # Suppression des lignes avec des valeurs non convertibles
    df = df.dropna(subset=['Datetime']).reset_index(drop=True)
    # Conversion de 'Datetime' en nombre d'heures depuis l' poque  Unix
    df['Datetime'] = df['Datetime'].astype('int64') // 10 ** 9 // 3600
    # Tri des donn es par datetime
    df = df.sort_values('Datetime').reset_index(drop=True)
    df['AQI'] = df.apply(aqi_bon_ou_mauvais, axis=1)
    # Cr ation de la colonne AQI du lendemain    la m me heure
    df['AQI_lendemain'] = df['AQI'].shift(-24)
    # Suppression des lignes o   AQI_lendemain est NaN (celles pour lesquelles nous n'avons pas de
     donn es du lendemain)
    df = df.dropna(subset=['AQI_lendemain']).reset_index(drop=True)
    df_statique = df.copy()
    if type_dataset == "Statique" :
        colonnes_a_normaliser = [col for col in df_statique.columns if
                                 col not in ['Date', 'Time', 'Datetime', 'AQI', 'AQI_lendemain']]
        df_statique = normaliser_donnees(df_statique, colonnes_a_normaliser)
        return df_statique
    if type_dataset == "Dynamique" :
        # Pr papration du dataset pour modeles dynamiques :
        df_dynamique = df.copy()
        # Cr ation des colonnes pour les polluants    h-2 et h-4 pour tous les polluants
    disponibles
        polluants = [col for col in df.columns if col not in ['Date', 'Time', 'Datetime', 'AQI', '
    AQI_lendemain']]
        for p in polluants:
            df_dynamique[f'{p}_h-2'] = df[p].shift(2)
            df_dynamique[f'{p}_h-4'] = df[p].shift(4)

        # Suppression des lignes avec des valeurs manquantes dues au d calage
        colonnes_a_verifier = ([f'{p}_h-2' for p in polluants]
                               + [f'{p}_h-4' for p in polluants])
        df_dynamique = df_dynamique.dropna(subset=colonnes_a_verifier).reset_index(drop=True)
        colonnes_a_normaliser = [col for col in df_dynamique.columns if
                                 col not in ['Date', 'Time', 'Datetime', 'AQI', 'AQI_lendemain']]
        df_dynamique = normaliser_donnees(df_dynamique, colonnes_a_normaliser)
        return df_dynamique

def normaliser_donnees(df, colonnes_a_normaliser):
    scaler = StandardScaler()
    df[colonnes_a_normaliser] = scaler.fit_transform(df[colonnes_a_normaliser])
    return df

def aqi_bon_ou_mauvais(row):
    if (
        #PM10 dans le tableau
        (row['C6H6(GT)'] <= 34) and
        (row['NO2(GT)'] <= 134)
    ):
        return 1
    else:
        return 0

def optimize_KNN(model_type = "Statique"):

    print("Mod le de classification KNN pour un mod le de pr diction : " + model_type)

    # Choisir de dataset selon le mod le que l'on souhaite ("Statique" ou "Dynamique") :
    df = dataset_formatting(type_dataset=model_type)
    X = df.drop(columns=['AQI', 'Date', 'Time', 'AQI_lendemain', 'Datetime'])
    y = df['AQI_lendemain']

    # Mod le KNN :
    nb_donne = len(df['Date'])
    N = int(nb_donne/10)
    err_Nfold = []
    cv_Nfold = KFold(n_splits=N, shuffle=True, random_state=0)  # 10-Fold CV
```

```
    k_vect = np.arange(1,15)
    for k in k_vect:
        knn = KNeighborsClassifier(n_neighbors=k)
        print(f"k = {k}")
        error = 1 - np.mean(cross_val_score(knn, X, y, cv=cv_Nfold, scoring='accuracy'))
        err_Nfold.append(error)

    # Courbes
    plt.plot(1 / k_vect, err_Nfold, label=f'{N}-Fold', color='green')
    plt.xlabel('1 / K')
    plt.ylabel('Erreur de la validation crois ')
    plt.title(f'Erreur de validation crois  pour {N}-Fold CV')
    plt.legend()
    plt.grid(True)

    # Choix du nombre de voisin utilis  dans KNN :
    k_best = k_vect[np.argmin(err_Nfold)]
    print(f"On utilise k = {k_best} qui donne une erreur de {np.min(err_Nfold)}")
    return k_best

def analyse_KNN(k_best, model_type = "Statique", seuil_p_value =0.1):

    print("Mod le de classification KNN pour un mod le de pr diction : " + model_type)

