Projet chef d’oeuvre E1

Titre professionnel "Développeur en intelligence artificielle" de niveau 6 enregistré au RNCP sous le n°34757

Passage par la voie de la formation - parcours de 19 mois achevé le 20 octobre

Décembre 2020

L’Email Classifier

Fatima MOQRAN

SOMMAIRE

Introduction - l'Email Classifier

Le projet présenté dans ce document est un classifier d'emails.
Un classifier d'emails est une intelligence artificielle qui comprend un texte (ici le texte d'un email) et qui le classe dans le bon dossier ou la bonne catégorie.

Ce projet peut être utilisé pour plusieurs cas :

- la priorisation : Une personne ou un service qui reçoit une très grande quantité d'emails et qui doit prioriser les emails qui doivent être traités en premier, ceux par exemple qui apportent le plus de valeur à l'entreprise.

- L'affectation : Un service client par exemple qui reçoit des mails qui doivent être affectés à des personnes différentes pour traitement.

Pourquoi avoir choisi ce sujet :

L'intérêt que je porte au traitement du texte et du langage m'a poussé tout d'abord à choisir un sujet dans ce domaine. Ensuite, j'ai discuté avec des amis et des collègues (notamment commerciaux) qui me disaient qu'un outil de ce type leur serait très utile dans leur travail face à la quantité de mails à traiter qu'ils reçoivent chaque jour.

Par ailleurs, un projet similaire chez Orange (entreprise chez laquelle j'ai fait mon contrat de professionnalisation) m'a inspiré : Le service clientèle emploi des personnes qui lisent les mails et qui les distribuent aux différents services concernées. Ce projet permettrait d'automatiser cette tâche et de libérer ces employés qui effectuaient cette tâche là.

Enfin j'ai eu accès à des données intéressantes : Une base d'emails professionnels classés dans différentes catégories. Ce sont les emails d'un alliance manager qui s'occupait de développer l'activité de l'entreprise grâce à des partenariats avec d'autres entreprises.

Le cas d'usage que je vais présenter dans ce rapport est un cas d'usage fictif mais il est inspiré des cas d'usages mentionnées plus haut.

Nous verrons ainsi dans une première grande partie de ce rapport tous les aspects fonctionnels et techniques, ce qui comprend les besoins en développement d'une application d'intelligence artificielle, le développement d'une API d'intelligence Artificielle de classification d'emails et le développement d'une application qui utilise cette API.

Ensuite dans une deuxième grande partie nous étudierons la mise en oeuvre du projet en faisant un focus sur la gestion de projet, un retour d'expérience sur les outils, les compétences et les techniques utilisées ainsi que l'organisation technique et l'environnement de développement du projet.

Enfin nous ferons un bilan sur le projet et les suites à donner au projet.

Dans ce rapport nous ferons référence aux **annexes** qui se trouvent à la suite de ce rapport. Vous trouverez dans ces annexes du code et aussi des informations et détails supplémentaires pour chaque partie.

Partie 1. Aspects fonctionnels et techniques

1 Le besoin en développement d'une application d'intelligence artificielle

1.1 Analyse du besoin client

La société ReadChain est une entreprise qui développe un nouveau moyen de communication et de transaction inter-entreprises en utilisant la blockchain. Ce moyen permet aux entreprises de sécuriser les paiements avec des entreprises clientes et fournisseurs et de pouvoir disposer de garanties de paiements et d'assurance.
Suite à un contexte exceptionnel lié à la crise de la Covid19, beaucoup d'entreprises dans le monde ont dû gérer des retards de paiements ainsi que beaucoup d'impayés qui ont mis ces entreprises dans une situation critique avec de très gros problèmes de trésorerie.
Ainsi la solution que propose ReadChain répond à un problème que rencontre de plus en plus les entreprises ce qui va changer les futurs usages des organisations en matière de paiements.

L'équipe commerciale de ReadChain connaît donc une explosion de son activité et doit se focaliser sur les demandes les plus importantes. Les commerciaux reçoivent une grande quantité d'emails par jour pouvant aller jusqu'à la centaine d'emails par jour.
Afin de faciliter leur travail, la société ReadChain souhaiterait disposer d'un outil qui pourrait comprendre un email entrant et le classer dans une catégorie ce qui permettrait aux commerciaux de gagner du temps en traitant les emails qui les intéressent le plus en premier.

Vous trouverez ces informations synthétisées dans le cahier des charges fonctionnels en annexe n°1

Notre client souhaite que nous développions une Intelligence artificielle qui pourra être intégrée dans différents types d'outils. Cette intelligence artificielle pourrait être intégrée à une application mais aussi à une boite email, ou autre (assistant vocal etc..).

A cet effet et après discussion avec le client nous avons identifié plusieurs besoins : des besoins explicites, des besoins implicites et des livrables potentiels.

<u>Les besoins explicites :</u>
Les besoins explicites sont des besoins qui sont explicitement exprimés par le client : ici c'est une intelligence artificielle qui comprend un email et le classe dans la bonne catégorie.

C’est également le fait que cette intelligence artificielle soit intégrable dans plusieurs types d'applications, elle pourra être intégrée sous forme d’application web mais aussi à une boite email ou autre.

Les besoins implicites : Ce sont les besoins qui ne sont pas exprimés clairement par notre commanditaire mais qui après discussions avec le client se sont avérés nécessaires.
Le besoin implicite du client est que l’intelligence artificielle réponde au mieux au besoin dans un contexte réel et que nous ayons des indicateurs de performances ainsi que des axes d'amélioration.

Les livrables : A l’issue de l’expression de ces besoins nous avons décidé de développer le projet en deux grandes étapes qui vont donner lieux à deux livrables :

- Le développement d’un modèle de classification d’email qui sera accessible sous forme d’api

- le développement d’une application qui interagit avec cette API et qui nous permettra de tester le modèle dans la réalité et nous permettre d’avoir des statistiques sur les performances du modèle et de voir si le modèle est bien adapté à la réalité.

Nous pouvons résumer l’analyse du besoin dans ce tableau

Besoins explicites	**Besoin implicites**	**Livrables**
Intelligence artificielle qui comprend le texte		API d’intelligence artificielle
Intelligence artificielle qui sera intégrable dans différents types d’outils		API d’intelligence Artificielle
	Avoir des statistiques sur les performances de l’intelligence artificielle confrontée à la réalité.	Développement d’une application qui interagit avec cette intelligence artificielle dans une situation réelle et qui permet d’avoir des statistiques.

1.2 Les spécifications fonctionnelles

Il y a deux utilisateurs qui vont interagir avec l'application.

Un utilisateur qui classe les emails et un administrateur qui consulte les performances de l'intelligence artificielle.

L'utilisateur final :
L'utilisateur veut classer ses emails dans la bonne catégorie.

Ainsi l'application devra :

1) Récupérer un email et l'analyser grâce à l'appel à l'api d'intelligence artificielle
2) Dire à l'utilisateur vers quelle catégorie le modèle a classé son email et s'il est d'accord ou pas
3) S'il n'est pas d'accord, il pourra changer de catégorie
4) Enregistrer en base l'email dans la catégorie choisie et enregistrer également la prédiction du modèle

L' administrateur :

L'administrateur a accès à des statistiques lui permettant de mesurer les performances du modèle dans un contexte réel afin de le rendre plus performant.
Il veut connaître le taux d'échec du modèle ainsi que les taux d'échec par catégorie.

Ainsi l'application devra :

1) Enregistrer les emails en base avec la catégorie proposée par le modèle et le modèle choisi par l'utilisateur.
2) Donner le pourcentage d'échec du modèle ainsi les emails qui ont été mal classées avec la catégorie prédite et la catégorie choisie.

Voici un tableau récapitulatif des spécificités fonctionnelles.

Besoin Client	Spécifications fonctionnelles
Classer les emails dans la bonne catégorie (utilisateur final)	● enregistrement utilisateur ● connexion utilisateur ● Récuperer un texte ● Analyser grâce à l'appel API ● Afficher à l'utilisateur la prédiction de l'API ● Afficher un menu déroulant pour le choix de l'utilisateur ● Enregistrer en base l'email dans la catégorie choisie
Accès aux statistiques du modèle (Administrateur)	● identification ● Enregistrer les emails en base avec la catégorie proposée par le modèle et le modèle choisi par l'utilisateur. ● Donner le pourcentage d'échec du modèle ainsi les emails qui ont été mal classées avec la catégorie prédite et la catégorie choisie.

1.3 Le parcours utilisateur et schéma fonctionnel

A partir de ces fonctionnalités, nous avons élaboré avec le client un parcours utilisateur.

Parcours utilisateur :

L'utilisateur final entre le texte d'un email dans l'application, via une interface utilisateur. Ce texte est envoyé à l'API qui retourne une prédiction. Si l'utilisateur est d'accord avec cette prédiction, l'application enregistre l'email dans la catégorie proposée par le modèle.Si l'utilisateur n'est pas d'accord avec la prédiction, l'application enregistre l'email dans la catégorie choisie par l'utilisateur.

L'administrateur consulte les statistiques du modèle dans l'application. Il va voir le nombre d'emails classés, le nombre d'emails bien classés par l'API et le nombre d'emails mal classés par l'API et de telle catégorie vers une autre catégorie.

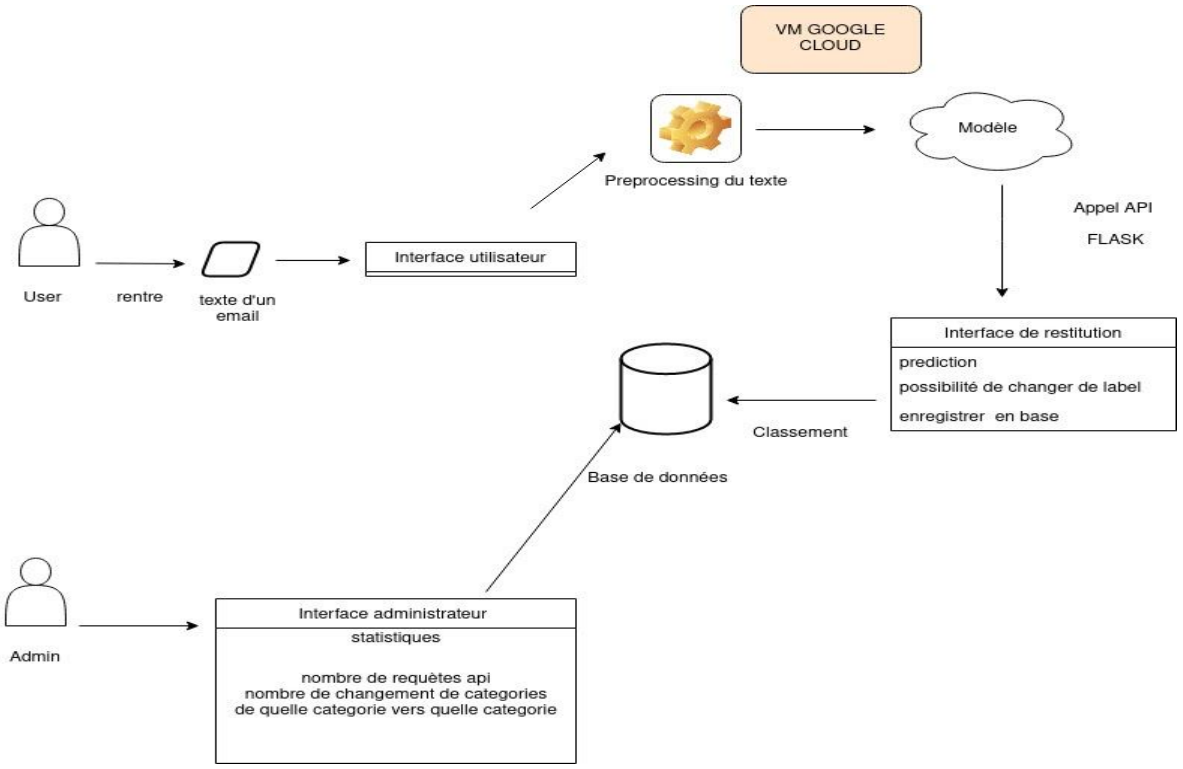


Fig 1 Schéma fonctionnel

1.4 Choix des outils utilisés pour le développement de l'application

Pour cette application nous avons choisi de développer avec **Flask** qui est un framework qui permet de développer des applications en langage python.

Nous avons choisi ce framework pour plusieurs raisons :

C'est un micro framework facile à apprendre lorsque l'on commence à développer des applications web. Ce qui nous a permis de développer une application assez rapidement.
Il est léger, flexible et il comporte plusieurs modules que nous pouvons installer au fur et à mesure des besoins.
Flask comporte un un moteur de templates Jinja qui s'intègre bien avec flask et simple à utiliser.
Ce framework était donc bien adapté à nos contraintes qui était de développer une application simple dans un temps minimum.

Il existe un autre framework Django qui permet de développer des applications en Python. Néanmoins nous avons trouvé flask pour facile à prendre en main que Django.

1.5 L’interface de l'application

L’interface de l’application a été maquettée avec l’application **FIGMA.** Figma est un outil pour le design des interfaces et permet un travail collaboratif.
Il permet le versioning des fichiers et des commentaires sur les interfaces créées. Travailler avec cet outil a permis de partager au fur et à mesure l’avancée du travail avec l’équipe.

En collaboration avec le client, nous avons élaboré des maquettes pour l’interface de l’application. Le client souhaitait un style simple et épuré et qui soit aussi professionnel pour cette application de classification de mails à destination des commerciaux.
Vous trouverez dans **le cahier des charges fonctionnel en annexe 1** les maquettes de l’interface de l’application.

2 Développement d'une API d'intelligence Artificielle de classification d'emails.

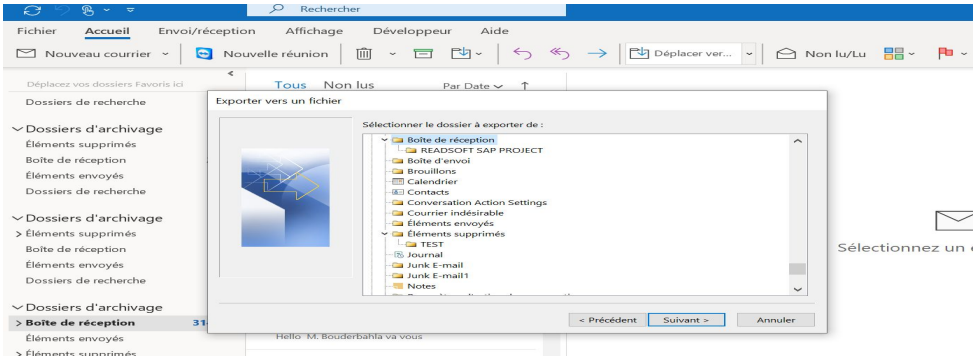
Afin de développer l'intelligence artificielle de classification de mails, nous avons tout d'abord récolté les données nécessaires à l'entraînement d'un modèle.

2.1 Les données

Le client nous a fourni une base de 5000 emails classés par sujet et archivés dans 7 dossiers différents dans outlook.

2.1.1 Extraction et exploration

L'extraction des fichiers s'est faite manuellement sur outlook. Puis les fichiers ont été transformés en csv.
Nous avons ajouté à chaque fichier csv une colonne 'label' qui correspondait à la catégorie dans laquelle la personne a classé son email.



Ensuite nous avons explorés et analysé les données avec la librairie **pandas**

	Objet	Corps	.	Deadresse	Detype	Anom
0	TR: ReadSoft Online : New Release 2 Juillet.	Chers partenaires\n \nLe 2 juillet dernier à 1...	[redacted]/O=READSOFT/OU=READSOFT/CN=RECIPIENTS/CN=BRUNO...			
1	RE: ReadSoft Mobile / ReadSoft Online	Hi again Bruno,\n \nI have checked with the de...	/O=READSOFT/OU=READSOFT/CN=RECIPIENTS/CN=STEFA...		EX	

Fig 2 : extrait du dataframe des emails avec pandas

Nous avons ici des fichiers csv comprenant différentes colonnes : objet : intitulé de l'email, corps : contenant le corps du texte de l'email , l'adresse email de l'expéditeur, l'adresse email de la personne qui a envoyé l'email, et les nom et adresses des personnes s' il y en a.

Quelques visualisations ont été faites afin d'explorer un peu plus nos données :
Nous avons tout d'abord regardé la part de chaque catégorie dans l'ensemble des données afin de se rendre compte de la disparité du nombre d'emails par catégorie.

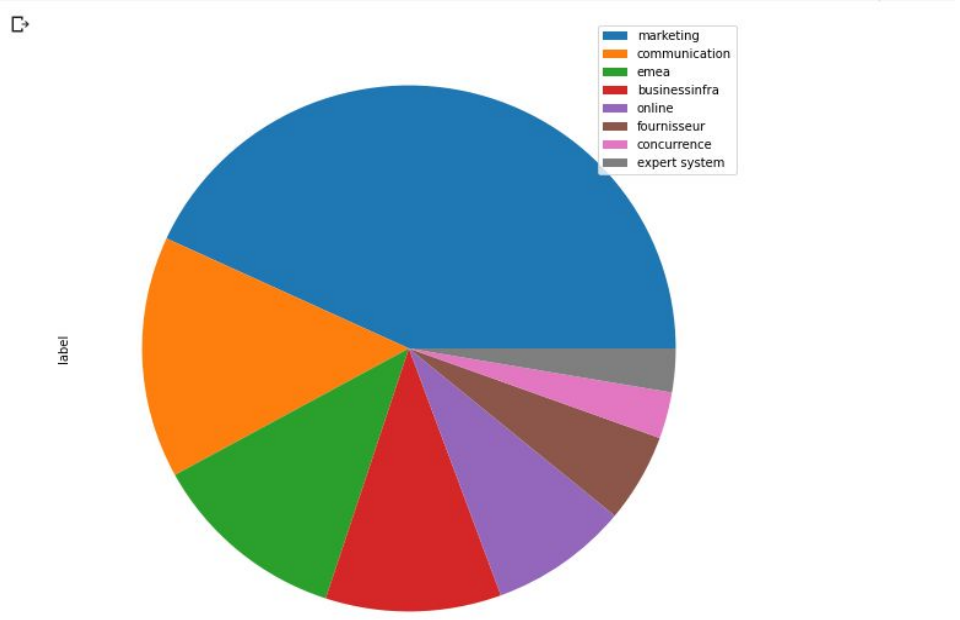
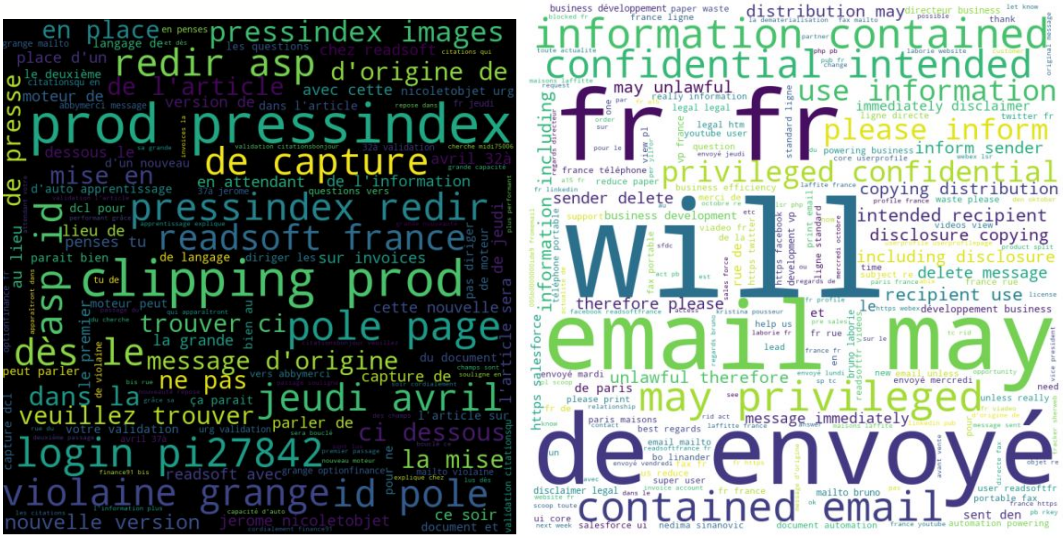


fig 3: diagramme circulaire de répartition des emails par catégorie

Nous observons que les emails sont répartis de manière très inégalitaire, il y a des catégories qui comportent beaucoup d'emails, d'autres qui en comportent très peu. Cette analyse est intéressante pour la suite car il est important de connaître la répartition des données afin de pouvoir ajuster l'entraînement du modèle.

Nous avons ensuite représenté chaque label avec un word cloud afin de voir si nous pouvions tirer des informations sur les mots qui sont les plus utilisés par catégorie .
En voici quelques un :



Catégorie : Communication *Categorie : Business infrastructure*

fig 4 : WordCloud des emails par catégorie

2.1.2 Le nettoyage des données

En analysant nos données nous remarquons que le texte comporte des caractères de contrôles comme les" \n \r" , des " - " des" |. "

"Hi Bruno,\n \nI am going thru the details and trying to setup a plan for the first iterations in the Online project starting n
ext week. I came across a request you had in your Word-doc that you sent to Dan that I think is a bit too vague.\n \nStart work
flow on Supplier\tCurrently Workflows can only be started on a rule based on values found on an invoice (or manual selected fie
ld) or on the supplier name etc. We are seeing more requirements to enable the Workflow based on other values that are not on
an invoice such as a value from the Supplier Master Data (Description/Location values). There is also a need to consider autom
atically routing through Workflows based on coding conditions so rather than values it should be against a coding suggestion.\t
\nStarting wf on supplier works as of today as you noted. Starting a wf based on Supplier Masterdata would be possible to imple
ment but since Masterdata is very different from customer/supplier it would be nice to know what specifics you have since I wou
ld have a hard time explaining the business behind \x93anv masterdata\x94 to my developers. Is it only the Description/Location
values or are there other reqs as well?\n \nBR\n \n/ \n <http://www.readsoft.com/> \r Manager SW LAB |
ReadSoft Tel: Skype nRegistered Office: Helsingborg | Reg. No: 55658
1-3176\n \nHelp us reduce paper waste. Please don't print this email unless you really have to.\n \nThe information contained i
n this email may be privileged and/or confidential. If you are not the intended recipient, use of this information (including d
isclosure, copying or distribution) may be unlawful, therefore please inform the sender and delete the message immediately. Dis
claimer: www.readsoft.com/legal/maildisclaimer"

fig 5 : extrait d'un email

Il est très important de faire un prétraitement du texte afin de pouvoir travailler sur les données. Cela permet aux librairies du traitement du langage de pouvoir analyser correctement le texte, de séparer les différentes entités d'un texte afin de pouvoir détecter par exemple les verbes, les noms propres , les url par exemple.

