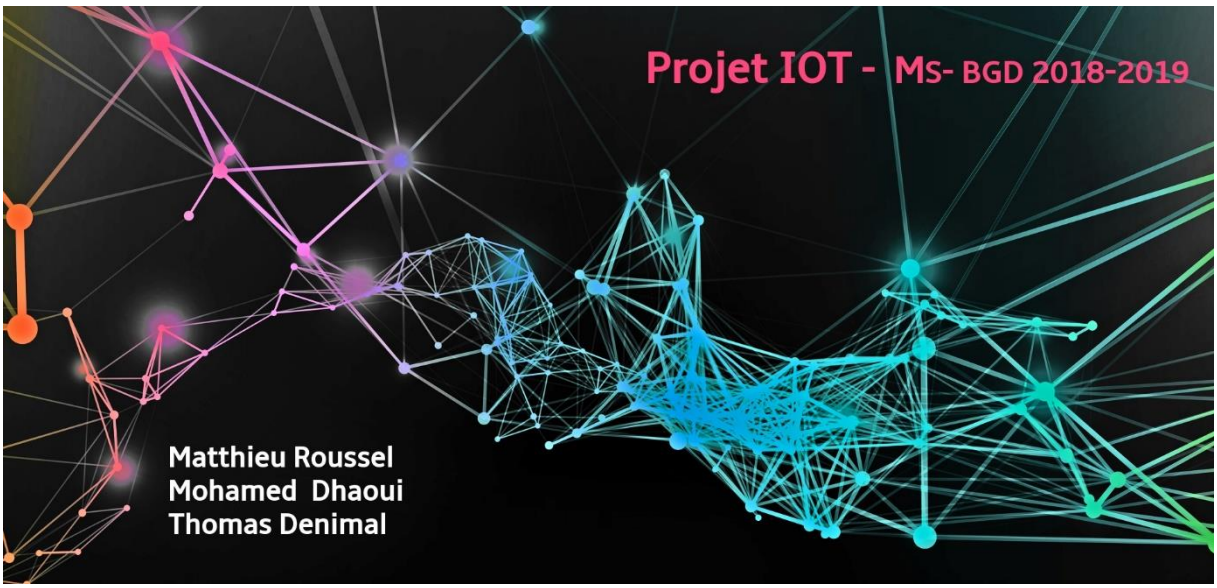
Projet IOT : Prédiction de la position des objets connectés



Les objets connectés envoient à des intervalles de temps réguliers des messages qui transitent par le réseau GSM. Des antennes appelées stations de base reçoivent ces messages et sont chargées de les transmettre via un réseau filaire. Notre objectif est d'implémenter un algorithme d'apprentissage automatique pour prédire la position d'un objet à partir des messages réceptionnés par les stations de base.

Pour ce faire, on commencera par une analyse descriptive et graphique des données, ensuite, on testera les approches suivantes :

- Une approche RSSI / Distance avec ajout de la densité urbaine puis régression XGBOOST
- Metric Learning & KNN ajusté
- KNN adaptatifs

Plan :

1- Chargement & Exploration des données

L'extraction des données se fait simplement sous Python via la librairie pandas. On dispose de 3 jeux de données. Deux jeux de données pour le train composés des différentes données géographiques de base et de réception d'un message suivant une puissance définie et un jeu de test.

Le fichier pos_train_list contient les coordonnées réelles des messages (notre target) . Pour les fichier Mess_train , on se focalisera sur les variables suivantes :

- Messid : L'identifiant du message
- RSSI : Le RSSI en dBm (une mesure de puissance du message réçu)
- bsid : L'identifiant de la station
- did : L'identifiant du device
- bs_lat : La latitude de la station ayant reçu le message
- bs_long : La longitude de la station ayant reçu le message

Avant de nous lancer dans le traitement des données , nous avons essayé de mieux les comprendre en utilisant des tableaux et des plots descriptifs. La première partie du notebook contiendra :

