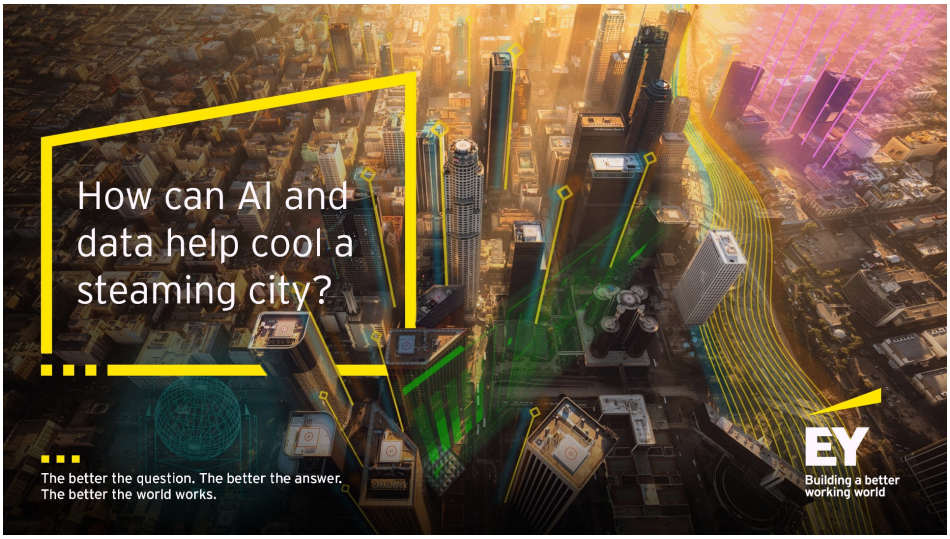
EY Open Science AI & Data Challenge 2025 : Cooling Urban Heat Islands

- Enzo Sebiane -



1 Introduction

Ce document retrace le cheminement suivi pour le challenge **EY Open Science AI & Data Challenge 2025 : Cooling Urban Heat Islands**, depuis la génération des données jusqu'à l'entraînement du modèle et la soumission des résultats, de manière concise et avec une discussion centrée sur les idées et difficultés rencontrées.

2 Observation rapide des données fournies

On se place alors dans le cas d'un problème de régression, concernant l'**UHI index**, qui est un indice directement lié à la température. Donc, pour une prédiction, il semble qu'un modèle extrêmement complexe ne soit pas essentiel, en espérant que les données/features extraites en entrée soient suffisamment représentatives. Par ailleurs, les données mises à disposition semblent assez logiques : **traces infrarouges des satellites**, **présence d'espaces verts**, **bâtiments**, etc. Le lien entre ces métadonnées et la prédiction attendue semble cohérent à première vue.

Tout d'abord, le jeu de données d'entraînement est déjà normalisé et sa répartition ressemble à une gaussienne centrée en 1, comme indiqué dans les slides de présentation du challenge. On peut alors prendre du recul : les données ont été récupérées sur une journée et relativement au même endroit géographiquement (Manhattan), donc on ne s'attend pas à de grandes variations ni à beaucoup de valeurs différentes ; et dans les faits, sur **11 229 points** de coordonnées différentes constituant le training set, on a seulement **595 valeurs uniques**, ce qui ne représente pas une continuité parfaite.

C'est pourquoi mes premières réflexions ont tourné autour d'une **classification** plutôt que d'une régression, car si cela s'était avéré possible, il aurait été facile d'obtenir un **modèle simple très performant**. Dans les faits, **595 classes représentent une trop grande proportion** comparée à la taille du training set, d'autant plus que certaines classes ne possédaient qu'**un seul point**, donc pas

assez de diversité par classe, et le lien entre les points de mêmes indices n'était pas forcément évident en traçant les coordonnées sur une carte.

Rapidement, pour ce qui est du *submission template*, on a envrion **10% des 12 309 échantillons globaux** et si on trace les coordonnées, elles semblent avoir été tirées de manière aléatoire, ce qui nous incite bien à ne pas nous servir des coordonnées comme d'une suite spatiale dans laquelle on pourrait prédire la position suivante.

Une fois ces informations prises en compte, on peut passer à l'étape de la **génération des données**.

3 Préparation et Génération des Données

J'ai pris connaissance de ce challenge relativement tard (3 semaines avant la fin), je n'ai donc pas pris le temps d'étudier en détail les données les plus pertinentes et j'ai, de manière générale, utilisé un jeu de données assez conséquent, avec beaucoup de métadonnées probablement peu utiles.

Dans les faits, les données principales proviennent des satellites **Landsat** et **Sentinel**, enrichies par des données issues du fichier `BuildingFootprint.kml`. La logique suivie est la suivante : au vu du nombre de données uniques assez faible pour l'UHI index, j'ai donc trié mes recherches en mettant plus de poids aux **données contenant beaucoup de valeurs uniques**.

