.

Rapport de projet

NLP et classification d'intention

CLOAREC Maelle
CROISSANT Pierrick

1 Analyse du modèle

1.1 Données fournies

Loss	Precision	Recall	F1-score
0.015	0.883	0.665	0.758

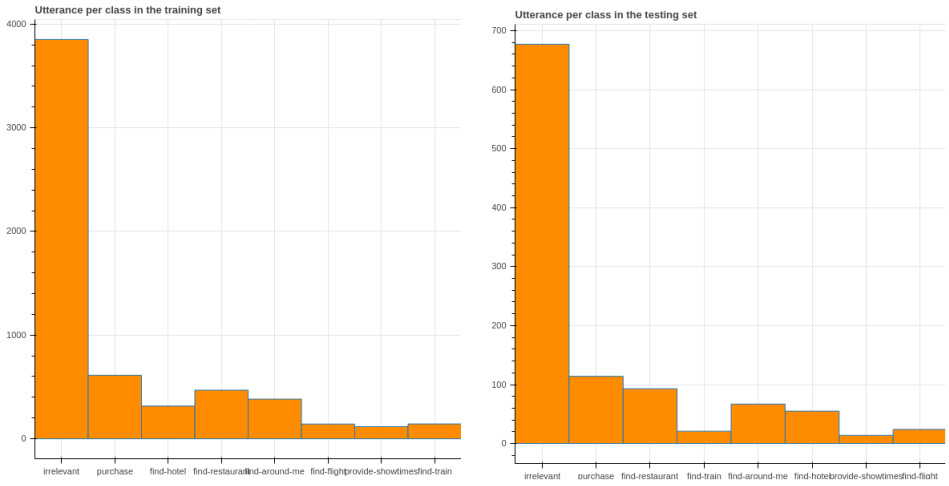
Tout d'abord la précision est élevée, mais le *recall* est plus faible, cela baisse la valeur du F1-score. Même si globalement le F1-score est correct, il n'est pas indiqué s'il s'agit du F1-score macro ou micro. Ainsi, la seule information est que le modèle dans son ensemble prédit bien, mais répond-t-il bien spécifiquement à ce qu'il lui est demandé ?

1.2 Informations manquantes

Effectivement, aucune des métriques ne fournit d'information sur les performances du modèle indépendamment sur chaque *intent*, ce qui est important puisque les besoins sur la précision varient en fonction de l'intention. Par exemple, il vaut mieux que le modèle ait un F1-score de 0.99 sur « *find-restaurant* » et de 0.50 sur « *irrelevant* » que d'être à 0.70 sur les deux. Plusieurs visualisations classiques sur les performances du modèles manquent également, telles que les matrices de confusions, pour voir quelles sont les intentions confondues. De plus, il n'y a pas de statistiques fournies sur le *dataset*. Bien que les linguistes soient de confiance, il est appréciable d'avoir des chiffres, notamment sur la répartition des *samples* dans les *intents*, afin de s'assurer que les données sont bien équilibrées.

1.3 Éléments ajoutés et analyse

Nous avons réalisé tous les éléments cités dans le paragraphe précédent. Premièrement, le *dataset* de *training* est complètement déséquilibré, il est composé à 64% de phrases étiquetées en « *irrelevant* ». Cela est évidemment problématique, les métriques globales peuvent être faussées puisque les autres classes ne sont pas assez représentées, avoir des bons résultats uniquement sur l'*intent* « *irrelevant* » hausse le F1-score global, alors que ce n'est pas l'intention la plus importante.



