DUSSARD
Claire

Bordeaux INP
ENSC

Rapport final de projet informatique
Mood Insight

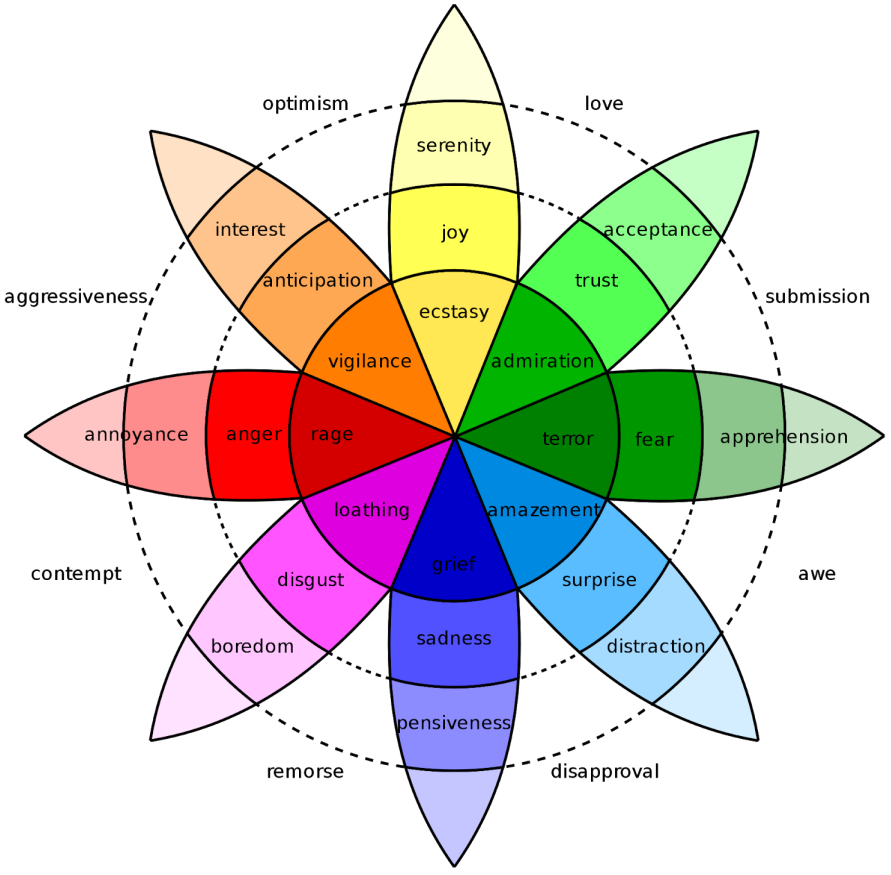
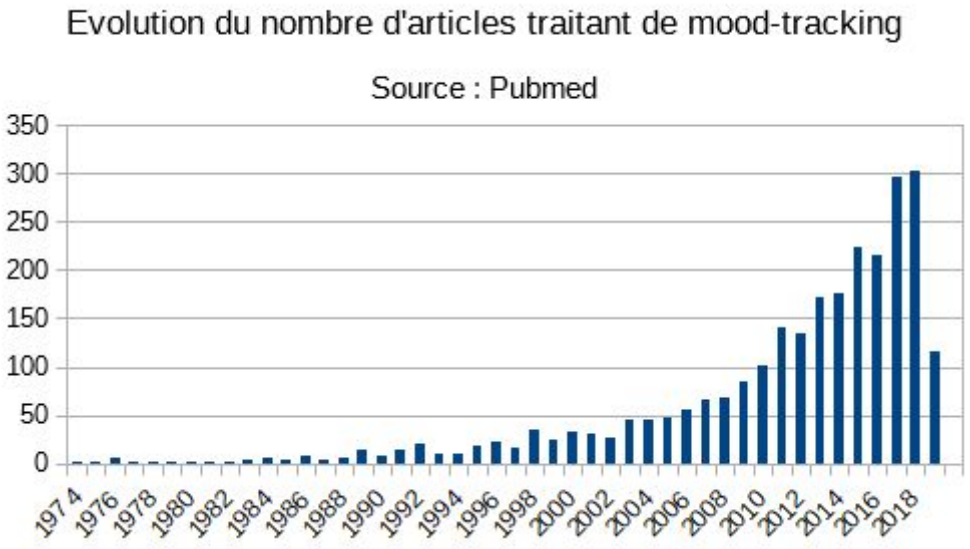


Table des matières

Le projet Mood Insight

Introduction

Le suivi de l'humeur est une méthode primordiale à la détection de plusieurs troubles de l'humeur. Par nature, notre humeur est dynamique, elle varie en permanence au cours du temps. Les méthodes actuelles pour rendre compte de l'évolution de l'humeur sont souvent biaisées et limitées dans leur capacité à collecter des données au cours du temps.
L'intérêt de la recherche pour le suivi de l'humeur a évolué ces dernières années.



Les avancées technologies permettent l'émergence de méthodes innovantes, efficaces et spontanées de suivre son humeur au quotidien.
L'analyse du sentiment est un domaine du traitement du langage.
Nous l'utiliserons ici afin de déterminer les émotions exprimées par un utilisateur à travers un récit de sa journée, à la façon d'un journal intime.
Cela permettrait de mettre en place une manière naturelle de récolter ces données pour l'utilisateur.

Sujet

Mon projet consiste à mettre en place un suivi de l'humeur de l'utilisateur à travers un récit de sa journée, à la façon d'un journal intime. **On se référera à ce texte entré par l'utilisateur sous le terme de souvenir** dans la suite de ce document.
L'émotion exprimée par l'utilisateur est déduite grâce à l'utilisation de traitement du langage.
L'utilisateur a un retour sur les émotions déduites et leur évolution.

Le but est donc de parvenir à mettre en place un **suivi des émotions** de l'utilisateur. L'émotion de l'utilisateur sera inférée à partir de ses phrases dans une **interface de type chatbot**, réalisée sur ordinateur pour faciliter la saisie et le traitement des données. Une fois l'humeur déterminée, celle-ci sera **stockée**, en lien avec la date et l'heure de sa saisie, et le souvenir

associé. Une **analyse** sera réalisée afin d'éclairer l'utilisateur sur ses émotions, leur fréquence voire des causes possibles.

Objectifs

L'objectif de ce projet est de développer un outil permettant de :

1. inférer l'émotion dominante dans un texte saisi (après nettoyage)
2. stocker les émotions déterminées, en lien avec le contexte de saisie
3. effectuer du machine learning pour inférer sur ces données en :
    a. recherchant des motifs temporels
    b. recherchant des corrélations
4. proposer des aides adaptées selon les résultats

Les fonctionnalités envisagées étaient les suivantes :

Repère	Désignation de la fonctionnalité
FP1	Saisir un souvenir
FP2	Accéder à un suivi de son humeur
FP3	Accéder à des corrélations selon l'heure, le jour
FP4	Accéder à des corrélations entre l'émotion et le contenu du texte (par *key phrase extraction*)
FP5	Accéder à des corrélations entre émotions si simultanées
FP6	Accéder aux souvenirs précédents
FP7	Accéder à des aides selon les émotions fréquentes / en cours

Organisation du projet

Phases du projet

Le projet s'est organisé en 3 grandes phases :

- traitement du langage
- réalisation de l'interface et intégration du pipeline de traitement
- machine learning et intégration

Implémentation du projet

Traitement du langage

J'ai commencé par me **familiariser avec les concepts principaux** du Traitement du Langage Naturel (TALN) en lien avec l'objectif de mon projet.

Pour cela, j'ai suivi deux tutoriels de Sentiment Analysis avec Python [1,2]. Cette démarche m'a permis de découvrir les étapes et les concepts majeurs derrière les API (Application Programming Interface) de TALN.
Les textes analysés sont en anglais et proviennent de revues de films issues du site IMDB. Ils sont analysés mot par mot (unigram) afin de rester dans l'esprit d'un tutoriel simple.
Les étapes sont les suivantes :

- nettoyage des données avec expressions régulières
- Vectorisation : convertir chaque revue en matrice (One hot encoding)
- construction d'un classifieur logistique
- entrainement du modèle (Régression Logistique)

Après cela, j'ai **étudié différentes API** de traitement du langage permettant de faire de l'analyse de sentiment. Après avoir exclu les API payantes et celles ne permettant pas d'analyser des textes en français, j'ai sélectionné Apico[3] et Tone Analyzer[4].
Malgré une prise en main plus simple, la technologie d'Apico ne donne accès qu'à la positivité du texte : un réel compris entre 0 et 1 (0 étant le plus négatif et 1 le plus positif).
En comparaison, Tone Analyzer donne accès non seulement à la positivité, mais aussi à l'émotion inférée du texte. De plus, cette API est plus fréquemment utilisée, par conséquent, un support plus important pour la prise en main et l'utilisation est disponible.
Après étude de la documentation de ces deux outils, j'ai donc **sélectionné la technologie d'IBM** : Tone Analyzer.

J'ai pris en main Tone Analyzer à partir de la documentation fournie et de plusieurs dépôts Github [5,6].
Les langages supportés par l'API sont les suivants :

Fig.1 : Langages supportés par Tone Analyzer

J'ai ensuite mis en place la **structure de stockage des données**, initialement un fichier .txt puis un fichier csv aux colonnes suivantes :

date	souvenir(texte)	émotion	score

Le fichier csv est plus adapté car il est supporté par la majorité des bibliothèques de traitement des données, et permet de structurer les données sous forme de tableau.

Le score est un réel compris entre 0 et 1 qui quantifie la précision de l'inférence de l'émotion par le système et donc sa confiance en la décision réalisée. Seules les émotions avec un score supérieur à 0.5 sont renvoyées par l'API[7]**.** La documentation de Tone Analyzer considère qu'une émotion est significative avec un score supérieur à 0.75.

