UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL

iut
CRÉTEIL-VITRY

2018

Rapport mini projet IoT

Etudiant

Yacine BOUSLAHI

20/02/2018

Introduction

L'Internet des objets connectés représente les échanges d'informations et de données provenant de dispositifs du monde réel avec le réseau Internet.
Considéré comme la troisième évolution de l'Internet, baptisé Web 3.0, revêt un caractère universel pour désigner des objets connectés aux usages variés, dans le domaine de la e-santé, de la domotique.
La quantité énorme de données produites par les objets connectés demandent un espace de stockage, vitesse de traitement et souvent bande passante pour le streaming de données audio ou vidéo. Pour certains, la solution idéale à ces problèmes est le cloud computing.
Google et IBM sont les principaux fournisseurs de services de IoT, ils offrent des plateformes pour le stockage, la visualisation et l'analyse de données.

L'Objectif du travail

l'objectif de ce travail est de créer une application dans le domaine des objets connectés qui consiste à simuler un réseau de bus et de stations , en considérant que chaque bus possède un capteur qui transmis sa position vers le cloud , chaque station de bus reçois la position des bus puis calcule le temps d'arriver de chaque bus.
La deuxième partie de ce travail consiste à appliquer un des algorithmes d'apprentissage automatique et de prédire le temps d'attente des bus.

Structure du programme

pour bien décortiquer le problème, nous avons créés 2 classe *Bus.py*, *Station.py* , un fichier *utils.py* pour les fonctions utilitaires et *main.py* pour la fonction principale.
Le fichier *bus_predicition.py* contient analyse les données collectées par les stations, le temps d'attente calculé, puis il prédit le temps d'attente avec un algorithme d'apprentissage automatique.

Diagramme de classe

Dans ce digramme de classe montre la modélisation des différentes classes et interface de notre application. Les deux classes Bus et Station hérite de la classe Client de la librairie de *ibmiotf*.