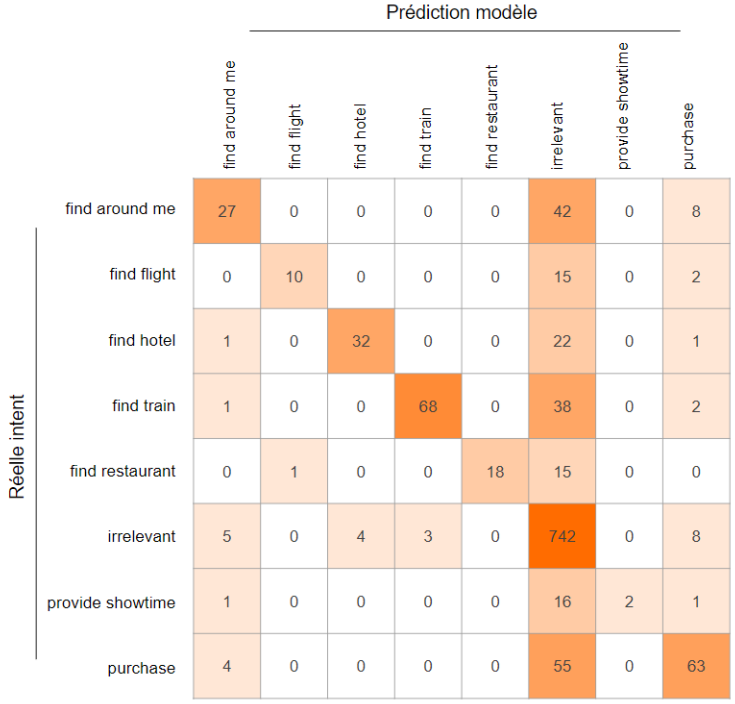
	Accuracy	Precision	Recall	F1-score
Micro	80.5%	0.805	0.805	0.805
Macro	80.5%	0.523	0.853	0.624

Pour en venir justement aux métriques, nous les avons calculées de façon micro et macro sur le dataset de test fourni. Aucun des scores ne concorde avec ceux obtenus par les anciens stagiaires (stagiaires 2020 »> stagiaires 2019), mais cela les valeurs fournies doivent être celles obtenues avec toutes les *intents* et non le *dataset* réduit. Tout de même, la différence est importante entre les métriques en micro et en macro, cela s'explique encore une fois par la répartition inégale du *dataset*, en micro les prédictions sont considérées dans leur ensemble, alors qu'en macro elles le sont par classe. Les scores en macro sont donc plus faibles, mais la précision reste très correcte, même meilleure qu'en micro, ce qui est une bonne nouvelle car dans le cas d'utilisation d'Iwidii, la précision est à favoriser au rappel.

Intent	Precision	Recall	F1-score
find-around-me	0.88	0.43	0.58
find-flight	0.88	0.29	0.44
find-hotel	0.78	0.38	0.51
find-restaurant	0.98	0.56	0.71
find-train	0.93	0.67	0.78
irrelevant	0.79	0.98	0.87
provide-showtimes	0.80	0.29	0.42
purchase	0.79	0.59	0.67

Seul le *recall* de « *irrelevant* » est élevé, mais sa précision fait partie des plus faibles. Pour les autres classes, c'est le cas inverse. Autrement dit, les phrases étiquetées « *irrelevant* » sont bien identifiées par le modèle, mais surtout un

nombre important de phrases sont prédites « *irrelevant* » alors qu'elles ne le sont pas. Par conséquent, le *recall* pour les autres classes sont beaucoup plus faibles, mais quand une phrase est prédite dans une autre classe, il y a plus de chance que le modèle ait correct. Ainsi, le modèle a tendance à classer les phrases en « *irrelevant* » la majorité du temps, ce qui est bien cohérent avec la matrice de confusion.