Les émotions inférées par l'API Tone Analyzer sont sous la forme d'un objet au format JSON détaillé ci-dessous :


```
{
  "document_tone": {
    "tones": [
      {
        "score": 0.6165,
        "tone_id": "sadness",
        "tone_name": "Sadness"
      },
      {
        "score": 0.829888,
        "tone_id": "analytical",
        "tone_name": "Analytical"
      }
    ]
  },
  "sentences_tone": [
    {
      "sentence_id": 0,
      "text": "Team, I know that times are tough!",
      "tones": [
        {
          "score": 0.801827,
          "tone_id": "analytical",
          "tone_name": "Analytical"
        }
      ]
    },
```


Fig.2 : Structure de l'objet JSON

Lorsque le texte d'entrée n'a qu'une seule phrase, ou qu'aucun ton n'a été déterminé par l'API, l'objet comporte le ton du document : un objet composé du membre *tones*, tableau contenant des objets à 3 membres : *score*, *tone_id* et *tone_name* (équivalents).

Lorsque la donnée d'entrée a plusieurs phrases, à ces données générales sur le document s'ajoutent des données spécifiques par phrase : un objet comportant autant d'objets que de phrases, à 3 membres : numéro de la phrase, texte et l'objet *tones* à la structure décrite ci dessus.

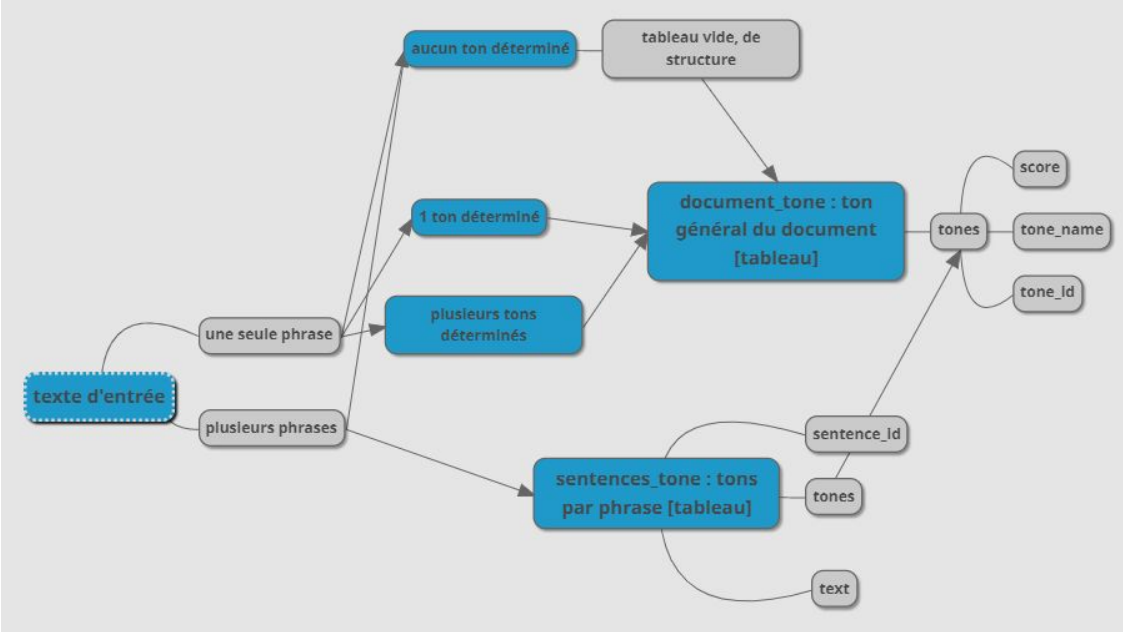


Fig.3 : Structure selon le texte d'entrée

Les émotions couvertes par l'API sont les suivantes :
tristesse, joie , peur, colère, confiance, analyse et indécision[8].
Les 5 premières correspondent à des émotions primaires comme définies par Plutchik [2].
Ce modèle étant initialement prévu pour analyser des avis clients, les deux derniers labels : analyse, indécision ne sont pas des tons émotionnels mais des tons de langage. En pratique, sur une soixantaine d'entrées de test, je n'ai jamais obtenu ces labels.

Interface utilisateur

Front-End de l'interface

Je souhaitais **implémenter une interface de type ChatBot** afin de recueillir les données.
J'ai commencé à la mettre en place sous React.js.
J'ai rapidement réalisé que l'envoi des informations à l'aide d'un composant TextInput devrait s'effectuer via un serveur NodeJS. J'ai essayé de mettre en place ce serveur plusieurs fois sans succès. La réalisation de l'application web se révélait par ailleurs assez chronophage pour un rendu assez moyen.
Ne souhaitant pas axer la majorité de mon projet sur l'apprentissage de ces technologies, j'ai décidé de **réaliser temporairement l'interface** à l'aide de la bibliothèque **Tkinter de Python**.

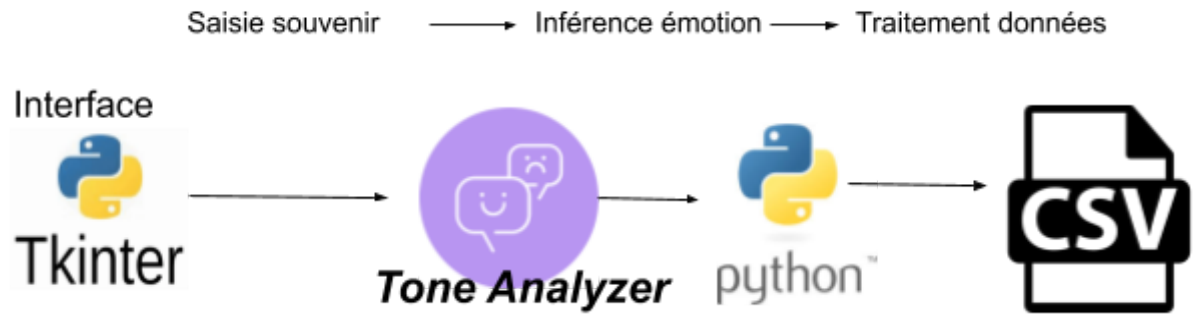


Fig.4 : Pipeline de traitement initial des données

Souhaitant réaliser une interface plus engageante pour l'utilisateur, j'avais prévu d'implémenter une interface de type ChatBot.
Après recherche des outils gratuits et intuitifs afin de créer ces assistants conversationnels, j'ai choisi d'utiliser la technologie **SnatchBot**.
L'avantage de cette technologie réside dans sa simplicité et le fait qu'elle permet de centraliser le script conversationnel pour déployer le chatbot sur différentes plateformes (interface web, email, Messenger ...).

La création du bot repose sur un ensemble d'interactions[9] liées entre elles par des évènements.
Ces interactions peuvent être de différents types : réponse textuelle, extraction d'informations particulières comme une adresse e-mail ou une date, traduction ou appel à une API.
J'utilise deux types d'interactions, la réponse textuelle et l'appel à une API.

Lors des interactions avec réponse textuelle, l'utilisateur peut entrer du texte en réponse à une question, ou sélectionner une "Quick Reply" : une carte lui permettant d'interagir avec le bot sans avoir à saisir de texte en sélectionnant une réponse pré-écrite.
Ces interactions sont liées par des connexions. Une connexion est un lien entre la réponse à une interaction n et une autre interaction n+1. Elle définit le passage à une autre interaction selon la vérification d'une condition sous une structure :
if (condition)
then { lancer interaction n+1}

La plupart du temps, la condition est la présence dans la réponse à l'interaction d'un mot ou un groupe de mots prévus. SnatchBot utilise des mécanismes de traitement du langage naturel et permet de repérer les synonymes afin que la condition soit remplie aussi en cas d'utilisation de termes ayant la même signification.
Il est également possible de définir un comportement "fallback" : si aucune condition n'est remplie, cette interaction est déclenchée.

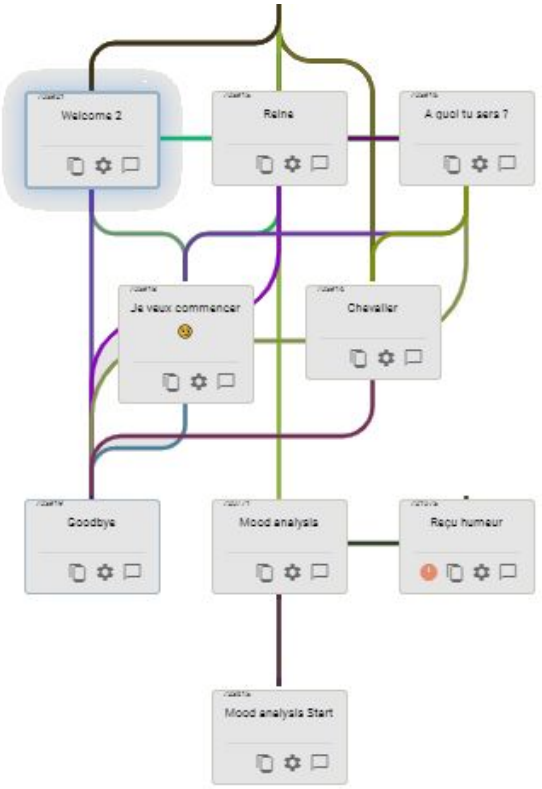


Fig.5 : Schéma des interactions du chatbot

Le chatbot réalisé est disponible par web et par mail à l'adresse moodtracker@bot.snatchbot.me.
Il y a plusieurs avantages à utiliser cette interface chatbot :

- l'engagement utilisateur : interface plus jolie, pouvant envoyer des rappels
- facilité de conception

Back-end de l'interface

Une fois l'interface réalisée, la partie qui s'est avérée plus complexe est celle des appels à l'API. En effet, il fallait trouver une manière de récupérer le texte entré par l'utilisateur pour le fournir au code Python qui appelle l'API Tone Analyzer, récupère le ton déterminé et le sauvegarde au format csv.