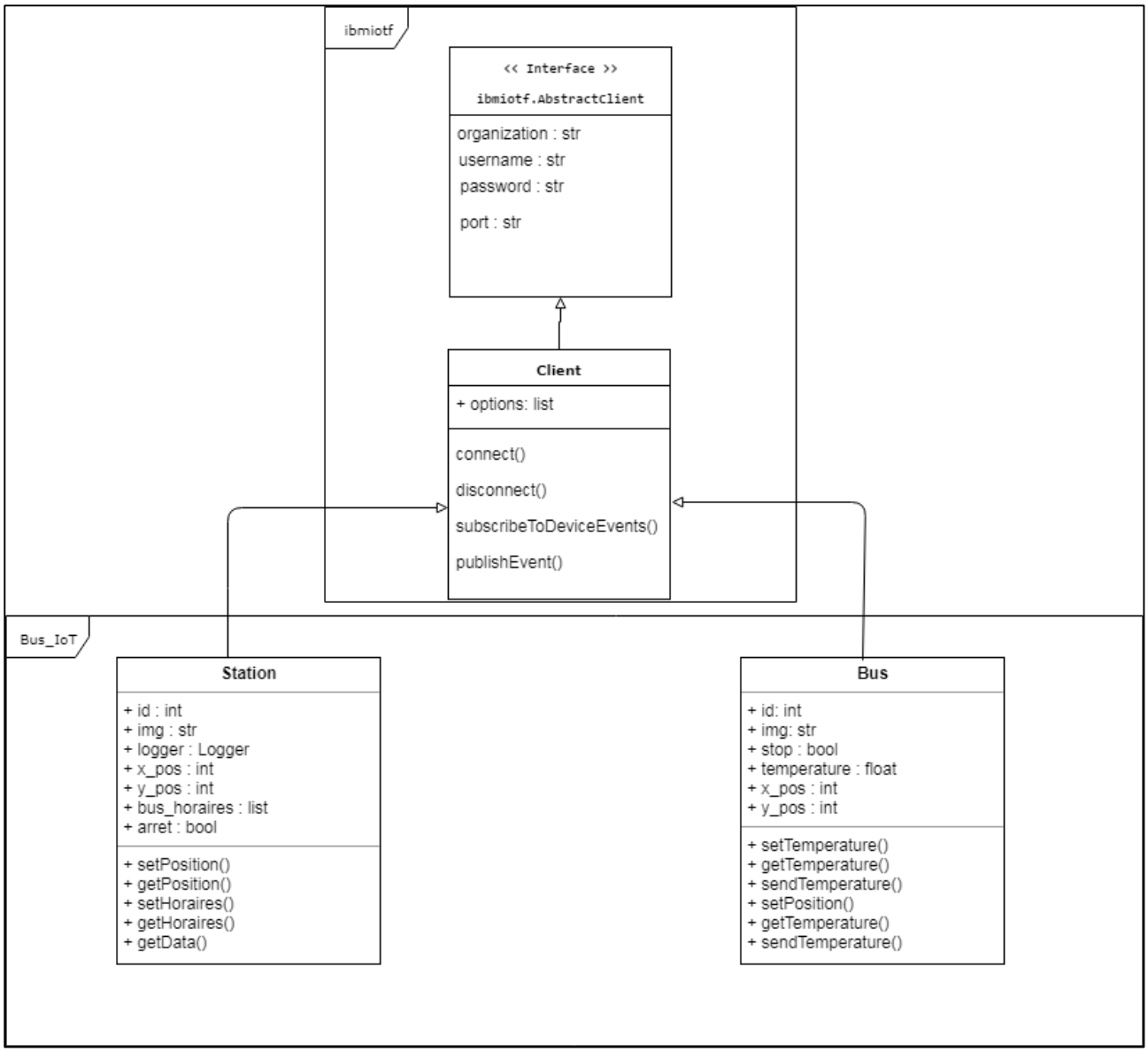


Figure 1 : Digramme de classes

Classe Bus :

La classe bus contient 6 attributs et 6 méthodes:
Id : un identifiant unique de type entier pour chaque bus.
Img : une image qui représente le bus sur l'interface graphique.
Stop : une variable de type booléenne, vraie dans le cas où le bus s'est arrêtée.
Température **:** désigne la température du bus de type réelle, généré d'une manière aléatoire.
X_pos : un entier qui représente la position du bus dans l'interface graphique par rapport à l'axe X.
Y_pos : un entier qui représente la position du bus dans l'interface graphique par rapport à l'axe Y.
setPosition() : une méthode qui sert à initialiser la position du bus.
getPosition() : une méthode qui sert à récupérer la position du bus.
sendPosition() : une méthode qui sert à envoyer la position du bus au cloud.
setTemperature() : une méthode qui sert à initialiser la température du bus.
getTemperature() : une méthode qui sert à récupérer la température du bus.
sendTemperature() : une méthode qui sert à envoyer la température du bus au cloud.

Classe station :

La classe bus contient 7 attributs et 5 méthodes:
Id : un identifiant unique de type entier pour chaque station.
Img : une image qui représente la station de bus sur l'interface graphique.
logger : un fichier de log qui set sauvegarder les données reçues et le temps d'attentes calculé.
Bus_horaires **:** vairiable de type liste
X_pos : un entier qui représente la position du bus dans l'interface graphique par rapport à l'axe X.
Y_pos : un entier qui représente la position du bus dans l'interface graphique par rapport à l'axe Y.
setPosition() : une méthode qui sert à initialiser la position du bus.
getPosition() : une méthode qui sert à récupérer la position du bus.
sendPosition() : une méthode qui sert à envoyer la position du bus au cloud.
setTemperature() : une méthode qui sert à initialiser la température du bus.
getTemperature() : une méthode qui sert à récupérer la température du bus.
sendTemperature() : une méthode qui sert à envoyer la température du bus au cloud.

Génération de vitesse aléatoire

Les bus avancent d'un seul pixel chaque t = `time.sleep(uniform(T_MIN, T_MAX))`.

Utilisation des Thread

Dans notre application, nous avons utilisé Les ``threads" qui sont des unités d'exécution autonomes qui peuvent effectuer des tâches, en parallèle avec d'autres threads, comme pour l'affichage de chaque bus sur l'interface graphique, et l'envoie de données, et les calculs faits par chaque station.

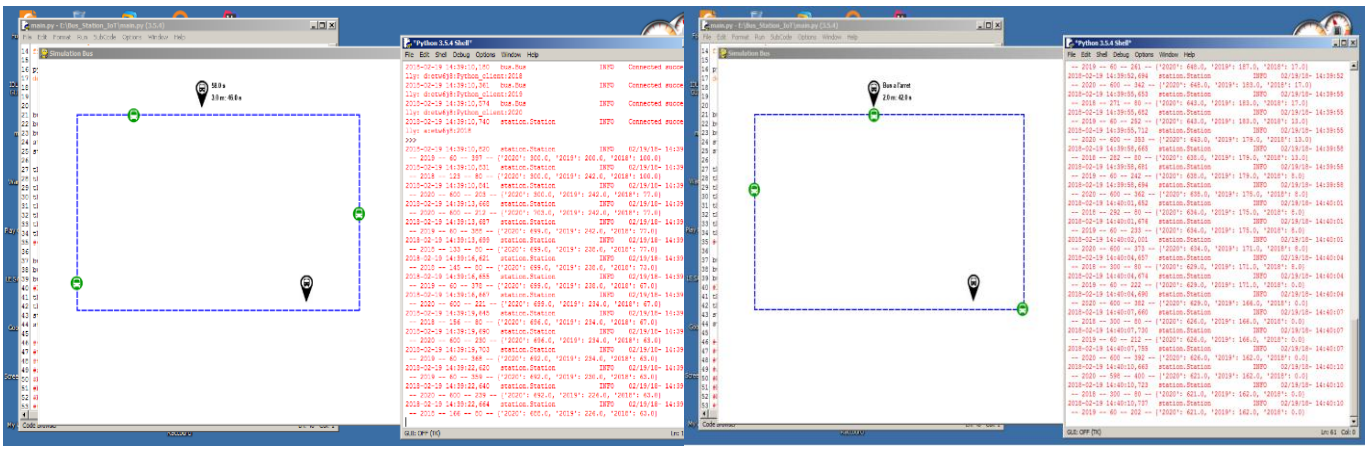
```
thread_bus_1 = Thread(target=running_bus_1, args=(bus_1, False))
thread_bus_1.start()

thread_send_data = Thread(target=send_data, args=(bus_1, bus_2, bus_3))
thread_send_data.start()
```