A cet effet, nous avons utilisé des expressions régulières (Regex) afin de nettoyer les caractères de contrôles à l'aide de la librairie re.

Vous trouverez l'ensemble du code de nettoyage des données en annexe 2 nettoyage et anonymisation des données.

2.1.3 Anonymisation des données

Les emails sont des données sensibles. Les emails que nous possédons sont la propriété d'une entreprise. Cette entreprise n'existe plus, mais il y a des données personnelles dans ces emails comme des noms de personnes, les emails des personnes, les numéros de téléphones.
Aussi la réglementation RGPD (Règlement Européen de Protection des Données) entré en application en 2018, nous impose de protéger les données personnelles. Si nous devons mettre en tant que développeur les données à disposition d'une autre équipe par exemple, hébergés dans le cloud, il est primordial de s'assurer que les données sont bien anonymisées.
Nous avons identifiés plusieurs champs à anonymiser : le champ expéditeur, le champs destinataires. Dans le corps des emails nous avons anonymisé plusieurs informations : les numéros de téléphone, les emails et les noms.

- Anonymisation champs expéditeur et destinataire

Pour l'anonymisation des champs expéditeurs et destinataires, nous avons utilisé la librairie Faker.
Faker est une librairie qui transforme des données en fausses données : elle peut générer des faux noms, des fausses adresses postales, des faux emails etc…
Nous avons donc utilisé cette librairie afin de remplacer les expéditeurs et les destinataires des emails par des faux, les noms et prénoms ainsi que les adresses emails.

- Anonymisation du corps du texte

Pour l'anonymisation du corps du texte, nous avons utilisé une librairie de traitement de données textuelles *Spacy.*
Spacy est une librairie qui est très pratique pour le traitement des données textuelles. Cette librairie permet de traiter le langage naturel avec les techniques de NLP classique. Elle permet de tokenizer un texte par exemple, c'est -à -dire de séparer le texte en petites unités. Elle peut également identifier dans le corps d'un texte des entités comme des noms propres (*Name Entity Recognition*). *Spacy* permet également de comprendre les propriétés grammaticales dans un texte, par exemple si le mot est un sujet, un verbe ou un complément(*Part-of-Speech*) tagging.
Ainsi dans le corps du texte la librairie a pu identifier les noms et les emails et nous avons

pu les supprimer.
Pour les numéros de téléphone nous avons utilisé des regex.

Vous trouverez le code d'anonymisation en annexe 2 : nettoyage et anonymisation des données.

2.2 Base de données analytique

Nous avons choisi d'utiliser une base de donnée NoSQL pour la base de données contenant nos données d'entrainements.
Nous avons utilisé une base de données MongoDB.

2.2.1 La base MongoDB pour l'analyse de données textuelles

La base de données MongoDB est une base de données orientée documents qui a une structure en Json. Cette base de données est particulièrement adaptée au format des emails qui ne suivent ni typologie ni format précis. En effet le corps des emails peut avoir des longueurs très différentes , il peut y avoir des personnes en copie ou pas….
La base de données MongoDB offre une souplesse pour la modélisation. Cette souplesse accepte une flexibilité et nous permet d'ajouter d'autres emails si à l'avenir nous voulons enrichir et réentrainer notre modèle d'Intelligence Artificielle avec de nouvelles données.

2.2.2. La procédure de mise en place de la base de données

Nous installons tout d'abord la base de données sur notre machine, ensuite nous nous connectons à celle-ci via le client du driver **Pymongo** que nous avons installé préalablement.
Le driver Pymongo permet de faire des requêtes à notre base de données avec le langage Python.
Il suffit de configurer un client grâce à l'URI de notre base et ainsi pouvoir communiquer avec notre base.

Ensuite nous créons une base de données **emails_database.** Cette base de données va stocker tous les emails que nous avons récupéré de cette entreprise mais aussi également d'autres emails que nous pourrons avoir par la suite.
Nous avons également créé une collection **readsoft _collection** où nous allons mettre les emails que nous avons récupérés et labellisés. Cette collection ne concerne que les emails que nous avons récupéré de cette entreprise. Si à l'avenir nous avons d'autres emails d'autres entreprises , nous pourrons créer une collection différente et les stocker.

Nous transformons les csv en dictionnaires et nous les insérons dans la base via la méthode insert_one() avec un script qui se trouve **dans l'annexe 3 Base de données.**

2.2.3 Modélisation de la base de données

Le data modèle de la base de données suit la méthode de l'embedding et permet de faire plusieurs types de requêtes.
Ainsi il est possible de faire des requêtes sur l'expéditeur, le destinataire de l'email, des requêtes par objet, et plus essentiel pour nous de faire des requêtes sur le **label** ce qui nous permettra de pouvoir ajouter des emails d'une catégorie particulière. Cette modélisation permettra d'enrichir la base de données avec de nouveaux emails qu'il sera très facile d'ajouter.
Lorsque nous recevrons d'autres emails avec de nouveaux labels il suffira de les ajouter à notre collection de la même manière.

Data Model :

```
{
"_id": {
"$oid": id de l'email
},
"Objet": objet de l'email,
"Corps": corps de l'email,
"Denom": nom de l'expediteur de l'email,
"Deadresse":adresse email de l'expediteur,
"Anom": nom du destinataire de l'email,
"Aadresse":adresse email du destinataire de l'email,
"Ccnom":nom du destinataire en copie,
"Ccadresse":adresse du destinataire en copie,
"label": classe dans laquelle l'email a été classé
}
```

fig 6 : Data Model de la Base de données

2.2.4 Interaction avec la base de données

Pour interagir avec la base de données nous avons créé des fonctions qui nous permettent de faire des requêtes

Toutes les requêtes se trouvent en annexe 3 base de données analytique.

Voici le résultat que nous obtenons pour une requête avec la fonction find_query()

```
email = find_query('localhost:27017','emails_database','readsoft_collection',{"label": "products"})
```

```
{"_id": {
"$oid": "5fac22e9ed66ed49e7d0bd52"
},
"Objet": "Business Objects Registration Confirmation",
"Corps": "[REDACTED], Thank you for registering CR Standard 10.0. Your Registration number is 8065564240 Please enter this number into the Registration Wizard, so you can readily access this information in the future. You can launch the wizard by going to the Help toolbar and selecting 'Register/Change Address'. You may also wish to record this number elsewhere”,
"Denom": "Adam Thomas",
"Deadresse": "lekatie@yahoo.com",
"Anom": "Deborah Ross",
"Aadresse": rachel82@hotmail.com","
"Ccnom": null,
"Ccadresse": null,
"CcType": null,
"label": "businessinfra"
}
```


fig 7 résultat d’une requête

2.2. Entraînement du modèle

2.2.1 L'échantillonnage

Ce cas d’usage de classification d'e-mail était un cas assez complexe compte tenu de la ressemblance des catégories. En effet, cette classification avait été faite par le professionnel qui avait ses propres repères, sa propre logique et il était difficile de classer un email dans une catégorie même pour un être humain.
Il y avait également des labels qui se ressemblaient beaucoup et des catégories déséquilibrées.
C’est pourquoi afin que notre modèle puisse tout de même apprendre, nous avons effectué un rééchantillonnage :
Nous avons supprimées certaines catégories : marketing , concurrence
Nous avons regroupé d’autres catégories : fournisseurs et experts system.

Nous avions donc un échantillon de 3523 emails regroupés en 5 catégories d'emails.

```
label
businessinfra          714
communication          996
emea                   803
fournisseur/expert     545
online                 565
```

fig 8 : nombre d'emails par catégorie

2.2.2. Les choix techniques

Pour ce projet, nous avons choisi d'utiliser la librairie *K-train* qui est une surcouche de *Tensorflow* et qui permet d'entrainer des modèles très rapidement avec un minimum de code. Cette librairie facilite la préparation des données, aide à trouver le taux d'apprentissage adéquat et permet d'implémenter des architectures différentes de réseau de neurones assez facilement. Notamment les architectures de transformers.
Nous avons choisi cet outil car notre client a besoin d'un résultat assez rapidement et que nous pouvons tester des méthodes différentes afin de lui proposer une solution dans les meilleurs délais.

2.2.3 L'algorithme

Le problème auquel nous sommes confrontés est un problème de classification de texte.
Historiquement ce problème de classification de texte est traité avec du feature engineering c'est-à-dire la transformation des données texte en matrice et des classifiers comme des régressions logistiques et des SVM qui sont des algorithmes de classification du machine learning.

Ces dernières années ont connu de grandes avancées dans le NLP avec des réseaux de neurones profonds(d*eep neural network)*.
Nous avons choisi d'utiliser un des modèles de référence dans ce domaine : le modèle BERT(*Bidirectional Encoder Representations from Transformers*)**.**

En effet ce modèle permet de prendre en compte de très longues séquences et de bien comprendre un long texte ce qui est le cas des emails et c'est le modèle qui a donné les meilleurs résultats dans tout ce que nous avons testé.

Développé par Google AI Language en 2019 ce modèle compte parmi les modèles de référence aujourd'hui. Il utilise les transformers et le mécanisme de l'attention qui lui permet d'apprendre les relations entre les mots.

Les Transformers et le mécanisme de l'attention :
Le modèle *Transformer* développé par Google en 2017 représente l'état de l'art aujourd'hui dans le traitement de texte. Ce modèle a été présenté dans le papier "attention is all you need".

Dans un *transformer* il y a des *encoders* et des *decoders* et des connexions entre les deux.

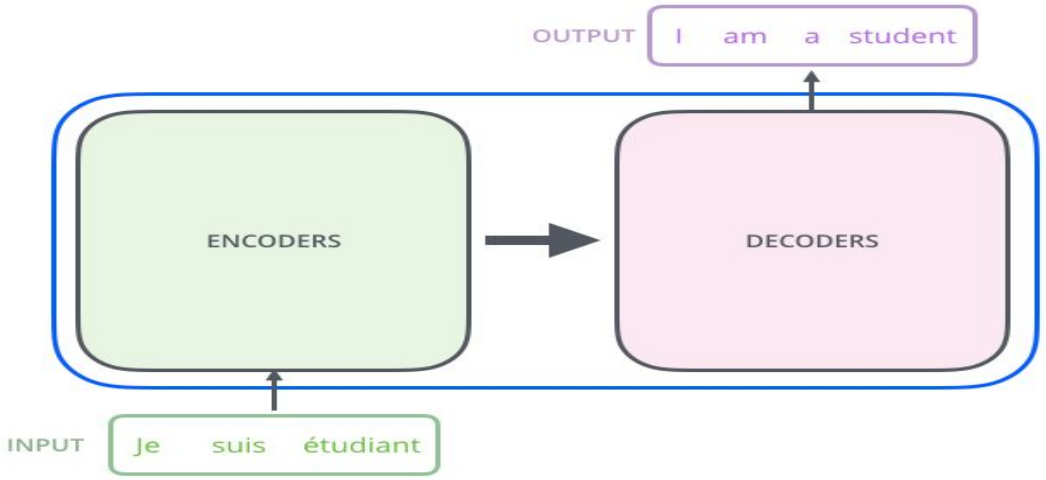


fig 9 : Les transformers [1]

Dans chaque encoder il y a deux couches : une couche de *self attention* et une couche de réseau de neurones. La couche de self attention aide l'encoder à voir tous les autres mots dans une phrase alors qu'il encode un mot spécifique .
Le *decoder* lui a trois couches : une couche de *self attention* , une de réseau de neurones comme l'encoder mais entre les deux, il a une couche *Encoder-Decoder Attention*.

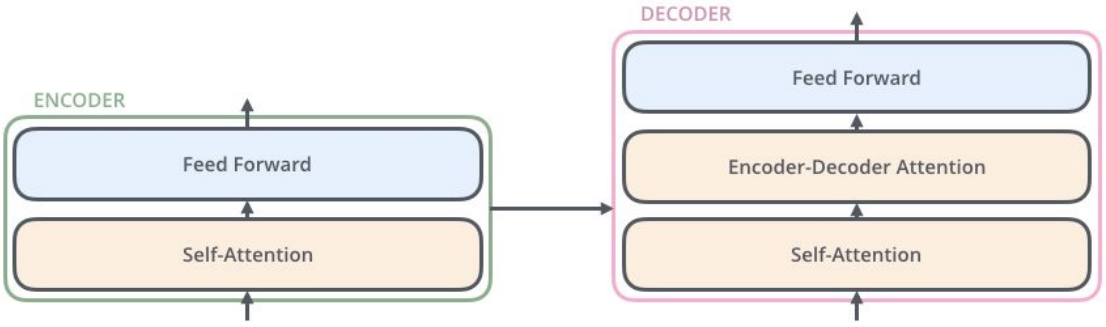


fig 10 : A l'intérieur des encoders [2]

Les *transformers* intègrent dans le réseau de neurones un **mécanisme d'attention**. Ce mécanisme permet une fois les mots transformés en vecteurs, de trouver les mots les plus importants à l'analyse de la phrase. Ce mécanisme d'attention va permettre à *l'encoder* de garder les mots les plus importants pour comprendre la phrase. Ceci va permettre au *décoder* de beaucoup mieux prédire la sortie de la phrase.
En d'autres termes pour chaque input qu'un réseau de neurones lit, le mécanisme d'attention va prendre en compte l 'input mais aussi tous les autres inputs en même temps et décide quel autre input est important pour mieux comprendre cet input.
Cette architecture est vraiment très performante et donne d'excellents résultats. C'est une des avancées majeures dans le NLP.

---

[1] The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.

[2] The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.

On utilise en général un modèle Bert pré entraîné dans le langage de son choix ou bien un modèle bert multilingue, auquel on ajoute nos données pour qu'il puisse se spécialiser sur le type de données qu'on lui donne et classifier correctement les données du même type. C'est ce que l'on appelle communément le transfer learning.

Pour notre cas nous avons utilisé un modèle BERT multilingue puisque nous avons à la fois des emails en français et en anglais.

Une revue de littérature plus complète se trouve dans le document E3.

2.2.4 L'implémentation

Le code de l'implémentation se trouve en Annexe 4

L'implémentation du modèle est assez simple à réaliser avec la librairie *Ktrain* :
Après l'import des librairies nécessaires comme k*train, tensorflow* ….nous préparons nos données en séparant les données et le label.
Plusieurs modèles sont déjà disponibles pour le traitement de texte dans la librairie *ktrain:*
le modèle *fasttext* , des *nbvsm,* le modèle BERT ou encore le modèle Distilbert.
Le **prétraitement des données** est fait automatiquement par *Ktrain* avec la fonction text.texts_from_df(). Cette fonction transforme les données en arrays qui vont entrer directement dans le modèle.
Cette fonction nous retourne une variable **preproc** qui normalise les nouvelles données. Ce fichier preproc sera aussi utile pour faire des prédictions sur de nouveaux exemples. On spécifie sur quelle colonne nous souhaitons faire l'entraînement, dans notre cas nous entraînons sur le corps des emails.

On définit ensuite notre modèle : nous utilisons un modèle bert déjà pré-entraîné en utilisant la fonction text classifier().
Il est possible de voir l'architecture de notre modèle avec la méthode model.summary() et un objet learner sera instancié.

Cet objet learner regroupe nos données (les données d'entraînement et celles de la validation) et notre modèle.
Nous pouvons trouver le meilleur taux d'apprentissage : le *learning rate* en appelant la fonction lr_find et lr_plot.
Le *learning rate* est un hyper paramètre qui nous permet de trouver le point où la fonction de coût est la plus basse(là où l'erreur est minimale). En d'autres termes, en fonction du *learning rate* nous allons trouver plus ou moins efficacement et rapidement le point qui nous permet d'avoir l'erreur la plus basse.

La fonction lr_find() simule l'entraînement pour des *learning rates* différents et regarde la fonction de coût. Il faut donc choisir le plus grand *learning rate* associé à une l*oss* (une fonction de coût) qui descend.

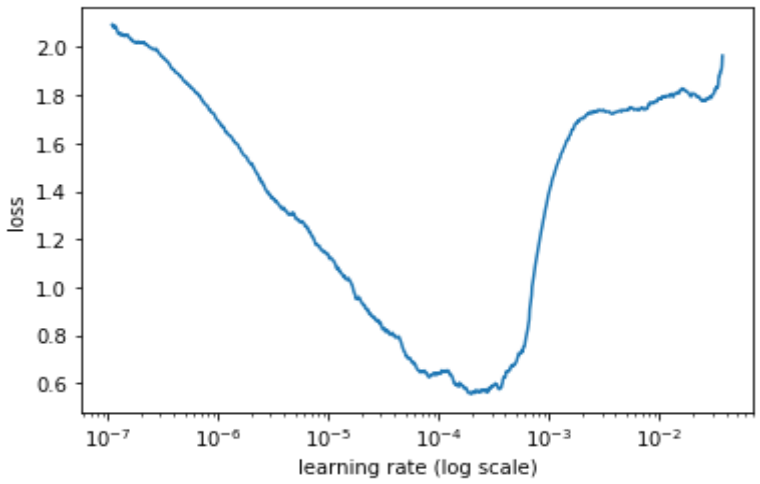


fig 11 : simulation de l'entraînement pour trouver le learning rate

Ensuite commence **l'entraînement :**

Nous avons utilisé la méthode learner.autofit(), cette méthode permet de laisser le *learning rate* varier entre deux tranches de valeur. Le l*earning rate* passé en argument est le maximum *learning rate*. L'entraînement va s'arrêter automatiquement quand la fonction de coût de validation ne s'améliore plus avec le early stopping qui a une patience de 5. C'est-à-dire que l'entraînement continue pendant 5 cycles après avoir trouvé la l*oss* la plus basse.

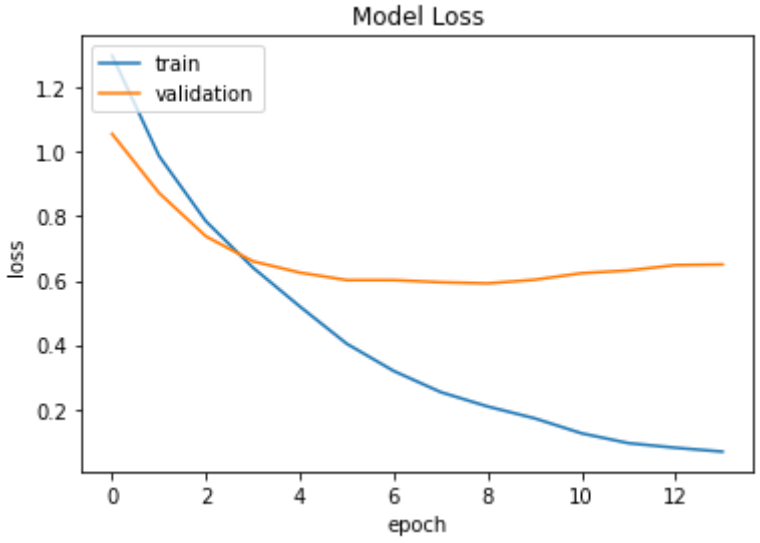


fig 12 : La fonction de coût du modèle en fonction de la durée de l'entraînement

Les **poids** du meilleur modèle sont enregistrés. Le modèle enregistre les poids de l'entraînement de l'epoch 9 juste avant que la *loss* de la validation augmente de nouveau.

2.2.5 Résultats du modèle

```
                precision    recall  f1-score   support

     businessinfra       0.67      0.83      0.74        69
     communication       0.90      0.92      0.91        87
              emea       0.84      0.76      0.80        84
fournisseur/expert       0.77      0.71      0.74        58
            online       0.78      0.72      0.75        65

          accuracy                           0.80       363
         macro avg       0.79      0.79      0.79       363
      weighted avg       0.80      0.80      0.80       363
```

L'**accuracy** est la part des observations prédites correctement sur total d'observations: ici l'accuracy est de 80%.Ce qui est plutôt correct compte tenu des sujets des classes d'e-mails qui sont assez similaires.

La **précision** représente les vrais positifs prédites dans le total des observations positives prédites: dans la catégorie business infra par exemple nous avons une précision de 0.67 c'est à dire que dans 33% des cas le modèle prédit qu'un email fait partie de la catégorie business infra alors que ce n'est pas le cas (faux positif).

Le **recall** est la part des des vrais positifs sur toutes les observations prédites de la classe par exemple pour la classe business infra le recall est de 83% c'est à dire sur tous les emails qui sont dans la catégorie business infra, notre modèle a réussi à en classifier 83% correctement.

Le **F1 score** est une moyenne pondérée de la précision et du recall et permet de se faire une idée de la performance du modèle en prenant à la fois les critères de recall et précision.

Une **matrice de confusion** est associée au résultat :

```
array([[57,  2,  1,  6,  3],
       [ 3, 80,  1,  1,  2],
       [12,  2, 64,  0,  6],
       [ 9,  3,  3, 41,  2],
       [ 4,  2,  7,  5, 47]])
```

Les chiffres en diagonales sont les prédictions qui sont bien prédites dans la bonne classe. Sur les côtés de la diagonale sont les prédictions qui sont prédites dans la mauvaise classe par ex le chiffre 12 indique qu'il y a 12 emails qui doivent être classés dans la catégorie 3 (emea) mais qui ont été mal classés dans la catégorie 1 (businessinfra).