La première solution que j'ai envisagée est de récupérer le texte en téléchargeant les conversations. SnatchBot permet de télécharger les conversations sous forme d'un rapport en fichier excel.
L'inconvénient est que ce fichier contient toutes les conversations et il est très compliqué de distinguer les utilisateurs entre eux. En admettant pouvoir trier les utilisateurs et accéder aux souvenirs entrés, j'aurais sauvegardé les souvenirs au format csv date | souvenir | émotion | score utilisé précédemment. Puis, j'aurais lancé l'appel à Tone Analyzer sur toutes les entrées.
L'inconvénient majeur de cette solution est la nécessité de devoir aller télécharger à chaque fois l'ensemble des conversations sur le site de SnatchBot, puis les trier et les convertir pour le code Python.

Une solution plus satisfaisante aurait consisté à appeler directement l'API de traitement du langage une fois le souvenir entré par l'utilisateur. C'est dans cette direction que j'ai décidé d'approfondir. Cela revient donc à réaliser une *Third Party API Integration* [10] : l'intégration d'une API développée par un tiers (ici, au sens d'une entreprise différente de SnatchBot) dans un autre service, ici le chatbot.
Après étude de la documentation de SnatchBot[11] et recherche des différentes solutions disponibles, j'ai sélectionné Zapier[12] et Integromat[13]. Ce sont des outils qui proposent d'automatiser des solutions en interfaçant différents modules entre eux, construisant ainsi un scénario de traitement des données.
Le scénario est constitué d'un ensemble d'événements en série, la sortie de chaque module constituant généralement l'entrée du suivant.

L'avantage de Zapier est de présenter un plus grand nombre de modules que Integromat dont un module qui permet de déclencher du code Python : "Run Python". Cependant, ce module ne supporte pas toutes les bibliothèques Python et en l'occurrence le module watson_developer_cloud contenant ToneAnalyzerV3 n'est pas supporté. Il faudrait appel effectuer l'appel à l'API de Tone Analyzer par requête HTTP mais Zapier ne possède pas de module réalisant des requêtes HTTP.

Integromat n'a pas de module permettant de déclencher du code Python mais il possède un module de requête HTTP. J'ai choisi cette solution car le module de requête HTTP est plus versatile, et permet à la fois de réaliser l'appel à l'API mais également de lancer du code Python.

Une fois l'outil sélectionné, les tâches à réaliser étaient les suivantes :

- récupérer le message posté par l'utilisateur en réponse au chatbot
- envoyer le texte de ce message à l'API Tone Analyzer
- récupérer la sortie de Tone Analyzer et la fournir au code Python déterminant le ton maximum

- enregistrer la date, le souvenir, le ton déterminé et le score associé dans un tableau
- faire un retour à l'utilisateur sur le ton déterminé et le score

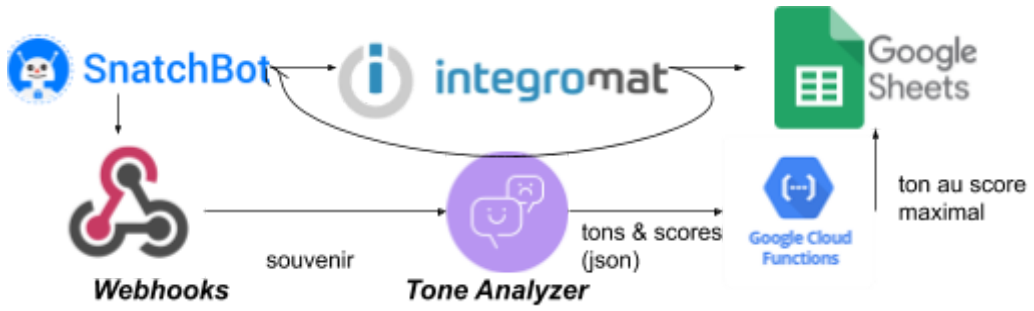


Fig.6 : Pipeline de traitement actuel des données

La communication entre le bot et les autres modules, que ce soit pour récupérer un message ou en envoyer un pour faire le retour à l'utilisateur sur l'émotion déterminée est gérée par WebHooks[14].

Le concept est le suivant : un WebHook est un rappel HTTP : une requête POST réalisée lorsqu'un événement se produit, notifiant ainsi de sa réalisation. Il peut porter des attributs, propres au contexte d'exécution.

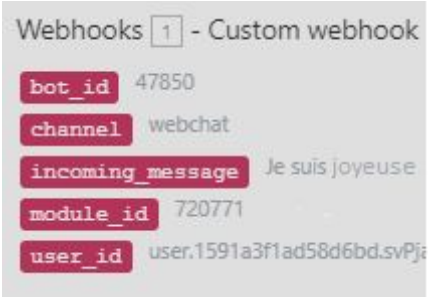


Fig.7 : Attributs du webHook envoyés par SnatchBot

Le texte (souvenir) entré par l'utilisateur est donc récupéré à l'aide de l'attribut "incoming_message" de WebHooks.
Afin de le fournir à l'API Tone Analyzer, j'ai utilisé le noeud final générique[15] (General Purpose Endpoint) via une requête GET. Cette requête est effectuée à l'aide du module HTTP Basic Auth d'Integromat.
La documentation d'IBM m'a permis d'identifier les différents paramètres nécessaires à l'envoi de cette requête. La méthode GET /v3/tone accepte un contenu en entrée via son paramètre de requête obligatoire text.
Tout d'abord, il est nécessaire de disposer de la clé d'API afin de s'authentifier, l'URL à appeler est la suivante : https://gateway-lon.watsonplatform.net/tone-analyzer/api/v3/tone.
Quant aux paramètres de la requête, la version à utiliser et le texte doit être fourni dans la requête tandis que la langue de la requête est insérée dans l'en-tête.

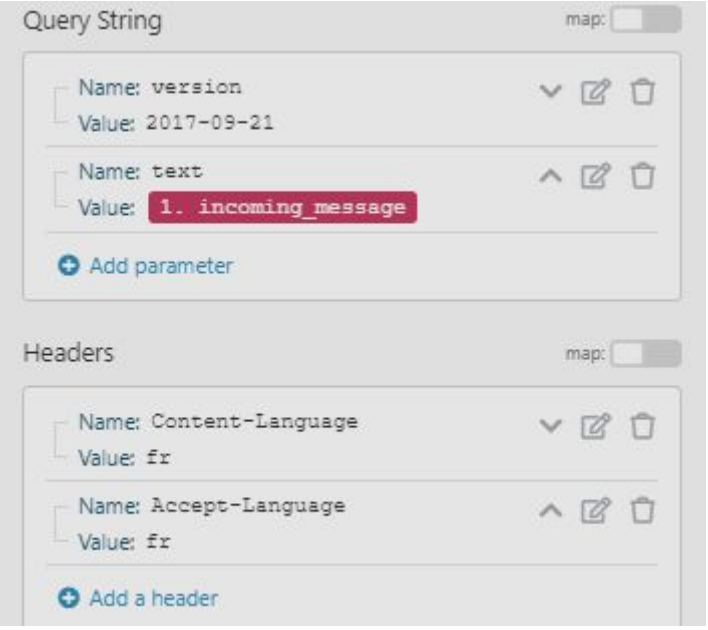


Fig.8 : Paramètres de la requête

Le résultat de cette requête est un objet JSON, qu'il faut traiter à l'aide du code déjà implémenté sous Python.

Integromat ne disposant pas d'un module permettant d'exécuter du code Python, il a fallu trouver un moyen d'exécuter du code Python en ligne en lui fournissant l'objet JSON renvoyé par Tone Analyzer.
Ce tutoriel[16] m'a beaucoup aidée. Il explique comment effectuer cette tâche en utilisant Google Cloud Functions. C'est un service proposant de la programmation événementielle sans serveur grâce à la plateforme Google Cloud.
Le code est déclenché par une autre requête HTTP effectuée à l'aide du module HTTP Request d'Integromat.
J'ai transformé le code afin de pouvoir l'exécuter avec cet outil. Très peu de bibliothèques Python sont supportées par Google Cloud Functions[17] car l'exécution de code Python était en Bêta-Test pendant ce projet. J'ai restructuré le code en conséquence afin qu'il prenne en entrée un objet request, contenant le json renvoyé par l'API Tone Analyzer et renvoie un objet json contenant :

- date
- souvenir (texte)
- émotion au score maximal
- score maximal associé

L'objet JSON renvoyé par le code Python exécuté en ligne contient toutes les informations, il ne reste plus qu'à effectuer le retour à l'utilisateur et le sauvegarder dans un tableau.
Le code que j'avais réalisé sous Python permettait d'écrire dans un fichier csv stocké en local. Or, migrer le code en ligne ne permet plus d'écrire dans des fichiers locaux.
La solution la plus simple a été d'ajouter un module Integromat Google Sheets afin d'écrire directement dans un tableur en ligne le résultat de la requête.

En parallèle, un retour à l'utilisateur est donné grâce à la transmission de l'objet JSON contenant l'humeur déterminée et le score à un WebHook qui permet d'afficher le résultat dans la discussion du chatbot. La fréquence de l'humeur est également renvoyée à l'utilisateur sous forme d'un pourcentage.

Pour l'obtenir, j'utilise le module Search Rows d'Integromat, et je renvoie $\frac{nb\ lignes\ où\ colonne\ mood = émotion\ en\ cours}{nb\ total\ lignes}$.

Le nombre de lignes correspondant à l'émotion en cours est déterminé à l'aide d'une requête select réalisée en *Google Charts Query Language*.
En précisant les identifiants uniques (bot_id, user_id et module_id), on peut envoyer le message à la bonne discussion.