- Des statistiques basique sur le jeux de données (moyenne , densité , écart-type ..)
- Des cartes permettant de visualiser les stations et les messages
- Une analyse d'outliers avec DBSCAN

2- XGboost avec densité urbaine

Notre première approche va consister à tenter de mettre en valeur la relation entre RSSI / distance et on démontrera l'insuffisance de ces données pour établir notre modèle :

- L'affaiblissement n'est pas seulement fonction de la distance mais aussi du terrain, par exemple la densité du tissu urbain.
- On a tenté d'enrichir notre modèle avec des données de densité de population donnant une indication supplémentaire pour expliquer l'affaiblissement.

En plottant le RSS en fonction des distances des messages , On a vu qu'il existe une relation RSSI / distance mais celle-ci est imparfaite : certaines rssi très affaiblies sont pourtant issu de devices très proches .Une explication à ce phénomène serait la nature du terrain et notamment la densité urbaine. L'idée serait d'ajouter cette information à notre modèle (à partir du dataset Population Density du SEDAC). On a constaté que la densité urbaine permet d'expliquer en partie l'anomalie concernant le rssi.

Pour la partie modélisation, on a choisi la régression avec XGBoost. On a défini une feature matrix avec l'ensemble des stations et le rssi associé par station/message.

Malheureusement cette approche n'était pas performant dans notre contexte et on a obtenu une erreur de prédiction après cross-validation égale à **4.40 km**

3. Metric Learning & KNN

La méthode XGBOOST n'a pas donné des résultats satisfaisants, de ce fait, on va essayer d'autres méthodologies.

Notre seconde approche consiste à créer une métrique qui mesure la distance entre deux messages en se basant sur leurs vecteurs des RSSis . Pour ce faire, on utilisera une approche supervisée :

- On part de la matrice des features basée uniquement sur les RSSI
- Pour chaque couple de messages on calcule des features de distance (norm1 ,norm 2) et la distance de vincenty (c'est notre target)
- On construit une matrice de features et un vecteur de distance . A partir de cela on estime un modèle qui sera notre métrique pour la problématique de géolocalisation.

Après avoir valider le modèle de la métrique , on l'a utilisé comme mesure de distance dans KNN.

Le calcul de distance au niveau de la classe KNeighborsRegressor de sklearn n'est pas parallélisé et si on utilise une métrique personnalisée , l'algorithme prend énormément du temps pour s'exécuter sur notre jeux de données . De ce fait, on a implémenté une version de KNN « from scratch » avec un calcul vectoriel de distance .

Comme on travaille sur une problématique de régression, le KNN consiste à trouver les k plus proche voisins de chaque message et moyenner leurs positions (longitude et latitude) . Dans notre contexte , on a utilisé la médiane des positions au lieu de la moyenne pour éviter les outliers .

La meilleure valeur de k trouvé est 15. L'erreur de prédiction associé est égale à **4.18 km**

On ne réalise pas de gain par rapport à la première méthode et les résultats de prédiction de cette approche sont similaires à ceux de la modélisation supervisée classique avec XGBOOST

4. KNN adaptatifs

Une approche, plus simple des KNN a au final été retenue :

La modélisation en utilisant les vecteurs de puissance (RSSI), approche judicieuse et largement rependue dans la littérature, a été ici conservée. La question du choix et du calcul de la métrique restait donc posée pour estimer la position par rapport à cette mesure de dispersion entre les vecteurs de puissance. Une recherche exhaustive sur l'ensemble des distances disponibles dans les algorithmes a été réalisée. Les distances précédemment testées ont également été prises en

considération. Au final la distance de Minkowski ($sum(|x-y|p)(1/p),p=5$) a été retenue. Le nombre de voisins à considérer a été tuné à 7 par validation croisée.