C'est pourquoi je me suis principalement attaché aux **données du satellite Landsat** : sur de simples tests en prenant les notebooks d'exemple mis à notre disposition, il présentait une précision sur les données plus pertinente selon moi (à savoir de meilleurs résultats pour moins de données) que le satellite Sentinel. En observant sur le site officiel de quoi étaient constituées les données, j'ai également pu savoir depuis quand le satellite est en service. De cette manière, j'ai trié toutes les images disponibles avec un seuil de nuages très faible (visuellement, moins de 10% semblait suffisant) et j'ai extrait d'abord les **bandes LWR11**, car directement reliées à la température, mais aussi les autres bandes par la suite, de toutes les années depuis **2013**.

Cette première génération de dataset m'a permis assez vite d'atteindre les **80% de précision**, ce qui est assez intéressant.

Pour enrichir ce jeu de données, je me suis tout de même intéressé au **satellite Sentinel**. Même si les distinctions des bandes entre les différentes positions sont moins importantes, combinées avec mes données de Landsat, j'obtenais au final plus de valeurs uniques que les données de Landsat seules sur le jeu d'entraînement. Cela me permettait alors d'enrichir les features, car on apportait encore plus de précision pour les points du training set donnant le même UHI index.

Cependant, j'étais bien conscient que les features de Landsat permettaient de bons résultats, donc je ne voulais pas polluer mes données en ajoutant trop d'informations issues de Sentinel. C'est pourquoi je me suis restreint uniquement aux bandes **B01, B04, B06, B08**, de sorte à garder une quantité de features plus importante (en proportion **70-30**) issues de Landsat.

Remarque : J'ai tenté d'ajouter d'autres indices ou encore de combiner les bandes pour obtenir des indices (**NDVI**, **NDBI**), mais les résultats ne semblaient pas drastiquement changer à première vue, donc je suis resté avec les **bandes brutes**.

4 Choix du Modèle

Il était alors temps de passer aux premiers modèles : après avoir exploré les bibliothèques de `scikit-learn` (Random Forest, Hist Gradient Boosting) ou encore XGBoost, j'ai trouvé plus intéressant de **passer à mon propre modèle**, d'une part d'un point de vue instructif, mais aussi car j'avais ainsi la main sur tous les paramètres — ce qui, comme on le verra par la suite, s'est avéré utile.

Un **modèle dense** (DNN) a alors été choisi afin d'éviter d'ajouter des traitements spécifiques aux données d'entrée. J'ai alors testé plusieurs architectures, et en termes de résultats bruts, les scores

étaient globalement similaires pour les modèles contenant trois couches et ceux en contenant davantage. Après plusieurs essais, le **modèle final retenu comporte six couches**.

L'idée derrière ce choix est intuitive : assurer une **transition progressive vers une régression**, en réduisant le nombre de neurones de façon douce. Pour les trois dernières couches, on divise en puissances de deux avant la régression (64-32-1), permettant ainsi, en cas de fine-tuning, de **ne modifier que les premières couches** (les plus larges), tout en conservant cette diminution progressive du nombre de neurones.

Plusieurs avantages à cette architecture : les données brutes en entrée sont suffisantes, et grâce à sa construction, le réseau dense peut identifier l'importance des features via des activations différenciées, et produire des résultats cohérents malgré l'absence de prétraitement fin.De plus le nombre de paramètres reste assez faible : de l'ordre de 350 000.

Le modèle a alors été implémenté en `PyTorch` et **entraîné sur GPU**.

5 Analyse des Résultats

À ce stade, le score ne dépassait pas **87%**. Ce plafond peut s'expliquer par le fait que les modèles denses, bien qu'efficaces, ne capturent pas nécessairement les relations entre les features. Par exemple, aucune information n'est fournie pour indiquer de quel **satellite** provient une colonne de données.

La question d'un changement de modèle s'est alors posée. Deux idées principales sont apparues :

— passer à un modèle **convolutif**, en traitant les features comme une **image 2D**, permettant aux noyaux convolutifs de détecter des **patterns** entre les différentes bandes ;

— passer à un modèle **récurrent**, en utilisant des couches **LSTM**, voire **Bi-LSTM**, pour considérer les suites de features comme des **séquences temporelles**.

Ces deux alternatives, bien que potentiellement plus performantes à long terme, augmentaient drastiquement le nombre de paramètres et le temps d'inférence, tout en nécessitant un **prétraitement complexe des données**.

En conclusion, les résultats ne s'amélioraient pas significativement par rapport au **DNN** initial, tandis que le temps de traitement augmentait considérablement.

Dans un souci d'optimisation rapide, j'ai donc conservé le **DNN** et concentré mes efforts sur l'amélioration des **données d'entrée**.

Remarque : j'ai volontairement conservé les **lignes et colonnes en double** ; curieusement, cela améliorait très légèrement les performances.

J'ai alors adopté une nouvelle stratégie : compenser l'absence de relations explicites dans les données par l'ajout de **contexte**.

6 Ajout de Contexte

6.1 Prise en compte des bâtiments

Dans un premier temps, j'ai intégré les **footprints des bâtiments**.