Le scénario final de traitement des données est le suivant ;

Fig.9 : Scénario global Integromat

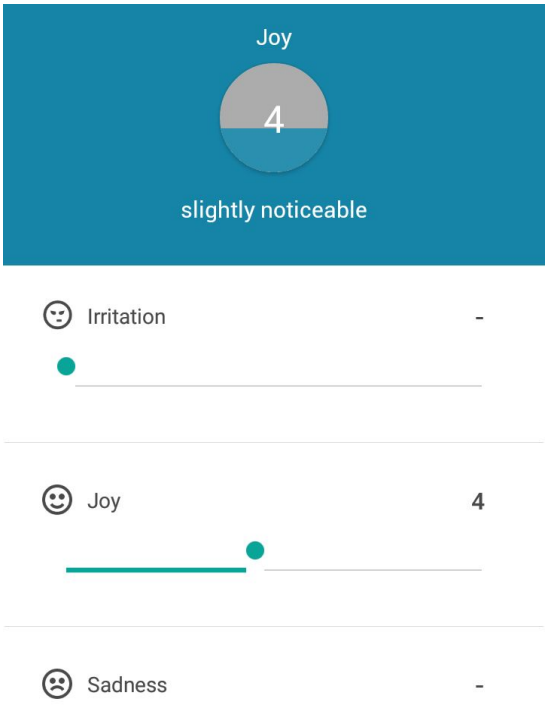
Machine learning

Avec mémoire des données : réseau de neurones

L'un des objectifs de mon projet était de donner à l'utilisateur des éléments d'analyse sur l'évolution de son humeur.

J'ai commencé par collecter des données et les mettre sous une forme qui me permettrait de m'en servir afin de réaliser un apprentissage. Pour cela, la taille de l'échantillon était importante, il fallait suffisamment de données pour produire un modèle d'apprentissage efficace et qu'il reste suffisamment de données inconnues au modèle pour évaluer son efficacité.

Or, je n'avais qu'une cinquantaine d'entrées issues de mon modèle fonctionnant sous Tone Analyzer au final. De plus, se posait le problème du formatage des données.
En effet, le score associé par l'API est un réel compris entre 0 et 1. Selon la documentation, le score est considéré comme significatif à partir de 0.75. Cependant, ce score ne reflète pas l'intensité mais la confiance du système en sa classification de l'émotion. L'évolution de l'émotion serait alors modélisée seulement en termes d'absence et de présence (0 ou 1). Pour cela, l'apprentissage résultant de ces données aurait été assez difficile. J'ai donc choisi d'utiliser des données récoltées sous MindCare, une application de suivi de l'humeur que j'avais commencé à utiliser dans le cadre de mon projet transpromotion.
Cette application permet de suivre l'évolution et l'intensité de plusieurs émotions à la fois.



:

Fig.10 : Capture d'écran de l'interface de l'application MindCare

Les données recueillies à l'aide de cette application sont :

- plus complètes : minimum 2 fois par jour, une dizaine d'émotions suivies et l'intensité de chacune notée entre 0 et 10 à chaque entrée
- nombreuses : 150 entrées.

Les émotions suivies sont les suivantes :
Joy (joie), *Anger* (colère), *Sadness* (tristesse), *Self-hatred* (haine de soi), *Irritation* (irritation), *Hopelessness* (désespoir), *Boredom* (ennui), *Worry* (inquiétude), *Stress* (stress), *Fatigue* (fatigue), *Emptiness* (sensation de vide).

Ces données permettent de constater qu'il semble effectivement y avoir des liens d'un jour à l'autre entre les différentes émotions suivies : ainsi les pics de fatigue semblent précéder les pics de stress par exemple.

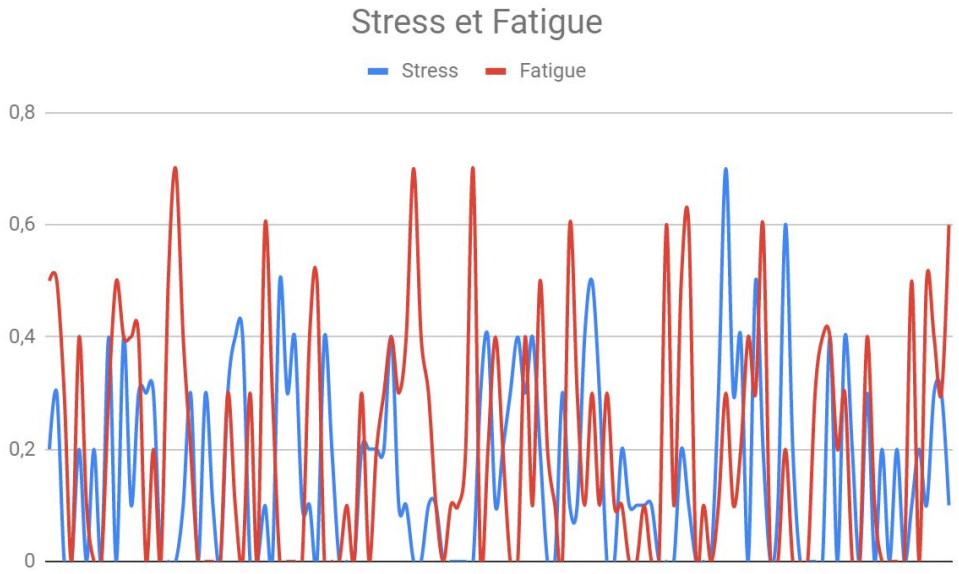


Fig.11 : Evolution de la fatigue et du stress

Ces données sont personnalisées et, dans le cas du développement d'une interface destinée à des utilisateurs multiples, le parti pris serait de proposer la construction d'un modèle spécifique à chaque individu. En effet, ces données sont hautement individu-spécifique et développer un modèle moyen à partir des données de l'ensemble des utilisateurs impacterait fortement la qualité de la prédiction.
J'ai reporté ces données depuis l'application vers un tableur et ai pris soin de relever l'heure, la date et le jour de la semaine. En effet, plusieurs articles ont indiqué que le jour de la semaine pouvait influer sur les émotions.
Selon une étude [3], Mardi est le jour le plus déprimant, et Samedi le plus joyeux.
Il est donc important de connaître le jour de l'entrée car il influence l'humeur.

Les données sont dans un tableau de la forme :
Jour | Date | Heure | Intensité Émotion 1 | … | Intensité Émotion 11

Une fois ces données récoltées, j'ai fait des recherches sur le développement de modèles prédictifs, et plus particulièrement de l'humeur.
Deux articles m'ont particulièrement éclairée : [4] et [5]. L'approche privilégiée par ces deux articles est l'utilisation d'un réseau de neurones de Deep Learning : l'utilisation de Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN) ou réseau de neurones récurrents à mémoire court-terme et long terme.
L'intérêt de l'utilisation de tels réseaux ici réside dans leur mémoire des entrées précédentes. L'idée du suivi de l'humeur de l'utilisateur repose en effet sur l'idée que des motifs temporels agissent sur l'évolution de ces émotions et leur intensité. Pouvoir prédire ces données avec un modèle d'apprentissage suppose donc de disposer d'une mémoire à court voire long terme.
Les réseaux de neurones récurrents sont particulièrement adaptés pour l'analyse de séries temporelles et à ce cas de par le fort impact supposé du jour et de l'heure sur l'humeur).
Cependant, en pratique, les réseaux de neurones récurrents ont peu de mémoire à long terme [6]. Le problème provient de la nature même de ces réseaux qui basent leur apprentissage sur la rétropropagation du gradient. Pour un événement lointain, la correction du modèle diminue exponentiellement avec l'écart temporel entre cet événement et le présent.
Pour pallier ce problème, les LSTM-RNN ont été développés à la fin des années 90.

Pour les présenter rapidement, les réseaux de neurones récurrents ont pour particularité d'avoir leur valeur d'activation à l'instant t h_t générée à l'aide de la valeur précédente h_{t-1} et de l'input à l'instant t x_t. C'est ce que la figure ci-dessous met en évidence.

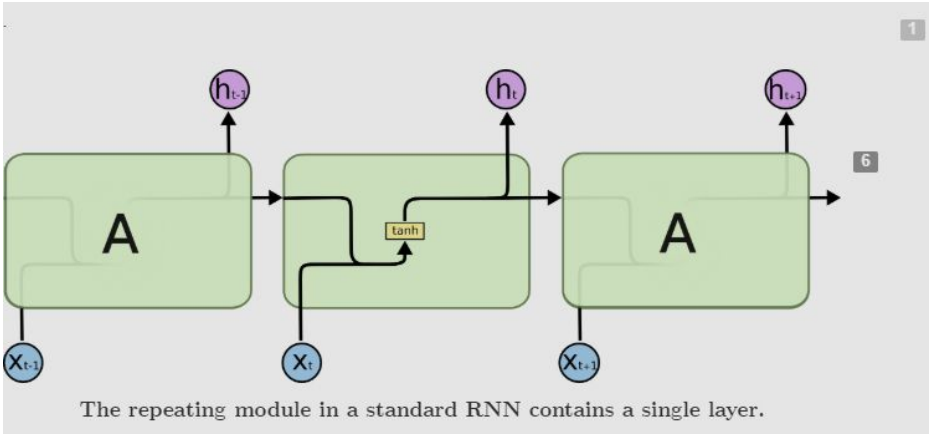


Fig.12 : Motifs standard d'un RNN : une seule couche [7]

Les LSTM-RNN ne sont pas constitués de neurones mais de blocs de mémoire (*memory blocks*). Chaque bloc dispose de portes qui gèrent son état et sa sortie.
Il y a trois portes :

- Porte d'oubli (*Forget Gate notée Ft*) : gère l'information qui doit être oubliée par le bloc
- Porte de sortie (*Output Gate notée Ot*) : définit la sortie du bloc en se basant sur la mémoire et l'entrée
- Porte d'entrée (*Input Gate notée It*) : définit les valeurs de l'entrée utilisées pour mettre à jour sa mémoire

Le o_t sur la figure ci-dessous représente la mémoire du bloc à l'instant t.