Quel langage de programmation

Nous avons opté pour le langage Python, qui est un langage de programmation très puissant, de plus en plus utilisé pour développer des applications rapidement et en même temps efficaces. Il est très utilisé dans le monde scientifique pour sa syntaxe aérée et une vitesse d'exécution rapide.

Ce langage est aussi excellent pour la création de prototypes car sa simplicité permet d'implémenter un projet aussi vite que l'on y réfléchit.
Nous avons utilisé plusieurs bibliothèques, qui sont mentionnées dans le fichier *requirements.txt*, et installable avec la commande suivante `pip install -r requirements.txt`.
Pygame : Pour l'interface graphique
Ibmiotf : Bibliothèques client et exemples de connexion à IBM Watson IoT.
matplotlib : Bibliothèque pour tracer et visualiser des données sous formes de graphiques.
numpy : Bibliothèque destinée à manipuler des matrices ou tableaux multidimensionnels.
paho-mqtt : implémentation open source du protocole de messagerie MQTT.



Messagerie MQTT

MQTT est le principal protocole utilisé par les terminaux et les applications pour communiquer avec IBM Watson™ IoT Platform. MQTT est un protocole de transport de messagerie de publication et d'abonnement conçu pour optimiser les échanges de données en temps réel entre le capteur et les terminaux mobiles.

MQTT s'exécute sur TCP/IP, et bien qu'il soit possible d'effectuer directement un codage pour TCP/IP, vous pouvez également choisir d'utiliser une bibliothèque qui traite pour vous les détails du protocole MQTT. Une vaste gamme de bibliothèques client MQTT est disponible. IBM contribue au développement et au support de plusieurs bibliothèques client, y compris celles qui ne sont pas disponibles au niveau des sites suivants :

Il existe également des limitations de taille pour le contenu des messages sur Watson IoT Platform. Les messages peuvent contenir n'importe quelle chaîne valide, cela dit, les formats JSON ("json"), texte ("text") et binaire ("bin") sont les plus couramment utilisés.

Le tableau suivant présente les restrictions relatives au contenu des messages pour les différents types de format

Format de contenu	**Instructions pour des cas d'utilisation spécifiques**
JSON	JSON est le format standard pour Watson IoT Platform. Si vous prévoyez d'utiliser les tableaux de bord, les tableaux et les cartes et les fonctions d'analyse Watson IoT Platform intégrés, assurez-vous que le format de contenu de message est conforme au texte JSON correctement mis en forme.
Texte	Utilisez un codage de caractères UTF-8 valide.
Binaire	Aucune restriction.

Important : La taille maximale de contenu de message sur Watson IoT Platform est 131072 octets. Les messages dont le contenu est supérieur à la limite sont rejetés. Le client en cours de connexion est déconnecté et un message s'affiche dans les journaux de diagnostic

Apprentissage automatique

Prétraitement

Dans un premier temps nous avons collecté les donnes dans un fichier log *activity.log* qui contient un journal d'activité de notre application sous forme de donnée séparé par des --

02/01/18- 20:16:36 -- 2018 -- 131 -- 80 -- {'2019': 1000.0, '2018': 1000.0, '2020': 1000.0}.

Puis nous avons supprimé les lignes dupliquées et générer un fichier *train.txt*, par la fonction `delete_duplicate()`.

la fonction `text_to_csv()` nous a permis de sélectionner toutes les données(lignes) qui correspondent au bus 2018 et générer un fichier au format csv train.csv qui sera utilisé dans la prédiction du temps d'attente du bus.

id_bus	**x_position,**	**y_position**	**wainting_time**
2018	141	80	74.0

Prédiction

Concernant la partie prédiction nous avons implémenté une fonction *predict()* qui applique la régression bayésienne pour prédire le temps d'attente du bus 2018.