Les données du fichier `BuildingFootprint.kml` ont été utilisées de manière originale : les polygones représentant les bâtiments ont été élargis via un `buffer` de manière à couvrir toute la zone étudiée (Manhattan), puis convertis en une image, agissant comme une carte d'occupation du sol. Ces informations ont été concaténées au dataset principal, en complément des features satellites.

Pourquoi est-ce intéressant ? Même sans utiliser directement la donnée principale de ce dataset, on obtient **512 clusters distincts**. En ajoutant cette information brute comme première colonne des features, on fournit au réseau une forme de pré-classement : une sorte de clustering implicite, comme un apprentissage non supervisé intégré.

Le score atteignait alors près de **90%**, validant l'hypothèse que c'est bien le contexte qui faisait défaut.

6.2 Génération Multi-Résolution

Pour continuer dans cette direction, je suis reparti de mes scripts pour les satellites **Landsat** et **Sentinel**, afin de générer les données de manière similaire, mais cette fois à plusieurs résolutions : 1, 20, 30, 50 et 80 mètres par pixel. Ces choix, bien que quelque peu arbitraires, permettent de couvrir plusieurs échelles. Jusqu'ici, les données étaient uniquement à 30 m/pixel ; on quadruple donc notre volume de données tout en conservant la répartition 70-30 entre Landsat et Sentinel.

Intuitivement, cela enrichit notre dataset. D'abord, les résolutions plus fines ajoutent de la précision : on passe à environ 9 000 lignes de features uniques pour quelques 10 000 points d'entraînement, augmentant la diversité des données. Ensuite, on ajoute du contexte implicite : deux points proches auront des valeurs similaires à haute résolution, mais seront regroupés à basse résolution.

Ainsi, on injecte du contexte dans notre **DNN** sans avoir recours à un traitement lourd (comme recommandé pour les modèles plus complexes) et sans trop augmenter le nombre de paramètres du modèle. Cette approche a permis d'atteindre une précision de plus de **96%**.

7 Fine-tuning

Pour optimiser ce résultat, et donc concrètement pouvoir prétendre à la finale française, je suis passé à l'étape de fine-tuning afin d'améliorer la performance d'environ un point de pourcentage.

J'ai alors utilisé la bibliothèque **Optuna**, qui permet d'effectuer des recherches bayésiennes de minimums locaux sur différents hyperparamètres. Dans mon cas : les tailles des trois premières couches, le *learning rate* et le *dropout*.

En prenant soin de définir un jeu de test comparable en taille au jeu de soumission final, et après une centaine d'itérations, j'ai pu entraîner un modèle final avec les meilleurs paramètres trouvés sur la totalité des données d'entraînement disponibles. Cela m'a permis d'atteindre mon score final de **97,48%**.

8 Pistes pour la suite

Un premier axe d'amélioration serait d'étudier plus en détail les activations du modèle, ou encore d'analyser des matrices de confusion, afin d'identifier les *features* non pertinentes qui pourraient polluer le dataset.

Ensuite, l'ajout des données issues des **stations météo** représente une piste prometteuse. Une idée, même un peu naïve, serait de considérer les positions comme ayant une dépendance temporelle, et d'associer les températures relevées par les stations à ces points sous forme de *séquences*.

Par ailleurs, approfondir l'analyse des indices **NDVI** et **NDBI** pourrait permettre de réduire le volume de données tout en augmentant leur utilité. Une exploitation plus fine des **empreintes de bâtiments** (*footprints*) est également envisageable.

Enfin, concernant le modèle lui-même, si l'on parvient à organiser les données de manière plus structurée et séquentielle en amont, l'utilisation de réseaux **convolutifs**, **récurrents**, voire de *Transformers*, pourrait mener à l'utilisation de modèles plus complexes, avec de meilleurs résultats. Néanmoins, il est clair que l'essentiel du gain reste à obtenir du côté des *features*.

9 Conclusion

En résumé, l'approche choisie a été de produire davantage de données que strictement nécessaire pour obtenir une régression performante rapidement. C'est pourquoi l'idée du **clustering des bâtiments** et de la **multi-résolution** permettait à mes *features* de contenir intrinsèquement toutes les informations nécessaires, de sorte qu'un simple **DNN** puisse en extraire un modèle performant.

Maintenant, en prenant un peu de recul sur ce projet, il a été très enrichissant, déjà techniquement : chercher des données, trouver les plus pertinentes pour le problème, ou encore explorer les modèles les plus performants dans un intervalle de temps limité...

Cependant il est aussi important de souligner la portée intellectuelle de cette démarche, comme mettre en avant les effets indirects du réchauffement climatique ou encore comprendre le poids de l'urbanisation et son impact sur la température, et ce, même à petite échelle. Ce challenge nous pousse ainsi à prendre conscience des enjeux environnementaux actuels, tout en montrant que les modèles d'IA — parfois critiqués pour leur empreinte carbone — peuvent aussi devenir des outils précieux dans la lutte contre ces défis.