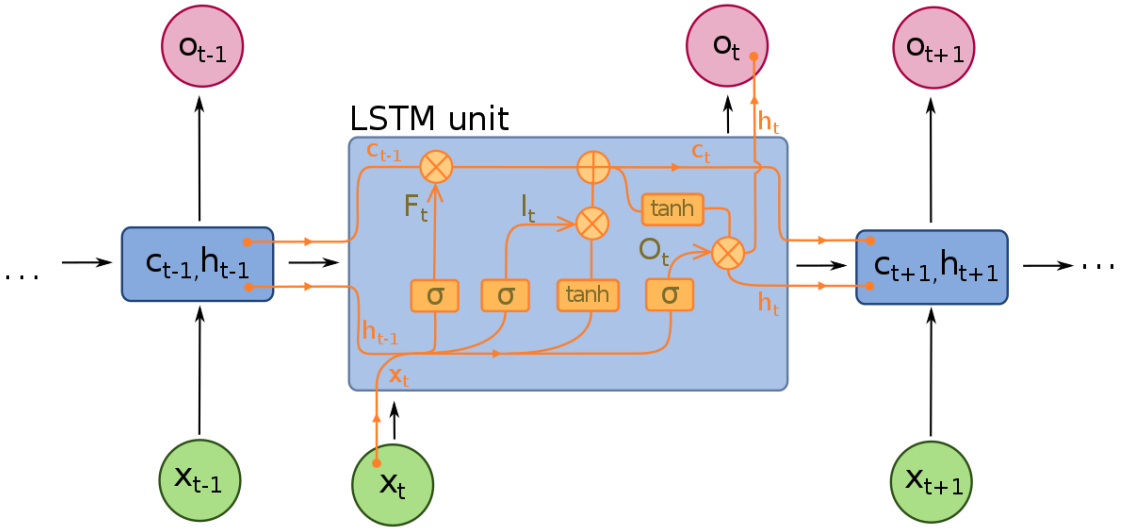


Fig.13 : Motif standard d'un LSTM RNN standard : 4 couches en interaction

J'ai commencé par implémenter une forme simple de LSTM-RNN à l'aide ce tutoriel [18]. J'ai choisi d'utiliser Keras pour l'implémentation. C'est une bibliothèque Python qui s'utilise en surcouche à Tensor Flow et qui est plus adaptée pour la création de modèles de Deep Learning. La prise en main est simplifiée et c'est donc tout à fait approprié pour l'implémentation du modèle de Machine Learning que je cherche à réaliser.
Le code est réalisé sur un notebook Google Colaboratory, Cela permet de bénéficier d'une puissance de calcul bien supérieure et de gagner du temps sur l'installation et la configuration de Keras qui est assez longue.

Essentiellement, les tâches effectuées sont les suivantes :

- Import des bibliothèques
- Chargement des données
- Mise en forme des données :
  - mise à l'échelle à l'aide d'un scaler (les données comprises entre 0 et 1 évitent que le réseau de neurones ne diverge)
  - séparation des données en données pour l'entraînement et données pour le test du modèle (⅔ - ⅓)
  - création d'une matrice de longueur de la taille de la mémoire choisie
    ex : avec une mémoire de 2

x_t	x_{t-1}	x_{t-2}
x_{t-1}	x_{t-2}	x_{t-3}

- Création du réseau de neurones : un neurone pour l'entrée, 4 blocs LSTM pour l'apprentissage et un neurone de sortie
- Entraînement du réseau à l'aide des données d'entraînement (*model.compile* et *model.fit*)
- Test du réseau entraîné en prédiction des données inconnues (*model.predict*)
- Affichage des résultats sur un graphique

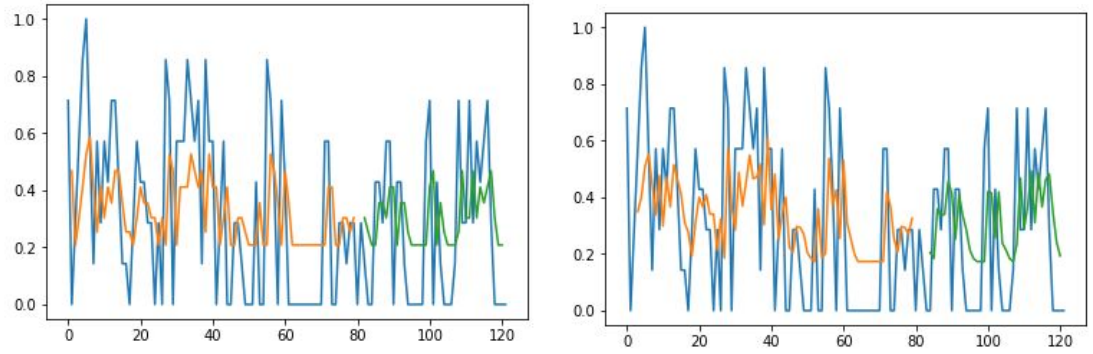
L'implémentation que j'ai réalisé est assez simple, la donnée d'entrée étant une colonne seule constituée des intensités dans l'ordre chronologique des entrées. La date en soi n'est donc pas ici une donnée d'entrée même si elle intervient dans l'ordre des données (considérées comme des séries temporelles par le réseau).
On peut faire l'hypothèse que ces données étant reportées régulièrement, les données prédites par le réseau seraient valides dans le cas d'un report similaire des données.
Le résumé du modèle est le suivant :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 4)                 320
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 5
=================================================================
Total params: 325
Trainable params: 325
Non-trainable params: 0
_________________________________________________________________
None
```

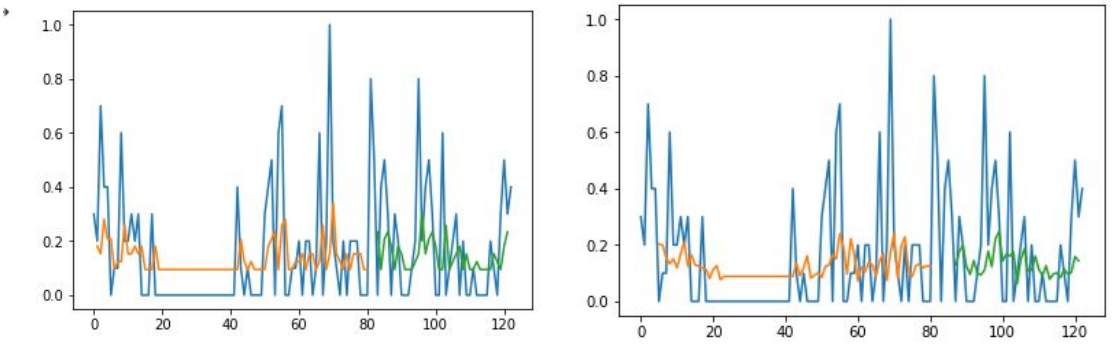
Fig.14 : Modèle du réseau de neurones entraîné

Le modèle de base possède un neurone d'entrée, une couche de 4 blocs mémoriels LSTM et un neurone de sortie.

- Joie, 100 epochs, batch_size = 2

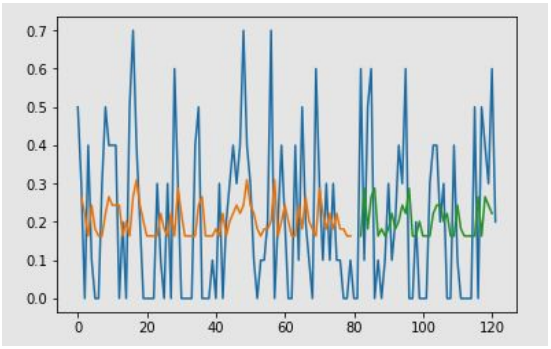


- Tristesse, 100 epochs batch_size = 2



- Fatigue 100 epochs batch_size = 2

```
Train Score: 0.20 RMSE
Test Score: 0.21 RMSE
```




```
Train Score: 0.27 RMSE
Test Score: 0.28 RMSE
```

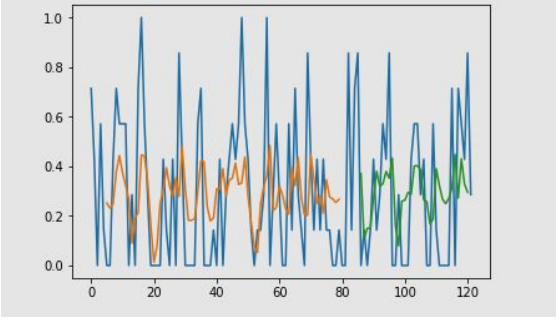


Fig.15 : Modèles obtenus (mémoire 1 à gauche contre 3 à droite), en orange les données d'entraînement et en vert les données de test

Sans mémoire des données : régression linéaire multiple

Une approche complémentaire à la méthode avec mémoire des données consiste à déterminer à un instant t quelles émotions sont corrélées.
On peut pour cela utiliser un modèle de régression linéaire multiple.
J'ai réalisé cette modélisation à l'aide du langage R.
En tant que variables, j'ai utilisé toutes les émotions provenant des données que j'utilisais pour le réseau de neurones. J'ai également utilisé l'heure de l'entrée, le jour du mois ainsi que le jour de la semaine. Pour représenter le jour de la semaine j'ai utilisé des *dummy variables*, *7* variables valant 1 lorsque le jour de la semaine correspond et 0 sinon.

J'ai mis toutes les variables à la même échelle (entre 0 et 1) afin de leur donner toute la même importance dans le modèle.
J'ai utilisé la méthode step de sélection des variables, qui utilise un critère statistique (le critère d'Akaike) pour réduire le nombre de variables du modèle à celles significatives.

Ainsi, par exemple, pour expliquer la variable *sadness* , le modèle obtenu est le suivant :
(self-Ha représente la self-Hatred (haine de soi) , empty l'emptiness (sensation de vide))

```
Call:
lm(formula = sadness ~ joy + irrit + boredom + selfHa + empty +
    hour + thursday + friday)

Residuals:
     Min       1Q   Median       3Q      Max
-0.34578 -0.07673 -0.00250  0.06550  0.52223

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.05167    0.06448  -0.801 0.424574
joy         -0.18445    0.08633  -2.137 0.034772 *
irrit       -0.14536    0.07926  -1.834 0.069275 .
boredom     -0.22303    0.07723  -2.888 0.004643 **
selfHa       0.27677    0.07683   3.602 0.000469 ***
empty        0.45730    0.06569   6.962 2.27e-10 ***
hour         0.18318    0.07191   2.547 0.012190 *
thursday     0.06163    0.04194   1.469 0.144456
friday       0.11023    0.03243   3.399 0.000933 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1344 on 114 degrees of freedom
Multiple R-squared:  0.6379,    Adjusted R-squared:  0.6125
F-statistic:  25.1 on 8 and 114 DF,  p-value: < 2.2e-16
```

Fig.16 : Modélisation finale de la tristesse

On constate que le modèle parvient à expliquer plus de 61% de la variance, ce qui est élevé. Utiliser cette approche en complément du réseau de neurones pourrait donc être particulièrement utile.

A l'inverse, cette modélisation permet également de déterminer quelles émotions sont caractéristiques selon le jour de la semaine, du mois, ou l'heure pour l'utilisateur.
Les données obtenues vont dans le sens que plusieurs articles sur l'humeur cités précédemment : il y a bien un effet significatif du jour sur l'humeur.