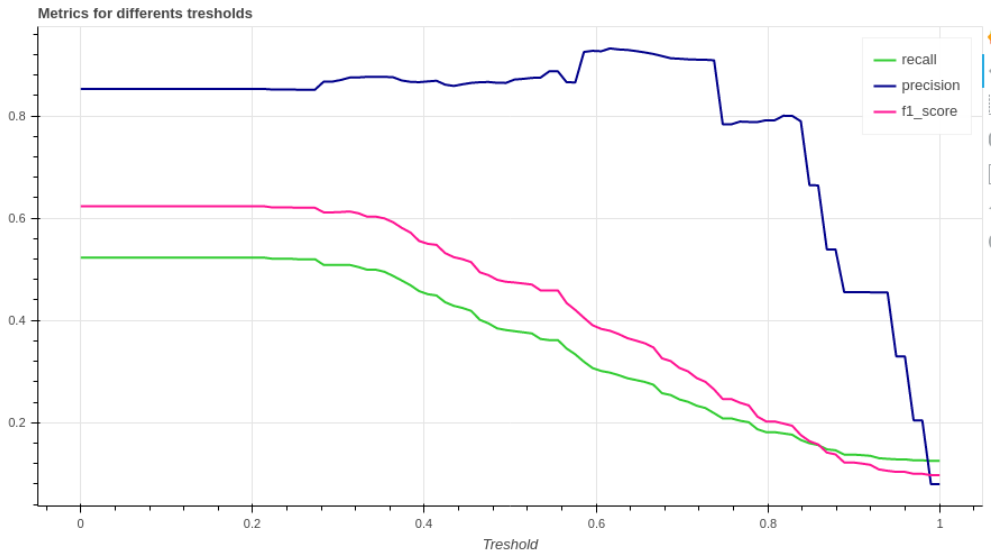


De plus, les intentions « *found around me* » et « *purchase* » ressortent car elles sont confondues avec toutes les autres *intents*, tout de même ces valeurs sont négligeables par rapport à la colonne « *irrelevant* ».

2 Seuil de validation

2.1 Global

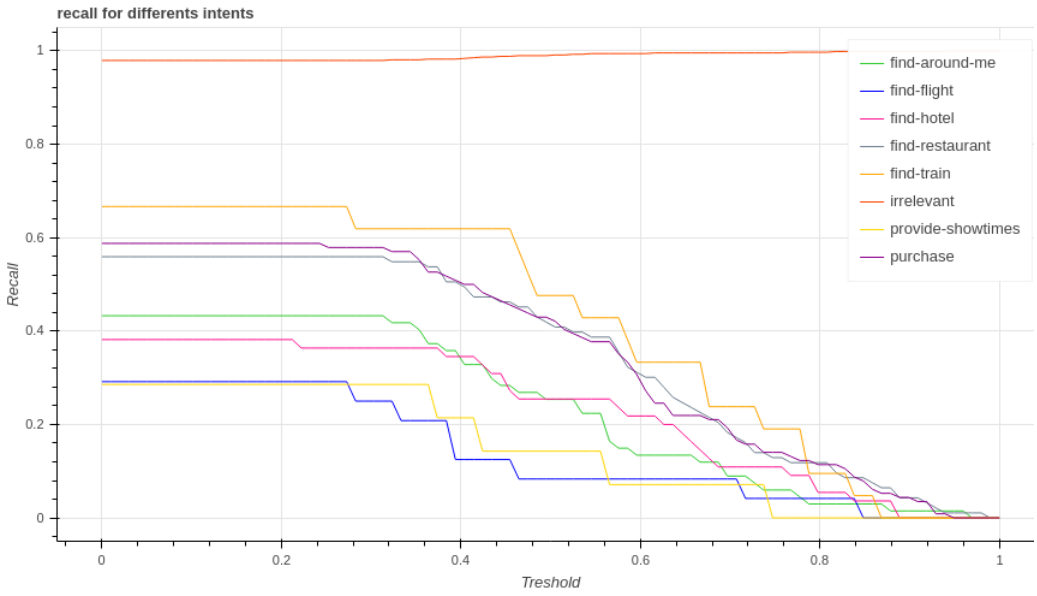
Un premier graphe global montre les différentes métriques obtenues en fonction du seuil de validation. Sachant que lorsque le *treshold* n'est atteint par un seuil de confiance sur aucune *intent*, alors la phrase est prédite en « *irrelevant* ».