Un premier ajustement a été tenté : pondérer la simple moyenne opérée sur les voisins considérés en fonction de leur proximité. Plusieurs pondérations ont été testées (exp(-x), 1/d, ...) améliorant sensiblement les résultats. La pondération a cependant trop tendance à opérer un cut-off trop importants sur les derniers voisins, ayant pour conséquence globale de réduire le nombre de voisins considérés. Autre inconvénient : bien que les performances soient améliorées sur l'échantillon de test, le procédé créé un réel écart entre le train et le test (fort sur apprentissage en train). Ce chemin a donc été finalement écarté.

A partir de ce premier essai, les erreurs ont été analysées, révélant une forte moyenne et une grande dispersion pour les messages captés par peu de balises, et notamment un très fort poids porté par les messages reçus par une unique balise. Un simple abaissement du nombre de voisins à considérer dans ces cas augmentait sensiblement les performances. Le nombre de bs réceptrices a donc été intégré à la matrice de feature, et le knn adapté est tuné pour prendre en compte un nombre différent de voisins selon ce paramètre ("**Adaptive KNN**"). Au final, une légère amélioration.

La vraie difficulté réside dans la présence d'outliers, de données erronées ou falsifiées, rendant la tâche de généralisation très difficile. D'abord focalisés sur les bs très éloignées (> 40km), les balises "anormales" ne posent au final pas trop de souci pour la stabilité par estimation sur les puissances de signal. En revanche, un certain nombre de positions de messages posent question. Des essais d'exclusion d'une partie des messages identifiés des jeux d'apprentissage et test ont été réalisés mais sans arriver à en déterminer l'exhaustivité ni comprendre la réelle nature de ces bruits. Une première sélection , cependant, a été réalisée manuellement, en analysant les trajectoires des devices comportant des points très surprenants. Des sauts dans ces trajectoires ont pu être identifiés, laissant penser à des modifications de données ou dans certains cas à des trajets exceptionnels pour des devices plutôt habituellement urbaines et relativement statiques. Les messages associés n'ont cependant pas été exclus car l'amélioration globale sur ces quelques exclusions ponctuelles n'était pas concluante, et la nature réelle du phénomène incomprise. Il conviendrait d'avancer d'avantages dans cette voie afin d'établir une méthodologie pour isoler/retraiter automatiquement ces points et ajouter de la précision et de la consistance aux modèles proposés. Venant à cette conclusion tardivement, nous n'avons pas eu le temps d'investiguer plus sur le sujet, mais la clé se situe sans nul doute dans l'isolation/le traitement de ce bruit.

La trame restituée dans cette démarche est la suivante :

- Prétraitements : construction de la matrice de features avec ajout du nombre de balises réceptrices par message et d'une catégorisation de ce champs (quelques plots de distribution sur ce champs)
- Ajout de la cible à la matrice de features ; échantillonnage simple servant pour une première évaluation des modèles

- Modélisation : knn adaptatifs, fit et scoring sur les trains et validation set.
- Anomalies : un exemple de première piste de réflexion sur les points abérrants
- Validation croisée à 100 folds sur le modèle retenu

L'ensemble du process a été réalisé en boucle, avec analyse humaine des anomalies permettant de repérer des trajectoires présentant des irrégularités.

La méthode des « adaptives KNN » obtient en moyenne sur 100 folds, une erreur de **2.879 km** en validation.

5 . Prédiction sur le test

Algorithme	**Erreur de Prédiction**
XGboost avec densité urbaine	4.40 km
Metric Learning & KNN ajusté	4.18 km
Adaptives KNN	2.879 km

La méthode des "adaptives KNN" permet d'obtenir le meilleur erreur de prédiction sur notre jeux de données . C'est avec ce dernier modèle que l'échantillon de test a été scoré.

Les résultats de prédiction sont dans le fichier « pred_pos_test_list.csv »

Point d'attention : l'ordre initial des données de test n'est pas forcément respecté. La clé (messid) est néanmoins associée aux prédictions