Pour avoir une prédiction on instancie un *predictor* avec la fonction get_predictor() qui comprend le model et le fichier preproc qui permet de mettre les données dans le bon format.

Nous pouvons entrer un texte dans le *predictor* et il nous classe le texte dans la catégorie.
Nous pouvons voir la probabilité calculée pour chaque classe.
Pour chaque classe, nous avons la probabilité calculée par le modèle que le texte appartienne à la bonne classe.

Enfin nous enregistrons le modèle avec la fonction predictor.save() .

2.2.6 Amélioration du modèle

Nous avons envisagé plusieurs solutions afin d'avoir un modèle plus performant :

- l'entraînement sur les destinataires et expéditeurs des emails :

Ceci n'est pas possible car nous avons dû par soucis de protection des données anonymisées tous les noms des expéditeurs , des destinataires.
Nous avons dû également anonymiser le corps des emails en supprimant les noms et les emails dans le corps des emails.

- entraîner sur le titre des emails : le modèle donnait de moins bon résultats.

Nous avons donc décidé d'entraîner sur l**e corps des emails** puisque l'état de l'art du NLP arrive bien à traiter les séquences longues.

- testé plusieurs modèles :

le Naive Bayes Support Vector Machine.(NBSVM) qui n'est pas un réseau de neurones. Le NBSVM est une approche de classification de texte proposée par Wang et Manning qui utilise à la fois un SVM pour la classification et remplace le word embedding par le Naive Bayes log-count ratios. c'est un modèle assez simple mais qui a prouvé son efficacité et qui est bien d'utiliser avant de passer au deep learning. Cependant dans notre cas ce modèle n'a pas donné de bons résultats.

Nous avons également utilisé le modèle DistilBert avec la librairie Simple Transformers qui est un modèle Bert plus léger et donc moins long à entraîner, nous avons aussi obtenu des résultats moins bons.

- Amélioration du modèle Bert entraîné sur le corps des emails :

Nous avons tout d'abord équilibré le plus possible le dataset en faisant de l'oversampling : c'est-à-dire que nous avons raccourci certaines catégories afin d'obtenir un nombre d'emails dans chaque catégorie à peu près équivalent.

Nous avons **régulé** le modèle afin d'éviter que le modèle apprenne par coeur les données d'entrainements et soit dans de l'overfitting : pour cela le calcul du nombre d'epochs d'entraînements nécessaire avec early stopping a été appliqué avec la méthode learner_autofit() comme vu ci dessus.

Nous avons réglé le batch size : c'est-à-dire le nombre d'exemples que doit voir le modèle à la fois pour apprendre les données.
Nous avons décidé de mettre ce batch size à 32 qui donnait les meilleurs résultats

- Problème des données et pistes d'améliorations :

Ce cas d'usage n'est pas un problème facile. En effet le dataset a la particularité que les emails se ressemblent beaucoup et qu'il est difficile même pour un humain de classer les emails dans la bonne catégorie compte tenu qu'il s'agit d'un classement personnel et subjectif d'un professionnel.

L'application qui va être développé par la suite est une partie de la solution pour améliorer le modèle :
L'utilisateur pourra choisir la catégorie dans laquelle le modèle a classé le mail s'il n'est pas d'accord avec le modèle et nous pourrons nous servir de ces réponses pour améliorer le modèle.

2.2.7 Déploiement du modèle sous forme d'API

Le modèle a été sauvegardé en deux fichiers : un fichier de preprocessing qui prétraite le texte avant de le passer dans le modèle et un fichier de H5 qui contient les poids du modèle.

Le modèle a été déployé sur une **Machine Virtuelle sur Google Cloud Platform** sous forme d' API. Cette API offre la possibilité de faire des requêtes depuis n'importe quel type d'application.
Nous avons choisi de déployer sur une Machine Virtuelle car le modèle développé de deep learning était assez lourd (BERT multilingues) et il n'était pas possible de l'héberger dans des Plateformes as a Service gratuites (type heroku).

Nous avons choisi cette solution également pour des raisons de prix : Google offre des crédits qui nous permettent d'utiliser à très bas coût des machines virtuelles et s'en servir comme serveur.
En effet, déployer et maintenir un modèle d'intelligence artificielle peut être assez coûteux.

Pour faire cela il y a eu plusieurs étapes :

Vous trouverez le code de l'API dans l'annexe 5 code API du modèle de Classification d'email

- développer l'api avec Flask :

L'api commence par charger le modèle puis elle prédit la classe du texte qu'on lui a envoyé via une requête POST grâce à l'objet *predictor* de la librairie *ktrain.*
Nous avons développé deux fonctions différentes : load predictor() qui charge le modèle et une autre fonction qui prédit la catégorie du texte : cela pour éviter de charger à chaque fois le modèle dès que l'on veut faire une prédiction.Car cela serait trop long.

- Choisir et configurer notre serveur :

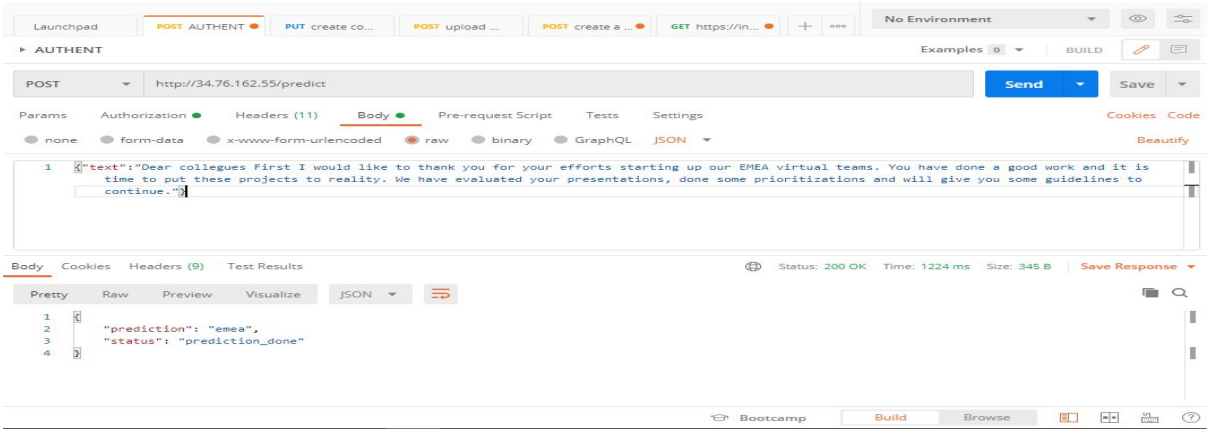
Nous avons choisi une VM CPU ubuntu standard. Nous avons également configuré une adresse ip externe afin d'avoir toujours la même adresse à appeler pour notre modèle.

- Docker file :

Nous avons rédigé un Docker file afin de tout mettre dans un conteneur et faire tourner l'API. Il était particulièrement important de faire un Docker afin d'avoir toujours le bon environnement avec les bonnes versions de librairies installées et uniquement ce dont nous avons besoin pour l'api via un fichier requirements.txt

- Test de l'API avec POSTMAN :

Une fois cela mis en place, nous avons effectué des tests unitaires avec POSTMAN pour tester l'API.



Puis nous avons testé l'API avec la méthode request en python.

```
import requests

response = requests.post("http://23.251.133.90/predict",json = {"text":"api ne fonctionne pas les clients sont furieux."} )
print(response)
print(response.text)
```

Nous avons pu communiquer avec le client afin de lui présenter le premier livrable.

3 L’application

L’API d'intelligence Artificielle étant prête, nous avons développé l’application qui va nous permettre de tester cette intelligence artificielle dans un contexte réel.

A partir des spécificités fonctionnelles explicitées dans la partie 1 du rapport, nous avons élaboré un schéma technique qui précise les outils techniques qui seront utilisés pour l’application.

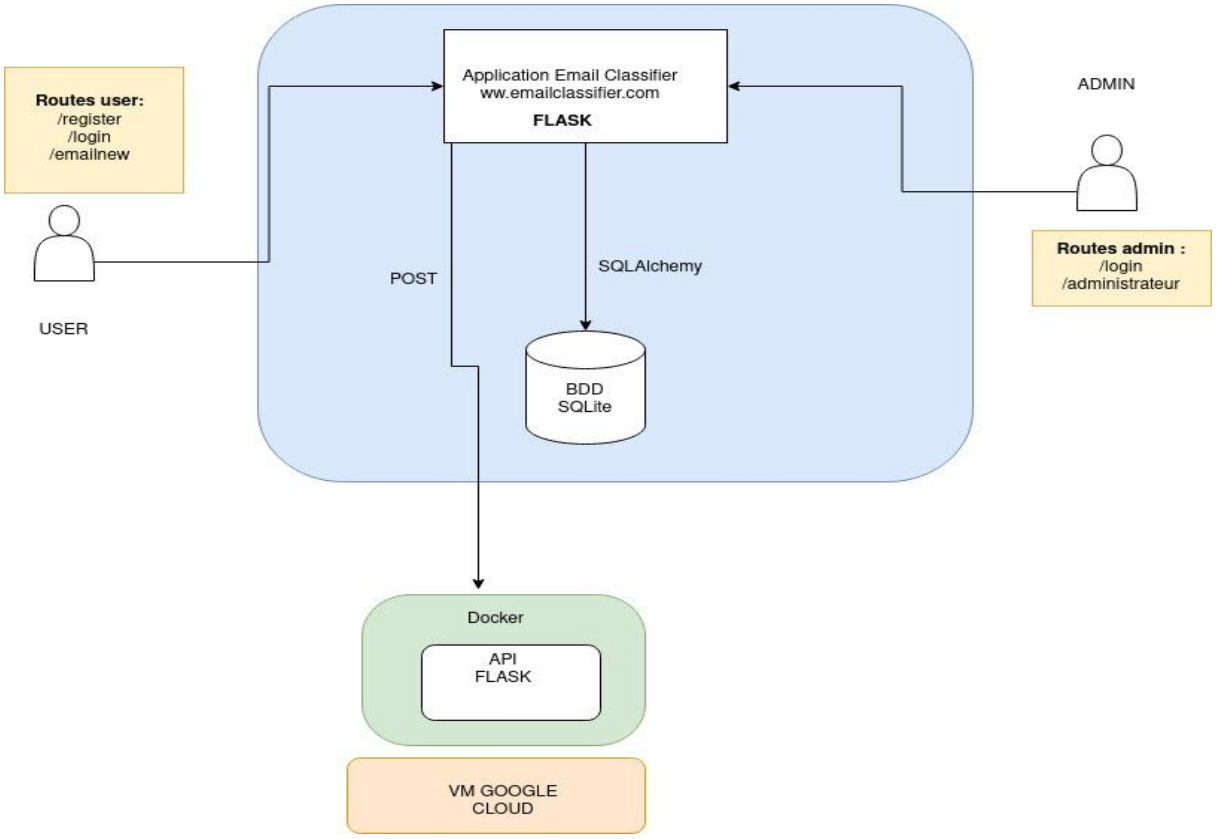


fig 13 : schéma technique de l’application Email Classifier

Nous verrons dans cette partie le détail de chaque élément de l'application.

3.1 La base de données relationnelle de l’application

Le but de la base de données est d’enregistrer plusieurs informations importantes nécessaires au fonctionnement de l’application et au développement de ses fonctionnalités.

La base de données devra donc :

- stocker les informations des utilisateurs : leurs données d’authentification comme le login, adresse email , mots de passe (cryptée)
- stocker le texte des emails pour chaque utilisateur avec la prédiction envoyée par le modèle et la classification choisie par l’utilisateur .

3.1.1.Modélisation

Nous avons utilisé un Diagramme de classe (UML) ainsi qu’un modèle physique de données (MPD) pour modéliser la base.

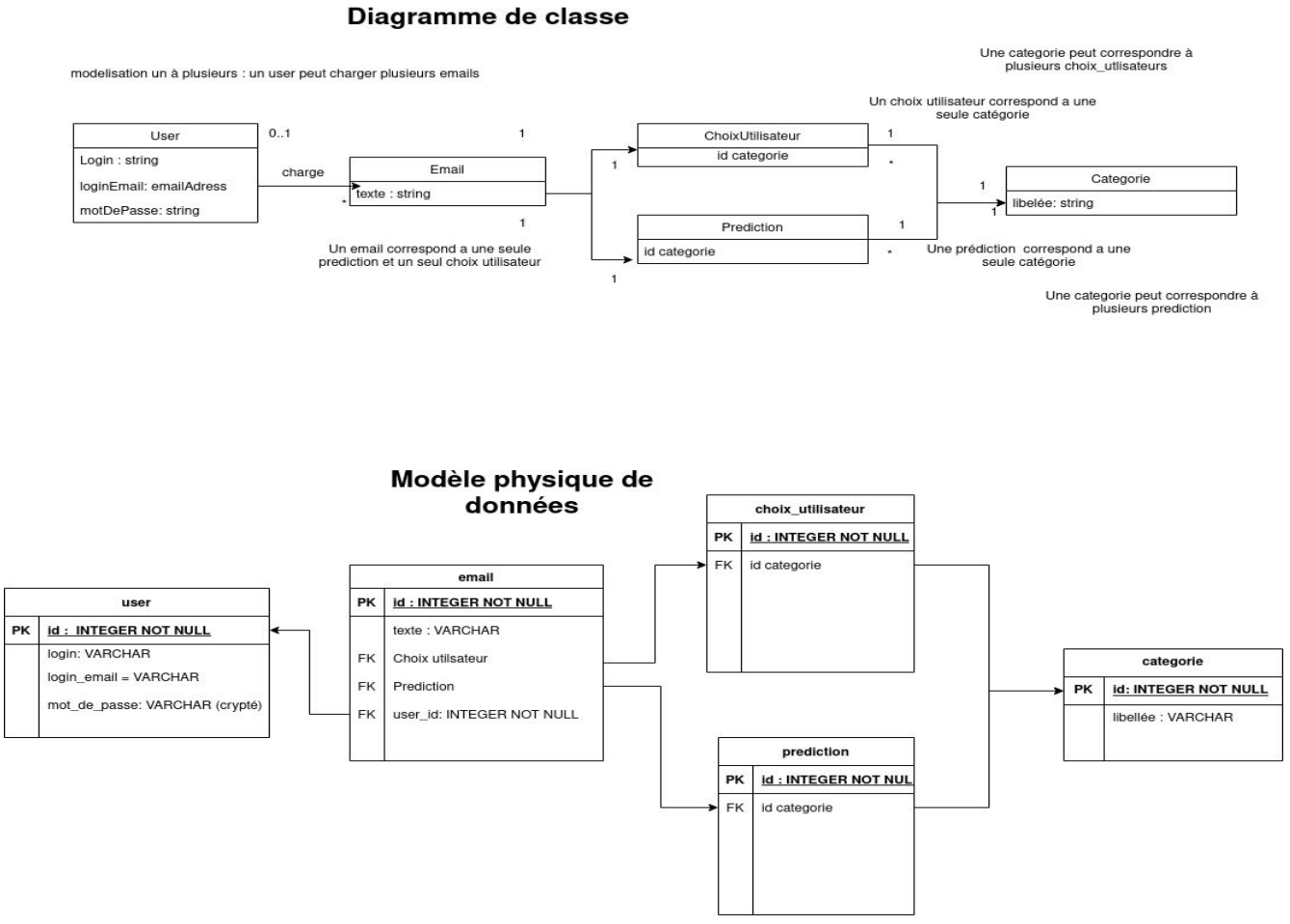


fig 14 : Diagramme de classe et modèle physique de données

Nous avons une base de données relationnelle avec 5 tables :

Une table **User** contenant les informations de l'utilisateur : son login, son login email et son mot de passe qui est stocké en crypté.
Une table **Email** qui contient le texte de l'email , une table **prédiction** qui contient la prédiction du modèle ,une table **Choix Utilisateur** avec le choix de l'utilisateur, enfin une table de référentiel catégorie avec les 5 catégories d'emails à classer pour référence.

3.1.2. Procédure de mise en place de la base de données

Vous trouverez le code du schéma dans l'annexe n°6

Nous avons utilisé une base SQLite avec Flask SQLAlchemy .
SQLAlchemy est un ORM (Object-relational mapping) qui nous permet de construire notre base de données avec des objets en python.

SQLAlchemy nous permet de définir le schéma de la base de données en code Python.
Nous avons choisi cet outil notamment pour sa facilité d'usage: les requêtes simples ne requiert que très peu de code, les requêtes plus complexes peuvent être écrites en python langage que nous connaissons. Nous avons utilisé la version intégrée avec Flask qui est Flask Sqlalchemy.

Pour notre base de données, nous avons défini une première table **User** contenant les informations de notre utilisateur. Notre table est reliée à la table **Email** .

La table **Email** contient le texte de l'email entré par l'utilisateur cette table contient une clé étrangère qui est notre utilisateur ainsi chaque email est relié à l'utilisateur qui a entré cet **Email.**
La table **Email** est reliée à la table **Prédiction** et à la table **ChoixUtilisateur** et contient deux clés étrangères qui est prediction_id et choix_utilisateur_id.

La table **Prediction** est la table qui contient les prédictions que le modèle renvoie à chaque fois que l'utilisateur classe un email.Cette table est reliée à la table email. Elle possède comme clé étrangère l' id de la catégorie.

La table **ChoixUtilisateur** est le choix que l'utilisateur fait pour classer son email. Son choix peut être le même que la prédiction ou il peut être différent.Cette table est reliée à la table email et contient une clé étrangère qui vient de la table Catégorie.

La table **Catégorie** contient les noms des catégories et cette table est reliée aux tables **ChoixUtilisateur** et **Prediction.**

3.1.3 Les Requêtes

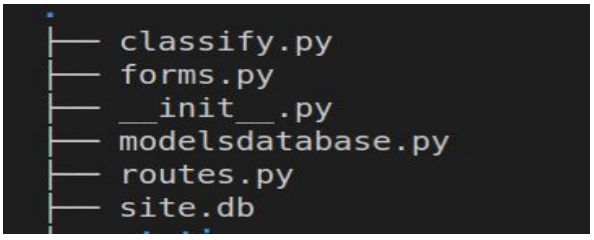
Pour le projet et au fur et à mesure du parcours de l'utilisateur nous avons eu besoin de développer des requêtes vers cette base de données afin de recueillir les informations nécessaires et d'en restituer d'autres.

Vous trouverez toutes les requêtes et le code de l'interaction avec la base de données en Annexe 7 .
Nous avons par exemple rempli la table Categorie correspondant aux différentes catégories où l'email pouvait être classé, avec des requêtes comme ceci: Les autres requêtes font partie du Back end de l'application.

3.2 Le Back-end de l'application

3.2.1 Les fichiers qui composent le back-end

Le back-end de l'application est réparti en plusieurs fichiers python :



- un fichier modelsdatabase où est définie notre base de données
- un fichier forms.py(annexe x) qui contient tous nos formulaires ,
- un fichier classify.py qui contient l'appel à l'api du modèle
- et le fichier routes.py avec les routes de l'application.

L'ensemble du code du back-end de l'application se trouve en annexe 6

Le fichier **modelsdatabase.py** contient le code de notre base de données .

Le **fichier form. py** dont le code se trouve en **annexe 8** est un fichier dans lequel nous avons développé les formulaires :

Les formulaires ont été développés avec flask_wtf et permettent l'intégration de formulaires dans l'application.
Le formulaire d'enregistrement est le formulaire qui permet l'enregistrement d'un nouvel utilisateur est défini dans la classe RegistrationForm . Ce formulaire permet l'enregistrement du pseudo, de l'adresse email, du mot de passe du User.

Le formulaire de connexion à l'application permet de se logger et de pouvoir classer son email. Ce formulaire est défini dans la classe Login Form.
Le formulaire pour entrer les emails est défini dans la classe PostForm et permet de poster les emails pour le classement.

Le fichier **classify.py** est le fichier qui permet de faire l'appel à l'api.
Dans ce fichier nous avons défini une fonction qui s'appelle classify et envoie le texte à notre serveur afin qu'il puisse être analysé par notre modèle.

3.2.2 Les routes du back end

Le fichier **routes.py** est le fichier où sont développés les différentes routes de l'application:

La route register est la route où l'utilisateur s'enregistre avec ses informations. Le mot de passe de l'utilisateur est crypté grâce à l'outil flask_bcrypt.
Dans cette route nous enregistrons en base les informations de l'utilisateur.

Vous trouverez tous les codes d'interaction avec la base de données en annexe 7

La route login : est la route qui permet à l'utilisateur de se logger. On fait une requête à la base de donnée pour s'assurer que l'utilisateur, si l'utilisateur est bien enregistré dans la base de données alors il peut se connecter et accéder aux pages accessibles pour les utilisateurs connectés.

La route logout : elle permet de se déconnecter grâce à la methode logout_user de flask_login.

La route 'email new' : c'est la route qui permet de classer un email. Dans cette route une fonction classer_email() permet à l 'utilisateur de rentrer le texte d' un email via le formulaire de classification d'emails qui se trouve dans le fichier forms.py. Ensuite dans cette route nous appelons la fonction classify du fichier classify.py. Cette fonction fait l'appel à l'api. Lorsque nous obtenons la réponse qui est la prédiction du modèle, cette prédiction est enregistrée dans la base de données dans la table Prediction.
L'email est également enregistré dans la base de données.

La route enregistrer_email : enregistre le choix de l'utilisateur. Ce choix peut être le même que la prédiction ou il peut être différent.

La route administrateur est la route qui permet à l'administrateur d'accéder à la page où il y a des informations et des statistiques pour analyser les performances du modèle.
Dans cette route nous récupérons de la base de données les informations que nous allons afficher pour l'administrateur.