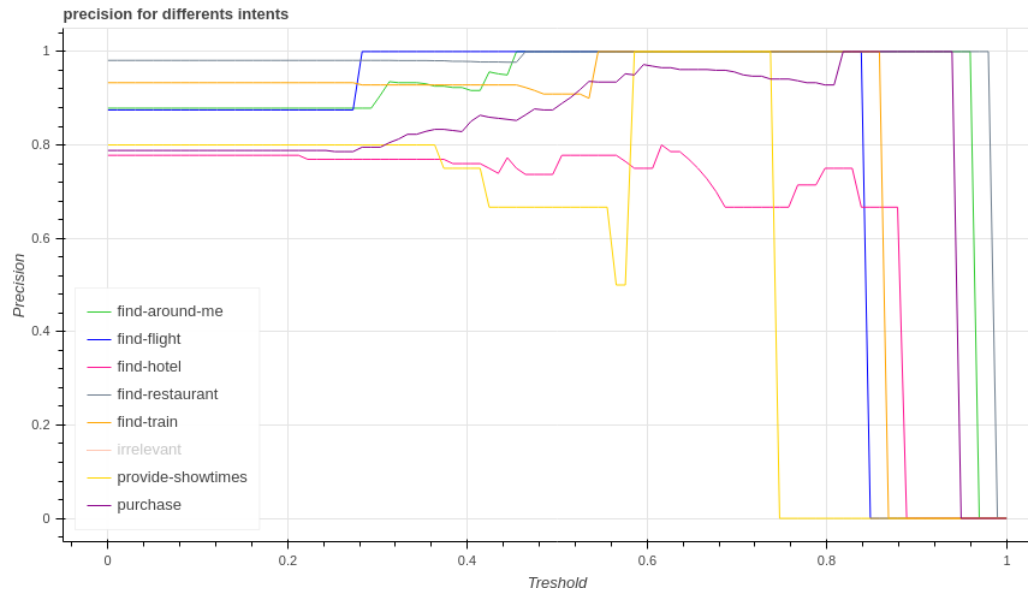


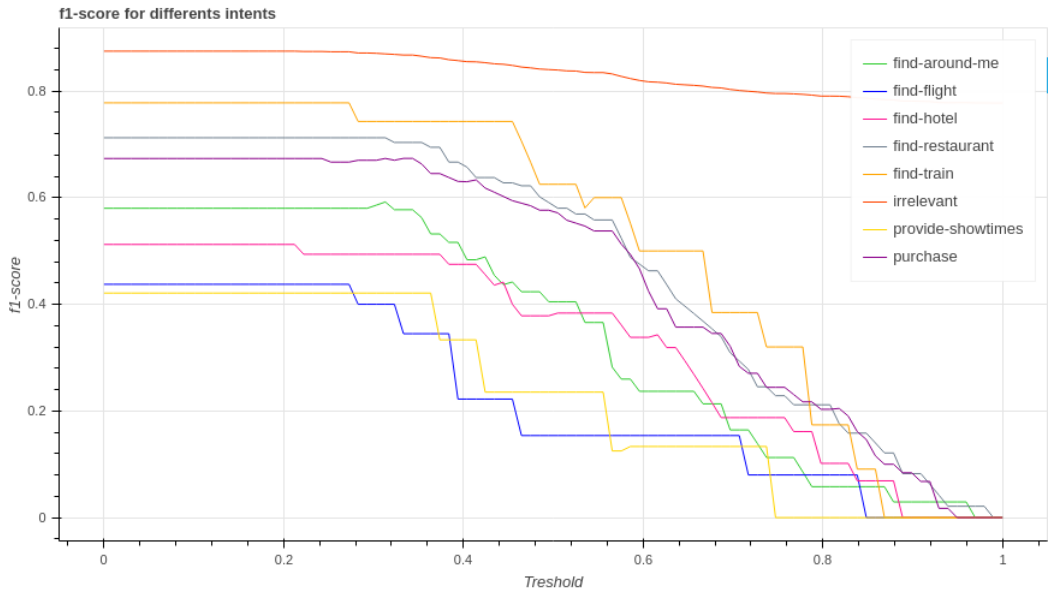
Les métriques sont constantes pour un seuil de validation inférieur à 0.27, après le *recall* diminue linéairement ce qui entraîne également la chute du F1-score. Cependant, la précision augmente jusqu'à un *treshold* de 0.71, puis chute drastiquement. Il est compliqué de trouver un seuil optimal avec ses informations, car le rappel et la précision ne sont pas tous les deux croissants, et il faut connaître jusqu'à quelle valeur de *recall* peut descendre le modèle dans le cas de son utilisation. Par exemple, pour un *treshold* de 0.2, le rappel est de 50% et la précision de 85%, donc le modèle trouve pour une intention la moitié des phrases correspondantes, mais peut se tromper dans la prédiction d'une phrase. Alors que pour un *treshold* de 0.7, le rappel est de 25% et la précision de 95%, le modèle est très précis, quand il prédit une phrase il se trompe peu, mais il ne trouve pas toutes les phrases correspondants à une *intent*, encore une fois car il a tendance à trop les classer en « *irrelevant* ».