Le résultat est représenté sous format d'un graphe ci-dessous.

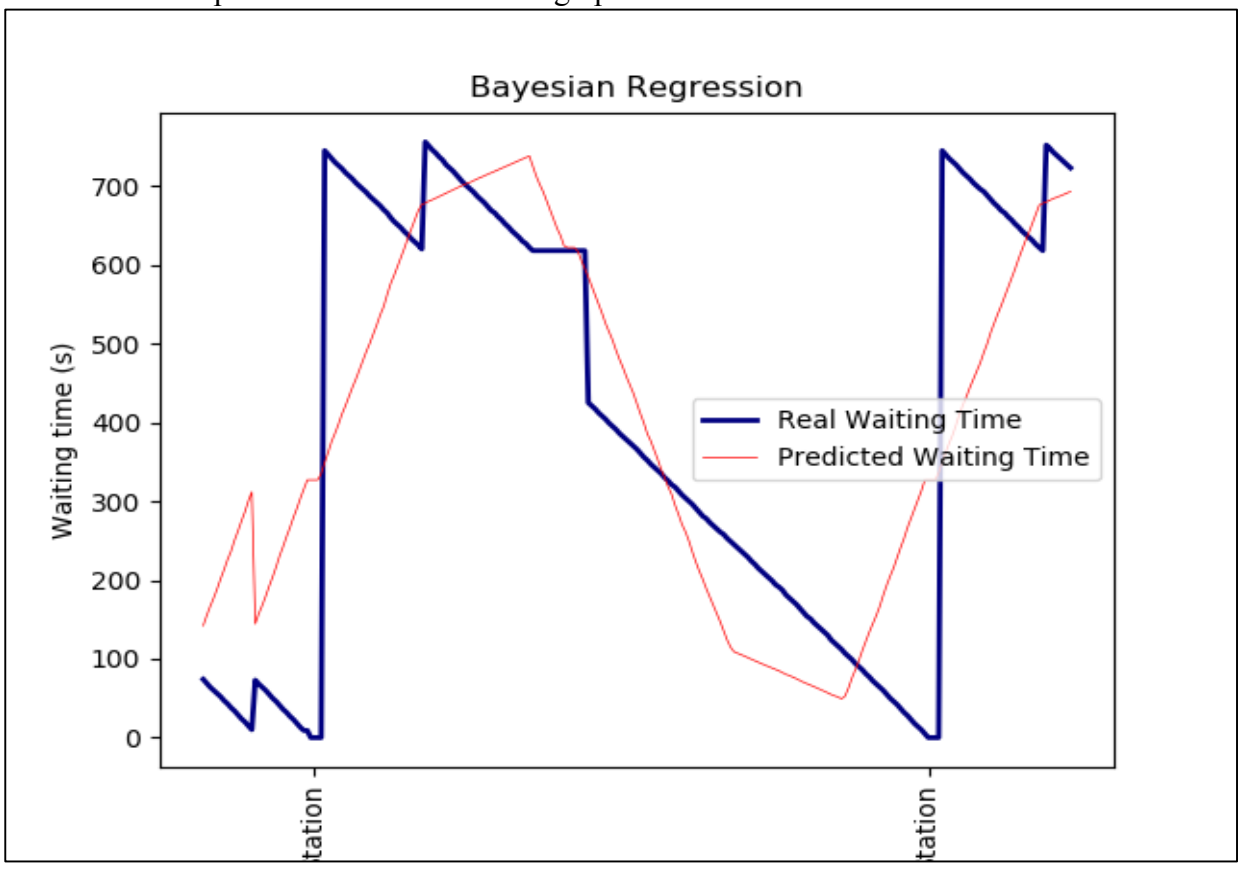


Figure 2 comparaison des temps d'attente (réel, prédis)

Conclusion

Pendant ce travail nous avons pu mettre en pratique nos connaissances théoriques du domaine des objets connectés, en utilisant les technologies IBM comme la plateforme *bluemix*, le protocole MQTT et Python comme langage de programmation, ainsi faire une analyse des données générées par les techniques de Machine Learning en appliquant l'algorithme de régression bayésienne afin de prédire le temps d'attente des bus.

Référence

http://www.digitaldimension.solutions/blog/avis-d-experts/2015/02/mqtt-un-protocole-dedie-pour-liot/
http://www.github.com/ibm-watson-iot/iot-python
https://console.bluemix.net/docs/services/IoT/applications/libraries/python.html#python
https://console.bluemix.net/docs/services/IoT/reference/mqtt/index.html#ref-mqtt
https://winpython.github.io
http://scikit-learn.org/stable/
http://scikit-learn.org/stable/modules/linear_model.html#bayesian-regression