Ensuite nous récupérons avec du code python les données qui nous intéressent dans les objets renvoyés par la requête.
Ces données seront transmises à la page html administrateur.

3.2.3.Les tests unitaires

Afin de s'assurer de la fonctionnalité de nos fonctions les plus critiques, nous avons réalisé des tests automatiques.
Pour cela nous avons utilisé la librairie pytest qui nous permet d'écrire les tests et de les exécuter en ligne de commande dans leur propre environnement.
Nos tests sont stockés dans leur propre espace : un dossier Tests à la racine de l'application et s'exécutent comme ceci :

```
python -m pytest -vv
```

Nous avons écrit plusieurs types de tests : des tests concernant les routes, des tests de la requête à l'api et des tests de la base de données.

L'ensemble des tests effectués se trouvent en annexe 9.

3.3 Le front end

Les interfaces utilisateur ont été développées à la fois à partir des maquettes élaborées en amont et validées par le client et aussi par le parcours utilisateur. (v**oir cahier des charges fonctionnel en annexe 1)**

Nous avons développé le front end dans des pages html qui se trouvent dans le dossier **templates** de notre projet.
Chaque page est développée en html et nous avons utilisé Bootstrap qui nous a permis de développer le design de notre application très facilement.
Bootstrap est un framework qui comprend une collection d'outils utiles à la création du design, de site internet.
Nous avons utilisé des tables, des buttons, des messages d'alertes provenant de bootstrap.

3.4 Le monitoring

Pour le monitoring de l'application nous avons utilisé l'outil Airbrake. Airbrake est un service qui nous permet de diagnostiquer et de résoudre des bugs et problèmes liés à notre application.

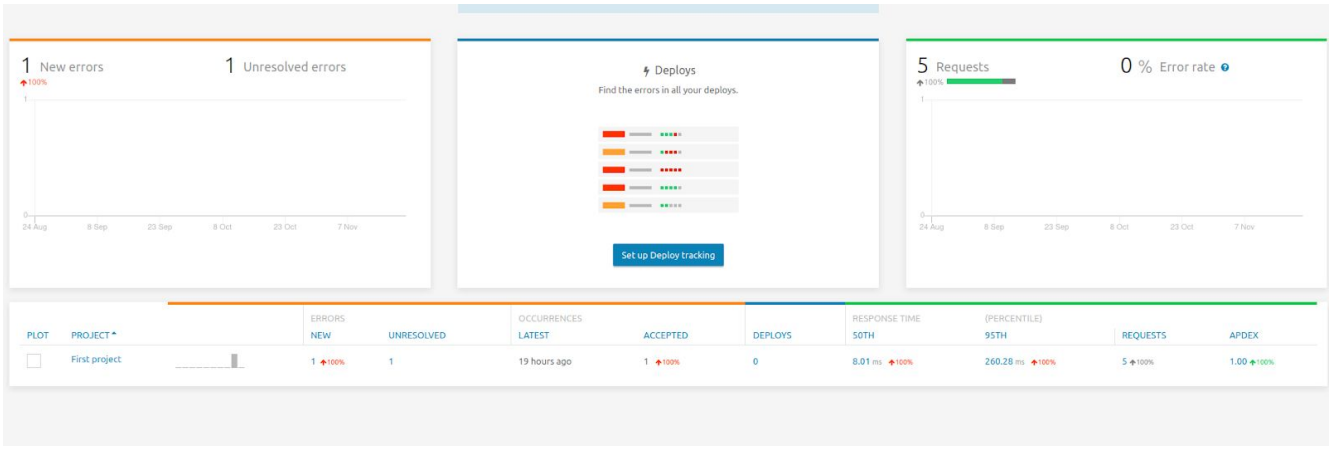
En effet nous avons des parties critiques à monitorer dans notre application :
Le login, la connexion avec l'API sont les fonctionnalités les plus critiques de notre application et doivent être monitorés.
Airbrake va recenser les erreurs renvoyées par l'application et nous avertir pour réparer le problème.

Procédure :

Il faut tout d'abord créer un compte sur airbrake.io. Ensuite nous recevons un identifiant de notre projet ainsi qu'une clé de notre projet.
Nous installons pybrake dans notre environnement de développement qui va nous permettre de notifier les exceptions de notre application .
Nous configurons dans notre fichier __init__ la connexion avec airbrake, comme ceci :

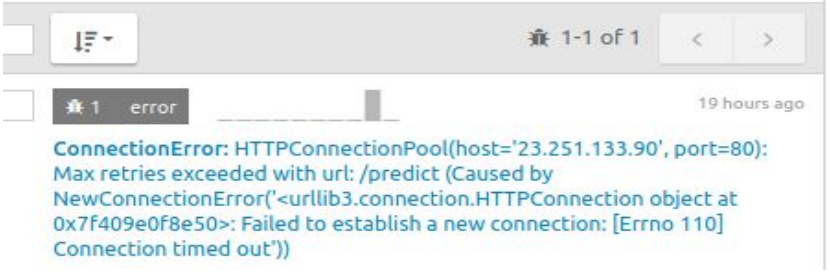
```
#configurer le monitoring avec Airbrake
app.config['PYBRAKE'] = dict(
project_id=id_du_projet,
project_key= cleduprojet',
)
app = pybrake.flask.init_app(app)
```

Ensuite nous pouvons consulter un dashboard dans le site d'airbrake pour notre application.
Nous obtenons un dashboard comme ceci :



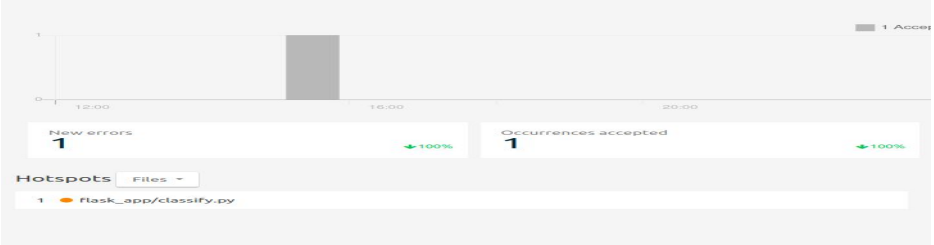
Nous voyons ci dessus que nous avons eu une erreur et que nous avons fait 5 requêtes sur notre application qui sont passées.

Lorsque l'on regarde l'erreur nous avons les détails de l'erreur :

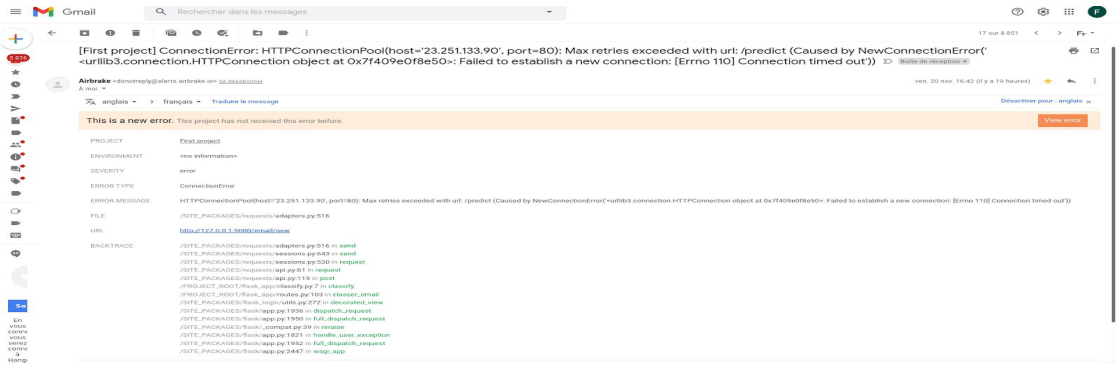


Ici l'erreur était due à un problème de connexion avec l'api de classification d'emails (le serveur était éteint). Ceci est une fonction critique et centrale de notre application et donc il est important de pouvoir traquer les éventuels problèmes sur cette fonctionnalité.

Nous pouvons voir sur quel fichier notre application a eu un problème et pouvoir détecter et résoudre rapidement la chose.



Nous avons également configuré une alerte par email afin que nous soyons très vite au courant du problème et intervenir dans les meilleurs délais.
Voici un email reçu nous informant du problème dans notre application.



Partie 2. Mise en oeuvre du projet

1. La gestion de projet

1.1 L'équipe et l'outil de gestion de projet

L'équipe chargée du projet était composée de deux développeuses : Fatima M et Fatima L .Fatima M est chargée de la relation avec le client, de la gestion du projet et aussi du développement. Fatima L s'occupe en priorité des développements mais peut aussi interagir avec le client lorsque Fatima M n'est pas disponible. Fatima M et Fatima L travaillent côte à côte et communiquent constamment ensemble afin que chacune puisse être en back up de l'autre.
Bruno L est la personne du côté client qui est chargée du projet. C'est un commercial, futur utilisateur de l'application et qui connaît bien les besoins de son équipe.

Nous avons opté pour une gestion **agile souple** du projet. Nous avons découpé le projet en sprints avec des livrables à chaque fin de sprint afin d'avoir une interaction avec le client et de pouvoir s'adapter aux changements.
Afin de gérer au mieux le projet nous avons utilisé l'outil **Trello.** Cet outil de gestion de projet est un outil gratuit pour sa version basique. Trello permet d'organiser le projet sous forme de tableaux et nous pouvons inclure dans ces tableaux plusieurs fonctionnalités comme un planning par exemple. Trello est aussi un outil collaboratif et permet donc de partager les tableaux du projet avec les différents membres de l'équipe et aussi avec le client.

Des plannings et des tableaux ont été élaborés afin de suivre au plus près l'évolution du projet. Chaque fin de sprint était suivie d'une réunion avec le client pour lui montrer les fonctionnalités développées durant le sprint et recevoir des feedbacks.
Cette méthode était particulièrement adaptée à ce projet car nous pouvions changer de cap au fur et à mesure du projet en fonction du besoin du client.
Une fois les besoins en developpement identifiés et les outils choisis, nous avons commencé par développer l'API d'intelligence artificielle de classification d'emails. La priorité dans ce projet était de pouvoir rendre compte au client le plus souvent possible que le projet avançait et de pouvoir lui livrer régulièrement et rapidement différentes fonctionnalités.
Ainsi nous avons établi avec le client des points très réguliers afin de pouvoir bien comprendre les attendus du client et pouvoir changer les choses au cours du projet pour coller aux vrais besoins du client.

1.2 Le backlog et les users stories

Après deux réunions avec notre Client Bruno L, nous avons collecté les besoins des utilisateurs sous forme de users stories. Ces users stories mettent les utilisateurs finaux au centre du produit et nous permettent de bien retranscrire leurs besoins.
Les users stories dans notre backlog ont constitué le point de départ de notre projet. Ceci va nous permettre de définir toute l'organisation du projet en termes de temps, de coût et de découpage du projet.

Notre backlog se présente sous la forme d'un tableau Trello comme ceci:

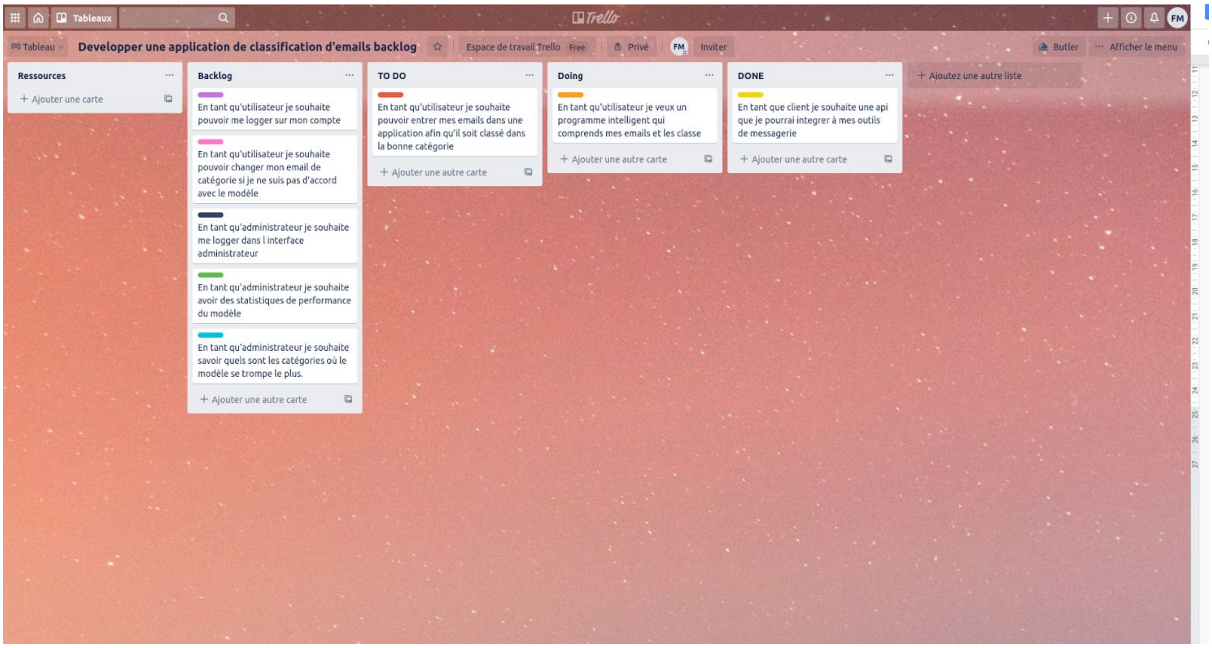


fig 14 : Le Backlog produit

Le Backlog contient 4 colonnes : une colonne Backlog avec toutes les users stories du projet, une colonne To Do où il y a les users stories à faire dans la période en cours (sprint), une colonne Doing qui contient les users stories que nous développons actuellement et une colonne Done qui liste les users stories qui ont été faites.

Une fois le backlog élaboré nous avons pour chaque user story listé les tâches à faire et nous avons discuté pour chaque tâche du temps que cela prendra.

1.3 Les planning et les sprints

1.3.1 Les sprints

Vous trouverez les tableaux pour chaque sprint en annexe 10 gestion de projet .

Ainsi nous avons découpé le projet en 3 sprints qui ont comporté chacun un tableau et un planning :

- Un premier sprint d'une durée d'un mois pour le développement d'une api de classification d'emails.
  A l'issue de ce sprint nous avons livré au client une API intégrable dans une application.
- Un deuxième sprint d'une durée de 15 jours pour le développement d'une application de classification d'emails, les fonctionnalités de l'utilisateur.
  A l'issue de ce sprint nous avons livré l'application avec les fonctionnalités de l'utilisateur.
- Un troisième sprint de 15 jours pour le développement des fonctionnalités de l'administrateur et des tests et du monitoring
  A la fin de ce sprint nous avons livré l'application complète.

Le tableau d'un sprint se décompose comme ceci :

- Une colonne ressources : avec des éléments de veille et des liens pour accomplir les différentes tâches(tutoriels, articles, éléments de code…)
- Une colonne Backlog avec les users stories du sprint.
- Une colonne To do avec les tâches à accomplir durant le sprint.
- une colonne Doing avec les tâches qui sont en cours durant le sprint.
- une colonne Done avec les tâches faites.

Pour chaque tâche il y a une étiquette qui correspond à une spécialité cela permet au développeur de savoir de quelle partie il s'agit plus rapidement.

Voici le tableau du Sprint 1 vous trouverez les autres tableaux en annexe 10:

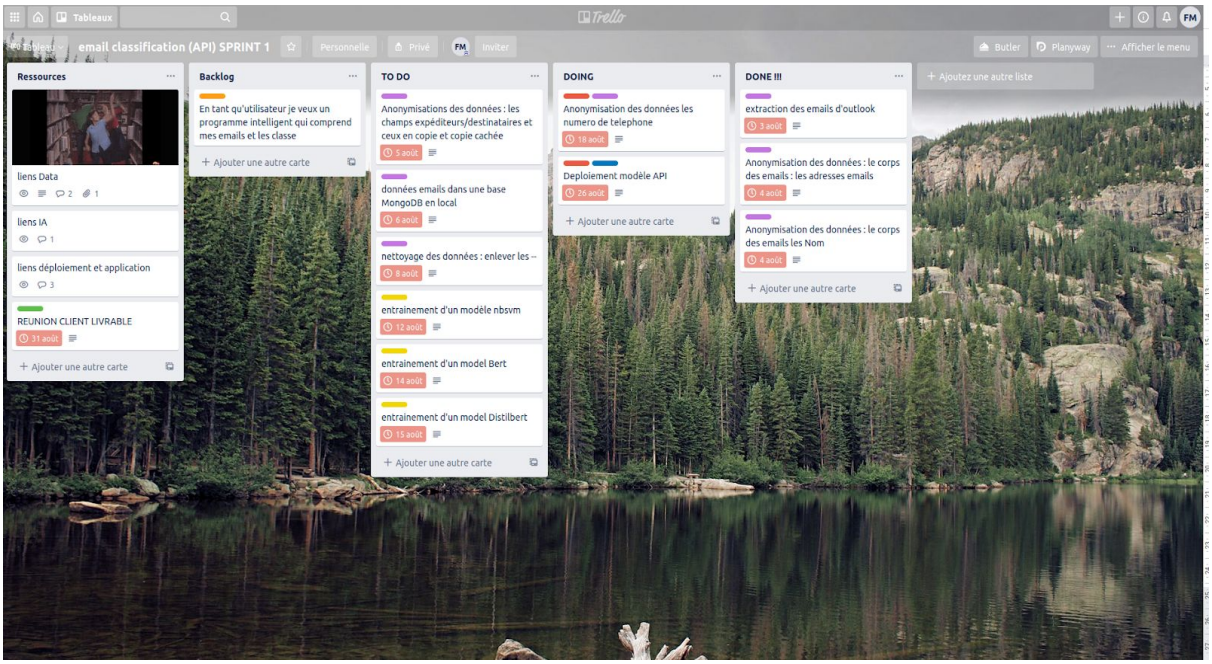


fig 15 :Sprint 1 Développement de l'API de classification d'emails.

1.3.2 Le planning des sprints

Vous trouverez les planning pour chaque sprint en annexe 10 gestion de projet .

Pour chaque tâche nous avons assigné une durée et un moment dans le calendrier.
Nous avons utilisé le plug in **Planyway de Trello** , le calendrier est donc intégré à notre tableau et toute l'équipe peut y avoir accès. Planyway récupère les tâches du tableau et nous avons mis une date de début et de fin pour chaque tâche conformément à ce que l'ensemble de l'équipe a décidé. Nous y avons également ajouté des marges de temps afin de nous permettre de faire face à d'éventuelles difficultés durant le sprint.
Les dates butoires pour chaque tâche sont reportées sur le tableau du sprint et permettent une bonne visibilité du suivi des tâches.

Voici le planning du sprint 1, vous trouverez les planning des autres sprints en **annexe 10.**

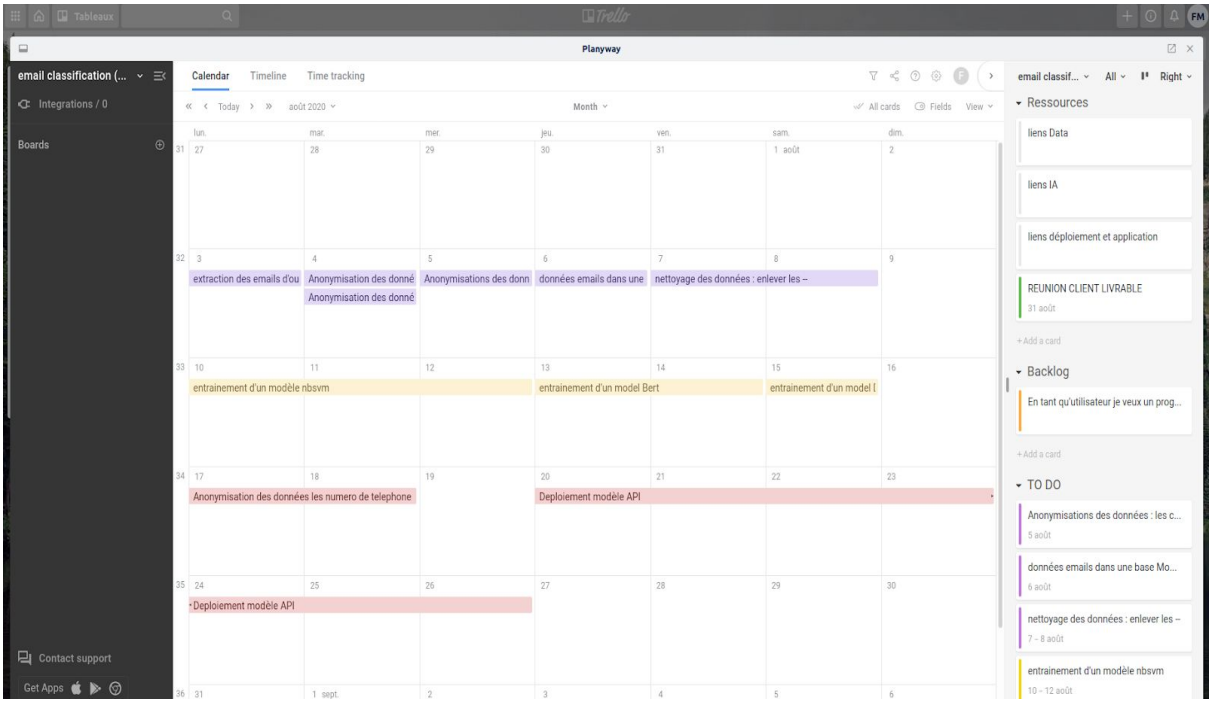


fig 16 : Planning du sprint 1

1.4 Le suivi des tâches : le burndown chart

Pour chaque sprint nous avons suivi l'avancement du travail afin de s'assurer que la livraison se fera dans les temps avec le client. Nous avons ainsi élaboré des tableaux qui nous permettaient de suivre l'avancement du sprint au jour le jour afin de pouvoir à tout moment nous adapter.