2.2 Par intention

Pour obtenir plus d'informations et bien distinguer la fameuse classe « *irrelevant* » des autres, nous avons fait une étude par *intent*.





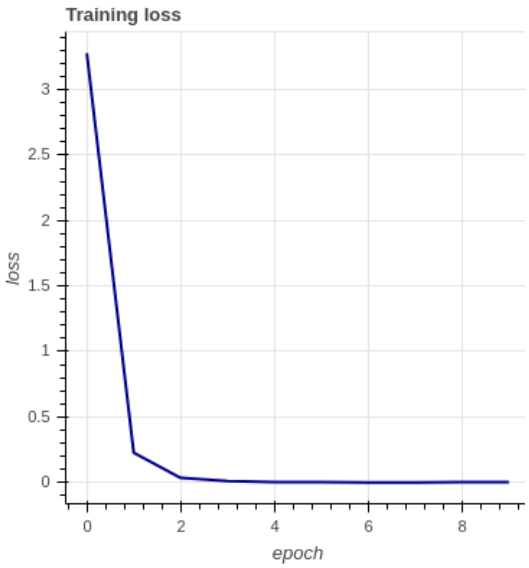


La classe « *provide showtime* » sur le graphe de précision est un peu étrange, mais c'est la classe la plus sous-représentée (14/1034), elle peut être mise de côté par rapport aux autres classes qui ont des courbes plus similaires.
En comparaison à s'il n'y a aucun *treshold*, la précision commence à être plus intéressante à partir d' un seuil de 0.4, et le rappel n'est pas tellement plus bas. A 0.40, la précision augmente de 0.02 sur la plupart des *intents*, et le *recall* diminue de 0.04, soit une perte de 0.03 au f1-score, mais au profit d'une meilleure précision.

3 Création et entraînement d'un modèle

3.1 Training

Le modèle est implémenté avec la librairie SpaCy, plus précisément il s'agit du plus petit modèle français de SpaCy « *fr_core_news_sm* ». Le jeu de données utilisé est le *training set* fourni, mais découpé en 80% d'entraînement, 10% de validation et 10% de test. Un premier training sur 10 époques a permis d'évaluer l'évolution du *loss*.