```
Call:
lm(formula = friday ~ joy + worry + sadness + anger + empty)

Residuals:
     Min       1Q   Median       3Q      Max
-0.88433 -0.22886 -0.10489  0.05267  0.88493

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.04947    0.11373   0.435   0.6644
joy          0.45162    0.21191   2.131   0.0352 *
worry       -0.38936    0.20008  -1.946   0.0541 .
sadness      0.50964    0.22619   2.253   0.0261 *
anger        0.46478    0.23290   1.996   0.0483 *
empty        0.34939    0.19768   1.767   0.0798 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3774 on 117 degrees of freedom
Multiple R-squared:  0.2093,    Adjusted R-squared:  0.1755
F-statistic: 6.193 on 5 and 117 DF,  p-value: 3.952e-05
```

Fig.17 : Modélisation du vendredi

Gestion de projet

Planning envisagé

En début de projet, lors de la réalisation du cahier des charges, mon planning prévisionnel était le suivant :

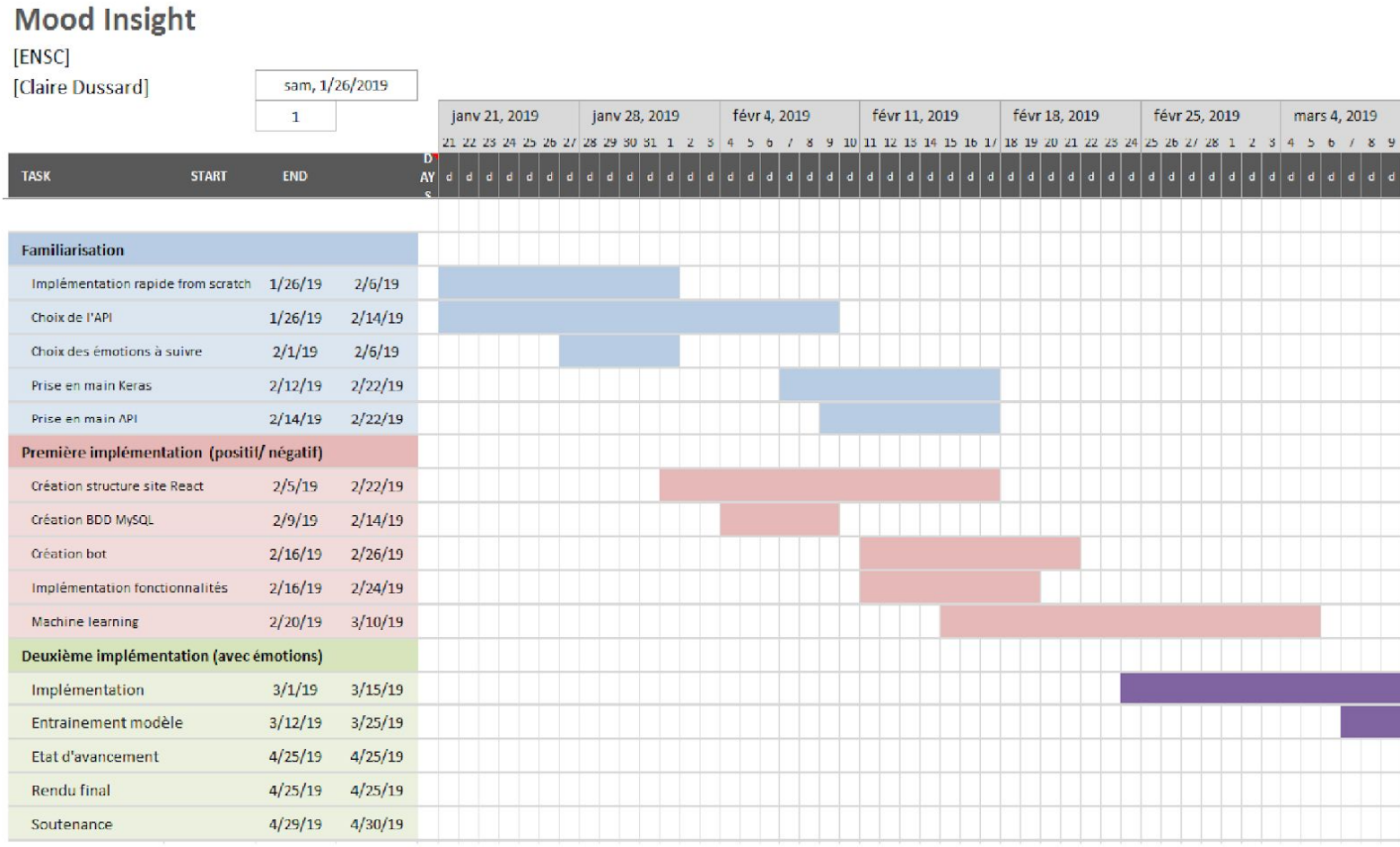


Fig.18 : Planning prévisionnel

Planning effectif

J'avais prévu 3 phases. Tout d'abord, une phase de familiarisation avec le traitement du langage et le sujet. Lors de cette phase je devais également choisir l'API de TALN que j'allais utiliser, la prendre en main et prendre en main Keras.
En pratique, je n'ai pris en main Keras qu'à partir de la 3ème phase mais ai réalisé toutes les autres tâches. J'ai consacré le temps libéré à prendre en main l'API Tone Analyzer ce qui m'a permis d'implémenter plus rapidement le script Python par la suite.

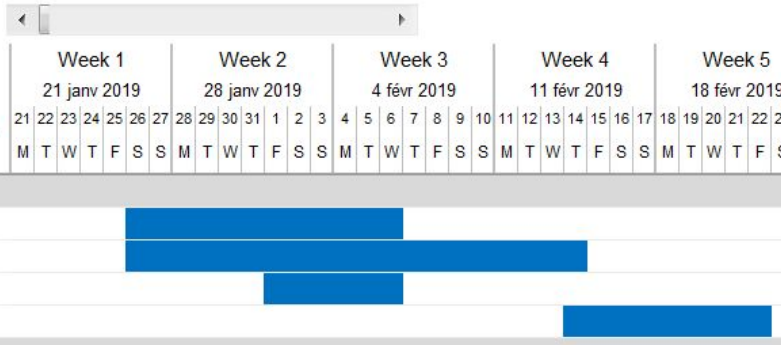
[MoodInsight] Project Schedule

Gantt Chart Template © 2006-2018 by Vertex42.com.

[ENSC]

Project Start Date 1/26/2019 (samedi) Display Week 1
Project Lead Claire Dussard

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	**Familiarisation**			-			-
1.1	Implémentation rapide from scratch		sam 1/26/19	mer 2/06/19	5	0%	8
1.2	Choix de l'API		sam 1/26/19	jeu 2/14/19	5	0%	14
1.3	Choix des émotions à suivre		ven 2/01/19	mer 2/06/19	4	0%	4
1.4	Prise en main API		jeu 2/14/19	ven 2/22/19	4	0%	7



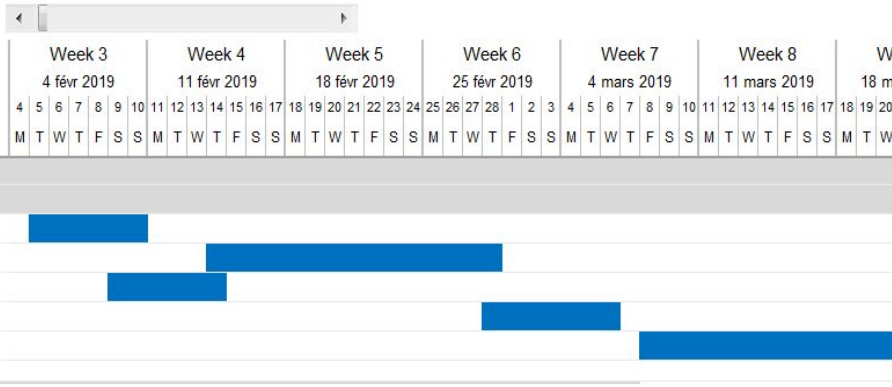
J'avais initialement prévu d'effectuer deux phases d'implémentation, avec positivité/ négativité puis avec des émotions. Cependant, l'API d'IBM permet directement d'avoir accès aux émotions, j'ai donc re-structuré le planning en conséquence.

Lors de la deuxième phase, après avoir changé de technologie pour l'interface, j'ai adapté la durée des autres phases du projet. La création du bot a pris plus de temps que prévu car il a fallu l'interfacer avec les autres parties du projet. Le temps gagné avec la vitesse de réalisation de l'interface permise par SnatchBot a été consacré à l'interfaçage avec Integromat.

[ENSC]

Project Start Date 1/26/2019 (samedi) Display Week 3
Project Lead Claire Dussard

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	**Familiarisation**			-			-
2	**Première phase**			-			-
2.1	Création interface basique		mar 2/05/19	dim 2/10/19	4	0%	4
2.2	FP1 : saisie souvenir		jeu 2/14/19	jeu 2/28/19	3	0%	11
2.3	Création structure stockage données		sam 2/09/19	jeu 2/14/19	3	0%	4
2.4	FP2 : suivi humeur		jeu 2/28/19	mer 3/06/19	6	0%	5
2.5	Création bot SnatchBot		mar 3/05/19	ven 3/08/19	3	0%	4
2.6	interfaçage avec Integromat		ven 3/08/19	mer 3/20/19	3	0%	9

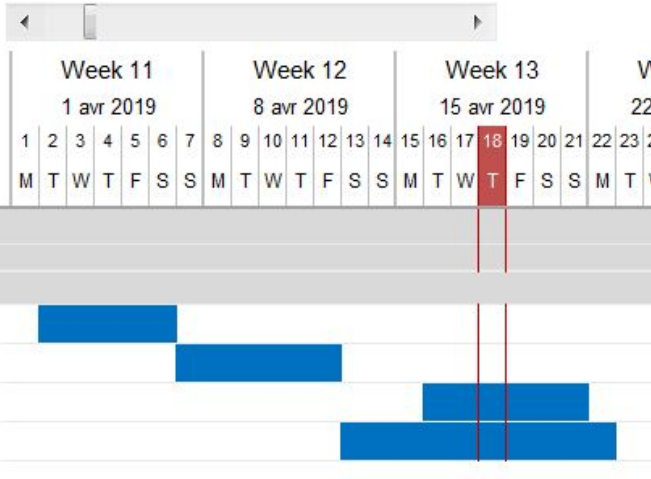


Après ces restructurations, la troisième phase du projet a été très différente de la phase prévue. Elle a été effectué sur un temps bien plus court. Cela a notamment été facilité par les séances de TD d'I.A à l'école, qui ont posé les bases et m'ont permis de construire un modèle plus rapidement par la suite.

[ENSC]

Project Start Date 1/26/2019 (samedi) Display Week 11
Project Lead Claire Dussard

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	**Familiarisation**			-			-
2	**Première phase**			-			-
3	**Deuxième phase**			-			-
3.1	Recherches biblio		ven 3/29/19	ven 4/05/19	4	0%	6
3.2	Récolte données		mar 4/02/19	sam 4/06/19	3	0%	4
3.3	Implémentation modèle univarié		dim 4/07/19	ven 4/12/19	3	0%	5
3.4	Retour utilisateur sur modèle		mar 4/16/19	dim 4/21/19	6	0%	4
3.5	Rédaction livrables		sam 4/13/19	lun 4/22/19	3	0%	6



Travail réalisé

Choix des outils & technologies

Ce projet m'a permis de découvrir et me familiariser avec de nombreux outils et technologies.

Outil	Description rapide	Signet vers justification
Integromat	outils proposant d'automatiser des solutions en interfaçant différents modules entre eux	lien