    # Choisir de dataset selon le mod le que l'on souhaite ("Statique" ou "Dynamique") :
    df = dataset_formatting(type_dataset=model_type)
    X = df.drop(columns=['AQI', 'Date', 'Time', 'AQI_lendemain', 'Datetime'])
    y = df['AQI_lendemain']

    # Classificateur KNN optimal :
    knn_final = KNeighborsClassifier(n_neighbors=k_best)
    knn_final.fit(X, y)
    y_pred = knn_final.predict(X)

    # Matrice de confusion :
    conf_matrix = confusion_matrix(y, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(y))
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f"Matrice de confusion mod le {model_type}")
    plt.show()

    # Calcul des p-values pour les variables
    f_stat, p_values = f_classif(X, y)

    # Cr ation d'un DataFrame pour les p-values
    p_values_df = pd.DataFrame({
        'Variable': X.columns,
        'p_value': p_values
    })

    # Tri des variables par importance (p-value croissante)
    p_values_df = p_values_df.sort_values(by='p_value').reset_index(drop=True)
    print("\nClassement des variables par importance (p-value) :")
    print(p_values_df)

    # Affichage des p-values les plus importantes
    plt.barh(p_values_df['Variable'], -np.log10(p_values_df['p_value']), color='blue')
    plt.xlabel("-log10(p-value)")
    plt.title("Importance des variables selon la p-value")
    plt.tight_layout()
    plt.show()

    # Variables inutiles (p-value > seuil_p_value)
    variables_inutiles = p_values_df[p_values_df['p_value'] > seuil_p_value]['Variable'].tolist()
    print(f"\nVariables consid r es comme inutiles (p-value > {seuil_p_value}) :")
    print(variables_inutiles)

    return variables_inutiles

def optimize_KNN_reduit(model_type = "Statique",variables_inutiles = []) :

    print("Mod le r duit de classification KNN pour un mod le de pr diction : " + model_type)
```

```
    # Choisir de dataset selon le mod le que l'on souhaite ("Statique" ou "Dynamique") :
    df = dataset_formatting(type_dataset=model_type)
    X,y = suppr_diff_time_var(df,variables_inutiles)

    # Mod le KNN :
    nb_donne = len(df['Date'])
    N = int(nb_donne/10)
    err_Nfold = []
    cv_Nfold = KFold(n_splits=N, shuffle=True, random_state=0)  # 10-Fold CV

    k_vect = np.arange(1,150)
    for k in k_vect:
        knn = KNeighborsClassifier(n_neighbors=k)
        print(f"k = {k}")
        error = 1 - np.mean(cross_val_score(knn, X, y, cv=cv_Nfold, scoring='accuracy'))
        err_Nfold.append(error)

    # Courbes
    plt.plot(1 / k_vect, err_Nfold, label=f'{N}-Fold', color='green')
    plt.xlabel('1 / K')
    plt.ylabel('Erreur de la validation crois ')
    plt.title(f'Erreur de validation crois  pour {N}-Fold CV')
    plt.legend()
    plt.grid(True)

    # Choix du nombre de voisin utilis  dans KNN :
    k_best = k_vect[np.argmin(err_Nfold)]
    print(f"On utilise k = {k_best} qui donne une erreur de {np.min(err_Nfold)}")
    return k_best

def analyse_KNN_reduit(k_best, model_type = "Statique", seuil_p_value =0.1, variables_inutiles =
    []):

    print("Mod le de classification KNN pour un mod le de pr diction : " + model_type)

    # Choisir de dataset selon le mod le que l'on souhaite ("Statique" ou "Dynamique") :
    df = dataset_formatting(type_dataset=model_type)
    X,y = suppr_diff_time_var(df,variables_inutiles)

    # Classificateur KNN optimal :
    knn_final = KNeighborsClassifier(n_neighbors=k_best)
    knn_final.fit(X, y)
    y_pred = knn_final.predict(X)

    # Matrice de confusion :
    conf_matrix = confusion_matrix(y, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(y))
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f"Matrice de confusion mod le r duit {model_type}")
    plt.show()

    # Calcul des p-values pour les variables
    f_stat, p_values = f_classif(X, y)

    # Cr ation d'un DataFrame pour les p-values
    p_values_df = pd.DataFrame({
        'Variable': X.columns,
        'p_value': p_values
    })

    # Tri des variables par importance (p-value croissante)
    p_values_df = p_values_df.sort_values(by='p_value').reset_index(drop=True)
    print("\nClassement des variables par importance (p-value) :")
    print(p_values_df)