Pour cela nous avons choisi le burndown chart qui est un graphique d'avancement du sprint emprunté à la méthodologie agile. Ce graphique fut très pratique pendant le projet car il permettait à toute l'équipe de savoir où nous en étions globalement y compris le client chargé du projet. Cela a permis de mettre l'accent sur la performance collective de l'équipe.

Voici le Burndown Chart pour le sprint 1 vous trouverez les Burndown chart des autres sprints en annexe 10 gestion de projet .

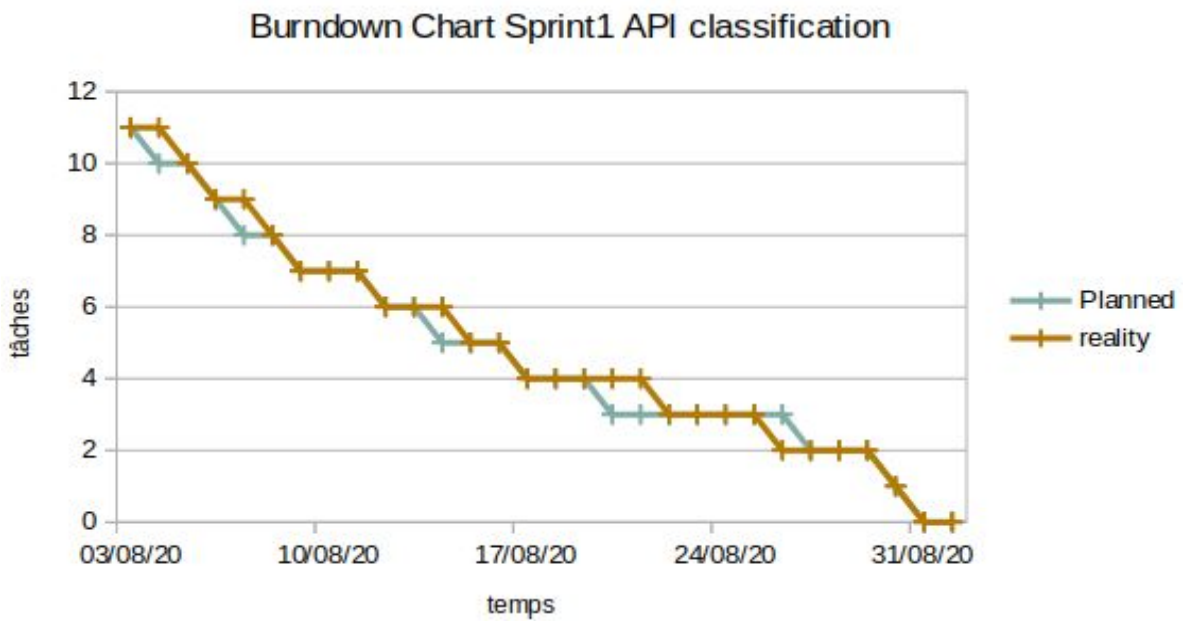


fig 17: Burndown Chart Sprint 1

1.5 La communication avec le client et rapports d'avancement

Afin de tenir au courant toutes les parties prenantes du projet, nous transmettons un compte rendu d'avancement assez régulièrement . Ainsi, comme l'ensemble des parties prenantes étaient au courant de l'avancée du projet. Cela nous a évité de recevoir des demandes à répétitions et nous a permis de rester concentré sur notre travail de fond.
En cas de retard ce projet d'avancement permettait au client de prendre connaissance de la situation et de savoir ce que nous faisions pour y remédier.

Cela a permis également d'instaurer une transparence et une relation de confiance avec le client qui pouvait nous poser des questions suite à cela.

Vous trouverez dans l'annexe 10 du projet **trois états d'avancement** que nous avons envoyés au client à trois moments différents du projet, ainsi que les échanges avec le client qui se sont suivis.

2. Retour d’expérience

Notre choix s'est tourné vers l’utilisation d’outils open source afin de limiter les dépenses et la dépendance vis-à-vis des solutions logicielles du marché.
Ces solutions se sont avérées très performantes et disposent d’une communauté importante en ligne qui permet de poser des questions et de disposer d’un support.

Parmi ces **outils** là je citerai notamment :

- Ktrain qui est une librairie pour l'entraînement de modèles. Cette librairie nous a permis d'entraîner plusieurs modèles très rapidement afin de choisir le meilleur.
  La personne qui a développé cette librairie est vraiment très disponible et répond très rapidement aux questions. Cette librairie dispose d’une communauté assez active avec beaucoup de sujets de discussions sur Github.

- Trello outil de gestion de projet, cet outil est vraiment très performant avec la possibilité d'intégrer des plug in comme le planning par exemple ou un burndown chart.
  Nous avons apprécié le fait que l’outil soit collaboratif et qu’il soit très intuitif et facile à utiliser. Nous avons pu utiliser avec le client un même outil que nous maîtrisons tous.

- Flask, un framework de developpement web en python. Nous avons aimé le fait que ce framework soit léger et flexible et aussi la facilité d’apprentissage de ce framework. Le coût d’entrée en termes d’apprentissage est assez petit contrairement au framework Django qui lui est plus long à prendre en main.

Concernant les **techniques appliquées**, nous avons retenu la méthode de gestion de projet agile qui a facilité la mise en œuvre du projet tout en impliquant le client. Cette méthode itérative a permis au client de faire évoluer son besoin au fur et à mesure des résultats obtenus.

Nous avons rencontré quelques difficultés dans l'installation d’environnements virtuels avec des problèmes de dépendances et de versions des librairies.
Ce problème aurait pu être évité par l’utilisation dès le départ de conteneurs avec **Docker** qui permettent d’installer un environnement avec les bonnes versions de librairies sur n’importe quel appareil.

La **compétence** qui a été particulièrement importante dans le développement de ce projet était tout d’abord savoir écouter notre client. En effet, il a fallu décrypter son besoin, poser plus de questions afin de comprendre exactement les enjeux du projet pour lui et son équipe.
La **compétence technique** qui était assez importante et que nous avons pu acquérir était le traitement du langage naturel en machine learning et en deep learning: comprendre les

différentes architectures qui permettaient de traiter des données textuelles ainsi que les techniques de nettoyage et d'anonymisation de texte était une vrai plus value dans le projet.

3. Organisation technique et environnement de développement

Pour ce projet nous avons utilisé plusieurs environnements de développement.
Nous avons utilisé tout d'abord des notebooks pour la partie développement de l'intelligence artificielle. Les notebooks permettent d'afficher des tableaux et des visualisations de données très facilement et de manière aisée.
Nous avons utilisé des notebooks sur google collab entre autres afin de pouvoir profiter des GPU disponibles dessus pour entraîner nos modèles.
Ensuite pour le développement de l'application ainsi que le développement de l'api, nous avons utilisé Visual Studio code qui est notre éditeur de code préféré. Nous aimons cet éditeur de code pour sa facilité d'installation, sa facilité d'utilisation. VS code dispose d'une interface graphique responsive et customisable via des thèmes déjà installés. Nous apprécions tout particulièrement la fonction IntelliSense qui assiste au développement en simplifiant l'édition du code, l'appel des propriétés, ou encore l'importation des dépendances.

Partie 3. Bilan du projet

Nous avons pu avec ce projet répondre au besoin du client qui était de classer ses emails dans la bonne catégorie. Nous avons eu des échanges en continu avec le client et cela nous a permis de répondre de manière précise à son besoin et même de lui proposer des fonctionnalités qu'il n'avait pas envisagées mais que nous pensions utiles pour son besoin. Cette collaboration avec le client est à notre avis une des raisons du succès de ce projet.

Par ailleurs, une bonne préparation en amont des outils, des technologies et de la recherche de l'état de l'art dans le NLP et la classification de texte furent une clé également de la réussite du projet.

Enfin la mise à disposition de données intéressantes et pertinentes pour répondre au besoin du projet ont grandement contribué à la réalisation de cette intelligence artificielle.

Le projet ne s'arrête pas pour autant.

Maintenant il faut tester au maximum le modèle avec l'application. Nous avons pour objectif de classer 400 emails (100 mails par utilisateur testeurs) afin de mesurer la performance de notre modèle dans la réalité.
Une fois cela fait nous pourrons réentrainer notre modèle avec les nouveaux emails et mesurer à nouveau la performance du modèle.

Une autre étape sera de développer une extension sur google chrome qui pourrait classer des emails directement dans gmail grâce à l'API d'intelligence artificielle.

Conclusion

Le projet “Email Classifier” que nous avons mené durant 3 mois et qui vise à classifier automatiquement des emails a permis d’obtenir des résultats significatifs. Notre modèle a obtenu un taux de 85 % d’accuracy suite à son entraînement.

Cette solution aidera les utilisateurs à se concentrer sur les emails les plus importants et donc de travailler plus efficacement et ainsi améliorer leur productivité.

Tout au long de ce projet, j’ai réalisé l’importance de consacrer le temps nécessaire à la préparation. Cela comprend le choix des techniques, des outils, de la méthodologie ainsi que la sélection de données pertinentes pour l'entraînement du modèle.
Autre point fondamental : la compréhension du besoin client. Il est indispensable en prérequis à tout développement technique de s’assurer que le besoin est clairement exprimé et compris par toutes les parties prenantes du projet. Le choix de la méthodologie agile permet néanmoins des ajustements pour s’adapter aux évolutions du besoin.

La réalisation du projet m’a beaucoup appris, j’ai pour la première fois mené un projet de bout en bout : du recueil des besoins du client au développement d’une application en passant par l'entraînement d’un réseau de neurones en deep learning.

J’ai néanmoins réalisé que si j’avais beaucoup appris, il me restait encore beaucoup à apprendre : j’ai été notamment sensibilisé durant ce projet à la protection des données sensibles, et aux différentes règles en vigueur en Europe sur ce sujet.
C’est un sujet que je souhaiterai approfondir dans l’avenir et qui me semble d’une importance majeure pour tous les développeurs qui traitent les données personnelles.

Annexe 1 Cahier des charges fonctionnel

Présentation de l'entreprise :

L'entreprise est Readchain une entreprise qui développe un moyen de communication et de transaction inter-entreprises en utilisant la blockchain.

Ce moyen permet aux entreprises de sécuriser les paiements avec des entreprises clientes et fournisseurs et de pouvoir disposer de garanties de paiements et d'assurance.

Suite à un contexte exceptionnel lié à la crise de la Covid19, beaucoup d'entreprises dans le monde ont dû gérer des retards de paiements ainsi que beaucoup d'impayés qui ont mis ces entreprises dans une situation critique avec de très gros problèmes de trésorerie.
Ainsi la solution que propose ReadChain répond à un problème que rencontre de plus en plus les entreprises ce qui va changer les futurs usages des organisations en matière de paiements.

L'entreprise compte 50 salariés et dispose d'une équipe de 10 commerciaux et business developers. Cette équipe connaît une explosion de son activité et doit se focaliser sur les demandes les plus importantes. Les commerciaux reçoivent une grande quantité d'emails par jour pouvant aller jusqu'à la centaine d'emails par jour.

Le client Bruno L qui est le chef de cette équipe de commerciaux souhaite disposer d'un d'un outil qui pourrait comprendre un email entrant et le classer dans une catégorie ce qui permettrait aux commerciaux de gagner du temps en traitant les emails qui les intéressent le plus en premier

Le besoin client :

Notre client souhaite que nous développions une Intelligence artificielle qui pourra être intégrée dans différents types d'outils. Cette intelligence artificielle pourrait être intégrée à une application mais aussi à une boite email, ou autre(assistant vocal ect..)

Objectifs du projet:

1) développer une API d'intelligence artificielle
2) développer une application de test pour tester notre modèle dans la réalité et permettre d'avoir des statistiques sur la performance du modèle.

Objectifs du livrable 1:

API de classification d'email:

Une api qui classe des emails dans des catégories prédéfinies par le client

Une api intégrable dans une application .

Objectif du livrable n°2 :

Application web qui interagit avec l'api développée et qui permet de tester cette API.

Les cibles du projets:

Le profil des clients :

Nos clients sont des commerciaux ou des business developper qui sont submergés de demandes de prospects. Ils doivent pouvoir prioriser les demandes.

Le profil type de nos clients est : un homme 38 ans, commercial qui doit engendrer beaucoup de ventes.

Les fonctionnalités et les modules :

Livrable 1 API : L'API est une intelligence artificielle qui comprend le texte des emails et qui les classe dans la catégorie adéquate.

Cette API doit être intégrable dans différentes applications y compris une application web.

Livrable 2 application WEB :

Un module Utilisateur:

- Utilisateur doit se logger à son compte pour accéder au site
- Utilisateur accède à une page pour classer ses emails
- L'utilisateur voit la prédiction du modèle et peut choisir s'il est d'accord avec la prédiction.
- S'il n'est pas d'accord avec la prédiction, il peut changer de catégorie

Un module Administrateur:

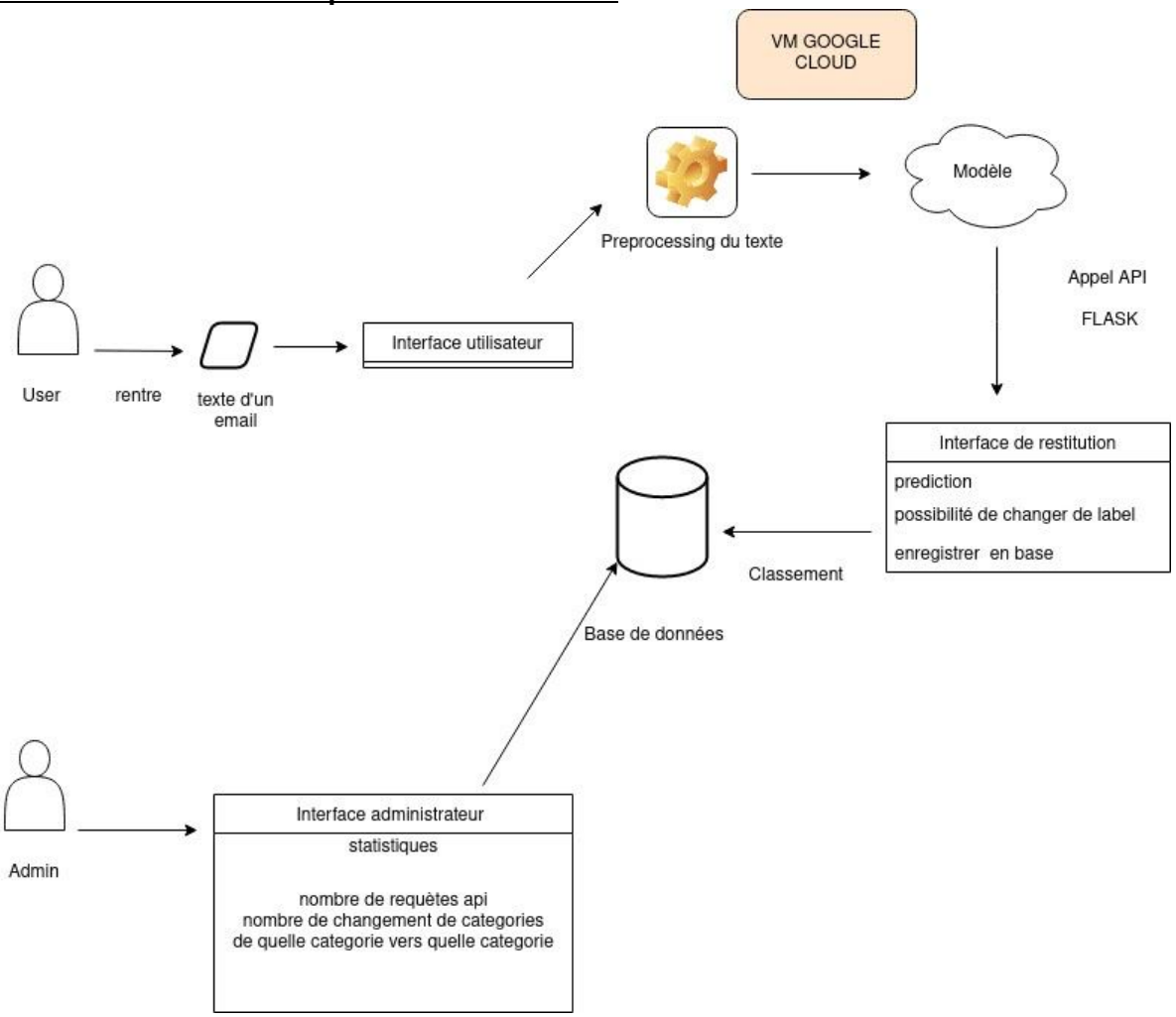
- Administrateur doit se logger à son compte pour accéder à la page administrateur
- Administrateur accède à une page de statistiques
- Administrateur voit le taux de réussite du modèle : quand le modèle a bien classé les emails
- Administrateur voit les emails mal classés et voit dans quelle catégorie le modèle l'a classé et dans quelle catégorie l'utilisateur l'a classé.

Les contraintes :

Le modèle d'intelligence artificielle doit être dissocié de l'application car il doit dans le futur pouvoir être installé dans un autre outil.

Le budget est un budget minimal.

Le schéma fonctionnel et parcours utilisateur :



Parcours utilisateur final :

L'utilisateur entre sur le site et se connecte à son compte.
Ensuite il peut accéder à une page ou classer son email via une interface.Il entre son email et l'application lui renvoie une prédiction. S'il est d'accord avec la prédiction, alors il confirme la prédiction du modèle sinon il peut changer de catégorie.

Parcours Administrateur :

L'administrateur entre dans l'application et accède à un tableau via une interface avec les statistiques du modèle.

L’interface de l’application :

L’interface de l’application doit avoir un styte épuré, son design doit rappeler la simplicité et l’organisation.
L'application est destinée à des professionnels elle doit donc par ses couleurs rappeler le côté professionnel et l’efficacité dans le travail.

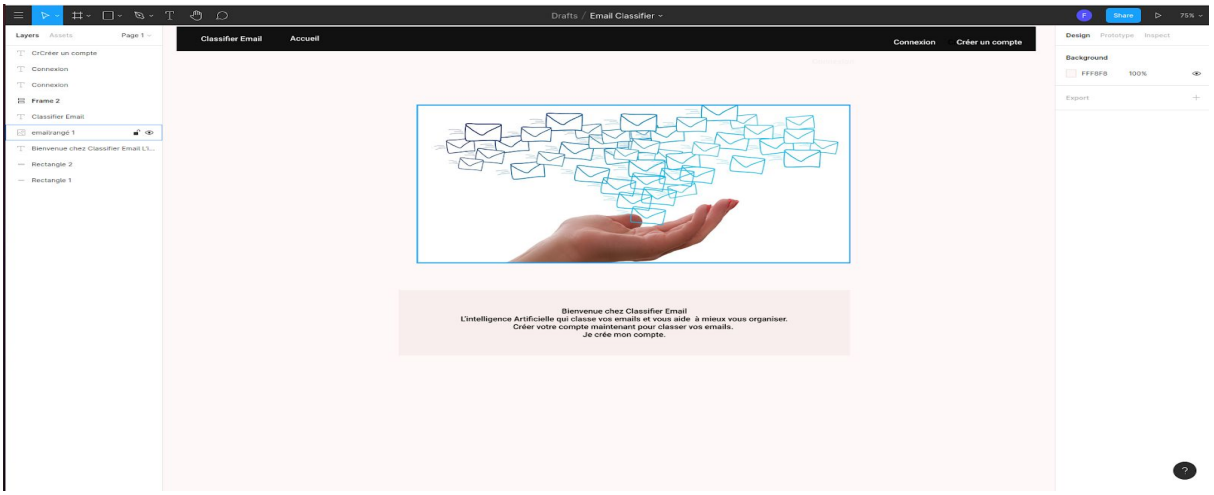
Les couleurs: Les couleurs doivent être donc épurés et simples nous seront dans du blanc du noir du bleu ces couleurs qui évoquent la simplicité, l’éfficacité et l’organisation.

Images : plusieurs images ont été sélectionnées mais il y a une image qui ressort du lot.

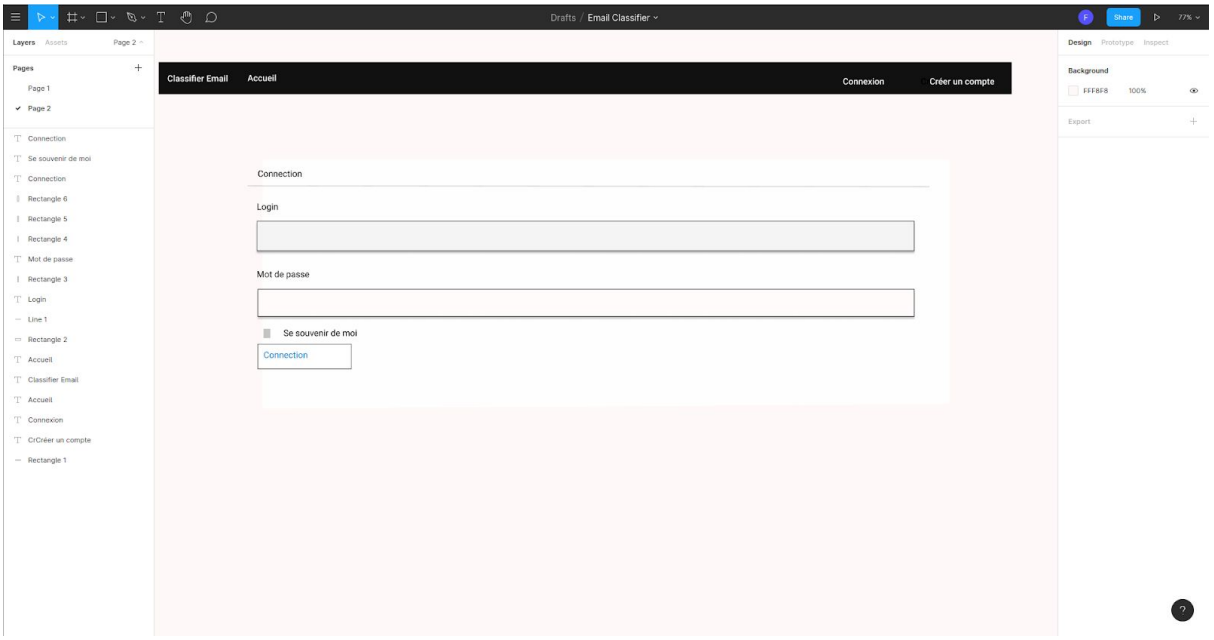
texte : Un texte présentant l’application devra être en dessus écrit en noir avec un fond blanc.