Google Cloud Functions	service permettant de réaliser de la programmation événementielle sans serveur	lien
Google Sheets	Outil de tableur en ligne permettant le stockage des données	lien
SnatchBot	Service de création de chatbots automatisant leur diffusion sur plusieurs plateformes (mail, web…)	lien
Webhooks	Rappel HTTP déclenché par un évènement , notifiant ainsi de sa réalisation	lien
Tone Analyzer IBM	API de traitement du langage donnant accès à l'émotion et au score de certitude associé à l'analyse d'un texte	lien
Google Colab	Service de programmation en ligne dans un environnement cloud donnant accès à des processeurs graphiques (GPU)	lien
Keras	bibliothèque Python qui s'utilise en surcouche à Tensor Flow et qui est plus adaptée pour la création de modèles de Deep Learning.	lien

Structure du code et fonctionnalités réalisées

J'ai choisi de réaliser le code en Python. C'est un langage que j'avais brièvement appris lors de mes années précédant l'ENSC.
Il est supporté par l'API et cela me permettra d'utiliser les nombreuses bibliothèques de manipulation des données (Pandas), et de traitement du langage (NLTK) disponibles.
L'avantage réside aussi dans la quantité de contenu disponible sur les principaux forums de programmation.

Afin de gérer les deux types d'objets JSON possibles, j'ai mis en place deux fonctions : *tone_handlerSentence* et *tone_handlerDoc*.
L'API renvoie souvent plusieurs émotions différentes. Afin de simplifier l'analyse et le stockage, j'ai décidé de sauvegarder :

- le texte entré par l'utilisateur
- la date
- l'émotion déterminée avec le score, et donc le degré de certitude le plus élevé
- le score associé

Pour cela, dans le cas d'une donnée d'entrée à plusieurs phrases, j'appelle successivement les deux fonctions, afin de déterminer l'émotion à sauvegarder.
L'utilisation de variables globales permet de gérer le score à écrire et l'émotion à écrire de façon unifiée à l'échelle du script.
J'ai pris soin de réinitialiser ces variables au début de chaque nouveau souvenir (texte entré) à analyser afin d'éviter la persistance de données entre deux analyses.

Après avoir réalisé le chatbot, j'ai eu besoin de pouvoir exécuter ce code en ligne. J'ai donc revu sa structure afin qu'elle fonctionne avec Google Cloud Functions.
J'ai ajouté un paramètre déterminant l'endroit du journal (ensemble des entrées) où commencer l'analyse. Ce paramètre est déterminé par défaut comme le premier indice où l'analyse n'a pas été réalisée la dernière fois. Cela permet d'accélérer le code et d'économiser des appels à l'API.
J'ai utilisé la méthode *jsonify* de Flask afin de renvoyer un objet json contenant l'entrée, la date, l'émotion inférée et le score associé.

Pour le réseau de neurones, j'ai principalement réalisé le code en m'inspirant d'exemples.
Il est structuré en blocs de codes voués à être exécutés en série.

Le code est commenté afin d'être compréhensible plus facilement. J'ai utilisé des bibliothèques classiques de Python (Pandas, Matplotlib) afin d'optimiser la vitesse des traitements effectués sur les données.

Résultats (démonstration)

L'interface du chatbot à la première connexion est la suivante :

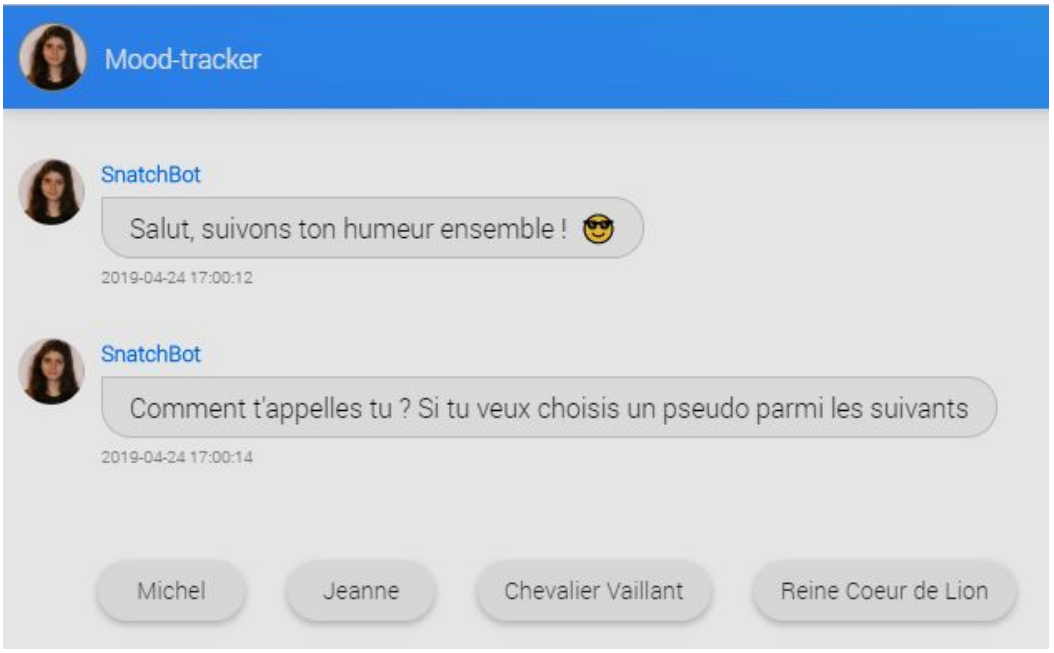


Fig. 17: Interface web du chatbot

Lorsque l'utilisateur entre un souvenir, l'émotion est inférée par Tone Analyzer puis elle lui est renvoyée dans la conversation avec le score associé.

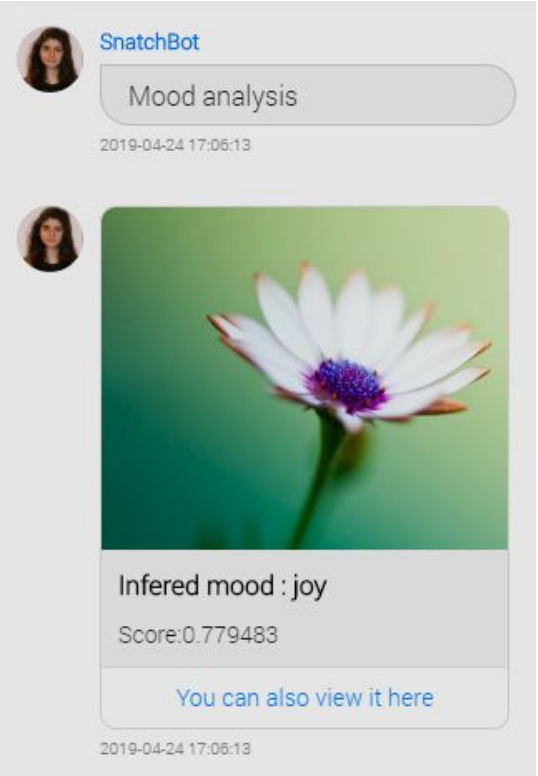


Fig.18 : Affichage des résultats de l'inférence de l'émotion

En parallèle, cette émotion inférée est inscrite dans le tableur Google Sheets, avec le score, la date et l'entrée.

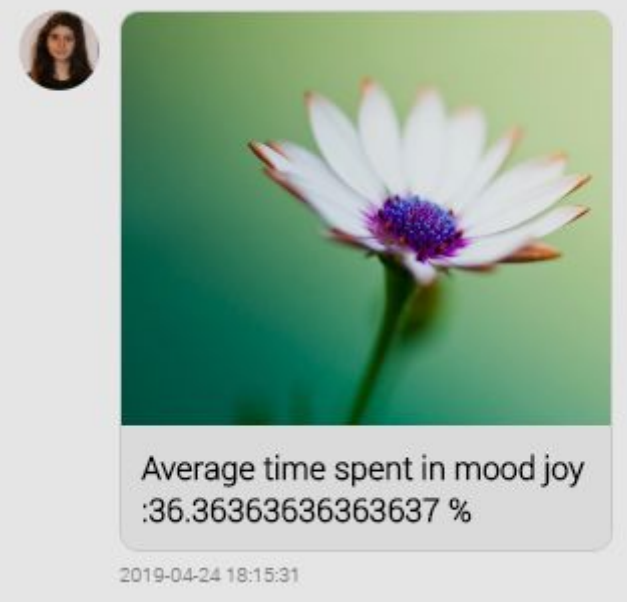


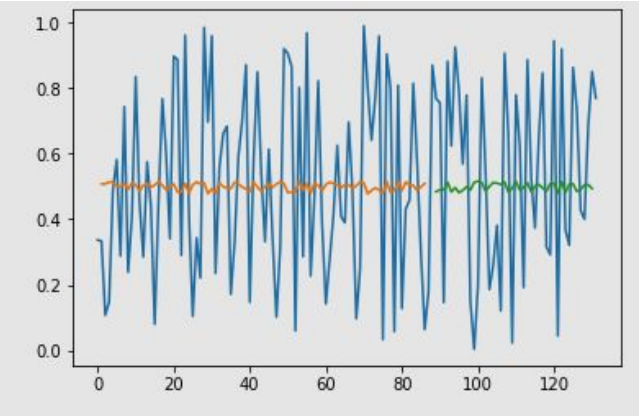
Fig.19 : Affichage de la fréquence de l'émotion

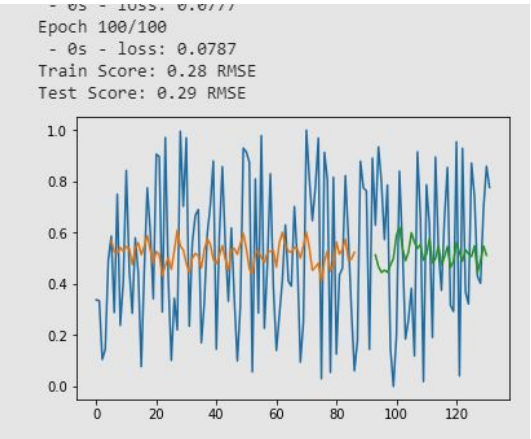
	B	C	D	E
1	date	entry	mood	score
28	2019-04-24T15:06:11.684Z	J'ai eu une super journée ! J'ai bien ri avec mes amis, c'était vraiment sympa ! Vivement demain	joy	0.779483

Le modèle développé à l'aide de Keras a les performances suivantes (mesurées en écart de la prédiction par rapport à la moyenne : RMSE) :
Concernant le réseau de neurones, après plusieurs itérations, il apparaît que le modèle avec une mémoire de 3 données est plus robuste à la diminution du nombre *d'epochs* (nombre de fois où le modèle peut itérer et améliorer sa précision). La mémoire des données semble donc intéressante.
De plus après avoir généré des données aléatoire et créé un modèle d'apprentissage avec les mêmes paramètres (batch_size= 2, epochs = 100), les résultats sont :