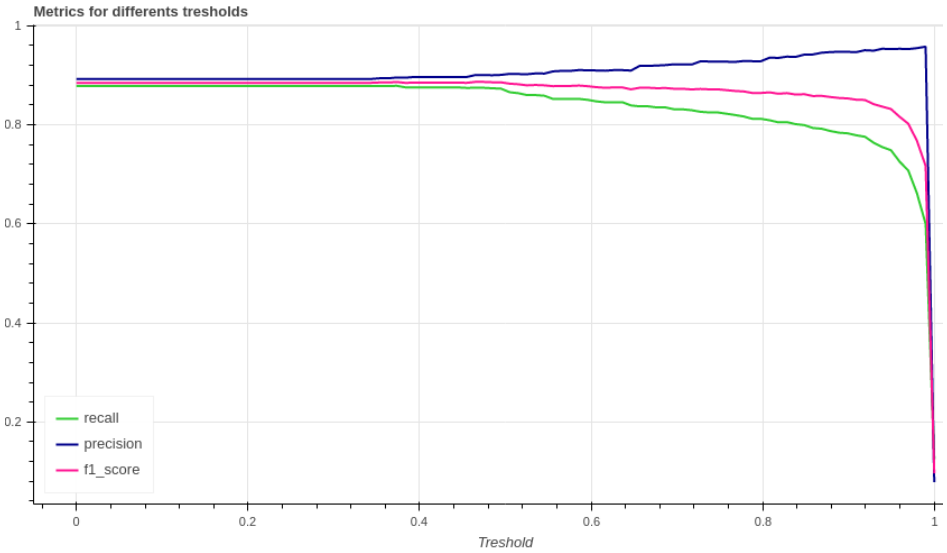
Epoch	Loss	Precision	Recall	F1-score
3	0.034	0.866	0.849	0.858
4	0.009	0.866	0.854	0.860
5	0.002	0.877	0.857	0.864
6	0.001	0.877	0.853	0.861

Les performances enregistrées sur le *dataset* de validation sont constantes à partir de 5 périodes, voire diminuent légèrement, et pour aussi éviter l'*overfitting*, le modèle final est entraîné sur 5 périodes.

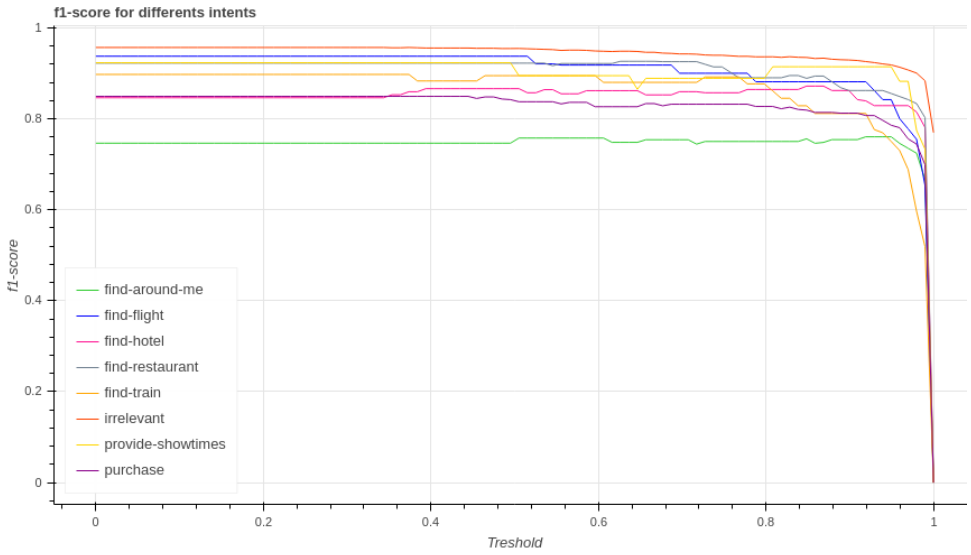
3.2 Évaluation

A partir du modèle entraîné, une image dockerfile a été construite afin de pouvoir tester le modèle dans le notebook ayant servi pour les parties précédentes.

	Accuracy	Precision	Recall	F1-score
Macro	82.0%	0.88	0.88	0.88



Sur les métriques en macro, les résultats sont plus réguliers et meilleurs et ceux obtenus avec le premier modèle. Cela se comprend en regardant la matrice de confusion, le soucis avec la classe « *irrelevant* » n'est plus aussi important, ce qui augmente le recall sur toutes les autres *intents*.



Les F1-score par intention sont donc plus élevés et similaires entre eux par rapport à ceux du modèle Idiwii, et le *treshold* a moins d'importance car les taux de confiance des prédictions sont plus démarqués. Ainsi, sur la tâche de classification d'intention le modèle obtenu avec SpaCy est bien plus performant, car la précision est la même mais le rappel est meilleur.