Maquettes du site internet :

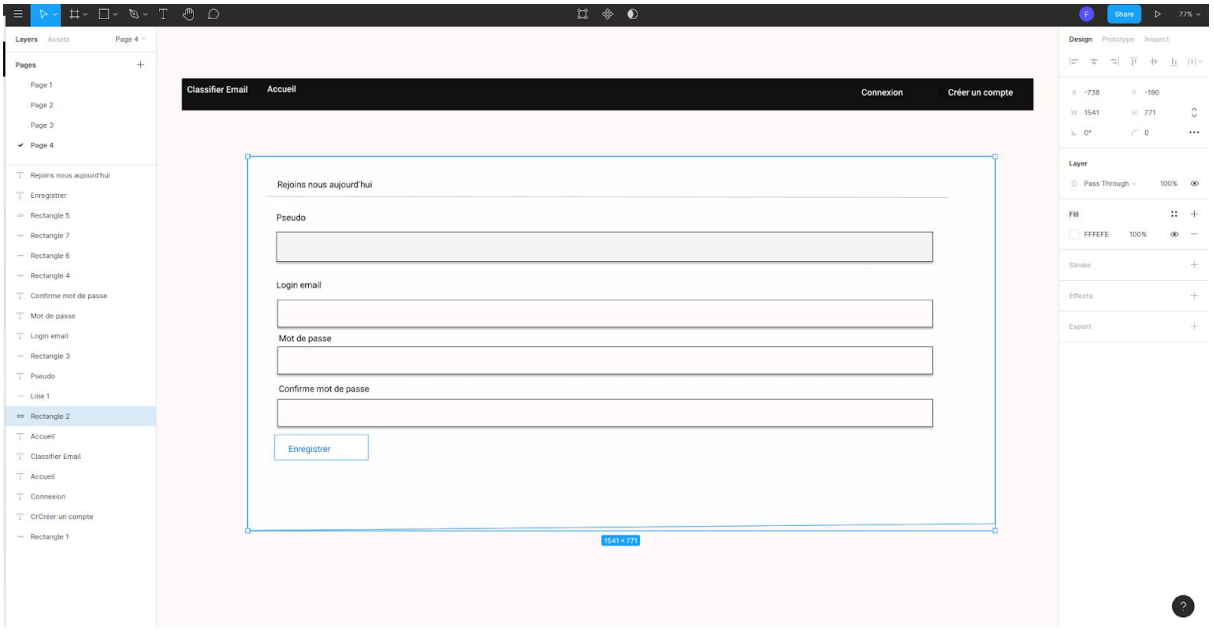
page d’accueil :



Page de connexion



La page d’enregistrement



Annexe 2 code du nettoyage et anonymisation des données

Nettoyage des caractère de contrôle

```
#enlever les control characters
df_online_cleaned['Corps']   =   [re.sub(r'\n','',str(x))   for   x   in
df_online_cleaned['Corps']]
df_online_cleaned['Corps']   =   [re.sub(r'\t','',str(x))   for   x   in
df_online_cleaned['Corps']]
df_online_cleaned['Corps']   =   [re.sub(r'[_]','   ',str(x))   for   x   in
df_online_cleaned['Corps']]
df_online_cleaned['Corps']   =   [re.sub(r'[-]','   ',str(x))   for   x   in
df_online_cleaned['Corps']]
```

Anonymisation des colonnes adresses et noms des expéditeurs et destinataires.

```
def anonymize_rows(rows):

    # Load faker
    faker  = Factory.create()
    # Create mappings
    names = defaultdict(faker.name)
    emails = defaultdict(faker.email)

    # Iterate over the rows from the file and yield anonymized rows.
    for row in rows:
        #replace with faked fields
        row['Denom']= names[row['Denom']]
        row['Deadresse'] = emails[row['Deadresse']]
        row['Anom']= names[row['Anom']]
        row['Aadresse'] =  emails[row['Aadresse']]
        #row['Ccnom']= emails[row['Ccnom']]

        yield row
```

```
#build the function that takes our source file and outputs our target file

def anonymize(source,target):
    with open(source, 'rU') as f:
        with open(target, 'w') as o:
            reader = csv.DictReader(f)
            writer = csv.DictWriter(o, reader.fieldnames)

            #pour avoir les headers
            writer.writeheader()
            for row in anonymize_rows(reader):
                writer.writerow(row)
```

Anonymisation du corps des emails :

```
def redact_names(text):
    doc = nlp(text)
    redacted_sentence = []
    for ent in doc.ents:
        ent.merge()
    for token in doc:
        if token.ent_type_ == "PER":
            redacted_sentence.append("[REDACTED]")
        else:
            redacted_sentence.append(token.string)
    return "".join(redacted_sentence)
```

```
def redact_mail(text):
    doc = nlp(text)
    redacted_sentence = []
    for token in doc:
        if token.like_email == True:
            redacted_sentence.append("[emailanonyme]")
        else:
            redacted_sentence.append(token.string)
    return "".join(redacted_sentence)
```

Annexe 3 Code base de données analytique

Insertion des données via un script dans la base de donnée:

Script d'insertion :

```
#inserting a csv on database
import csv
f = open("data_anonyme/product_anonyme_true5.csv", encoding = 'utf8')
reader = csv.DictReader(f)#renvoi quelque chose qui ressemble à un dictionnaire
header = reader.fieldnames
database = client.emails_database
collection = database.readsoft_collection
for i in reader:
    row = {}
    for c in header:
        row[c] = i[c]
    collection.insert_one(row)
f.close()
```

requête des emails par label

```
client = MongoClient('localhost:27017')
database = client.emails_database
collection = database.readsoft_collection
query = {"label": "fournisseur"}
doc = collection.find_one(query)
```

Fonctions d'interaction avec la base de données

Insérer des données :

```
#insert data :
def insert_bdd( db_uri,db_name,db_collection ,data):
      client = MongoClient(db_uri)
      database = client[db_name]
      collection = database[db_collection]
      collection.insert_one(data
return
```

Supprimer des données :

```
#remove data : supprime toutes les données pour une collection
def remove_data(db_uri, db_name, db_collection):
      client = MongoClient(db_uri)
      database = client[db_name]
      collection = database[db_collection]
      remove= collection.delete_many({})
      return remove
```

Mettre à jour des données :

```
#update data : update toute ou une partie
def update_data(db_uri, db_name, db_collection,db_query,new_data):
      client = MongoClient(db_uri)
      database = client[db_name]
      collection = database[db_collection]
      update = collection.update_one(db_query,new_data,upsert = False)
      return update
```

Récupérer une collection

```
#get collection with database client
def get_collection(db_client,db_name,db_collection):
      database = db_client[db_name]
      collection = database[db_collection]
      return collection
```

Annexe 4 implémentation de l'entraînement du modèle

Import des librairies

```
#import tensorflow
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
import ktrain
# module for text classification
from ktrain import text
```

Préparation des données

```
y = df.label
X = df.drop(['Unnamed: 0','Ccnom','Ccadresse','CcType','label'], axis = 1)
```

Plusieurs modèles disponibles :

```
#print some available models

text.print_text_classifiers()
```

```
fasttext: a fastText-like model [http://arxiv.org/pdf/1607.01759.pdf]
logreg: logistic regression using a trainable Embedding layer
nbsvm: NBSVM model [http://www.aclweb.org/anthology/P12-2018]
bigru: Bidirectional GRU with pretrained fasttext word vectors
[https://fasttext.cc/docs/en/crawl-vectors.html]
standard_gru: simple 2-layer GRU with randomly initialized embeddings
bert: Bidirectional Encoder Representations from Transformers (BERT)
[https://arxiv.org/abs/1810.04805]
distilbert: distilled, smaller, and faster BERT from Hugging Face
[https://arxiv.org/abs/1910.01108]
```

Le prétraitement des données se fait automatiquement par ktrain.

```
# Load and Preprocess the Data
#first, we use the texts_from_df function to load and preprocess the
data in to arrays that can be directly fed into a neural network model.
#function return preprocessor instance (preproc variable): normalize raw
data when making predictions on new examples.
(X_train,y_train),(X_test,y_test), preproc = text.texts_from_df(df,
                                                                'Corps',

label_columns = ['label'],

maxlen=75,

max_features=100000,

preprocess_mode='bert')
```

On utilise un modèle pré entraîné : ici un modèle BERT

```
#defining a Model
#We use a precanned model we use a 'bert' model
model = text.text_classifier('bert',(X_train, y_train) , preproc=preproc)
```

On peut voir l'architecture de notre model

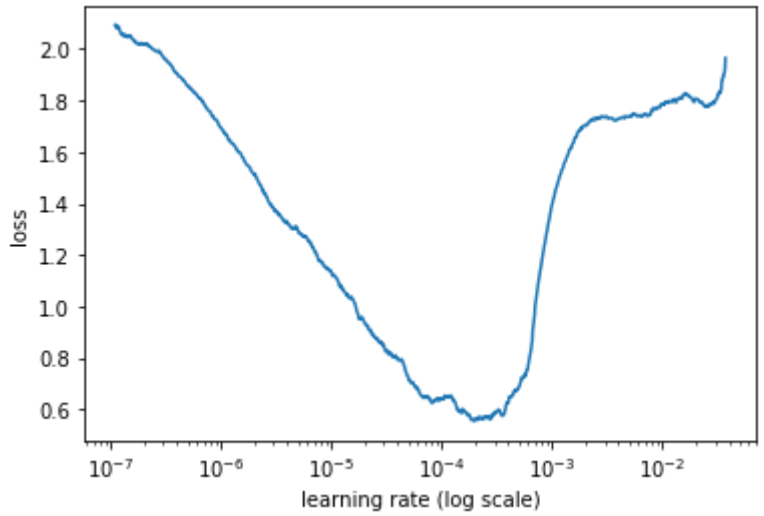
```
#print a summary of the network
model.summary()
```

On instancie un objet learner dans lequel il y a notre model et nos données

```
#wrapping Your Data and Model in a Learner Object
#permet de nous aider à entraîner notre model de plusieurs manières
learner    =    ktrain.get_learner(model,train_data    =(X_train,    y_train),
val_data=(X_test, y_test),batch_size= 32)
```

On cherche le meilleur learning rate

```
#find the best learning rate
learner.lr_find()

learner.lr_plot()
```



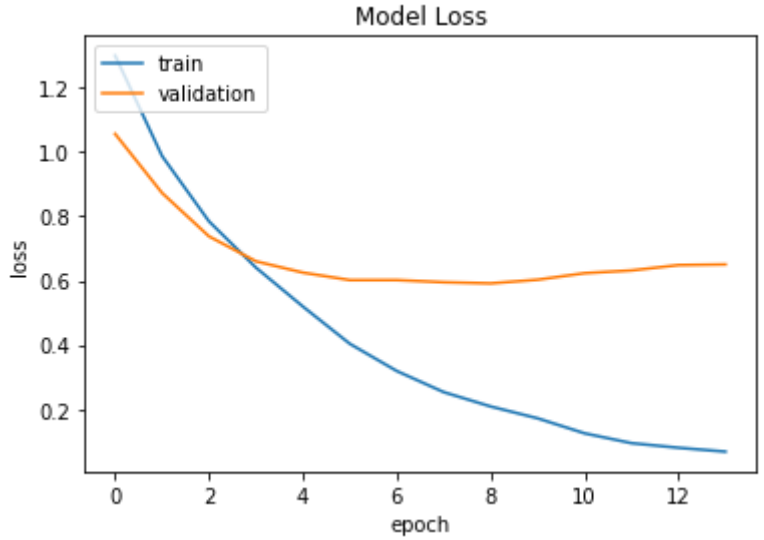
On entraîne le modèle en utilisant la méthode autofit() qui permet de laisser le learning rate osciller entre deux tranches de valeur.

```
learner.autofit(0.00001)
```

Entrainement :

On regarde la *loss* de notre entraînement afin de choisir les poids du meilleur modèle .

```
learner.plot('loss')
```



Le modèle enregistre les poids du meilleur modèle automatiquement.

Résultats du modèle et matrice de confusion :

```
learner.validate(class_names=preproc.get_classes())
```

```
                precision    recall  f1-score   support

     businessinfra       0.67      0.83      0.74        69
     communication       0.90      0.92      0.91        87
              emea       0.84      0.76      0.80        84
fournisseur/expert       0.77      0.71      0.74        58
            online       0.78      0.72      0.75        65

          accuracy                           0.80       363
         macro avg       0.79      0.79      0.79       363
      weighted avg       0.80      0.80      0.80       363
```

```
array([[57,  2,  1,  6,  3],
       [ 3, 80,  1,  1,  2],
       [12,  2, 64,  0,  6],
       [ 9,  3,  3, 41,  2],
       [ 4,  2,  7,  5, 47]])
```

Voir les emails pour lequel le modèle s'est le plus trompé :

```
learner.view_top_losses(n=1,preproc=preproc)
```

```
id:4 | loss:5.61 | true:online | pred:emea)

[CLS] fully agree : ) [ red ##act ##ed ] | managing director | reads
##oft uk switch ##board
```

faire une nouvelle prédiction

```
predictor = ktrain.get_predictor(learner.model, preproc)
```

```
data = ['the online api doesnot work correctly ']
predictor.predict(data)
['online']
predictor.get_classes()
['businessinfra', 'communication', 'emea', 'fournisseur/expert', 'online']
predictor.predict(data, return_proba=True)
array([[0.2658987 , 0.08177441, 0.22430994, 0.01971096, 0.40830603]],
      dtype=float32)
```

Enregistrement du modèle

```
predictor.save('/content/drive/My Drive/notebooks_chefdoeuvre_deep/model_corps_mail_bert')
```

Annexe 5 code API du modèle de Classification d'emails

Code API en Flask :

```
#import the necessary packages
import ktrain
from flask import Flask, request,jsonify
import json


# initialize our Flask application and the Ktrain model
app = Flask(__name__)
predictor = None

#we load our trained ktrainmodel
def load_predictor():
global predictor
predictor = ktrain.load_predictor("model_corps_mail_bert")
if hasattr(predictor.model, '_make_predict_function'):
predictor.model._make_predict_function()
return predictor

@app.route('/predict',methods=['POST'])
def predict():
#fonction user : prédiction d'un texte contenu dans le body
# return : une prédiction

data = request.get_json(force=True)
print(data)
print(type(data))
#initialize a data dictionary that will be returned from the view
result = {"status": "No_prediction"}
#récupération du text
if 'text'in data:
prediction = predictor.predict(data['text'])
result["prediction"] = prediction
result["status"] = "prediction_done"
else:
result["status"] = "no_text"
return result
if __name__ == "__main__":
print(("* Loading Keras model and Flask starting server..."
"please wait until server has fully started"))
```

```
load_predictor()
port = 8008
app.run(host='0.0.0.0', port=port)
```

Docker file:

```
FROM python:3.6-slim-stretch

RUN apt update
RUN apt install -y python3-dev gcc

ADD requirements.txt requirements.txt
ADD model_corps_mail_bert model_corps_mail_bert/
ADD api.py api.py

# Install required libraries
RUN pip install -r requirements.txt

# Run it once to trigger resnet download
#RUN python api.py

EXPOSE 8000
```

Annexe 6 backend de l'application

Le modèle de la base de données a été écrit dans un fichier **modeldatabase.py**

```
from flask_app import db, login_manager
from flask_login import UserMixin

#permet de logger les differents user
@login_manager.user_loader
def load_user(user_id):
return User.query.get(int(user_id))


class User(db.Model, UserMixin):
id = db.Column(db.Integer, primary_key=True)
pseudo = db.Column(db.String, nullable=False)
login_email = db.Column(db.String(20), unique= True, nullable= False)
mot_de_passe = db.Column(db.String(60), nullable = False)
#spécifier la relation entre les deux classes
email = db.relationship('Email', backref='sender', lazy=True)

#comment l'objet sera printé
def __repr__(self):
return f"User('{self.pseudo}','{self.login_email}')"#on ne montre pas le
mot de passe quand on voudra le représenter

class Email(db.Model):
id = db.Column(db.Integer,primary_key=True)
texte = db.Column(db.Text, nullable=False)

#foreign key
user_id = db.Column(db.Integer, db.ForeignKey('user.id'),nullable=False)
prediction_id= db.Column(db.Integer, db.ForeignKey('prediction.id'))
choix_utilisateur_id = db.Column(db.Integer,
db.ForeignKey('choix_utilisateur.id'))

#spécifier la relation entre les deux classes
prediction = db.relationship("Prediction", back_populates="email")
choix_utilisateur =
db.relationship("ChoixUtilisateur",back_populates="email")
#comment l'objet sera printé
def __repr__(self):
return
f"Email('{self.texte}','{self.prediction_id}','{self.choix_utilisateur_i
```

```
d}')"#on ne montre pas le texte car ça peut être très long

class Categorie(db.Model):
id = db.Column(db.Integer,primary_key=True)
label = db.Column(db.String(50), nullable=False)

#spécifier la relation entre les deux classes
prediction = db.relationship('Prediction')
choix_utilisateur = db.relationship('ChoixUtilisateur')

def __repr__(self):
return f"Categorie('{self.id}','{self.label}')" #afficher les categories
et leur libelle
class Prediction(db.Model):
id = db.Column(db.Integer,primary_key=True)
#foreign key
categorie_id = db.Column(db.Integer, db.ForeignKey('categorie.id'))

#spécifier la relation entre les deux classes
email =
db.relationship("Email",back_populates="prediction",uselist=False)

def __repr__(self):
return f"Prediction('{self.id}','{self.categorie_id}')" #afficher les
categories et leur libelle

class ChoixUtilisateur(db.Model):
id = db.Column(db.Integer,primary_key=True)
#foreign key
categorie_id = db.Column(db.Integer, db.ForeignKey('categorie.id'))
#spécifier la relation entre les deux classes
email =
db.relationship("Email",back_populates="choix_utilisateur",uselist =
False)

def __repr__(self):
return f"Choix Utilisateur('{self.id}','{self.categorie_id}')" #afficher
les categories et leur libelle
```

Le fichier form.py contient les formulaires de la base de données :

```
#import de flaskform pour faire un formulaire d'enregistrement
from flask_wtf import FlaskForm
#pour ajouter les champs pour les noms
from wtforms import StringField, PasswordField, SubmitField,
BooleanField, TextAreaField
#pour ajouter des validations aux champs requis
from wtforms.validators import DataRequired, Length, Email, EqualTo,
ValidationError
from flask_app.modelsdatabase import User

#formulaire d'enregistrement
class RegistrationForm(FlaskForm):
      pseudo = StringField('Pseudo',
      #obligation de mettre quelque chose et contrôle de la longueur
                           validators=[DataRequired(),
Length(min=2,max=20)])

      login_email = StringField('Login Email',
                        validators= [DataRequired(), Email()])

      mot_de_passe = PasswordField('Mot de passe', validators =
[DataRequired()])
      confirm_mot_de_passe= PasswordField('Confirme Mot de passe',
                                          validators=[DataRequired(),
EqualTo('mot_de_passe')])
      submit = SubmitField('Enregistrer')

      #on ajoute une validation error si jamais on se retrouve avec le
même pseudo

      def validate_pseudo(self, pseudo):
      user = User.query.filter_by(pseudo=pseudo.data).first()
      if user:
            raise ValidationError("Ce login existe déja. Merci de
choisir un nouveau login.")

      #on ajoute une validation error si jamais on se retrouve avec le
même pseudo
      def validate_login_email(self, login_email):
      user = User.query.filter_by(login_email=login_email.data).first()
      if user:
            raise ValidationError("Ce login_email existe déja. Merci
d'en choisir un nouveau.")
```

```
#formulaire pour se connecter
class LoginForm(FlaskForm):

      login_email = StringField('Login Email',
                        validators= [DataRequired(), Email()])

      mot_de_passe = PasswordField('Mot de passe', validators =
[DataRequired()])
      remember = BooleanField('Se souvenir de moi')
      submit = SubmitField('Connection')

#formulaire pour entrer un email
class PostForm(FlaskForm):
      texte = TextAreaField('Email', validators = [DataRequired()])
      submit = SubmitField('Classer')
```

Le fichier **classify.py** fait l'appel à l'api :

```
import requests, json

def classify(text_input):
      url = "http://23.251.133.90/predict"
      headers = {'Content-Type':'application/json'}
      body = {'text':text_input}
      response = requests.post(url,json=body, headers=headers)
      return response.json()
```