    # Affichage des p-values les plus importantes
    plt.barh(p_values_df['Variable'], -np.log10(p_values_df['p_value']), color='blue')
    plt.xlabel("-log10(p-value)")
    plt.title("Importance des variables selon la p-value")
    plt.tight_layout()
    plt.show()
```

```
    # Variables inutiles (p-value > seuil_p_value)
    variables_inutiles = p_values_df[p_values_df['p_value'] > seuil_p_value]['Variable'].tolist()
    print(f"\nVariables consid r es comme inutiles (p-value > {seuil_p_value}) :")
    print(variables_inutiles)

    return variables_inutiles

def compare_models(k_best, k_best_reduit, model_type, variables_inutiles):
    """
    Compare les performances des mod les non r duit et r duit.
    """
    # Chargement des datasets
    df = dataset_formatting(type_dataset=model_type)
    X_full = df.drop(columns=['AQI', 'Date', 'Time', 'AQI_lendemain', 'Datetime'])
    X_reduit,y = suppr_diff_time_var(df,variables_inutiles)

    nb_donne = len(df['Date'])
    N = int(nb_donne / 10) # Utilis  pour la validation crois

    # Mod le non r duit
    knn_full = KNeighborsClassifier(n_neighbors=k_best)
    scores_full = cross_val_score(knn_full, X_full, y, cv=N, scoring='accuracy')

    # Mod le r duit
    knn_reduit = KNeighborsClassifier(n_neighbors=k_best_reduit)
    scores_reduit = cross_val_score(knn_reduit, X_reduit, y, cv=N, scoring='accuracy')

    # Affichage des r sultats
    print("\n### Comparaison des mod les ###")
    print(f"Mod le non r duit (k = {k_best}):")
    print(f"  - Accuracy moyenne : {scores_full.mean():.4f}")
    print(f"  -  cart -type : {scores_full.std():.4f}")

    print(f"\nMod le r duit (k = {k_best_reduit}):")
    print(f"  - Accuracy moyenne : {scores_reduit.mean():.4f}")
    print(f"  -  cart -type : {scores_reduit.std():.4f}")

    # Graphiques des performances
    plt.figure(figsize=(10, 6))
    plt.bar(['Non r duit', 'R duit'], [scores_full.mean(), scores_reduit.mean()],
            yerr=[scores_full.std(), scores_reduit.std()], capsize=5, color=['blue', 'orange'])
    plt.ylabel(f"Accuracy moyenne (CV {N}-fold)")
    plt.title("Comparaison des performances des mod les")
    plt.grid(axis='y')
    plt.tight_layout()
    plt.show()

def suppr_diff_time_var(df,variables_inutiles):
    # Si une variable est inutile   h-i pour i=0,2,4 alors on estime qu'elle est
    # inutile pour toute heure. On la supprime donc du dataset de fa on globale
    var_inutiles_forall_h = variables_inutiles.copy()
    for var in variables_inutiles:
        # Retirer les suffixes "_h-2" ou "_h-4" si pr sents
        base_var = var.replace("_h-2", "").replace("_h-4", "")
        for colonne in df.columns:
            # V rifier si la colonne correspond   la variable inutile (pour toutes les heures)
            if base_var in colonne and colonne not in var_inutiles_forall_h:
                var_inutiles_forall_h.append(colonne)

    X = df.drop(columns=['AQI', 'Date', 'Time', 'AQI_lendemain', 'Datetime']+var_inutiles_forall_h
    )
    y = df['AQI_lendemain']

    print("On supprime : ",var_inutiles_forall_h)

    return X,y
def variable_inut(X, y, seuil_p_value):
    f_stat, p_values = f_classif(X, y)
    p_values_df = pd.DataFrame({
        'Variable': X.columns,
        'p_value': p_values
    }).sort_values(by='p_value').reset_index(drop=True)
    plt.barh(p_values_df['Variable'], -np.log10(p_values_df['p_value']), color='blue')
```

```
    plt.xlabel("-log10(p-value)")
    plt.title("Importance des variables selon la p-value")
    plt.tight_layout()
    plt.show()
    variables_inutiles = p_values_df[p_values_df['p_value'] > seuil_p_value]['Variable'].tolist()
    print(f"\nVariables consid r es comme inutiles (p-value > {seuil_p_value}) :")
    print(variables_inutiles)
    return variables_inutiles

def mean_error(model, X, y, cv):
    errors = []
    #pour chaque fold
    for i, j in cv.split(X):
        model.fit(X.iloc[i], y.iloc[i])
        predictions = model.predict(X.iloc[j])
        error = np.mean(predictions!=y.iloc[j])
        errors.append(error)
    return np.mean(errors)
```

Listing 4: Fonctions utilisées dans le projet