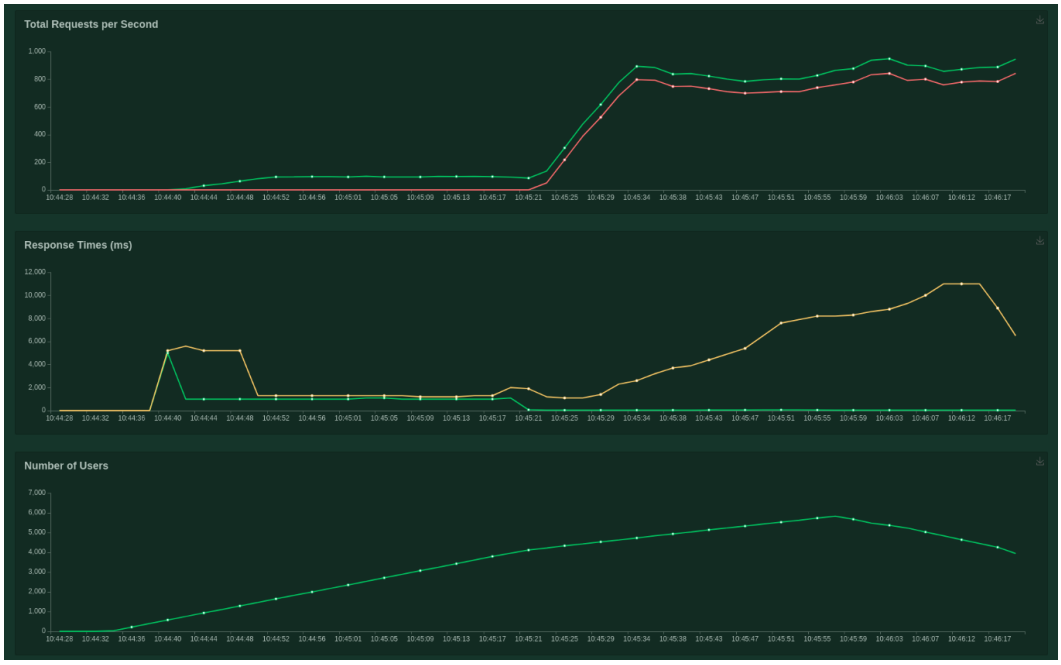
4 Exposition du nouveau modèle

Le nouveau modèle a ensuite été exposé à travers une API REST avec le même contrat d'échange que pour le modèle fourni :

GET */api/intent?sentence=<phrase-à-classifier>*

Afin de réaliser cette API, nous avons utilisé le *framework* Flask de part sa simplicité d'utilisation. Afin de documenter le service, nous avons utilisé Flasgger, une extension de Flask permettant d'utiliser Swagger et ainsi de créer une API RESTful respectant les spécifications OpenAI. Une fois l'image Docker contenant le nouveau modèle exécutée, le service exposera donc sur la route */apidocs/*, une documentation présentant ses caractéristiques et son utilisation. Pour le déploiement en production, nous avons décidé d'utiliser un serveur WSGI via *waitress*, une librairie python simple mais performante. Nous avons également implanté un *logger* renvoyant le temps d'inférence du service lorsque ce dernier est en production.

Afin de réaliser des tests de montée en charge, nous avons utilisé l'outil open source Locust :



Nous avons ainsi remarqué qu'à partir d'environ 4000 utilisateurs simultanés sur le service, ce dernier commence à saturer. Ces performances sont largement suffisantes dans le cadre de ce projet, cependant dans un contexte de production elles pourraient s'avérer problématiques. Afin de les améliorer, nous pourrions nous orienter vers un serveur WSGI plus rapide que *waitress* comme *gunicorn* par exemple. Nous pourrions également mettre en place une mise en cache afin de réduire le nombre d'appels sur le service et améliorer la latence des demandes.