Le fichier **routes.py c**ontient les différentes routes de l'application.

```
from flask import render_template, flash, redirect, url_for, request
from flask_app import app, db , bcrypt
from flask_app.forms import RegistrationForm, LoginForm, PostForm
from flask_app.modelsdatabase import User, Email, Prediction,
ChoixUtilisateur, Categorie
from flask_login import login_user, current_user, logout_user,
login_required
from flask_app.classify import classify

@app.route('/')
@app.route('/home',methods=['POST','GET'])
def home():
      return render_template('home.html')

@app.route('/administrateur',methods=['POST', 'GET'])
```

```
def admin():
      if request.method == 'POST':

      emails = Email.query.all()
      predictions = Prediction.query.all()
      choix_utilisateur = ChoixUtilisateur.query.all()
      categories = Categorie.query.all()
      prediction_ok = 0
      data = []
      #recupère chaque element des tuples renvoyés par la base de donnée
      for email, prediction, choix_utilisateur in zip(emails,
predictions,choix_utilisateur):
            if prediction.id == choix_utilisateur.id and
prediction.categorie_id == choix_utilisateur.categorie_id:
                  prediction_ok +=1
      #recupère quand les predictions ne sont pas ok les élements que je
vais afficher pour l'administrateur
            else:
                  tmp = []
                  tmp.append(email.texte)
                  tmp.append(prediction.categorie_id)
                  tmp.append(choix_utilisateur.categorie_id)
                  data.append(tmp)

      taux = (prediction_ok/ db.session.query(Email.texte).count())*100
      nb_elements = db.session.query(Email.texte).count()
      print("taux de prediction correctes %s"%(taux))


      return render_template('administrateur.html', title= 'Page
Administrateur', data = data, taux = taux, nb_elements = nb_elements )
      else:
      return 'Erreur permission denied'

@app.route("/register", methods=['GET', 'POST'])
def register():
      #si on est déja authentifié on n'a pas accès à la page
enregistrement
      if current_user.is_authenticated:
      return redirect(url_for('home'))
      form = RegistrationForm()
      # flash message pour nous dire si c'est validé ou pas
      if form.validate_on_submit():
      #crypter le mot de passe
      hashed_password =
bcrypt.generate_password_hash(form.mot_de_passe.data).decode('utf-8')
```

```
      #on crée une variable user
      user = User(pseudo=form.pseudo.data,
login_email=form.login_email.data, mot_de_passe=hashed_password)
      #on l'enregistre dans la base de données
      db.session.add(user)
      db.session.commit()

      flash(f'Votre compte a été crée! Vous pouvez maintenant vous
connecter', 'success')
      return redirect(url_for('login'))
      return render_template('register.html', title='Register',
form=form)

@app.route("/login", methods=['GET','POST'])
def login():
      #si on est déja authentifié on n'a pas accès à la page connexion
      if current_user.is_authenticated:
      return redirect(url_for('home'))
      form = LoginForm()
      if form.validate_on_submit():
      user = User.query.filter_by(login_email=
form.login_email.data).first()
      if user and bcrypt.check_password_hash(user.mot_de_passe,
form.mot_de_passe.data):
            login_user(user, remember= form.remember.data)
            if user.login_email == 'admin@blog.com':
                  return redirect(url_for('admin'), code = 307)
            #nous permet d'aller vers la page account que lorsque nous
sommes connecté
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else
redirect(url_for('home'))
      else:
            flash('La connection a été refusé. Veuillez vérifier votre
email et mot de passe','danger')
      return render_template('login.html', title='Login', form=form)

@app.route("/logout")
def logout():
      logout_user()
      return redirect(url_for('home'))

@app.route("/account")
@login_required
def account():
      return render_template('account.html', title='Account')
```

```
#nouvelle route pour poster les emails(il faut être connecté)
@app.route('/email/new', methods = ['GET'])
@login_required
def classer_email():
      form = PostForm()
      if form.validate_on_submit():
      if request.method == "POST":
            texte = request.form['texte']
            print(texte)
            response = classify(texte)
            if response["status"] == "prediction_done":
                  #print('je vais dans le prédiction')
                  #enregistrer la prediction et le texte

                  prediction = Prediction(categorie_id =
response['prediction'])
                  db.session.add(prediction)
                  db.session.commit()
                  email = Email(texte =texte,prediction_id=
prediction.id, sender = current_user )
                  db.session.add(email)
                  db.session.commit()
                  return render_template('prediction.html',title=
'prediction', response=response, form='form')
      else :
            render_template('classer_email.html', title='Classer votre
email', form=form)
      return render_template('classer_email.html', title='Classer votre
email', form=form)

@app.route('/enregistrer_email', methods = ['GET','POST'])
def enregistrer_email():
      form = PostForm()
      if request.method == "POST":
      #enregistrer le choix
      choix_utilisateur =
ChoixUtilisateur(categorie_id=request.form['select_categorie'])
      db.session.add(choix_utilisateur)
      db.session.commit()
      email= db.session.query(Email).order_by(Email.id.desc()).first()
      email.choix_utilisateur= choix_utilisateur
      db.session.commit()
      flash('votre email a été classé','succes')
      return render_template('classer_email.html', title='Classer votre
email', form=form)
```

Annexe 7 code interaction base de données de l'application

Remplir la table Categorie correspondant aux différentes catégories où l'email pouvait être classé.

```
categorie1 = Categorie(label = "emea")
>>> db.session.add(categorie1)
>>> categorie2 = Categorie(label = "communication")
>>> db.session.add(categorie2)
>>> categorie3 = Categorie(label = "online")
>>> db.session.add(categorie3)
>>> categorie4 = Categorie(label="expertsystem/fournisseurs")
>>> db.session.add(categorie4)
>>> categorie5 = Categorie(label="businessinfra")
>>> db.session.add(categorie5)
>>> db.session.commit()
>>> Categorie.query.all()
[Categorie('1',emea'),Categorie('2',communication'),
Categorie('3',online'),Categorie('4',expertsystem/fournisseurs'),
Categorie('5',businessinfra')]
```

Enregistrement des informations de l'utilisateur dans la base de donnée dans la route register :

```
#on crée une variable user

      user = User(pseudo=form.pseudo.data,
login_email=form.login_email.data, mot_de_passe=hashed_password)
      #on l'enregistre dans la base de données
      db.session.add(user)
      db.session.commit()
```

S'assurer que l'utilisateur est bien enregistré dans la base de données dans la route login :

```
user = User.query.filter_by(login_email= form.login_email.data).first()
```

Enregistrer la prédiction du modèle dans la base de données.

```
prediction = Prediction(categorie_id = response['prediction'])
db.session.add(prediction)
db.session.commit()
```

Enregistrer le choix de l'utilisateur :

```
choix_utilisateur =
ChoixUtilisateur(categorie_id=request.form['select_categorie'])
db.session.add(choix_utilisateur)
db.session.commit()

email= db.session.query(Email).order_by(Email.id.desc()).first()
      email.choix_utilisateur= choix_utilisateur
      db.session.commit()
```

Requête pour récupérer les informations à afficher à l'administrateur

```
emails = Email.query.all()
predictions = Prediction.query.all()
choix_utilisateur = ChoixUtilisateur.query.all()
categories = Categorie.query.all()
```

Nous récupérons avec le code python les données qui nous intéressent à afficher :

```
prediction_ok = 0
data = []
#recupère chaque element des tuples renvoyés par la base de donnée
for email, prediction, choix_utilisateur in zip(emails,
predictions,choix_utilisateur):
if prediction.id == choix_utilisateur.id and prediction.categorie_id ==
choix_utilisateur.categorie_id:
prediction_ok +=1
#recupère quand les predictions ne sont pas ok les élements que je vais
afficher pour l'administrateur
else:
tmp = []
tmp.append(email.texte)
```

```
tmp.append(prediction.categorie_id)
tmp.append(choix_utilisateur.categorie_id)
data.append(tmp)

taux = (prediction_ok/ db.session.query(Email.texte).count())*100
nb_elements = db.session.query(Email.texte).count()
print("taux de prediction correctes %s"%(taux))
```

Annexe 8 les formulaires

```
#import de flaskform pour faire un formulaire d'enregistrement
from flask_wtf import FlaskForm
#pour ajouter les champs pour les noms
from wtforms import StringField, PasswordField, SubmitField,
BooleanField, TextAreaField
#pour ajouter des validations aux champs requis
from wtforms.validators import DataRequired, Length, Email, EqualTo,
ValidationError
from flask_app.modelsdatabase import User

#formulaire d'enregistrement
class RegistrationForm(FlaskForm):
      pseudo = StringField('Pseudo',
      #obligation de mettre quelque chose et contrôle de la longueur
                           validators=[DataRequired(),
Length(min=2,max=20)])

      login_email = StringField('Login Email',
                        validators= [DataRequired(), Email()])

      mot_de_passe = PasswordField('Mot de passe', validators =
[DataRequired()])
      confirm_mot_de_passe= PasswordField('Confirme Mot de passe',
                                          validators=[DataRequired(),
EqualTo('mot_de_passe')])
      submit = SubmitField('Enregistrer')

      #on ajoute une validation error si jamais on se retrouve avec le
même pseudo

      def validate_pseudo(self, pseudo):
      user = User.query.filter_by(pseudo=pseudo.data).first()
      if user:
            raise ValidationError("Ce login existe déja. Merci de
choisir un nouveau login.")

      #on ajoute une validation error si jamais on se retrouve avec le
même pseudo
      def validate_login_email(self, login_email):
      user = User.query.filter_by(login_email=login_email.data).first()
      if user:
            raise ValidationError("Ce login_email existe déja. Merci
```

```
d'en choisir un nouveau.")

#formulaire pour se connecter
class LoginForm(FlaskForm):

      login_email = StringField('Login Email',
                        validators= [DataRequired(), Email()])

      mot_de_passe = PasswordField('Mot de passe', validators =
[DataRequired()])
      remember = BooleanField('Se souvenir de moi')
      submit = SubmitField('Connection')

#formulaire pour entrer un email
class PostForm(FlaskForm):
      texte = TextAreaField('Email', validators = [DataRequired()])
      submit = SubmitField('Classer')
```

Annexe 9 les tests automatiques

Initialisation des tests

```
from flask_app import app, db, bcrypt
import pytest
from random import randint
import requests
from flask_app.modelsdatabase import User, Email, Prediction,
ChoixUtilisateur, Categorie

@pytest.fixture
def client():
      app.config["TESTING"]= True
      app.config['SECRET_KEY'] = '606c4261b169756bba33c5e36525d288'
      with app.test_client() as client:
      yield client
```

Tests routes

```
def test_home(client):
      url = "/home"
      res = client.get(url)
      assert res.status_code == 200

def test_register(client):
      url = "/register"
      res = client.get(url)
      assert res.status_code == 200

def test_logout(client):
      url = "/logout"
      res = client.get(url)
      assert res.status_code == 302

def test_login(client):
      url = "/login"
      res = client.get(url)
      assert res.status_code == 200

def test_classeremail(client):
      url = "/email/new"
      res = client.get(url)
```

```
      #assert res.status_code == 200
      assert res.status_code == 302
```

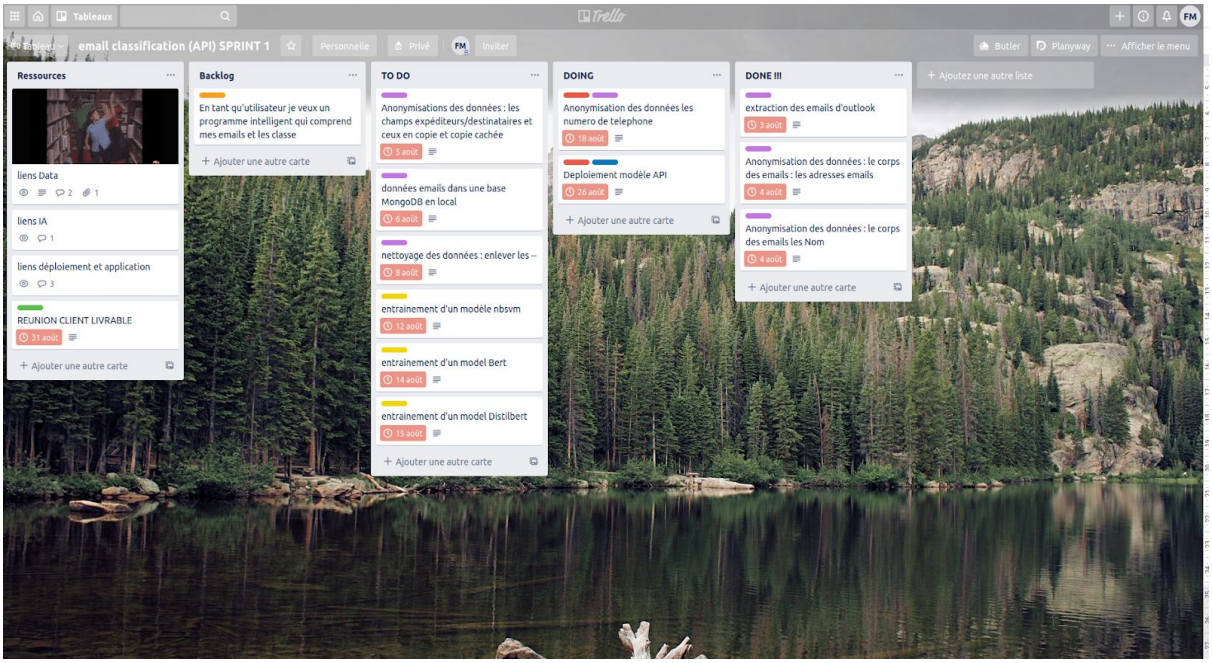
Tests connection API et base de données

```
#test de la connexion
def test_connexion():
      user = User.query.filter_by(login_email= 'admin@blog.com').first()
      assert bcrypt.check_password_hash(user.mot_de_passe, 'admin') ==
True


#test de la requête api
def test_requestapi():
      res = requests.post("http://23.251.133.90/predict",json =
{"text":"api ne fonctionne pas les clients sont furieux."} )
      response =
'{"prediction":"communication","status":"prediction_done"}'
      #enlever les caractères de control avec rstrip
      assert res.text.rstrip() == response

#test base de données
def test_insert_bdd():
      user_pseudo= 'test'+ str(randint(1,1000))
      user_test = User(pseudo = user_pseudo,login_email=
user_pseudo+'@test.com',mot_de_passe='python')
      db.session.add(user_test)
      db.session.commit()
      assert type(User.query.filter_by(login_email =
user_pseudo+'@test.com').first()) == type(User())
```

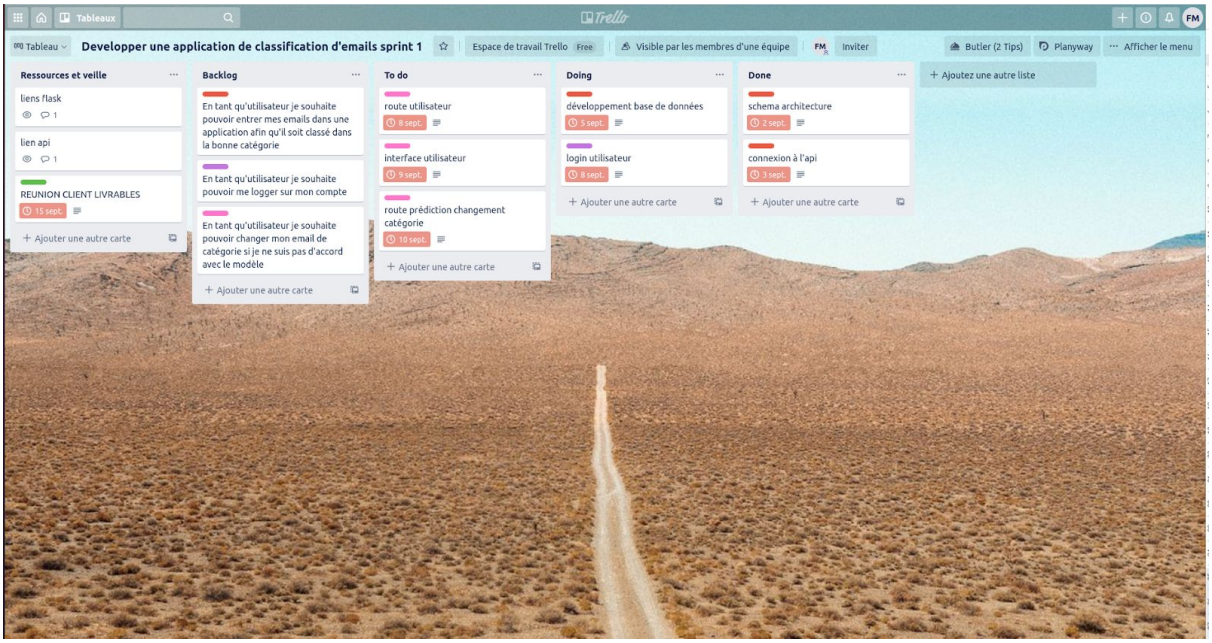
Annexe 10 la gestion de projet

Le tableau des sprints :

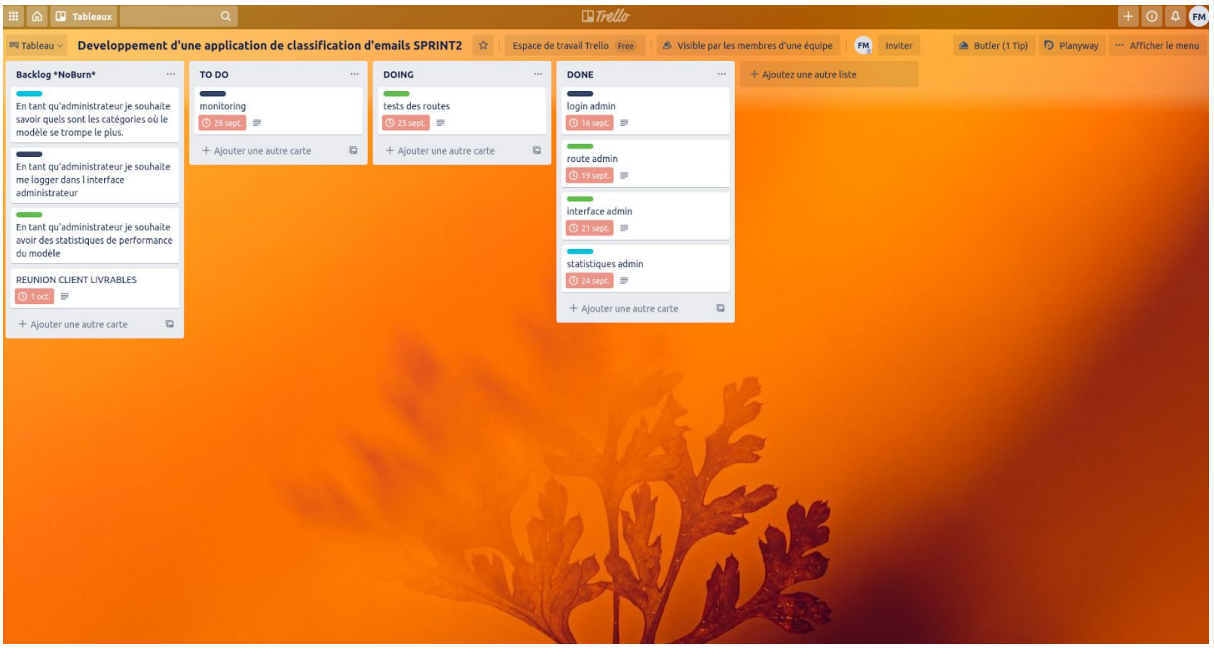
Sprint 1 Developpement de l'API de Classification d'emails.



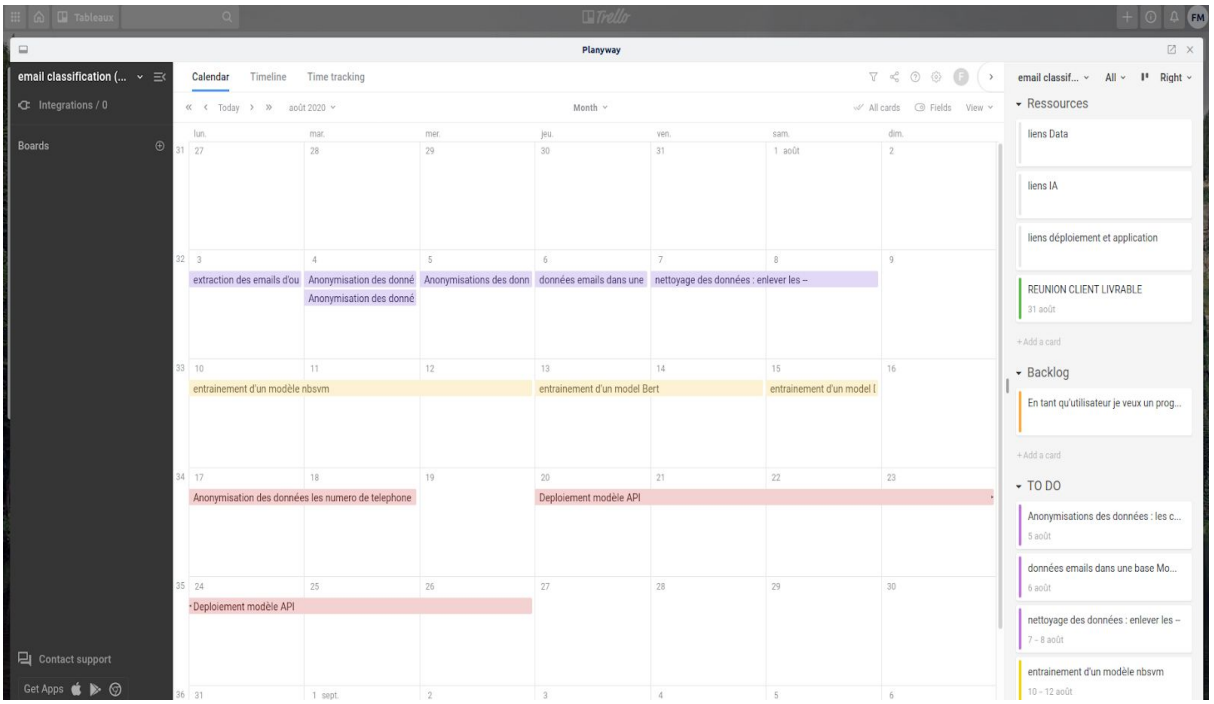
Sprint 2 Développer une application de classification d'emails 1ere partie



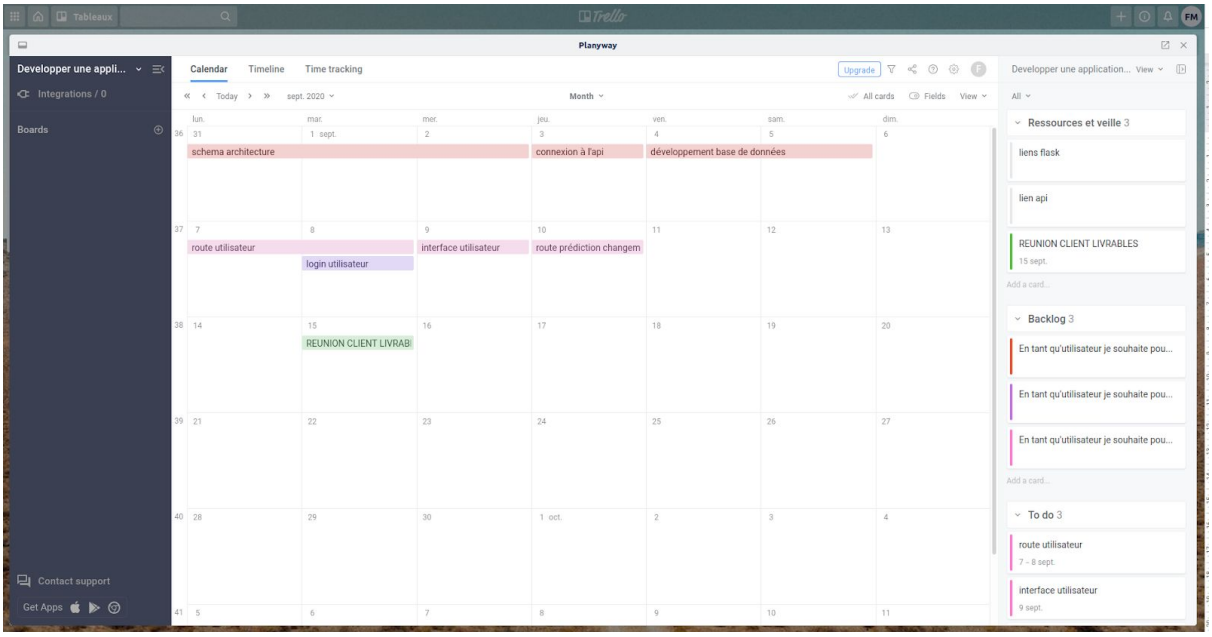
Sprint 3 Développer une application de classification d'emails 2eme partie.