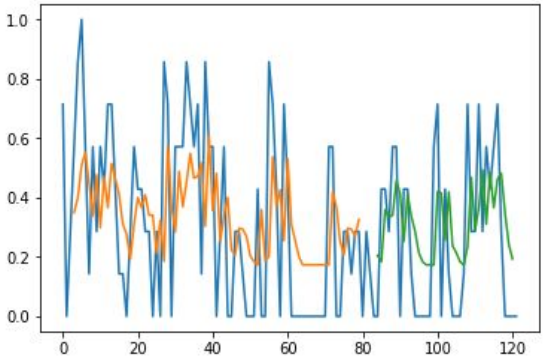
```
Epoch 100/100
 - 0s - loss: 0.0816
```

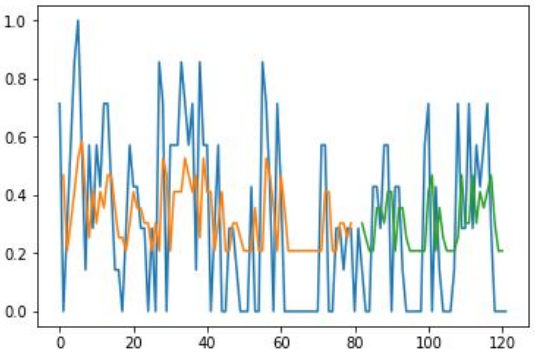
```
Train Score: 0.28 RMSE
Test Score: 0.29 RMSE
```





aléatoire look_back =1 VS 3





look back : 3 , joie VS look_back 1
RMSE : 0,27 - 0,24 VS à 0,27 - 0,25
La différence d'apprentissage montre qu'il y a bien une structure intrinsèque aux données et qu'il semble donc pertinent d'utiliser des réseaux de neurones récurrents.

Retour sur le projet

Fonctionnalités réalisées

Repère	Désignation de la fonctionnalité	Réalisation
FP1	Saisir un souvenir	oui : interface Snatchbot
FP2	Accéder à un suivi de son humeur	oui : tableur google Sheets téléchargeable en csv pour réaliser un graphique avec le script Python+ google Colaboratory
FP3	Accéder à des corrélations selon l'heure, le jour	oui : script R réalisé
FP4	Accéder à des corrélations entre l'émotion et le contenu du texte (par *key phrase extraction*)	non
FP5	Accéder à des corrélations entre émotions si simultanées	oui : script R réalisé
FP6	Accéder aux souvenirs précédents	oui : accessibles en ligne sur le tableur google Sheets
FP7	Accéder à des aides selon les émotions fréquentes / en cours	non

Points forts et améliorations possibles

L'inférence de l'émotion réalisée par Tone Analyzer est précise et rapide.
La partie machine learning gagnerait à être plus poussée, mais à titre de preuve de concept, les résultats du réseau démontreNT qu'il serait possible d'apprendre des données récoltées avec le chatbot et de prédire l'humeur d'un utilisateur à terme.

Si je devais refaire ce projet, je cadrerais mieux le sujet dès le départ afin de clarifier les fonctionnalités primordiales à réaliser. N'ayant que peu de connaissances préalables en traitement du langage et Machine Learning, il était assez difficile d'évaluer la difficulté et le temps que ces tâches prendraient.

Si je continuais ce projet, je souhaiterais implémenter une version plus avancée d'apprentissage sur les données (en croisant les variables afin de corréler les émotions entre elles). Je souhaiterais également effectuer plus de retours à l'utilisateur que la fréquence de l'émotion en cours. En effet, les requêtes en *Google Charts Query Language*[19] sont assez puissantes et permettraient de réaliser des traitements variés sur les données stockées dans le tableur en ligne.

Enfin, une amélioration possible consisterait à stocker non pas l'émotion maximale inférée mais toutes les émotions considérées comme significatives par l'API (score > 0.75). Cela impliquerait de revoir la structure des données, par exemple en :

Date : score émotion1 | score émotion2 | score émotion3 …

En effet, j'ai utilisé cette structure pour l'apprentissage des modèles de régression et du réseau de neurones et elle s'est révélée efficace et pratique.

Conclusion

Ce projet m'a permis de découvrir beaucoup d'outils et de technologies différentes.
J'ai également appris à m'organiser et gérer mon temps en autonomie afin d'obtenir les résultats escomptés sur un projet de 3 mois.
Effectuer régulièrement des comptes rendus d'avancement m'a aidée à situer mon avancement et planifier les tâches suivantes à réaliser.

Annexes

Bibliographie

[1] Malhi GS, Hamilton A, Morris G, et al. The promise of digital mood tracking technologies: are we heading on the right track?Evidence-Based Mental Health 2017;20:102-107.

[2] Plutchik, Robert. "A Psychoevolutionary Theory of Emotions." *Social Science Information*, vol. 21, no. 4-5, 1982, pp. 529–553., doi:10.1177/053901882021004003.

[3] S. A. Golder and M. W. Macy. Diurnal and seasonal mood vary with work, sleep, and daylength across diverse cultures. Science, 333(6051):1878–1881, 2011.

[4] Md. Golam Rabiul Alam, Sarder Fakhrul Abedin, Seung Il Moon, Ashis Talukder, Anupam Kumar Bairagi, Md. Shirajum Munir, Saeed Ullah, Choong Seon Hong. (2017). Affective Mood Mining Through Deep Recurrent Neural Network. 501-503.

[5] Suhara, Yoshihiko & Xu, Yinzhan & Pentland, Alex. (2017). DeepMood: Forecasting Depressed Mood Based on Self-Reported Histories via Recurrent Neural Networks. 715-724. 10.1145/3038912.3052676.

[6] Bengio, Y., et al. "Learning Long-Term Dependencies with Gradient Descent Is Difficult." IEEE Transactions on Neural Networks, vol. 5, no. 2, 1994, pp. 157–166., doi:10.1109/72.279181.

[7] Understanding LSTM Networks -- colah's blog, repéré à http://colah.github.io/posts/2015-08-Understanding-LSTMs/