Tableaux
Trello
Tableau
Developpement d'une application de classification d'emails SPRINT2
Espace de travail Trello Free
Visible par les membres d'une équipe
Inviter
Butler (1 Tip)
Planyway
Afficher le menu
Backlog *NoBurn*
En tant qu'administrateur je souhaite savoir quels sont les catégories où le modèle se trompe le plus.
En tant qu'administrateur je souhaite me logger dans l interface administrateur
En tant qu'administrateur je souhaite avoir des statistiques de performance du modèle
REUNION CLIENT LIVRABLES
1 oct.
Ajouter une autre carte
TO DO
monitoring
28 sept.
Ajouter une autre carte
DOING
tests des routes
25 sept.
Ajouter une autre carte
DONE
login admin
16 sept.
route admin
19 sept.
interface admin
21 sept.
statistiques admin
24 sept.
Ajouter une autre carte
Ajoutez une autre liste

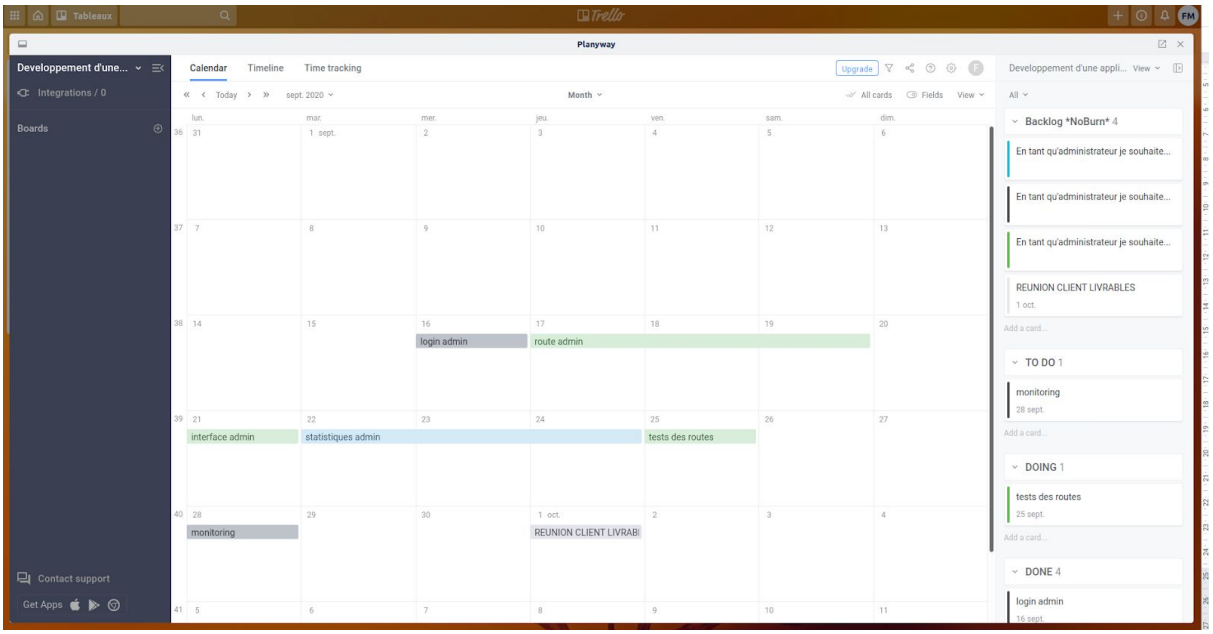
Planning du sprint 1 :



Planning Sprint 2:

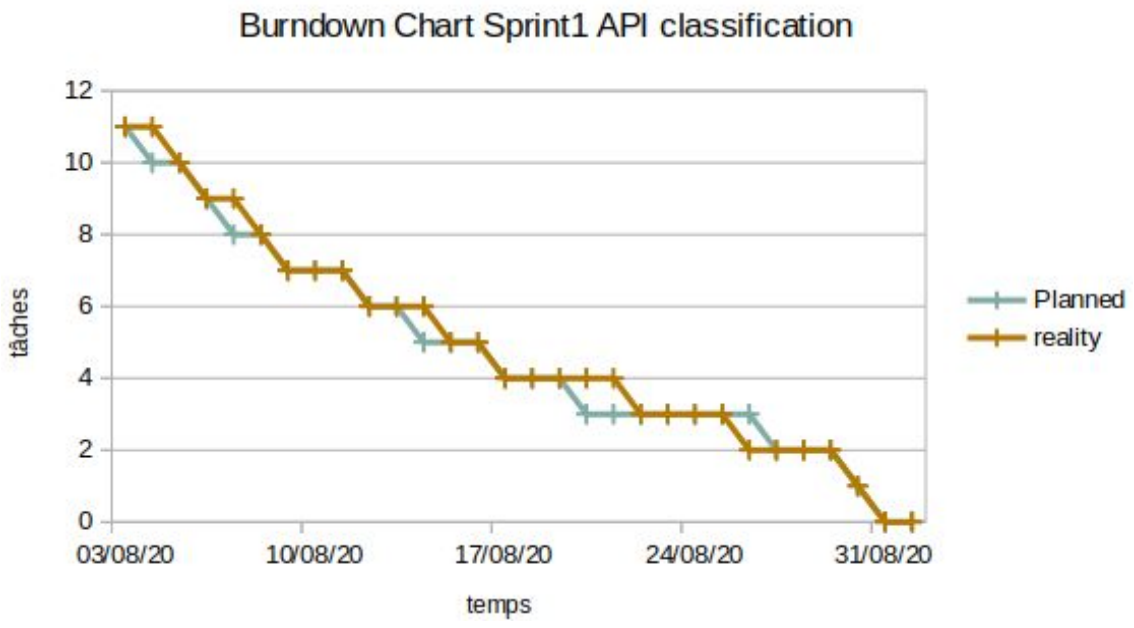


Planning Sprint 3:

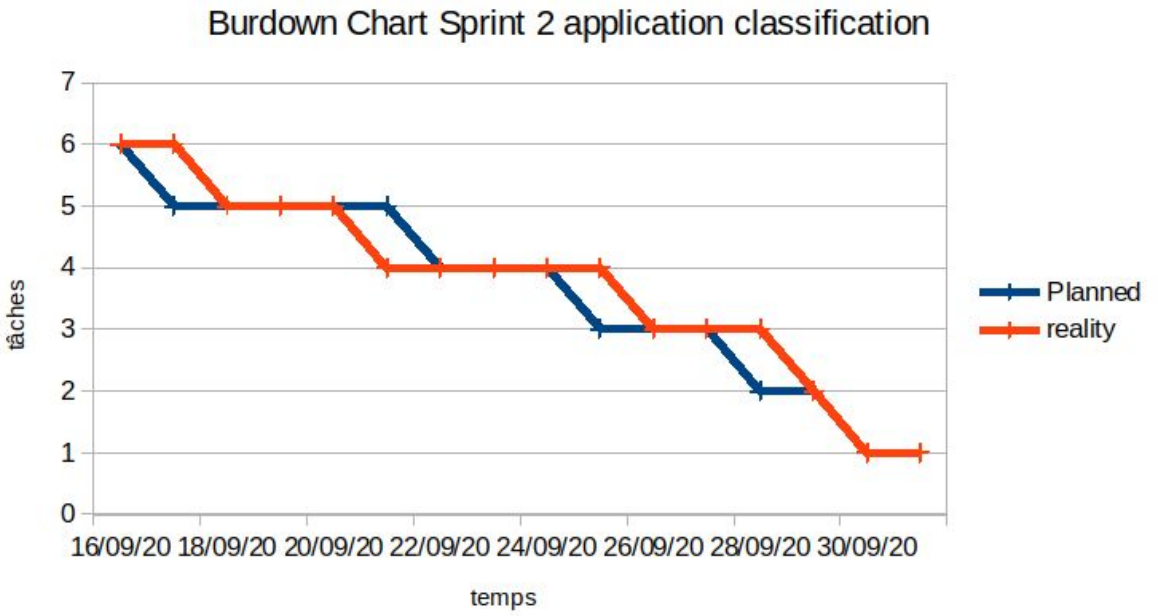


Les Burndown charts:

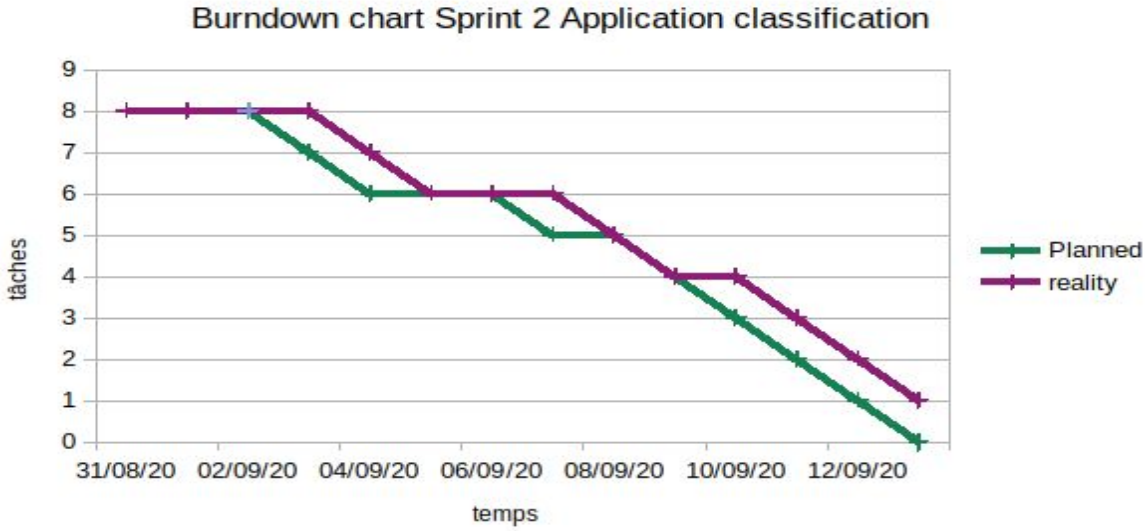
Sprint1



Sprint 2



Sprint 3



Les rapports d’avancements :

Etat d’avancement numero 1:

Nom du rapport :

Développement d' une application de classification d'emails 17/08/2020

statut du projet : dans les délais

Résumé:

Nous avons terminé d'entraîner nos modèles. Nous avons réussi à avoir un taux de réussite de 85% pour les prédictions du modèle.

L’équipe attaque le déploiement du modèle sous forme d’api.

Les modèles d’intelligence artificielle :

- L’équipe de développement a entraîné plusieurs modèles afin de choisir celui qui a le taux de réussite le plus grand.

- Les modèles performants sont des modèles assez lourds et le déploiement va nécessiter un peu de temps pour déterminer quelle est la meilleure manière de déployer le modèle.

Déploiement de l’api:

- L’équipe de developpement s’est réunie pour examiner les contraintes techniques liées au déploiement de l’api. Cela a fait prendre un léger retard au projet.

Remarques supplémentaires :

- Fatima L sera absente la semaine prochaine, vous pourrez adresser vos questions à Fatima M.

Problèmes/Défis:

Le délai pour rendre l’api est assez serré, il est donc important de continuer à travailler en équipe et à utiliser les outils de gestion projet pour être efficace.

Echange avec le client

Email du client Bruno L en charge du projet:

Bonjour Fatima M,

Je vous remercie pour ce rapport d'avancement. J'ai compris que le modèle d'intelligence artificielle était prêt mais qu'il y avait encore du travail pour le déploiement de l'api.

Qu'est ce que cela implique pour le projet et avez-vous des pistes pour le déploiement de l'api?

Cordialement,

Bruno L.

Réponse de Fatima M. développeuse en charge du projet :

Bonjour Bruno L,

Pour le déploiement de l'api nous avons eu un léger retard car notre modèle étant un modèle assez sophistiqué il était assez lourd. Nous n'avons pas pu le déployer dans une plateforme de service qui entrait dans notre budget.
Nous allons finalement opté pour le déploiement dans une Machine Virtuelle sur Google Cloud. Cette machine se comporte comme un ordinateur réel et entre dans notre budget.

Cela ne changera rien pour la suite du projet que ce soit en termes de coûts, de performance ou de budget.

Cordialement,

Fatima M.

Etat d'avancement numéro 2:

Nom du rapport : Développement d' une application de classification d'emails le 01/09/2020

statut du projet : dans les délais

Résumé: L'api a été développée et livrée au client. L'équipe commence un nouveau sprint de 15 jours sur le développement d'une application.
L'environnement de développement vient d'être mis en place.

Livraison de l'api :

- Suite à la réunion de fin de sprint durant laquelle il y a eu une démonstration de l'api, Bruno L , le client va tester l'API.
- Bruno L reviendra vers l'équipe de développement pour faire un retour sur ces tests.

Environnement de développement:

- L'environnement de développement et le Github de l'application est mis en place .

Remarques supplémentaires :

- Un grand merci à Bruno pour son implication dans les tests et pour ses réponses aux demandes concernant le besoin métier.

Problèmes/Défis:

- Le sprint est assez court mais les premières fonctionnalités devraient être livrées dans les temps.

.

Echange avec le client

Email du client Bruno L en charge du projet:

Bonjour Fatima,

Nous testons actuellement l'API avec Postman comme lors de la démo de lundi. Cependant lorsque nous essayons de rentrer un texte dans la partie Body de l'application Postman, cela ne nous renvoie rien. Pourriez vous vérifier?

Cordialement,

Bruno L.

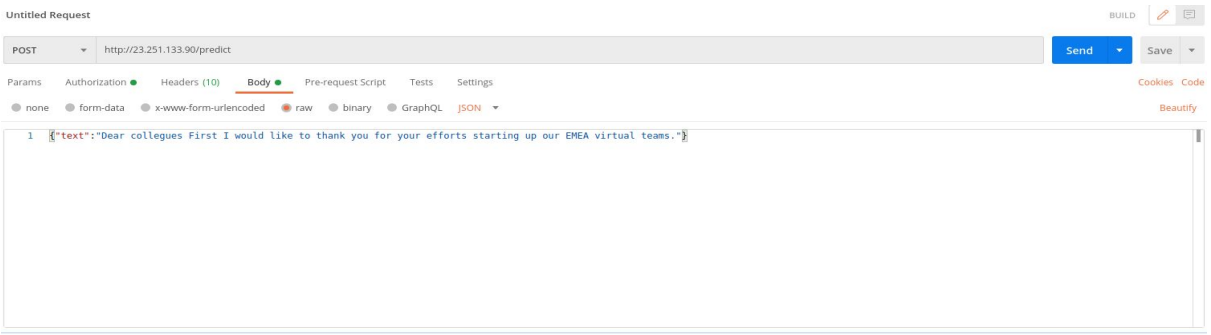
Réponse de Fatima M. développeuse en charge du projet :

Bonjour Bruno,

Nous allons résoudre ça. En attendant vous pouvez entrer votre texte en cliquant sur le bouton raw de l'api et entrer un texte comme ceci :

{text: "mon texte"}

Je vous met ci-dessous une capture d'écran avec un exemple :



Cordialement,
Fatima M.

Etat d'avancement numéro 3:

Nom du rapport : Développement d' une application de classification d'emails le 22/09/2020

statut du projet : dans les délais

Résumé: La première partie de l'application a été livrée la semaine dernière. L'équipe a bien entamé le nouveau sprint de l'application concernant les fonctionnalités restantes de l'administrateur. L'équipe va commencer à construire les KPI de la storie de l'utilisateur.

Developpement des fonctionnalités de l'administrateur:

- Les fonctionnalités de l'administrateur sont en cours de développement. Le login et la route sont mis en place.
- Il reste quelques statistiques et l'interface à terminer
- Il y a eu quelques bugs avec la base de données mais ça a été corrigé.

Remarques supplémentaires :

- Bruno a proposé de préparer une réunion avec toute l'équipe de son service afin que nous fassions une démonstration de l'application à l'ensemble de l'équipe.
  Cela permettra d'avoir plusieurs utilisateurs pour tester l'application et avoir un maximum d'informations dans le futur pour le modèle.

Problèmes/Défis:

- L'écriture des tests est plutôt longue. Il faudra travailler un peu plus pour respecter les délais

Echange avec le client

Email du client Bruno L en charge du projet:

Bonjour Fatima,

Actuellement je suis en train de monter l'équipe qui voudra bien tester l'application avec

leurs emails.
Combien de personnes auriez-vous besoin idéalement? et quels sont les profils?

Cordialement,

Bruno L.

Réponse de Fatima M développeuse en charge du projet :

Bonjour Bruno,

Une équipe de quatre volontaires parmi les commerciaux serait très bien.

Je vous appelle comme convenu demain à 10h pour les dernières modalités concernant la réunion avec cette équipe.

A demain.

Fatima M.

Annexe 11 tests fonctionnels

Module Enregistrement :

test n°1	criticité (entre 0 et 5)	Action	Attendu	Résultat
1	4	Aller sur le site et cliquer sur enregistrer	page d'enregistrement	ok
2	3	entrer le nom , l'email et un mot de passe	champs à remplir pour l'enregistrement	ok
3	4	soumettre le formulaire d'enregistrement	un message qui dit " Votre compte a été crée! Vous pouvez maintenant vous connecter"	ok
4	2	s'enregistrer avec un login qui existe déja	un message qui dit " Ce login existe déja. Merci de choisir un nouveau login."	ok
5	2	'enregistrer avec un email qui existe déja	Un message qui dit " Ce login_email existe déja. Merci d'en choisir un nouveau."	ok

Module Connection:

test n°	criticité (entre 0 et 5)	Action	Attendu	Résultat
6	4	Aller sur le site et cliquer sur login	page de connexion	ok
7	4	Entrer son email et son mot de passe	Champs à remplir pour la connection	ok
8	5	Soumettre le formulaire de connection	être connecté et avoir accès aux pages classer emails et account	ok
9	2	s'enregistrer avec une adresse qui n'existe pas	Un message qui dit "La connection a été refusé. Veuillez vérifier votre email et mot de passe"	ok
10	2	Entrer un mauvais mot de passe	Un message qui dit "La connection a été refusé. Veuillez vérifier votre email et mot de passe"	ok
11	2	Entrer un email sans mot de passe	Un message qui dit "La connection a été refusé. Veuillez vérifier votre email et mot de passe"	ok
12	2	Entrer un mot de passe dans email	Un message qui dit "La connection a été refusé. Veuillez vérifier votre email et mot de passe"	ok
13	1	Pré Enregistrer ses informations de connection	Une case à cocher avec "se souvenir de moi"	ok

Module Classer un email:

test	criticité (entre 0 et 5)	Action	Attendu	Résultat
14	5	Aller sur la page classer un email	page pour entrer un texte	ok
15	4	Entrer un texte	le texte apparaît dans le champ email	ok
16	4	cliquer sur envoyer	page de prediction et de choix de catégorie	ok
17	4	Voir la prédiction	prediction affiché et menu déroulant de choix de catégorie	ok
18	3	choisir sur le menu déroulant la prédiction choisie	Message avec “votre email a été classé”	ok

Module Administrateur:

test	criticité (entre 0 et 5)	Action	Attendu	Résultat
19	4	Aller sur la page et se connecter avec l’adresse administrateur	page avec le formulaire de connection	ok
20	4	Cliquer sur connexion	page administrateur avec le tableau de statistiques	ok
21	2	Ne pas être connecté comme administrateur et aller sur la page admin	page html avec message: “Erreur permission denied”	ok